EUNICE BSD

User Guide

WOLLONGONG

EUNICE SOFTWARE

User Guide

Manual Release 1.0
February 1989

The Wollongong Group, Inc.
1129 San Antonio Road
Palo Alto, California 94303
(415) 962-7100
TWX 910-373-2085
FAX (415) 968-4374

Acknowledgment

This software and documentation is based in part on the Fourth Berkeley Software Distribution under license from the Regents of the University of California. We acknowledge the Regents of the University of California and the Electrical Engineering and Computer Science Department at the Berkeley Campus of the University of California and other contributors for their roles in its development.

EUNICE USER GUIDE

CONTENTS

Preface

The EUNICE™ Software User Guide (P/N A005002-001) supplements the five-volume EUNICE documentation set. It is designed for experienced VAX™/VMS™ users who have limited or no knowledge of the UNIX® operating system. The intent of this guide is to orient users to EUNICE and the UNIX system, and to provide practical information on how to start using EUNICE immediately. If you are an experienced UNIX system user, read "Notes to UNIX Users" in this section.

This manual is organized into the following chapters:

Chapter 1: Introduction to EUNICE Software

Chapter 1 presents an overview of the EUNICE Software. EUNICE is discussed in relation to the UNIX and VMS operating systems. A functional description of EUNICE, as well as a discussion of its features are given.

Commands specific to EUNICE are also introduced.

Chapter 2: Getting Started

Chapter 2 provides the essential information needed to start working with EUNICE, such as logging in and out, and initializing home directories in the VMS and UNIX systems.

Chapter 3: Using UNIX and EUNICE

Chapter 3 introduces you to basic UNIX system commands and gives instructions for switching between VMS and EUNICE. This chapter also includes a tutorial on the UNIX full-screen text editor *vi* and an introduction to text formatting. Instructions for using the UNIX mail facility are also given.

Chapter 4: EUNICE, UNIX, and VMS

Chapter 4 discusses the interrelationship of EUNICE, UNIX, and VMS. The UNIX file system, the correspondence (mapping) of the VMS directory structure to the EUNICE directory structure; as well as filenames, file formats, and file-naming conventions are discussed.

Chapter 5: EUNICE Specifics

Chapter 5 discusses general programming concerns, such as EUNICE processes, compilers, and batch queues. The REX™ environment, which is the linked version of the EUNICE Runtime Library, is introduced.

Chapter 6: Troubleshooting

Chapter 6 presents a brief troubleshooting guide organized by command name. When you have problems working with a specific command, refer to this chapter before consulting your system administrator.

Related Publications

The *EUNICE User Guide* supplements the five-volume EUNICE documentation set. Installations with UNIX System V.2 or higher (with AT&T license) also receive Volume Ia, a reference manual for the Source Code Control System (SCCS).

The EUNICE documentation is similar to the UNIX documentation but is organized according to USENIX standards. While some documentation is EUNICE-specific, most of it consists of the UNIX documentation that has been annotated for EUNICE. Differences between the EUNICE and UNIX environments are indicated in the text by *EUNICE NOTES*. A description of each volume's contents follows.

Volume I: UNIX User's Reference Manual

The *UNIX User's Reference Manual* contains sections of the EUNICE manual pages. Manual pages, or *man pages*—the reference document for each EUNICE command—

- describe the command and its option
- show the command syntax
- list related files
- direct the reader to other references, which may be other commands or related documentation
- present any EUNICE-specific information (EUNICE NOTES)
- describe any known bugs

Man pages are divided into numbered sections. Each section is described in an introductory man page. The Commands(1), Games(6), and Miscellaneous(7) sections are included in Volume I. (Note that Wollongong does not provide support for Games.) EUNICE man pages are included, or EUNICE NOTES are added to existing man pages when they differ from the standard UNIX version. Man pages are also usually available online through the *man*(1) command by entering *man* followed by the command name. If the man command does not work, see your system administrator. Installation of manual pages is optional.

Volume Ia: SCCS Reference Manual

Volume Ia contains the SCCS man pages and the *Source Code Control System (SCCS) User's Guide*. It is distributed to sites licensed for SCCS (UNIX System V.2 or greater with the AT&T license).

Volume II: Programmer's Reference Manual

Volume II contains the *EUNICE Reference Manual*, which is EUNICE-specific. Note that this *Reference Manual* includes a troubleshooting guide ("Guide to Operations Problem Solving"). The rest of the manual, which is called the *UNIX Programmer's Reference Manual*, contains the following man pages related to programming:

- Section(2)—System Calls
- Section(3)—C Library Subroutines
- Section(3f)—FORTRAN Library Functions
- Section(4)—Special Files and related driver functions
- Section(5)—File Formats (of files created during the running of various programs)

Volume III: User's Supplementary Documents

This volume contains papers that supplement the manual pages in the *UNIX User's Reference Manual*. These papers cover such topics as learning how to use the UNIX system and its shells, electronic mail, text editing, document preparation, and games. Documents that do not apply to EUNICE have been omitted except for their title pages.

Volume IV: Programmer's Supplementary Documents (1 and 2)

Volume IV contains additional papers that supplement the manual pages in the *UNIX Programmer's Reference Manual*.

These documents are divided into two volumes:

> Volume 1 contains information on programming languages in common use, the *Berkeley Software Architecture Manual*, tutorials on interprocess communication, programming tools, and a discussion of the *curses* package.

Volume 2 includes documents of historical interest and papers related to programming languages not discussed in Volume 1. Documents that do not relate to EUNICE are omitted except for their title pages.

Volume V: EUNICE System Administrator's Guide

Volume V consists of the following documents:

- *EUNICE Product Release Notice*
- *EUNICE Administrator's Guide*
- *UNIX System Manager's Manual*
- *Sendmail Installation and Operation Guide*
- *Installation and Operation of UUCP*
- *Sendmail—An Internetwork Mail Router*

Additional documents cover data security, the portable C Compiler, writing *nroff* terminal descriptions, and setting up a dial-up network of UNIX systems.

The *EUNICE System Administrator's Guide* provides a site's system administrator with the information needed for the installation, maintenance, and administration of EUNICE. It includes the Product Release Notice, which accompanies the EUNICE distribution tape. The Product Release Notice gives information specific to the latest release, such as its contents, documentation notes, release notes, and completed requests for product modification.

Volume V includes a guide within a guide—the *EUNICE Administrator's Guide*, which is written expressly for EUNICE system administrators. This *Guide* gives step-by-step instructions for installing EUNICE and discusses setting up user accounts. Appendix B, *Guide to Installation Problem Solving* is a useful resource during installation.

The *UNIX System Manager's Manual* contains Maintenance Command(8) man pages that are relevant to EUNICE.

lanual Conventions

When the following symbols appear at the beginning of command lines, they are operating system prompts:

$	VAX/VMS Digital Command Language (DCL) prompt. Also, the UNIX Bourne Shell prompt. **Do not enter as part of your command.**
%	C Shell prompt (EUNICE's default prompt). **Do not enter as part of your command.**

When used as part of command lines, these symbols and font styles represent specific information:

[ ]	Optional parameters, arguments, or flags are enclosed in brackets.
...	Horizontal ellipses indicate that the preceding item can be repeated one or more times.
Bold	Literal commands to be entered exactly as shown. (Observe case-sensitivity rules.)
Italics	Filenames, variable parameters, or EUNICE/UNIX commands referred to in the text.
ALL CAPS	VMS commands.
Regular Type	System response to a user's command.

Case-Sensitivity Information:

VMS	Not case sensitive; enter commands in either uppercase or lowercase.
UNIX	Case sensitive; enter commands in uppercase or lowercase, as indicated.

Notes to UNIX Users

While working with EUNICE use VMS, not UNIX, terminal control characters.

VMS CONTROL CHARACTERS

CTRL Y	Interrupts execution of a command.
CTRL C	Cancels the execution of a job.
CTRL T	Interrupts execution of a command, displays a line of information—including process name, process state, used CPU time, and pages in physical memory—and resumes execution.
CTRL S	Stops scrolling of screen image.
CTRL Q	Releases screen image.
CTRL Z	Logs out from EUNICE, ends secondary shell processes, and exits from mail. Replaces CTRL D as the UNIX end-of-file (EOF) character.

Entering UNIX System Commands

Enter UNIX system commands on the blank line following the shell prompt (% or $).

```
%
```

This line, which is called the *command line*, consists of all the characters that you type after the shell prompt until you press Return.

A command line is made up of one or more entities, separated by one or more spaces. The first element is the name of a command or program. For example, a simple UNIX system command, such as *date*, is the only element on the command line:

```
% date
```

In more complex commands, the name of a command or program is followed by one or more *arguments* that modify the command's effect. Arguments are usually filenames or directories that the command operates on. *Options* modify commands; they usually begin with a hyphen and are followed by one or more letters. For example:

```
% ls -l filename
```

displays the long-form directory listing for the file *filename*. (This command is described in Chapter 3, "Basic UNIX and VMS System Commands.") If you look at the *ls*(1) man page in the *UNIX User's Reference Manual*, you will note that all of the options are enclosed in brackets, which means that they are optional. If you do not enter a directory or filename, a listing of all files in the current directory is given (except those beginning with "."). Files beginning with "." are also called "hidden files." See Chapter 3 for more details on *ls*.

NOTE: UNIX is case sensitive. Commands should be entered exactly as specified in the man pages in the UNIX User's Reference Manual.

Notes

Chapter 1: Introduction to EUNICE Software

This chapter introduces the EUNICE operating system environment. EUNICE is discussed in relation to the UNIX and VMS systems. A functional description of EUNICE, as well as a discussion of EUNICE features, follows. The chapter concludes with a description of commands specific to EUNICE.

METAPORT: The EUNICE UNIX/VMS Connection

When EUNICE 4.3 Software is installed on a VAX computer with VMS (4.0 and higher), VAX users have the option of working with VMS, UNIX, or an integrated VMS and UNIX system environment. UNIX-to-VMS connectivity is achieved through the Metaport approach. The Metaport is a software package that allows user-level programs developed for a foreign operating system to run on a host operating system with little or no source code modification. The EUNICE implementation of the Metaport is known as the REX environment, which is discussed in the next section.

Through the Metaport, EUNICE emulates UNIX on the VMS operating system, offering the features of a native UNIX port. From the user's standpoint, EUNICE is the same as UNIX.

With EUNICE, UNIX system calls are mapped to one or more VMS system services, without any modifications to the *kernel.* The kernel, which is the heart of an operating system, allocates resources and controls processes. UNIX programs can read and write to both VMS- and UNIX-style files, and can read VMS directory files as though they were UNIX directories. The EUNICE user can refer to files by either UNIX- or VMS-naming conventions.

EUNICE 4.3 contains most of the UNIX 4.3 commands and tools, except for networking tools, which are a separate Wollongong product, and the system administration tools. The majority of the system administration tools and the device drivers are controlled from the VMS level.

When you work with EUNICE, VMS remains the native operating system on the VAX computer, and the way you interact with VMS does not change.

Functional Description and Features

EUNICE permits VAX users to merge VMS and UNIX system commands and utilities, and to switch from VMS to UNIX as necessary. EUNICE resides on top of the VMS host-operating system, and duplicates the UNIX system with VMS system calls. VMS supplies device drivers, the operating system kernel, and I/O structures.

EUNICE 4.3 Software consists of the UNIX system utilities and the system call runtime library (EUNICE Runtime Library). The UNIX system utilities and applications provided with EUNICE 4.3 are derived from the native UNIX 4.3 utilities. The EUNICE Runtime Library contains the linkable object files used to create the REX environment. The REX (Runtime EXecutive) environment is the linked version of the EUNICE Runtime Library. It emulates the UNIX kernel and makes the UNIX utilities available on VMS. The REX environment, which is licensed separately, can provide runtime support for executable programs under VMS, even if EUNICE is not installed.

As shown in Figure 1-1, the UNIX system environment resides above the VMS operating system. EUNICE users access the UNIX system utilities and applications through the REX environment. VMS users can access DCL and VMS applications independently of UNIX. Users of both VMS and UNIX systems can access UNIX utilities and applications, as well as DCL and VMS applications.

Application programs written for UNIX 4.3 can be run under VAX/VMS with little or no modification to the source code (although often source code must be relinked and sometimes it must be recompiled). In many cases, UNIX 4.3 object code can be successfully relinked without the need for full source code.

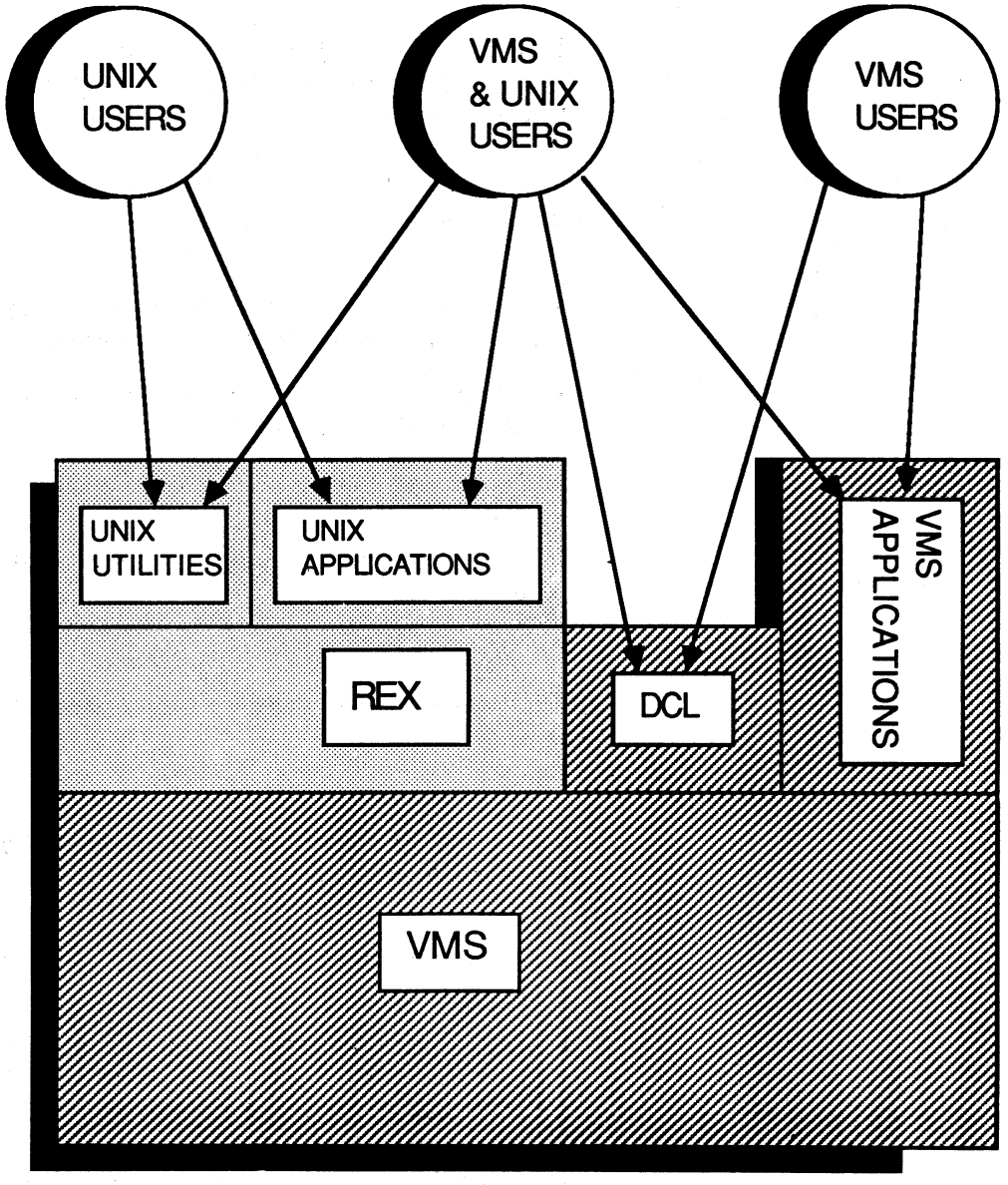


Figure 1-1. The Relationship of EUNICE, UNIX, and VMS

EUNICE Command Overview

EUNICE 4.3 commands include almost all of the standard UNIX 4.3 commands, as well as some EUNICE-specific commands. The EUNICE-specific commands are primarily utilities that facilitate working with both operating systems.

There are three kinds of utilities: user-level, EUNICE initialization utilities, and UNIX object libraries in VMS-style object code format.

For example, EUNICE user-level utilities, which are located in */usr/eun*, include:

- *unixtovms*—converts files from the 512-byte, fixed-record length UNIX format to the variable-length VMS format.
- *vmstounix*—converts VMS variable-record length files to UNIX fixed-record length files.
- *vms*—allows you to call a VMS command from the UNIX shell.
- *vmsmail*—sends mail from the UNIX shell to VMS users only.

These utilities are discussed in more detail in the following chapters.

Two of the EUNICE initialization and management utilities located in */etc/eunice* are

- *STARTEUNICE.COM*—the DCL command file that initializes all of EUNICE.
- *ROOT.COM*—the DCL command file that maps VMS directories to the UNIX file system tree. This file is called by *STARTEUNICE.COM*.

UNIX libraries in VMS-style object code format are located in */usr/libvms*; for example, *libc.olb*, which is the standard UNIX C library (*libc.a*) in VMS-style object code.

Notes

Notes

Chapter 2: Getting Started

This chapter gives you the basic information needed to start working with EUNICE 4.3 Software, such as logging in and out, and initializing home directories in the VMS and UNIX systems.

Logging In and Logging Out

EUNICE runs on top of the VMS operating system. To log in to EUNICE, first log in to VMS according to your standard procedures. Then you can invoke EUNICE by issuing a command at the DCL prompt.

When you log out of EUNICE, you return to VMS.

Invoking EUNICE from VMS

To invoke EUNICE from the VMS operating system, enter the following command at the DCL prompt:

```
$ UNIX
```

NOTE: If this command fails, then you do not have this symbol defined in your LOGIN.COM file. See your system administrator. In the meantime, you can always invoke EUNICE from VMS by entering:

```
$ @TWG$ADMIN:CSHELL.COM
```

EUNICE responds with the date and time of your last login and the message of the day. If you have new UNIX mail, a message informs you of it. EUNICE then displays the C Shell prompt (%) on an otherwise empty line. When this prompt is displayed, you can enter EUNICE and UNIX-supported commands.

Unlike native UNIX, EUNICE does not automatically place you at your login directory when you log in. You remain at the current directory position in the VMS file system upon invoking EUNICE, and can change to another directory via the UNIX *cd* command. (See the section "Changing Your Working Directory" in Chapter 3 for instructions.) From your login, or home directory, you are able to use the UNIX commands, tools, and utilities available with EUNICE.

If you intend to perform most of your work within the UNIX system environment, it is possible to invoke EUNICE automatically upon log in, and to log out of EUNICE directly. Instructions for doing so are given in this chapter (see "Setting Up the LOGIN.COM File").

Ending the EUNICE Session

To end your EUNICE session and return to VMS, you can type:

```
% logout
```

at the UNIX system prompt. Alternately, you can press **CTRL Z**, which ends the EUNICE Shell process (and all other EUNICE-initiated background processes in progress) and returns you to VMS.

EUNICE User Accounts

Each EUNICE user has a unique login name. Your login name may be your initials, or an abbreviated version of your name. It is the same as your VMS username, but in lowercase.

Your home directory is your private working space. As shown in Figure 2-1, all UNIX directories branch out from the *root* directory (/). The UNIX directory structure resembles an upside-down tree, with the *root* at the top. Users are grouped in the next level of directories, in a directory that is usually called */u* in EUNICE. Individual users have directories that branch out from the */u* directory. For example, assume that two users have the login names of *jkl* and *ron*, respectively. We can follow a path from the *root* directory to the */u* directory to each user's home directory. Then one path follows the branch *jkl* and the other branches to *ron*. When we speak of a *pathname* in the UNIX system, we mean the complete path down to the current directory or file in which you are working. (*Pathname* is the UNIX equivalent of the VMS term *file specification.*) For example, the home directories for the two users mentioned above are */u/jkl* and */u/ron*.

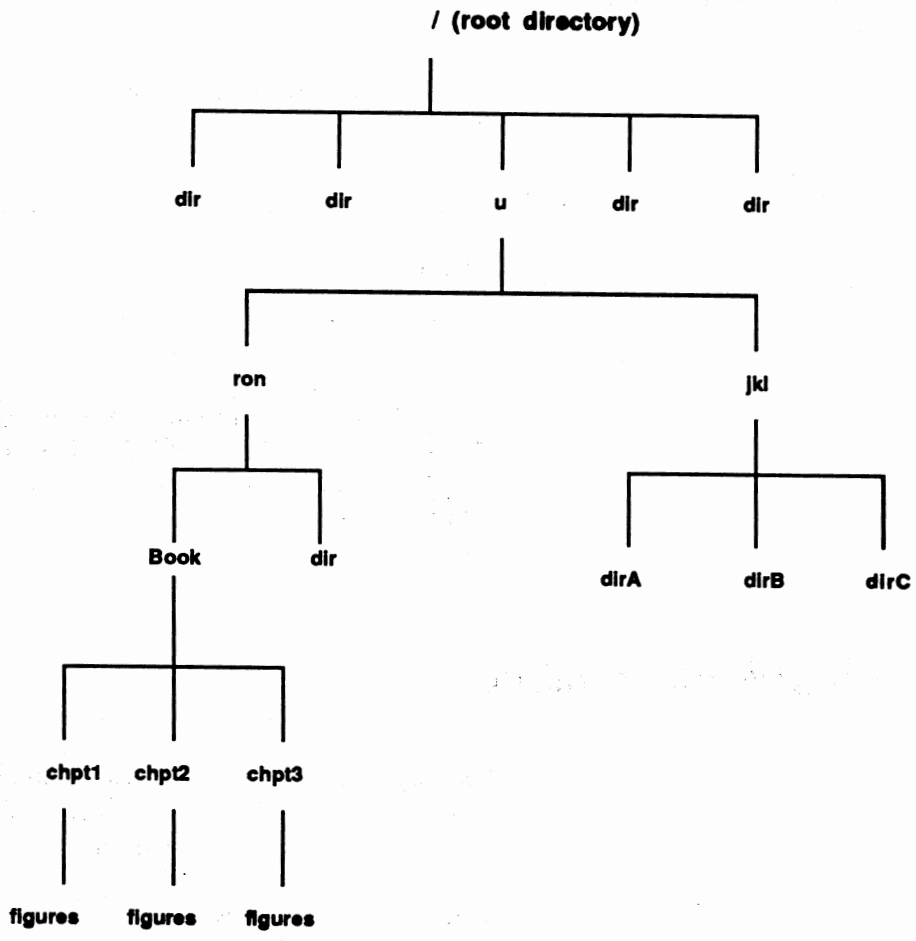


Figure 2-1. Following UNIX Paths through the File Structure

From their home directories, users create other directories to hold related files. Each file in the UNIX system has a pathname and can be traced backward through all of the higher directories to the *root* directory. For example, if *ron* creates a directory called *Book* and creates files within that directory called *chpt1*, *chpt2*, and *chpt3*, the pathname to Chapter 1 is */u/ron/Book/chpt1*, and to Chapter 2 is */u/ron/Book/chpt2*. Note that each chapter is also a directory that contains a file called *figures*.

Now let us return to the concept of a home directory. (The UNIX file system is discussed in more detail in Chapter 4.)

Six files must be included in your home directory to ensure that your environment is properly set whenever you log in: LOGIN.COM, *.login*, *.cshrc*, *.exrc*, *.mailrc*, and *.profile*.

LOGIN.COM is a VMS command file that should be modified in order to invoke EUNICE. The other files are UNIX system files that need to be placed in your home directory. Generic copies, or templates, of these six set-up files exist in the directory */usr/skel*. You can copy these files into your home directory and edit them as needed. A discussion of how to copy and edit UNIX system files is given in Chapter 3. (See "Copying, Renaming, and Moving Files" and "An Introduction to the Vi Text Editor.")

Each of these set-up files is discussed below. Enough information is given to get started working with EUNICE. To fine-tune these files to their specifications, more advanced users should refer to Section 6.2, "Setting Up User Accounts," in the *EUNICE Administrator's Guide*, and to the following man pages and documents in the *User's Supplementary Documents*:

.login	*csh*(1) and *An Introduction to the C Shell* (Basic Utilities).
.cshrc	*csh*(1) and *An Introduction to the C Shell* (Basic Utilities).
.exrc	*Ex Reference Manual* (Text Editing).
.mailrc	*Mail Reference Manual* (Communications).
.profile	*An Introduction to the UNIX Shell* (Basic Utilities).

Setting Up the LOGIN.COM File

As mentioned, the LOGIN.COM file is a VMS command file that is executed when you log in to the VMS operating system. This file, which exists in your VMS home directory, needs to be modified if you wish to use UNIX utilities from VMS. Figure 2-2 shows a sample LOGIN.COM file in */usr/skel*. It is called VMSLOGIN.COM to avoid overwriting your current LOGIN.COM file if you should copy the */usr/skel* files into your home directory. (Note that the exclamation point [!] indicates that comments follow.) You may copy this file from the */usr/skel* directory into your home directory and edit it using EDT. For example, EUNICE can be automatically invoked upon login to the VAX by adding one of the following lines, depending on the desired shell, to the LOGIN.COM file:

For the C Shell:

```
$ @TWG$ADMIN:CSHELL.COM
```

For the Bourne Shell:

```
$ @TWG$ADMIN:SHELL.COM
```

To bypass VMS when logging out, add the following line as the last line of your LOGIN.COM file:

```
$ LOGOUT
```

The LOGIN.COM file may also be modified so that you can use UNIX utilities while in the VMS environment. See the section "Using UNIX Utilities from DCL" in Chapter 3 for instructions.

```
$ ! ------------------------------------------------------------------------------------------

$ ! | VMSLOGIN.COM  -  a sample LOGIN.COM for users who wish to use UNIX     |
$ ! | utilities from their VMS environment. The required commands should    |
$ ! | be included in the user's own LOGIN.COM, or this file may be run      |
$ ! | directly from the user's LOGIN.COM file.  This includes EUNICE        |
$ ! | startup commands for your convenience.                                |

$ ! ------------------------------------------------------------------------------------------
$ !
$ !  Note:  Remember that the EUNLOGIN and EUNLOGOUT commands have to be
$ !         run to use EUNICE commands when you are in VMS.  To simplify
$ !         this, you use the UNIXIN and UNIXOUT symbols below.
$ !
$ !  Set up for vi(1) and other programs that access /etc/termcap.
$ !  This is equivalent to UNIX's "setenv TERM vt100".
$ !  (This supersedes the logical set system-wide in ETC:EUNICE.COM.)

$ DEFINE TERM "vt100"     !vt100 is site-dependent.
$ !
$ !
$ !                       Set up VMS symbols
$ !
$ BSD                     :== @TWG$ADMIN:CSHELL.COM          ! to login to EUNICE
$ CD                      :== $TWG$USR:[EUN]CD. CD           ! special 'cd' for VMS use
$ CMP                     :== $TWG$BIN:[000000]CMP. CMP      ! use the UNIX compare utility
$ CSH*ELL                 :== @TWG$ADMIN:CSHELL.COM          ! another login to EUNICE
$ CSS                     :== $TWG$BIN:[000000]CSH. CSH      ! use: CSS -F CSHELLSCRIPTNAME
$ DATE                    :== $TWG$BIN:[000000]DATE. DATE    ! get the UNIX date format
$ DF                      :== $TWG$BIN:[000000]DF. DF        ! show disk space
$ HOME                    :== SET DEF SYS$LOGIN              ! return to login directory
$ LS                      :== $TWG$BIN:[000000]LS. LS        ! use ls(1) for UNIX names
$ MAKE                    :== $TWG$BIN:[000000]MAKE. MAKE    ! UNIX compiler 'make' program
$ MKDIR                   :== $TWG$BIN:[000000]MKDIR. MKDIR  ! use UNIX directory creation
$ MORE                    :== $TWG$USR:[UCB]MORE. MORE       ! type a file in screenfuls
$ PAGE                    :== $TWG$USR:[UCB]PAGE. PAGE       ! like 'more', only page by page
$ PWD                     :== $TWG$BIN:[000000]PWD. PWD      ! show UNIX directory spec.
$ RESUME                  :== STOP/ID=0                      ! return to EUNICE after suspend
$ RM                      :== $TWG$BIN:[000000]RM. RM        ! use the UNIX delete command
$ UNIX                    :== @TWG$ADMIN:CSHELL.COM          ! yet another login to EUNICE
$ UNIXIN                  :== $TWG$ADMIN:EUNLOGIN            ! set up to run EUNICE commands
$ UNIXOUT                 :== $TWG$ADMIN:EUNLOGOUT           ! release EUNICE resources again
$ VI                      :== $TWG$USR:[UCB]VI. VI           ! use the UNIX screen editor

$ !
$ ! If WIN/TCP is present, remember to define the finger(1) from THAT software
$ ! rather than the following one:
$ !

$ FINGER    :== $TWG$BIN:[000000]FINGER. FINGER    ! user info. lookup program
$ !
$ EXIT
$ !
```

Figure 2-2. LOGIN.COM Template

Setting Up Your UNIX Home Directory Files

The EUNICE-related files that are required in your home directory are *.login* and *.cshrc* for the C Shell (or *.profile* for the Bourne Shell). The *.exrc* file is required for the *vi* editor, and the *.mailrc* file for the EUNICE *mail* facility. Wollongong provides templates of these files, called *dot* files because they begin with a period, in the */user/skel* directory. You can copy these files into your home directory and make modifications for your user account using the *vi* text editor. If you are unfamiliar with *vi*, see "An Introduction to the *vi* Text Editor" in Chapter 3 for instructions on editing your set-up files. There are two versions of each of these files in the */usr/skel* directory. The filenames that end in *.csh* and *.sh* are commented versions of the file. To understand how these files work, EUNICE users should review the commented versions of the files before copying the uncommented versions into their home directories. Lines that begin with the pound sign (#) are comments.

The following chart lists each EUNICE template file and gives its pathname. See the section on "Copying, Renaming, and Moving Files" in Chapter 3 for instructions on copying files.

Filename	Pathname
.login	*/usr/skel/.login.csh*
.cshrc	*/usr/skel/.cshrc.csh*
.exrc	*/usr/skel/.exrc.csh*
.mailrc	*/usr/skel/.mailrc.csh*
.profile	*/usr/skel/.profile.sh*

The .login File

The *.login* file is the UNIX equivalent to LOGIN.COM. The commands in this file initialize the UNIX environment upon login. Figure 2-3 shows the contents of the *.login* template.

The C Shell supports an *alias* feature, which is similar to the foreign command feature in VMS. You can set up aliases in your *.login* and *.cshrc* files to make standard commands perform nonstandard features, or to define new commands.

How does the alias feature work? The line on which you enter UNIX commands following the prompt is called the command line. When you enter a command line, the C Shell breaks it into separate commands. *Alias* performs a string substitution on

the command line according to the aliases you set up in your *.login* and *.cshrc* files. The *.login* is executed when logging into EUNICE. *.Cshrc* commands are executed for each subprocess *.csh*.

Create *alias* commands in this format:

```
$ alias entered-command executed-command
```

where the *entered command* is the command normally entered after the C Shell prompt, and the *executed command* is the string that the alias feature substitutes for the *entered command*. If the *executed command* contains spaces, enclose it within single or double quotation marks. For example:

```
$ alias art "cd /u/ron/Book/chpt3"
```

If this *alias* is entered in your *.login* file, each time you enter *art* after the prompt, the directory changes to *u/ron/Book/chpt3*. This is useful if you frequently work with files in Ron's Chapter 3 directory.

Besides pathnames, you can set up aliases for variations on commands, such as *cu12* for

```
$ alias cu12 "cu -s 1200 -a /dev/null -l /dev/tty8"
```

or abbreviated command names, such as *ll* for *ls -l*.

The *.login* template file includes standard useful aliases; for example, *lo* for *logout*. For a listing of your defined aliases, enter *alias* and press Return.

```
# A template .login file for EUNICE.
# SCCS_ID - "@(#)login.csh 1.2  (TWG) 87/07/08 "

# When using redirection (">"), cshell warns if file already exists:
set noclobber

# Prevent logout with ctrl-z "end-of-file" signals:
# set ignoreeof

# Set the research path for the csh:
setenv PATH .:/usr/ucb:/bin:/usr/bin:/usr/eun:/usr/local/bin

# Csh history will remember the last 30 events:
set history=30

# Remember the last 20 events for your next cshell login:
set savehist=20

# Set prompt to event number and csh prompt:
set prompt='\! % '

# ALIASES.
#   Note:  which(1) only recognizes aliases set in .cshrc, so you may
#   wish to move some to that file. That can, however, slow performance.

# Compiler switches for cc(1) and f77(1):
# With Eunice you have your choice of assemblers and loaders. You
# may choose UNIX or VMS executable style. The default is UNIX style.
# You may switch to VMS style using the vmsobj alias and switch back to UNIX
# using the unixobj alias.
#
# Have compiler use UNIX assembler and loader:
alias unixobj 'unsetenv AS_IMAGE; unsetenv LD_IMAGE'

# - or have compiler use VMS assembler and loader:
alias vmsobj 'setenv AS_IMAGE /usr/eun/vmsas; setenv LD_IMAGE /usr/eun/vmsld'

# This will make cc(1) use the UNIX assembler and loader now:
# unixobj

# This can make cc(1) use the VMS assembler instead:
# vmsobj

# Allow use of VMS's EDT from the csh:
alias edt  'vms -t edit/edt'

# Allow use of the VMS mailer program from the Cshell:
alias vmail  'vms -t mail'
```

Figure 2-3 the .login template (continued on next page)

```
# Set up some other shorthand calls for frequently used commands:
alias cp cp -i
alias a alias
alias cl clear
alias h history
alias ll ls -l
alias lo logout
alias mv mv -i
alias pu 'vms purge &'
alias rew 'vms set mag/rew mt0:'
# alias rm /bin/rm -i
alias so source
alias sus suspend
```

Figure 2-3. The .login Template (continued)

NOTE: After reading the commented lines—those beginning with #—you can delete them.

The .cshrc File

The *.cshrc* file is the initiation file for the C Shell.

Aliases can also be added to the *.cshrc* file.

Figure 2-4 shows the contents of the *cshrc.csh* template.

```
# A template .cshrc file for EUNICE.
# SCCS_ID - "@(#)cshrc.csh 1.1  (TWG) 87/06/22 "
# Login time can be sped up by removing comments from
# your own version of this (and other) login files.

# Remember:  which(1) only recognizes aliases which are
# set in .cshrc; however, as this file gets larger, performance slows down.

# DO NOT REMOVE!  Allows a background task to notify the shell immediately.
set notify

# Set up a path for cd.
set cdpath=(~ /usr/eun /usr/local /usr/man)
```

Figure 2-4. The .cshrc Template

The .exrc File

The *.exrc* file sets up the operating environment for the *vi* and *ex* editors. You can copy this file from the *.exrc* template, or you can read the *vi*(1) Reference Manual and *An Introduction to Display Editing with Vi* (found in the "Text Editing" section of the *User's Supplementary Documents*) for suggestions on customizing your environment. Delete all comment lines (with pound signs) from the file before using it.

NOTE: .exrc will not work with comments in it.

Figure 2-5 shows the contents of the *exrc.csh* template.

```
# The .exrc file is read every time vi, ex or ed is started by the user.
# See the text editing sections in Volume III for more details.
# "set noautoindent" turns off the automatic justifying feature of these editors
# "set noignorecase" In the case of "regular expressions" for these editors,
#  uppercase letters are not mapped to lowercase.
# "set noshowmatch" will not place the cursor for one second on the matching
#  left '{' or '('.
# "set redraw" This feature causes the line to be redrawn as you insert
#  characters on a line.
set noautoindent
set noignorecase
set noshowmatch
set redraw
```

Figure 2-5. The .exrc Template

The .mailrc File

The *.mailrc* file is used whenever the UNIX *mail* command is invoked. The template file can be used without alteration.

Figure 2-6 shows the contents of the *mailrc.csh* template.

```
# See your mail(1) man pages for further information
# The following "options" are included for mail.
# The .mailrc file is read whenever mail is started.
# "set ask" will ask for a subject line.
# "set askcc" when your message has been completed
# you will asked to send copies to other users.
# "set dot" this will interpret a "period" at the beginning
# of any line as a termination of the message.
set ask
set askcc
set dot
```

Figure 2-6. The .mailrc Template

The .profile File

The *.profile* file is the Bourne Shell equivalent to *.cshrc*. This file contains information that is read by the Bourne Shell. The commands in this file set up your search path, file permissions, and the terminal type. Edit *.profile* so that your login name replaces the term *loginname*. Replace the TERM (terminal) type *vt100* with your terminal type, if it differs. (The */etc/termcap* file lists supported terminal types.)

Figure 2-7 shows the contents of the *.profile* template.

```
: '.profile - Bourne Shell, sh(1), initialization script'
: SCCS_ID - '@(#)profile.s  1.1 (TWG) 87/06/22 '

: 'Pathnames'
: 'set HOME="/u/loginname" - uncomment if set'
: 'export HOME'
: 'USER="loginname"'
: 'export USER'

SHELL=/bin/sh; export SHELL
EDITOR=/usr/ucb/vi; export EDITOR

: 'Search lists'

PATH="/usr/ucb:/bin:/usr/bin:/usr/eun:/usr/local/bin:"
export PATH

: 'Terminal support'

TERM="vt100"; export TERM
```

Figure 2-7. The .profile Template

Notes

Notes

Chapter 3: Using UNIX and EUNICE

This chapter introduces basic UNIX system commands, describes UNIX's text editor and formatter programs, and explains how to issue VMS commands from EUNICE and vice versa. Instructions for using the UNIX mail facility are also given.

Basic UNIX and VMS Commands

This section introduces some of the basic UNIX commands and gives their VMS equivalents. A brief description of the UNIX commands follows.

For complete information, refer to the Commands(1) section of the *UNIX User's Reference Manual*, which describes each UNIX command on a separate page called a man (manual) page. The man page gives the command syntax and describes all options. Man pages are also available on-line on most systems. Enter the *man* command followed by the command name. For example:

```
% man ls
```

displays the on-line man page for the *ls* command.

List Contents of Directory—ls

The UNIX list directory command, *ls*, displays the names of all files and directories in the current or specified directory. It is equivalent to the VMS DIR command. Among the many options described in the *ls* man page is *-l*, which lists the directory in the *long* format. In this format the permissions, owner, number of bytes, date last changed, and file name are given for each entry. The *-a* option lists *all* entries. Without this option, files preceded by a period, such as *.profile*, are not

listed. The *-R* (recursive) option displays the filenames in the current directory, as well as files in any subdirectories under the current directory. Note that UNIX does not provide version numbers of files. With EUNICE, your system administrator can set version numbering either on or off at the system level in the EUNICE.COM file.

Examples of the *ls* command and three of its options follow. Entering the *ls* command without options produces a listing similar to this:

```
% ls
chpt_1
mail.mai
nic.log
```

Entering the *ls* command with the *-l* (long) option displays the contents of the working directory in this format:

```
% ls -l
total 15
drw-rw-r--1     jkl      9987 Jan 04 17:10 chpt_1
-rw---------1   jkl     37794 Jan 22 16:29 mail.mai
-rw-r-------1   jkl      1048 Jan 10 12:03 nic.log
```

where the first column shows the file permissions (see Chapter 4, "UNIX File Permissions"), followed by the owner's login name, number of bytes, date and time of last change to the file, and the name of the file or directory. Note that a directory is indicated by *d* at the beginning of the line.

Entering the *ls* command with the *-a* (all) option also lists the files that begin with a period (*dot*), as well as the current directory (.), and the parent directory (..). These *dot* files reside in your home directory:

```
% ls -a
 .
 ..
 .cshrc
 .login
chpt_1
mail.mai
nic.log
```

The options can also be grouped together. For example, the *ls -la* command lists all files in the long format, including files that begin with a period.

For comparision from VMS, entering the VMS DIR command displays a listing of the files in the current directory, including the *dot* files. For example:

```
$ DIR

 .CSHRC;1          HISTORY;1        .LOGIN;1
 .MAILRC;1         CHPT_1.TXT;1     MAIL.MAI;1
 LOGIN.COM;1       NIC.LOG;1
```

Copying, Renaming, and Moving Files—cp, mv

The UNIX copy command, *cp*, is equivalent to the VMS COPY command. It duplicates a specified file (*file1*) by copying it into another specified filename (*file2*), writing over any contents of *file2*. If *file2* exists, be certain that its present contents can be deleted. For example:

```
% cp spec spec.back
```

spec.back is now identical to *spec*, even if it already existed as another file.

You can rename files and directories, or move a group of files into another directory with the UNIX move command, *mv*. The VMS RENAME command performs a similar renaming function. The *mv* command in the following example moves *file1* into *file2*, writing over any existing *file2* (*file1* is renamed *file2*).

```
% mv file1 file2
```

Use the move command, *mv*, as follows to rename directories:

```
% mv dirname newdirname
```

To move one or more files into a directory, enter:

```
% mv file ... dirname
```

Removing Files—rm

The UNIX remove command, *rm*, deletes one or more files. For example:

```
% rm file1 file2
```

deletes *file1* and *file2*.

Unless the *-i* interactive option is in effect, *rm* removes the file without confirming that you want it deleted. This command is equivalent to the VMS DELETE command. With the *-i* option, *rm* is similar to the VMS DELETE/CONFIRM command.

The *-r* option enables you to delete a directory and its subdirectories and files with one command. For example:

```
$ rm -r Book
```

removes the directory *Book* and all of its contents.

Displaying a File's Contents—cat, more

The UNIX *cat* (concatenate and print) command displays the contents of a file on the screen. If more than a screenful of text is to be displayed, you can freeze the screen by pressing CTRL S and resume scrolling by pressing CTRL Q. The *cat* command is similar to the VMS TYPE command.

The *cat* command also appends files. For example:

```
% cat file1 file2
```

concatenates *file1* and *file2* and displays them on the screen. This output can also be redirected to another file by using the *redirect output* symbol (>). The redirect arrow indicates to the shell that the output is to be directed to the specified file instead of to the *standard output*, which is the terminal screen. For example:

```
% cat file1 file2 > file3
```

appends *file1* to *file2* and places the combined files in *file3*. *File1* and *file2* remain unchanged. If the redirect arrow were not present, the output would be displayed on the terminal screen. If *file3* already exists and you want to append the output to the end of the file (as opposed to writing over it), use the double redirect arrow (>>) instead of >. Similarly, input can be redirected to a file by using the *redirect input* (<) symbol.

The UNIX *more* command enables you to display the contents of a file screen by screen, or line by line. This command is similar to the TYPE/PAGE command in VMS. Press Return to display the next line, or press the spacebar to display the next screenful of text.

If you enter a question mark (?), a help list for the *more* command is displayed.

Working with Directories—mkdir, rmdir

As mentioned, UNIX directories usually consist of related files. For example, if you are working on a book that consists of chapters, you can create a directory called *Book* by entering the make directory command, *mkdir*:

```
% mkdir Book
```

The equivalent VMS command is CREATE/DIRECTORY DIRECTORYNAME.

Since UNIX filenames are usually lowercase, you may want to distinguish files from directories. Directory names, for example, might begin with a capital letter, or be all capital letters.

You can either create files within this directory, such as chpt1, chpt2, chpt3, chpt4, etc., through the UNIX text editor, or move files from another directory using the *mv* command.

You can delete a directory with the remove directory command, *rmdir*, only if it is empty. First remove all of the files within a directory and then issue the *rmdir* command. For example:

```
% rmdir Book
```

deletes the directory named *Book*. As mentioned, the *rm* command with the *-r* option deletes a specified directory and all of its contents.

Changing Your Working Directory—cd

The UNIX change directory command, *cd*, enables you to move to the directory specified, or if no directory is specified, to return to your home directory from any other directory.

This command is similar to entering the VMS SET DEFAULT DIRECTORYNAME command from the default directory.

The *cd* command followed by two periods (*cd* ..) takes advantage of the relative pathnames in UNIX. Since the UNIX files are organized hierarchically, files and directories can be described in relation to other files and directories. For example, the *chpt1* file in the directory *Book* can have a subdirectory called *figures*. To move from the *figures* directory to the *chpt1*

directory; that is, to move up one level in the UNIX hierarchy, enter:

```
% cd ..
```

From the *chptl* directory you can move up to the *Book* directory by re-entering *cd...* This action is equivalent to issuing the VMS command SET DEFAULT [-].

Displaying Your Working Directory—pwd

The UNIX *pwd*—print working directory—command is useful for orienting yourself within the UNIX hierarchy of files and directories. When you issue this command, the complete pathname of your current UNIX directory is displayed. Use the *cd* command to change to another directory.

The equivalent command in VMS to show the current directory is SHOW DEFAULT.

Process Status—ps

The *ps* (process status) command provides status information for currently active system processes. Only your processes are displayed unless you specify all processes. The process identification number (PID), originating terminal, amount of CPU time already consumed, and an indication of which commands are running, is displayed on your terminal screen.

The *ps* command format is as follows:

```
% ps [options]
```

The most frequently used options are *a*, which requests information for all terminals, and *l*, which gives a more complete description of processes status.

See the *ps*(1) man page in the *UNIX User's Reference Manual* for more information.

Search File for a Pattern—grep

Use the *grep* command to find a specific character string in one file, or in several files simultaneously. Whereas search-and-find editor commands locate patterns in a single file, *grep* can find patterns in more than one file.

The *grep* command has the following format:

```
% grep [options] ... pattern [file] ...
```

where:

options include:

-*v* = (variant) prints all lines except those that match the pattern.

-*c* = (count) prints only the number of matching lines.

-*l* = lists the names of the files that contain the indicated pattern.

-*n* = each line is preceded by the file's line number.

-*i* = ignores case; upper- and lowercase letters are considered identical.

pattern = the character string to be found.

file = the filename(s) to be searched.

For example:

```
% grep -n talent memo.*
```

displays all occurrences of *talent* by line number in the *memo* files. (The asterisk [*] is the UNIX wildcard operator; it stands for any character or characters. Therefore, *grep* searches through *memo.1*, *memo.2*, *memo.3*, memo.save, etc.)

To search for more than one word, enclose the string in quotation marks:

```
% grep "Project status" memo.*
```

Refer to the *grep* man page for a description of all available options.

Issuing VMS Commands from EUNICE

To enter VMS commands without exiting EUNICE, use either the *vms* or *suspend* command. In general, the decision to use one instead of the other is based on the number of VMS commands to be entered. Use the *suspend* command when you need to work more extensively with VMS.

The vms Command

Use the *vms* command to enter a VMS command while still logged on to EUNICE. For example, to issue the VMS DIR command from EUNICE, enter:

```
% vms dir
```

For interactive commands, such as EDT or HELP, include the *-t* flag before entering the VMS command.

```
% vms -t edt
```

The *vms* command enables you to access any VMS command while logged on to EUNICE. There are two distinctions to keep in mind. First, when editing text, precede any UNIX metacharacters with a backslash (\ ) to prevent EUNICE from interpreting the characters. These characters are as follow:

($ * [ ] ? { } - > < & ! ^ | ; ( ) \ ' ' ")

For example:

```
% vms type twg$tcp:[netdist.etc]gated.conf
```

or you can enter the vms command followed by metacharacters by enclosing the vms command string in quotation marks.

For example:

```
% vms "dir/full file.*"
```

Second, you cannot set VMS symbols or logical names using the *vms* command. Use the *suspend* command to set them without exiting from EUNICE.

The suspend Command

The *suspend* command spawns a new VMS DCL process from your current C Shell process. To issue the *suspend* command, enter:

```
% suspend
```

The VMS $ prompt is displayed. Enter VMS commands as if you were logged on to VMS directly. To return to EUNICE, enter:

```
$ RESUME
```

(The RESUME foreign command must be part of your LOGIN.COM file in order to use this command. It is set equal to STOP/ID=0.)

Using UNIX Utilities from DCL

EUNICE commands such as *ls* or *ps* can be run from DCL by defining the command as a *foreign command.* You can set foreign commands through a variation of the assignment statement either:

- as a DCL command (if the command is to be used once), or
- as a line in the LOGIN.COM file (if the command is to be used often).

A foreign command is defined as follows:

$ GLOBAL-SYMBOL-NAME :== "$FILE-SPEC"

where *global-symbol-name* is the UNIX command name, and *$file-spec* identifies the file containing the foreign command. The file specification must be prefaced with a dollar sign and include the device, directory, and filename.

For example, the following foreign command enables you to invoke the UNIX *ls* command directly from DCL:

```
$ LS :== $TWG$BIN:[000000]LS. LS
```

where:

ls = the UNIX command.

TWG$BIN = the logical that corresponds to *device:[eunice.bin].*

[000000] = the top-level directory.

ls = the file name of the UNIX command's location.

ls after the period = the UNIX command.

To determine the UNIX pathname for a command, enter the UNIX command *whereis* followed by the command name. For example:

```
% whereis ls
```

tells you that the pathname to *ls* is */bin/ls* and specifies the manual page directory in which the command can be found. The *which* command gives the pathname only.

To invoke the UNIX *more* command directly from DCL, follow these steps:

1. Determine the pathname for *more* using the UNIX *whereis* command: */usr/ucb/more*.

2. Convert the directory after the *root* (/) directory into the logical EUNICE equivalent by prefixing it with *TWG$* and capitalizing it; for example: */usr* equals *TWG$USR*.

3. Since the directory immediately above *more* is not a top-level directory, which is represented as *000000* in VMS, it is simply the directory name *ucb*. Therefore, the foreign command is as follows:

```
$ MORE :== $TWG$USR:[UCB]MORE. MORE
```

NOTE: The cd command is actually a C Shell internal command. To enable you to invoke it from VMS, a special VMS executable—/usr/eun/cd—has been provided.

To simplify an uppercase letter, use a caret (^). For example:

```
$ HOSTNAME ^VAX
```

sets hostname to Vax. To edit a file with a capitalized name, such as FILE, enter the following line:

```
$ vi ^F^I^L^E^
```

An Introduction to the Vi Text Editor

The full-screen editor available under EUNICE (equivalent to EDT) is *vi*, pronounced *vee-eye*. This section introduces *vi* and gives a brief tutorial. For more information, refer to *An Introduction to Display Editing with Vi* and *Advanced Editing on UNIX* in the text editing section of the *UNIX User's Supplementary Documents*. There are also many commercially available books that describe UNIX text editing and formatting.

The main difference between *vi* and EDT in keypad mode is that *vi* has one mode for entering commands and another for entering text. You cannot enter text from the command mode or enter commands from the *vi* text entry mode. At first *vi* seems confusing because you have to keep track of modes. Changing modes, however, is simple: press the ESC (or RUB OUT) key to switch to command mode.

After you create a new file by entering:

```
% vi filename
```

at the UNIX prompt, a screen with either blank or numbered lines is displayed, depending on the variables in your *.exrc* file. If the file exists, the first screenful of the file's contents is displayed.

If you were using the keyboard mode of EDT, you could begin to enter text or move the cursor around the screen. In *vi*, however, when a file is created or retrieved, the command mode is in effect. You can move the cursor but cannot enter text until you enter a text entry command, such as the insertion command, *i*. After entering *i*, any letters you type at the keyboard are entered as text.

To exit from the text entry mode, you must press the ESC key, which returns you to command mode. Now the cursor can be moved around the screen. What happens if you try to move the cursor to another position while in text entry mode? Since *vi* interprets the keys *h*, *j*, *k*, and *l* as cursor movement keys, any key you press in insert mode is entered as that character! With command mode editing, the keyboard adopts new functions: every character key can become a command key.

This changing of modes sounds more complicated than it really is. Let us edit a file using *vi* for practice. Type:

```
% vi testfile
```

which brings up a screen that is blank except for tildes (~) at the left margin. (If the line *set number* is added to the *.exrc* file, consecutive numbers replace the tildes.) A sample editing screen appears as follows:

```
~
~
~
~
~

"testfile" [New File]
```

You have created a new file called *testfile*. Note that the cursor is placed at the beginning of the first line of the file. When you enter *vi*, you are in the command mode and remain there until you enter one of the edit mode commands to move to the text entry or *insert* mode (see Table 3-1).

Table 3-1. Text Entry Mode Options

i = enter *insert* (text entry mode), starting at current cursor position.

I = enter *insert* mode, starting at beginning of current line.

a = append; enter *insert* mode immediately after current cursor position.

A = append to end; enter *insert* mode at end of current line.

o = open line below; enter text entry mode on new line beneath current one.

O = open line above; enter text entry mode on new line above current one.

A Sample Editing Session

To begin entering text, switch to text entry mode by pressing the lowercase i key. Start typing the following text exactly as shown, in spite of its obvious errors. If you make typing errors, do not be concerned. You will learn how to correct them in this session.

Press the Return key to move to the next line.

```
We hold these truths to bee self-evident; that all
men are created equal, that they are endowed by there
Creator with certain Rights, that all among
them are Live, Liberty and the pursuit
of Happiness.
of Happiness.
~
~
~
```

Now let's correct the errors. First, exit text entry mode by pressing ESC.

Note that the cursor can be moved around the screen by pressing one of the cursor movement keys described in Table 3-2. Table 3-3 lists the commands that enable you to move from the first to the last line in the file, and to move forward and backward through the file a screen at a time. Table 3-4 lists various editing commands. These tables serve as a reference. When you are ready to start making corrections, go to *Step 1*, which follows Table 3-4.

Table 3-2. Cursor Movement Commands

k = move cursor up one line.

j = move cursor down one line.

l = move cursor to the right one character.

h = move cursor to the left one character.

w = move cursor forward to the beginning of the next *w*ord (or next punctuation mark).

W = same as *w*, but ignores punctuation.

e = move cursor to the *e*nd of the next word.

E = same as *e*, but ignores punctuation.

b = move cursor *b*ackward to the beginning of the previous word.

B - same as *b*, but ignores punctuation.

$ = move cursor to the last character on the line.

0 = move cursor to the first character on the line.

On some terminals the arrow keys move the cursor keys. You can also move the cursor more than one line or character by specifying a number before the cursor movement commands; for example, *3w* moves the cursor forward three words. (Many *vi* commands accept number qualifiers.)

Table 3-3. Scrolling/Paging Commands

1G = move cursor to the first line of the file.

G = move cursor to the last line of the file.

CTRL b = move *b*ackward one screen.

CTRL f = move *f*orward one screen.

Table 3-4. Correction/Editing Commands

x = delete character at position of cursor.

r = replace character at position of cursor with next character typed.

R = overstrike all current characters until ESC key is pressed.

dl = delete letter at position of cursor (same as *x*).

dw = delete current word, starting from cursor position.

dd = delete current line.

d$ = delete to end of line.

Editing Steps 1 - 8

Step 1

In the sample exercise, let us correct the misspelling of *bee* in the first line. Type **5k** to move to the first line from the end of the file. (If *5k* is typed on the screen, you have not pressed ESC to exit text entry mode.) Then move the cursor to the extra *e* in *bee*. (The *e* command moves the cursor to the end of the next word.) Press *x*, the delete character key, to delete the *e*.

Step 2

On the second line *there* should be *their*. Move the cursor down one line using the *j* command. To quickly move to the end of the line, enter the *$* command. Back up the cursor,

using *h*, to the *r* in *there* and type **R** to replace more than one character. (Use the lowercase *r* command to replace one character only.) Type **ir** over the *re*. Press ESC to return to command mode.

Step 3

Move to line three with the *j* key. Delete the extra word *all* in line three by moving the cursor to the beginning of the word (*2b*) and entering the delete word command **dw**.

Step 4

Move to line four, where *Live* should be *Life*, and make the correction using the *r* command.

Step 5

Note that the last two lines are the same. To delete one of them, move the cursor anywhere in the line (*0* moves to the beginning) and enter the delete line command **dd**. (To delete more than one line, enter the number of lines before the *dd* command; for example, *3dd* deletes the line where the cursor is placed plus the following two lines.)

Step 6

Now let us insert an extra line. Select any line in the file and move the cursor to it (the cursor can be at any position on the line). Enter the *o* command to insert a blank line beneath the current line, or enter the *O* command to insert a blank line above it. The *open* command places you in text entry mode so you can begin entering text immediately. Type anything on the line. Then press ESC and delete the line as described in *Step 5*.

Step 7

Insert the word *inalienable* between *certain* and *Rights* in line three by moving the cursor to the last letter in *certain*. Enter the append **a** command, type a space, and type **inalienable**. Your file should look like this:

```
We hold these truths to be self-evident, that all
men are created equal, that they are endowed by their
Creator with certain inalienable Rights, that among
them are Life, Liberty and the pursuit
of Happiness.
```

The file contents now exist in a *buffer*, or temporary storage area. Any editing that you perform in *vi* takes place on a copy of the file stored in a buffer. If the computer were to fail before you *saved* the file by *writing* it to the permanent storage device, or hard disk, everything you typed would be lost. After a file is saved, it can be retrieved and re-edited at another time.

Step 8

Now let's save the file you created and quit the editor. Press ESC. Enter the colon (:) followed by the combined *write* and *quit* commands as shown here:

```
: wq
```

The following message is displayed at the bottom of the screen:

"testfile" [New File] 5 lines, 210 characters"

indicating that the file you created and edited is now written to disk and can be retrieved later.

Note that when you preface a command with a colon, you are issuing a line-mode command. These commands are discussed later in "The *Vi* Command Line" section.

This completes the sample editing session.

Editing an Existing File

An existing file that is retrieved by entering:

```
% vi oldfile
```

is displayed in the following format:

```
This is a sample text file using the vi
editor.  Text can be entered after you
enter one of the text entry mode commands.
~
~
~
~
~
~
~
"oldfile" 3 lines, 122 characters
```

You can begin editing the file in the same way that you edited the sample file.

A Few vi Pointers

Guidelines for Entering Text

When you enter text, follow these general rules:

- Start text at the left margin.
- Avoid hyphenating or splitting words across lines.
- Keep lines short.
- When writing documents, avoid backslashes (\ ) in text since they have a special meaning in UNIX document preparation (*nroff*).

Undoing a Mistake

One of the most useful *vi* commands is the *undo* (*u*) command. If you make a mistake while editing text, use the *undo* command to erase the effect of the last editing command. To restore your text as it was before the last change, press the ESC key to enter command mode, and then enter *u*. Re-entering *u* again undoes the most recent *undo*. Note that the dot command (.) repeats the last action.

Search and Replace

To search forward through the file for a string of text, enter the slash character (/) from command mode. The cursor moves to the last line of the screen (the *vi* command line). Enter the characters to be located and press Return.

To search backward through the file for a string of text, enter the question mark (?), followed by the characters to be located, and press Return.

This search action is equivalent to the FIND keypad function in EDT (keypad mode). To find the next occurrence of the string, press *n*. To repeat the search in the reverse direction, press *N*.

To substitute all occurrences of one string for another throughout a file; that is, to make a global substitution, enter:

```
: g/abc/s/123
```

where:

g = the *global* command.

abc = the search string.

s = the *substitute* command.

123 = the replacement string.

This command replaces every instance of *abc* with *123*.

The vi Command Line

Another feature of *vi* is the command line. If you press the ESC key from the command mode and enter the colon (:), the cursor moves to the last line of the screen. After the colon you can enter line commands that enable you to work globally with the file. Use these commands to save files, quit the editing session, move to specified lines within the file, combine files, and even to enter a UNIX command without exiting from the session.

For example, to save the contents of the file—which you should do at least every ten minutes—switch into command mode from editing mode. Enter the **w** (write) after the colon and press Return. The file you are currently editing is only a copy of the file. Until you enter the **w** command your changes are not written to a disk, which records them. If the system stops working ("goes down" or "crashes"), your file can be recovered up to the point that it was last saved, either by you or by the system.

The *recover* feature attempts to save your file immediately before the system crashes. This version of the file should be the most recent. To recover your file, enter:

```
% vi -r filename
```

When the file is displayed, verify the date that displays at the beginning of the file. If the system has been crashing frequently, the latest version may not have been captured. (The file may even be blank.) Assuming that you have made subsequent saves, your own version may be more recent. Note that when you save the file version displayed through the *vi -r* command, any other version of the file is overwritten and therefore no longer exists. Do not save the recovered file until you are certain that it is the latest copy.

You can exit from your editing session by entering the *write* and *quit* (*wq*) commands after the colon. If you do not want to save the changes you have just made, you can exit by quitting with the *q!* command, which means quit without saving. The next time you retrieve your file, it will be exactly as it was when you started the previous editing session, or last issued the *write* command. To exit from the file and return to the UNIX

prompt, enter **wq** to write the file to disk, thereby saving it. Or, if no changes have been made, you can exit the editing session by entering **q**. If you have made changes that have not been saved, *vi* warns you to write the changes before quitting. *Vi* line commands enable you to move the cursor to a specified line in the file simply by entering the line number after the colon. (The *f* command shows the current line.) Enter *G* to go to the last line of the file.

Another file can be inserted into your current file by using the *read* command. First place the cursor on the line above the line where you want the file to be inserted, or read in. Then enter the **read** command followed by the filename to be inserted after the colon. For example:

```
: read file2
```

places the contents of *file2* on the lines below the cursor in your current file. The *read* command inserts a copy of the designated file, leaving the original file intact. The *read* command may be abbreviated to *r*.

Two commands entered after the colon enable you to exit from *vi* temporarily: *!* and *sh*. To exit from *vi* to the UNIX shell to enter one command, enter an exclamation point (!) followed by the UNIX command. For example:

```
: ! ls
```

lists the contents of your current directory. Press the Return key to return to your editing session.

To exit from *vi* to the UNIX shell to enter more than one UNIX command, enter the *sh* command after the colon. The UNIX shell prompt is displayed. When you have finished entering UNIX commands, type **exit**, which returns you to your *vi* session.

An Introduction to Text Formatting

The text you enter using *vi* can be processed through a text formatting program into final documents, such as letters, manuscripts, and technical reports. In UNIX, text entry and text formatting are separate processes. UNIX provides two text formatters—*nroff* and *troff*. *Nroff* formats text for printing on line printers or letter-quality printers. *Troff* formats text for printing on a phototypesetter. If you want to use a laser printer, you must get an appropriate printer driver. Text formatting commands, also called *dot* commands because they are preceded by a period, are entered as text in the text entry mode. When the file is complete, the *nroff* or *troff* program is run to process the input file. Thus, while creating a document you enter text formatting commands with the text. Afterward, you will run the *nroff* or *troff* program with the input file, which creates an output file.

Nroff commands are either one or two characters in length and may be followed by arguments that further describe the command. For example, the command *.ce* causes the next line of text to be centered horizontally. If you want to center more than one line, follow the *.ce* command with the number of lines you want to center; for example, *.ce 10* for ten lines. After typing in the ten lines of text, turn off the *dot* command by entering *.ce 0*, which means center zero following lines.

Nroff gives you the capability to specify page layout, such as line length, margins, and page numbers, as well as document style, such as indented or blocked paragraphs. *Nroff* also controls options such as hyphenation, underlining, and spacing.

Macro Packages

Often a set of *nroff* commands is needed to perform certain functions, such as determining paragraph style and creating tables of contents, footnotes, page headers and footers, and cover sheets. To save time, macros, which combine several *nroff* instructions into one command, have been developed to perform these functions. Two macro packages, *-ms* and *-me*, are available with EUNICE.

Nroff commands consist of a period (.) followed by one or two lowercase letters. In contrast, *ms* instructions consist of a period (.) followed by one or two uppercase letters. *Nroff* and

ms instructions may be followed by one or more arguments.

Enter *nroff*, *-ms*, and *-me* commands separately on a line above, or if applicable, on the line above and below the text to be affected. All commands must start at the left margin. A few of the most common *nroff* and *-ms* commands are shown in Table 3-5.

Table 3-5. Sample nroff/-ms Commands

nroff Commands

.sp	*sp*aces one blank line.
.sp n	*sp*aces *n* number of blank lines.
.ce	*ce*nters next line.
.ce n	*ce*nters *n* following lines.
.bp	*b*reaks (ejects) *p*age and starts a new page.
.in +n	*in*dents the left margin *n* spaces.
.in -n	*in*dents margin to the left *n* spaces.
.ls 1	(*l*ine *s*pacing) single spaces text
.ls 2	double spaces text
.ls 3	triple spaces text

-ms Macro Commands

.PP	starts a paragraph, which is indented five spaces. A blank line is inserted above and below the paragraph.
.LP	starts a block-style paragraph.
.FS	starts a footnote.
.FE	ends a footnote.

The nroff Command

The *nroff* command directs the *nroff* program to process the input file that you created through the editor. When a macro package is specified as part of the command line, it is also invoked in the processing of the input file. The processed output is redirected into a file that you specify. Usually, this file has the same name as the input file, except for the suffix, which you can assign as *.lp* to indicate that the output is designated for a line printer.

Thus, the *nroff* command line consists of the *nroff* program, any macro package used, the input file, and any destination file. For example:

```
% nroff -ms filename > filename.lp
```

produces a formatted version of *filename* in the file *filename.lp*, which is input for the print file. Use the *lpr* command to print the formatted output file. For example, the *nroff* output file *filename* can be printed using the *lpr* command:

```
% lpr -[option] filename.lp
```

The *lpr*(1) command in the *UNIX User's Reference Manual* describes the command's options. You will probably use a variation of the *lpr* command since it is installation dependent.

Formatted output files produced by *nroff* can be displayed on the screen using the *more* or *cat* commands. Output produced by *troff* **cannot** be displayed on a normal screen.

See the document preparation section in the *UNIX User's Supplementary Documents* for additional information on *nroff*, *troff*, and the two macro packages.

Using Mail

EUNICE enables you to send mail from EUNICE to the VMS mailer, as well as to use the standard UNIX mail program.

Setting Up the Environment

The mail environment is set by the system file */usr/lib/Mail.rc*, which is maintained by the system administrator. This file sets up default parameters for mail.

The *.mailrc* file in your home directory enables you to customize the mail facility. To use it, copy the */usr/skel/.mailrc* template into your home directory. The options that you set in your *.mailrc* file override those set at the system level. The template file contains three options for the mailer:

1. *ask*—causes the mailer to prompt you for the message's subject.
2. *askcc*—causes the mailer to prompt you for the names of additional recipients (*cc* stands for carbon copies).
3. *dot*—causes mail to interpret a period alone on a line as indicating the end of the message. To change this option, include the unset dot command in your *.mailrc* file in your home directory.

The *.mailrc* options are enabled or disabled through the *set* and *unset* commands, respectively. For a more complete discussion of setting mail options, refer to the *Mail Reference Manual* in the *User's Supplementary Documents*. Also see the *mail*(1) command man page in the *UNIX User's Reference Manual*.

Sending Mail within UNIX

To send a message to another user in the UNIX system (with the */usr/skel/.mailrc* template in your home directory):

1. Enter the *mail* command followed by the login name of the person who is to receive the message. For example:

```
% Mail ron
```

UNIX responds by prompting you for the *Subject:*.

2. Respond by entering a short line stating the subject of your message. End the subject line by pressing Return. UNIX responds by positioning the cursor at the left-hand margin.
3. Enter the text of your message. If you want to correct any typing errors, press the Backspace key, which moves the cursor to the left margin. Retype the line. Press RETURN to advance to the next line. Press the DEL key to remove a single character.
4. To end the message, press CTRL Z or enter a period (.) at the beginning of a new line. UNIX responds with the *Cc:* prompt.
5. Enter the login names of any users who are to receive copies of the message, then press Return. If you do not want to send any copies, press Return to exit from the mailer and send the letter.

To cancel a mail message at any time, press CTRL C. This action exits the mail program and returns you to EUNICE.

To send an existing file to several users, enter:

```
% Mail user1 user2 ... < filename
```

where *filename* is the ASCII file to be mailed.

Reading Mail

If you have mail to read, the message *You have UNIX mail* is displayed each time you log in to EUNICE. Enter the *Mail* command after the prompt.

By default, you will not receive notice of mail sent to you while you are logged on, you can query for mail by entering the *Mail* command. If you do not have mail, the system

responds with the message:

No mail for *login-name*

If you do have mail, a list of your messages is displayed in the following format:

```
U   1   dave   Tue   Jan 12   18:45   13/324   "Conf. Call"
N   1   sam    Wed   Jan 13    9:13    33/416   "Meeting Overview"
N   2   jkl    Wed   Jan 13   10:36    19/369   "Reminder"
N   3   ron    Wed   Jan 13   11:45    19/326   "Lunch"
```

where the first column gives the message status:

N = New
R = Read
P = Posted
U = Unread

and the following columns list the message number, the sender's login name, the date and time the message is received, and the subject, respectively.

Your message list is displayed followed by an ampersand (&) prompt, after which you can press RETURN to display the next message, or issue one of the *Mail* commands. *Mail* commands, which are listed in Table 3-6, can be entered by their complete name, or by the first unique letter or letters, except for *help*, which must be written in its entirety.

Table 3-6. Mail Commands

t	*t*ypes (displays) the most recent new message on the screen.
2	types message number *2* (or any other number specified).
delete	*d*eletes the most recently accessed message; *d* followed by the message number deletes a specific message.
r	*r*eplies to the sender of the last message displayed (the subject is assumed to be the same).
-	displays the previous message.
help	provides a brief *help* list with definitions of *Mail* commands.
list	*l*ists valid *Mail* commands.
mail	*m*ail followed by the recipient's login name enables you to send mail from within the *Mail* program. (If you omit the login name, the mail is sent to your dead letter file.)
print	*p*rints all messages in the queue on the terminal.
quit	*q*uits the *Mail* program and updates any related files, folders, and the system mailbox, or the *mbox* file in the user's home directory. Use the *cat* command to display the contents of your *mbox*.
save	*s*aves messages. For example, entering **save 1 3 7 remind** saves messages 1, 3, and 7 in the file *remind*.
undelete	restores the previously deleted message.
x	exits the *Mail* program without modifying your mailbox.

Sending Mail to the VMS Mailer from UNIX

The *vmsmail* command enables you to send mail from UNIX to VMS users, who must read their mail through VMS. To send mail from UNIX to VMS:

1. Enter the *vmsmail* command followed by the login name of the person to receive the mail. For example:

   % vmsmail login-name

2. Enter the text of your message. To correct errors, press the Backspace key and retype the line. Press Return to advance to the next line.
3. To end the message, press CTRL Z.

The mail utility can deliver mail to a file or to another process. For example, to send mail to one or more users and to keep a record in a file, enter:

```
% Mail user1 user2 ... filename
```

which sends mail to user1, user2 (and other specified users) and also to the specified file.

To send mail to a process, enter:

```
% Mail user1 user2 ... "| process"
```

where the vertical bar | represents the *pipe* command. The *pipe* command sends the output of one command to the input of another. For example:

```
% Mail user1 user2 ... "| vmsmail username"
```

sends mail to one or more users and redirects that mail to the VMS mail utility.

To send an existing file to several users, enter:

```
% Mail user1 user2 ... < filename
```

where *filename* is the ASCII file to be mailed.

To cancel the message at any time, press CTRL C. This action exits you from the mail program and returns you to EUNICE.

Note that mail can be sent from VMS to EUNICE through the WIN™ /TCP mailer only.

This chapter has presented basic information to enable you to start using the UNIX system and EUNICE. To increase your knowledge, refer to the "Getting Started" section of the *User's Supplementary Documents*. The UNIX on-line tutorial, which is called *learn*, is available with EUNICE. Ask your system administrator for details.

Notes

Notes

Chapter 4: EUNICE, UNIX, and VMS

This chapter discusses the interrelationship of EUNICE 4.3 Software, UNIX, and VMS and the following topics:

- the UNIX file system
- the mapping of the VMS directory structure to EUNICE
- UNIX and VMS filenames and formats
- UNIX file-naming conventions
- UNIX file protections

VMS and the UNIX Shell

The EUNICE 4.3 Software provides transparent access to both the VMS and UNIX operating systems in a co-existent environment. The UNIX shell is a utility program that enables you to communicate with the UNIX system. The shell interprets the commands you type, then calls the program (or programs) that make up the command line from the computer's memory and executes them one at a time (or in a series called a "pipe"). It is also a high-level programming language, with standard utility programs that can be combined to build entire applications.

File Systems

In Chapter 2, we said that the UNIX file system could be visualized as an upside-down tree, with the *root* (/) at the top. As shown in Figure 4-1, *branches* extend out and downward from the *root*, ending in *nodes*. Nodes can branch to other nodes, and nodes can branch to *files*. The *root* and nodes are known as *directories*. Related files are organized into directories (files are attached to their directory via branches). This hierarchical system allows each user to have one primary directory with as many subdirectories as required.

When we speak of moving within the file structure, *up* refers to moving toward the *root*, while *down* refers to moving away

from the *root*.

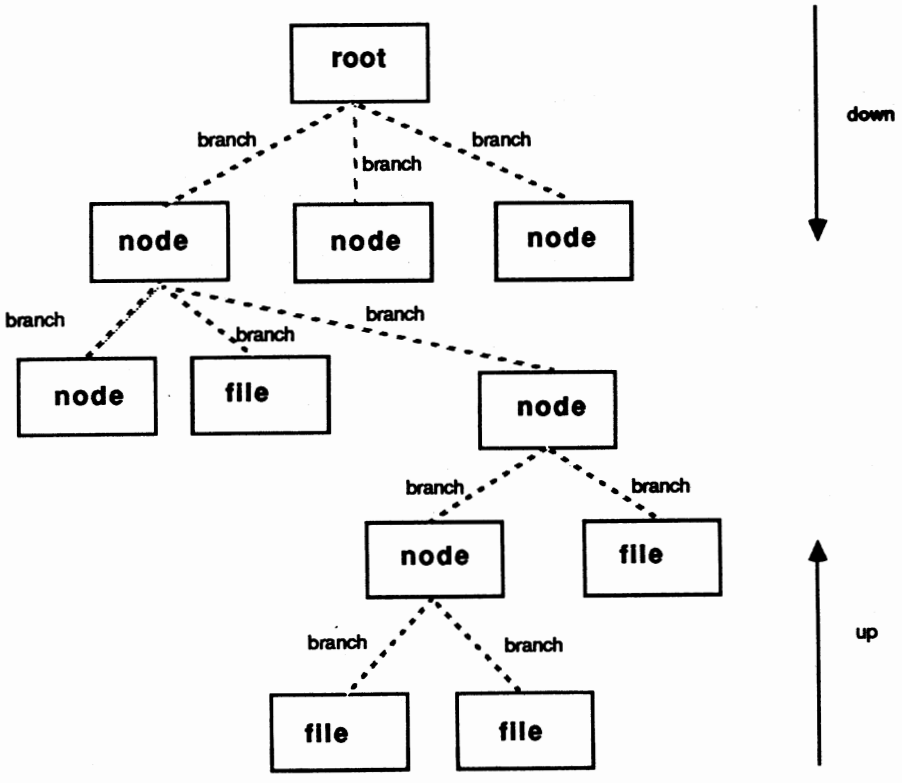


Figure 4-1. UNIX File System Tree

Absolute and Relative Pathnames

As mentioned, every file has a *pathname*, which can be determined by tracing a path from the *root* directory, through intermediate directories, down to the file. This complete path from the *root* to the file is known as an *absolute* pathname. The first slash (/) in a pathname represents the *root* directory. Each directory or filename is separated by a slash; for example, */u/ron/Book/chpt2*. The name following the final slash in the pathname is the *filename*.

You can also specify pathnames relative to your working directory. Absolute pathnames begin with the *root* (/) directory. *Relative* pathnames assume that the pathname for a file is the same up to your working directory; that is, it begins with the name of the directory that branches off the present working directory. For example, if Ron, whose login name is *ron*, is working in his home directory, he can specify filenames from the working directory, such as *Book/chpt2/figures*, omitting the */u/ron* part of the pathname to his working directory, which is assumed. (See Figure 2-1.)

When you work with EUNICE, the file system appears to be the same as the UNIX file system. Actually, EUNICE uses the standard VMS file system. VMS logical names are used to map UNIX file systems to the VMS file hierarchy. To complete the UNIX file hierarchy, the EUNICE initialization procedure sets up pseudo-root (/) and device (/dev) directories. A reference to */dev/device* causes EUNICE to look for a VMS logical *device* to map into its equivalent VMS device name. There is, as well, full support for relative pathnames.

Directory Structure and Mapping

The EUNICE directory structure is mapped to the VMS directory structure. In VMS, names are assigned to physical disks; for example, DRA1:, DRA2:, and DRA3:. Disks contain directories that are mapped to logical names, which are, in turn, mapped to a UNIX/EUNICE directory translation.

In the implementation of EUNICE shown in Figure 4-2, physical disk DRA1: contains the EUNICE binaries, which have logical names such as TWGBIN, TWGETC, and TWG$USR, and are mapped to the EUNICE subdirectories /bin, /etc, and /usr. Device DRA2: contains the user's directory [USER], which has the logical name TWG$U, and

the UNIX/EUNICE translation of /u. Disk DRA3: contains the [TMP] directory, which has the logical name TWG$TMP and the EUNICE translation of /tmp. The combination of the directories of these three logical devices creates the following EUNICE emulation of the UNIX directory structure under the *root* (/) directory:

/u /tmp /bin /etc /lib /usr

As mentioned, the *root* (/) and */dev* directories are emulated in EUNICE so that the complete UNIX file structure is present.

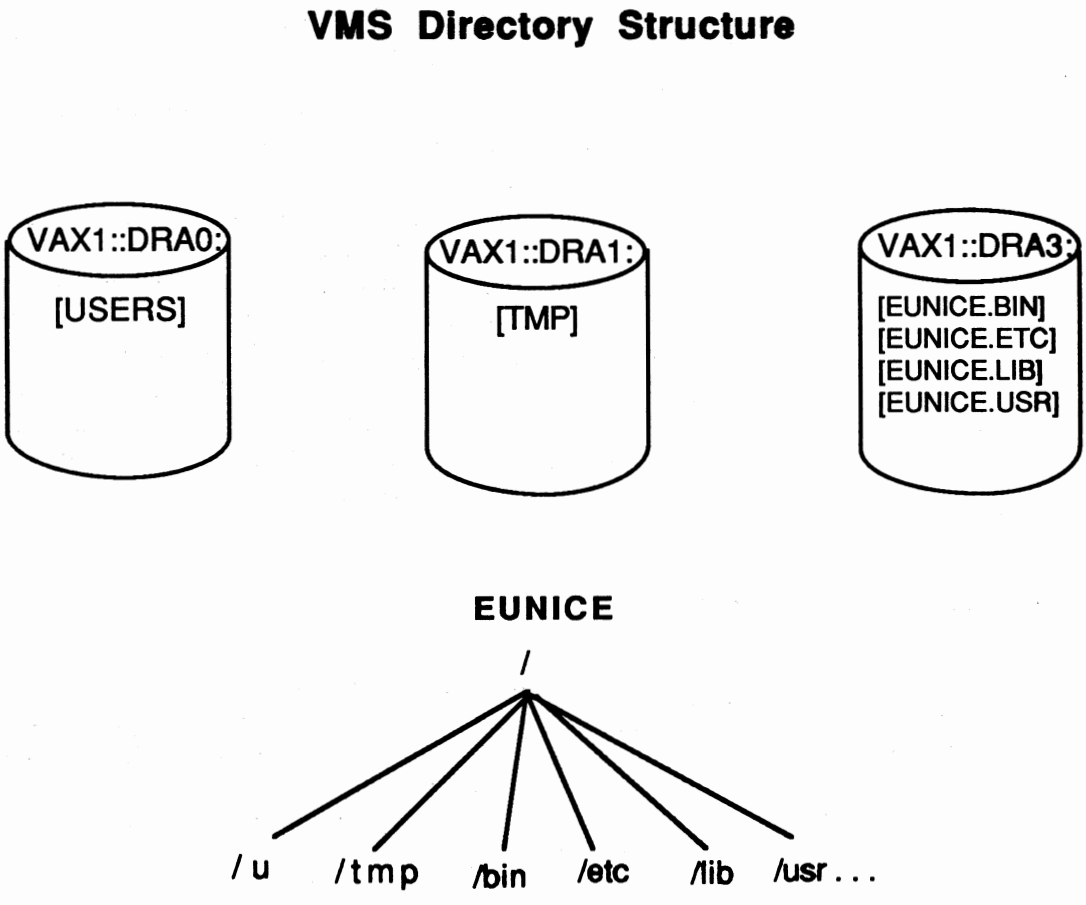


Figure 4-2. VMS to EUNICE Directory Structure and Mapping

UNIX and EUNICE File Naming Conventions

EUNICE supports the full range of UNIX filenames, including multiple extensions and case-sensitive filenames. UNIX filenames can be as many as 256 characters. The *vmsname* command can be used to display the name of a UNIX file in VMS. (See the *vmsname* man page in the *UNIX User's Reference Manual.*)

All references to filenames in EUNICE are passed through an internal file-parsing routine, which is transparent to the user. Any discrepancy between the UNIX and VMS filenames is noted and a translation is made. (This translation is referred to as *file mapping*.) Since lowercase UNIX filenames are translated into all uppercase in VMS, any uppercase letters that are part of a UNIX filename are preceded by the dollar sign ($). The first $ turns on capitalization; the next one turns it off. For example, the UNIX filename *Letters* is translated as LETTERS. in VMS. The UNIX filename *LETTER* is translated as $LETTER. in VMS. (If a UNIX filename does not contain a period when it is translated into VMS, a period is added at the end.)

The VMS filenames have only one extension. If a UNIX file has more than one extension, the period before the second (or greater) extension is converted to a *$5N* in VMS. For example, *file.ext.ext* in UNIX is translated as *FILE.EXT$5NEXT*.

The process that allows EUNICE to accommodate the syntax difference between VMS and UNIX filenames is called *file hashing*. Filenames can be standard VMS or UNIX filenames. EUNICE supports VMS filenames that consist of as many as 39 characters for the filename and 39 characters for the extension which is a VMS restriction. EUNICE creates a second file when VMS filenames exceed this number.

As mentioned, UNIX filenames can consist of as many as 256 characters without spaces. The following characters, called *metacharacters*, have special meanings to the shell and should not be used as part of a filename unless they are surrounded by quotation marks, or are preceded by a backslash (\ ):

| ; & ( ) < > >> * ? [ ] \ ‘ ’ "! $ { } ~ - ^

It is recommended that you restrict your filenames to letters (both uppercase and lowercase), numbers, periods, and underlines.

In EUNICE, you can specify filenames in either UNIX or VMS format. When specifying a VMS filename in EUNICE, enter the backslash (\) before brackets ([ ]) or dollar signs ($) so that they are correctly interpreted. The UNIX filenames are translated in the VMS structure as follows:

DISK:[dir]filename

where:

dir, which is the same name as the UNIX directory, can consist of as many as eight levels of directories (a VMS limitation). For example: *dir.dir*, *dir.dir.dir*, etc.

filename is the same filename as in the UNIX system.

-or-

DISK1:[user.username] filename

where:

username is the VMS login directory name.

filename can equal a UNIX pathname; for example, */u/george*.

As mentioned, UNIX does not provide version numbers of files unless you modify your LOGIN.COM file (see the section "List Contents of Directory" in Chapter 3). The latest version number of a VMS file is not translated by EUNICE. EUNICE appends an extension after earlier versions. Therefore, the VMS files *FILE.;1*, *FILE.;2*, and *FILE.;3*, in which *FILE.;3* is the most recent version, are translated by EUNICE as *file..1*, *file..2*, and simply *file*, respectively. When VMS files with numerical extensions are translated into EUNICE files, the most recent version numbers are deleted. For example,

FILE.1;1 = file.1
FILE.2;1 = file.2
FILE.3;1 = file.3

If version numbers exist for *FILE.3*, such as *FILE.3;1*, *FILE.3;2*, and *FILE.3;3*, EUNICE translates them as *file.3..1*, *file.3..2*, and *file.3..3*.

When naming VMS files, avoid extensions in which two or more extensions consist of numbers. For example, EUNICE interprets the filename *test.43* and *test.43.1* as identical filenames. When naming EUNICE files, avoid names or

extensions that consist only of numbers since VMS may interpret these as version numbers and overwrite existing files.

File Formats

UNIX and VMS file formats differ. UNIX requires that all files have a fixed length of 512-byte records (the last record can be truncated or shortened), with a linefeed character signifying the end of the line. VMS files can be either variable- or fixed-length record files with implied new lines.

The EUNICE *unixtovms* command converts the format of a UNIX file to a VMS variable format. The *vmstounix* command converts the format of a VMS file to a UNIX format. For example, where *filename* is a valid UNIX file,

```
% unixtovms filename
```

converts the file *filename* to the VMS format. Similarly, where *file* is a valid VMS filename,

```
% vmstounix file
```

converts *file* from VMS to the UNIX format.

File Protections

File protections ensure that only authorized users can perform specified functions on your files. Since UNIX and VMS establish file permissions differently, EUNICE translates file protection information and specifications.

Table 4-1 shows the correspondence between UNIX and VMS *access codes*. Access codes refer to which categories of users are authorized to gain access to a file. Note that UNIX has no equivalent system access code.

Table 4-1. UNIX and VMS Access Codes

UNIX	VMS
(u)user	(O)owner
(g)group	(G)group
(o)other	(W)world
none	(S)system

In EUNICE, the UNIX *su* (superuser) command is not supported. Users who have accounts with SYSPRV, however, have privileges equivalent to those of UNIX superusers.

Table 4-2 shows the correspondence between UNIX and VMS *protection codes*. Protection codes refer to whether an authorized user can read, write, execute, or delete a file.

Table 4-2. Protection Codes

UNIX	VMS
(r)read	(R)read
(w)write	(W)write
(x)execute	(E)execute
none	(D)delete

Note that UNIX has no delete protection code. Consistent with UNIX, EUNICE allows users who have write privileges for a file to also delete it.

In EUNICE, UNIX file protection options override VMS protections for files that you own. The *umask* line in your *.login* file sets the protections for newly created files, such as those created with editors or through the *cat* command. Be aware that *all* delete bits for *all* users are set. When *umask* is not present in the *.login* file, VMS default file protections are in effect.

VMS file protections can also be set from EUNICE using the VMS SET PROT command.

UNIX File Permissions

The UNIX categories of protection codes are *read*, *write*, and *execute*. Read permission grants access to display or print a file's contents, and list a directory. Write permission enables you to change or delete a file. If a file has execute permission, you can run it.

UNIX divides the user community into three—user, group, and others. The *user* owns the files in his or her home directory. *Group* affiliation is assigned by the system administrator. Group members may share a project or be members of the same department. They are a related subset of the entire user community, known as *others*. The UNIX file system was designed so that files could be shared among developers, and this is one of its advantages. Therefore, in assigning file protections, try to strike a balance between protecting your files and sharing your information with group members and other users.

Viewing File Permissions

To view the current file permissions, issue the *ls* command with the *-l* option followed by the filename. The file permissions are displayed as follows:

```
-rw-r-xr-x   4   david   1057   Sep 21   12:00   memo
```

where *-rw-r-xr-x* = file permissions. (The first character is either a hyphen (-), indicating a file; a *d*, for a directory; or others.)

As shown below, the three fields that make up the file permissions are user, group, and other. Each field contains three characters that indicate whether the user, group, or other category has permission to read, write, or execute the file.

User	Group	Other
|	|	|
{rwx}	{rwx}	{rwx}

If *r*, *w*, or *x* appears, the user, group, or other has read, write, or execute permissions, respectively. If a dash appears, permission is denied.

To interpret the file permissions for the file *memo*, divide the nine file permission characters into three sets of three characters: rw-, r-x, and r-x (excluding the first hyphen, which indicates that this is a file). Note that the owner has read and write, but not execute permission. Group members and others can read and execute the file, but cannot alter it.

Changing File Permissions

Use the change mode—*chmod*—command to change file permissions. Only file owners and system administrators can change file permissions. The *chmod* syntax is as follows:

chmod ugo filename ...

where:

u = the total value of permissions granted to the user/owner.

g = the total value of permissions granted to the owner's group.

o = the total value of permissions granted to the remaining users (others).

The user, group, and others are either assigned or denied permission to read, write, and execute a file.

To calculate permissions for the owner, group, or others, decide what permissions each category is to have. The following values are assigned to each permission:

Read permission = 4

Write permission = 2

Execute permission = 1

In the *memo* file example, the owner has read and write permissions. Calculate the permission values by adding the values associated with the read and write permissions. (*Read* equals 4 and *write* equals 2. Add 4 plus 2 for the user's total permission value, 6.)

In the *memo* example, group members can read the file and execute it, but cannot write to it.

Read = 4
Execute = 1

Total = 5

The rest of the user community can read and execute the file, but cannot write to it.

Read = 4
Execute = 1

Total = 5

To set the permissions for *memo* as described above, enter:

```
% chmod 655 memo
```

Notes

Notes

Chapter 5: EUNICE 4.3 Software Specifics

This chapter discusses general programming concerns, such as EUNICE processes, compilers, device drivers, and batch queues. In addition, the REX environment, *make* utility, and UNIX symbolic debugger (*dbx*) are introduced.

EUNICE Processes

UNIX differs considerably from VMS in that UNIX is a much more process-oriented operating system. VMS creates processes under the following circumstances: when you log in, submit a batch job, spawn a subprocess using the SPAWN command, or run a program. The UNIX system begins execution with a single process—Process Identification number (PID)1. PID 1 creates a login process for each terminal. When you log in, this process becomes a shell process. When you issue a command, the shell process creates a subprocess, called a *child.* (The process that creates a *child* is termed the *parent* process.) In UNIX, the *parent* process remains inactive, or *sleeps*, while the *child* process is running, which conserves computer resources. In contrast to VMS, UNIX operation is predicated on the creation of many processes and subprocesses.

With EUNICE, if a *parent* process releases a *child* process, the *child* is sent into a hibernation state and all resources are released to VMS. If another request for processing occurs within five minutes, the *child* process is revived and new VMS resources are created for it. If no request is received, the *child* process dies after five minutes. This recycling of processes eliminates the VMS overhead that would be needed if new EUNICE processes always had to be spawned. You may notice that EUNICE is a litte faster after these recyclable processes are created.

If you issue the SHOW PROCESS command from VMS less than five minutes after exiting EUNICE, EUNICE processes will still be hibernating. If you enter this command more than five minutes after exiting, the processes will have already been deleted. Because processes are recycled in EUNICE, they do not have unique PIDs as is the case in UNIX.

EUNICE Libraries

EUNICE provides three C libraries:

- Standard VMS-Style C Library
- Standard UNIX-Style C Library
- Sharable C Library

By default, the Sharable C Library is loaded when executables are created. When there are changes in this library's contents, you do not need to relink your programs that make calls to it. Relinking of programs is always required when there are changes to the Standard C Library.

To select the nonsharable Standard UNIX-Style C Library, issue the *noshare* option when calling the loader or compiler. (The *noshare* option is described in the "*cc* (UNIX C Compiler)" section.)

EUNICE Compilers

EUNICE supports all compilers provided with UNIX 4.3:

- *cc*—the UNIX C Compiler
- *f77*—the UNIX FORTRAN Compiler
- *pc*—Pascal
- *liszt*—Franz LISP

The *pc (Pascal)* and *liszt* compilers are the standard UNIX compilers. The *cc* and *f77* compilers have been modified for EUNICE. When using *cc* and *f77* UNIX compilers under EUNICE, you have the choice of generating either VMS- or UNIX-style object files. VMS object files may be linked with VMS libraries and with other VMS object modules (from other VMS languages).

cc (UNIX C Compiler)

To compile programs using the *cc* compiler, enter:

```
% cc [option] ... file ...
```

Major Options (cc and f77)

The following major options are the same for *cc* and *f77*:

-o Name	produces executable code with the filename *Name*.
-O	produces improved object code (optimizer).
-S	produces .s file only (assembler language output).
-c	produces .o file only (object file).
-g	creates additional symbol table for the debugger (*dbx*).
-d	is the verbose option.

For example, entering:

```
% cc -O test.c -o test
```

creates executable *test* from the source code (with *optimized* code).

If you enter:

```
% cc -g text.c -o test_debug
```

the executable *test_debug* is created from the source code, which has special (extended) symbol tables and can be run by the debugger (*dbx*).

Selecting the *verbose* option—*d*—shows all of the compilation steps. For example:

```
% cc -d test.c -o test
```

displays the following lines:

```
/lib/ccp:ccp cf.c /tmp/ctm001614
```

where *ccp* is the preprocessor, and *ctm001614* is the temporary file created by the preprocessor from C source code.

```
/lib/ccom: ccom /tmp/ctm001614 /tmp/ctm001613
```

where the compiler creates the temporary file /tmp/ctm001613 from the temporary file */tmp/ctm001614*.

```
/bin/as: as -o cf.o /tmp/ctm001613
```

where the assembler creates the object file from the temporary file /tmp/ctm001613.

```
/bin/ld: ld -X /lib/crt0.o -o test cf.o -lc
```

where *crt0.o* is the runtime startup and *-l* stands for library (and where *-lc* indicates that the C library is to be included), which causes the loader to load all object files with the start-up runtime object file and libraries.

By default, all objects are loaded with the shareable C library, which is located in SYS$SHARE:TWG_LIBC_43.EXE.

After the *cc* compiler is finished, control is returned to the shell.

EUNICE-Specific Options (cc and f77)

The following EUNICE-specific options are the same for *cc* and *f77*:

noshare	objects are loaded with the standard (nonsharable) C library: */lib/libc.a*.
notraceback	must be used for all installed images.
nop0bufs	stops the Record Management System (RMS) from intruding on P0 space in those programs that are sensitive to the state of P0 space, such as debuggers.

See the *cc*(1) man page in the *UNIX User's Reference Manual* for a more complete description of the various options.

Creating VMS- or UNIX-Style Objects

The *cc* compiler has been modified to create either VMS- or UNIX-style object modules. The *cc* compiler default is to create UNIX-style object modules, unless the following two environmental variables are set:

```
setenv AS_IMAGE /usr/eun/vmsas
setenv LD_IMAGE /usr/eun/vmsld
```

The compiler first checks the value of the C Shell variable AS_IMAGE to determine if the UNIX or VMS assembler should be used. Then it checks the LD_IMAGE variable to determine which loader is required. If these variables are not set, the UNIX-style assembler and loader are called by default.

See the section below on "Selecting UNIX- or VMS-Style Object Code" for instructions on specifying assemblers and loaders in your *.login* file.

f77 (UNIX FORTRAN)

EUNICE 4.3 supports *f77*, the standard UNIX FORTRAN compiler.

To compile a FORTRAN program, issue the command:

```
% f77 [option] file.f
```

The major options and EUNICE-specific options are identical to those described for the *cc* compiler. The steps for selecting UNIX- or VMS-style object code are also the same. (See the section on "Selecting UNIX- or VMS-Style Object Code" for instructions on selecting the *f77* assembler and loader in your *.login* file.)

When you port VMS FORTRAN to *f77*, note that it differs from VMS FORTRAN in the following ways:

— parameter statements (*f77* lacks any)

— variable name length (31 for VMS FORTRAN; 6 for *f77*)

— column size (80 for VMS FORTRAN; 72 for *f77*)

Refer to Section 4.6, FORTRAN 77 Notes, in the *Programmer's Reference Manual* (*EUNICE Reference Manual*) for additional information.

Our *f77* compiler complies with the ANSI standard, with the following exceptions:

— vertical format control

— default files

— lowercase strings

— exponent representation on Ew.dEe output

— preconnection of files

Selecting UNIX- or VMS-Style Object Code

As mentioned, the *cc* and *f77* compilers can create either UNIX- or VMS-style object modules. Objects generated under native UNIX 4.3 can be loaded with UNIX-style object modules developed under EUNICE. Currently, *liszt* and *pc* cannot create VMS-style objects.

In order to link routines created under EUNICE with VMS objects, or to create VMS executables from code developed under EUNICE, a VMS-style object file needs to be generated from the source files. The environmental variables AS_IMAGE and LD_IMAGE determine which object style *cc* or *f77* is to create. The environmental variables can be set in the *.login* file, from the command line, or by establishing an alias to do so in the *.login* or *.cshrc* file.

To set the environmental variables, add the following two lines to the *.login* file.

setenv AS_IMAGE /usr/eun/vmsas

setenv LD_IMAGE /usr/eun/vmsld

where:

vmsas is a modified UNIX assembler that produces VMS object code.

vmsld is a modified UNIX loader that converts a UNIX loader argument list into a valid argument list for the VMS linker.

To have the flexibility of selecting either VMS- or UNIX-style object modules, enter the following aliases in your *.login* or *.cshrc* file to change the environmental variables whenever you issue the command's *alias*.

Aliases for Creating VMS-Style Object Module

alias vmsobj 'setenv AS_IMAGE /usr/eun/vmsas; setenv LD_IMAGE /usr/eun/vmsld'

Issue the *vmsobj* command to create VMS-style objects.

Aliases for Creating UNIX-Style Objects

alias unixobj 'unsetenv AS_IMAGE;unsetenv LD_IMAGE'

Issue the *unixobj* command to create UNIX-style objects.

NOTE: These aliases are set up in the sample login file: /usr/skel/.login.

Figure 5-1 shows how the *cc* compiler creates executables in either VMS or UNIX style. The default is always UNIX style.

For additional information, refer to "VMS/UNIX Interface" in Section 4 of the *EUNICE Reference Manual.*

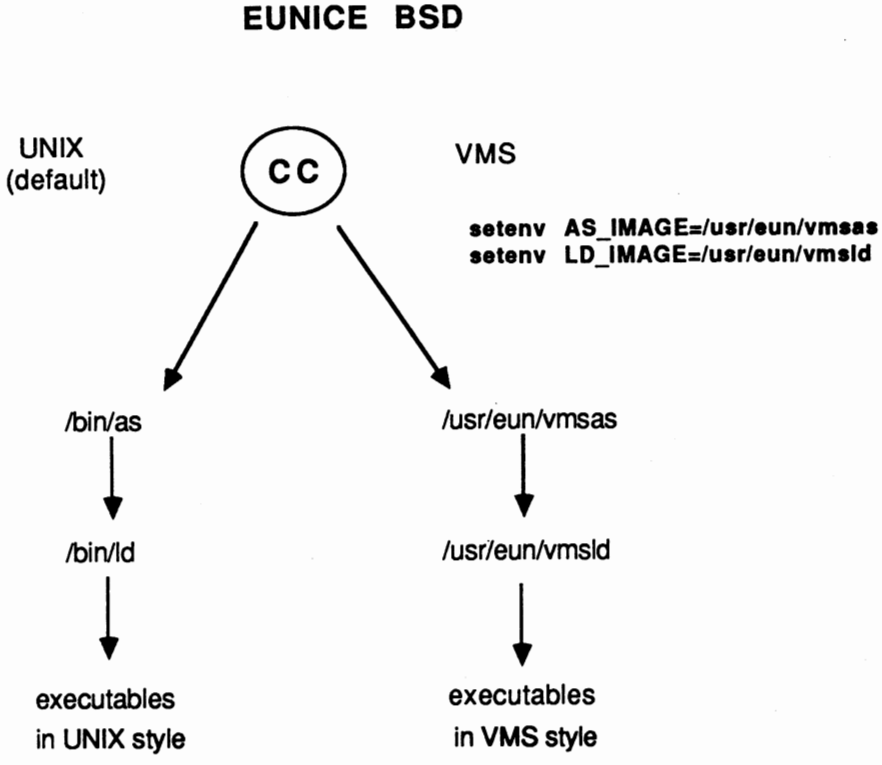


Figure 5-1. UNIX- or VMS-Style Object Code

pc (Pascal)

The *pc* compiler has not been modified for EUNICE. The major options are the same as those described for *cc* and *f77* above.

See the *pc*(1) man page in the *UNIX User's Reference Manual* for additional information.

Franz LISP

EUNICE 4.3 supports *Franz LISP*, rather than *Common LISP*, *MacLisp*, or *INTERLISP*.

The Franz LISP interpreter is called *lisp*. The Franz LISP compiler is called *liszt*. Franz LISP can *fasl* only UNIX objects created by EUNICE 4.3. Franz LISP can *dumplisp* to create pre-loaded executables.

See *The Franz LISP Manual* in the *Programmer's Supplementary Document* for additional information.

Device Drivers

Since EUNICE uses the standard VMS terminal driver, all terminal characteristics have to be set at the VMS level.

EUNICE tape devices are mapped to the UNIX file system in the same way as other EUNICE devices. Tape drives are controlled by VMS and accessed by EUNICE. *Tar*, the UNIX tape archiver command, creates and reads tapes in the UNIX format. Issue this command to access tapes in EUNICE. Before using it, however, you must mount a tape in VMS using the DCL MOUNT command. For example,

```
$ MOUNT/FOR/BLOCK=10240 TAPE_DEV:
```

(Tapes are dismounted through the DCL DISMOUNT command.)

Three examples of the *tar* command follow.

Example 1 lists the contents of *file.tar* on device *rmt0*, the tape unit.

```
% tar tvf /dev/rmt0 file.tar
```

where:

v is the verbose option, which makes *tar* print the name of each file it treats.

f before a device means the *tar* uses the next argument as the device instead of the default device */dev/rmt8*.

t when used with the *v* option, lists files.

Example 2 restores (reads) the contents of *file.tar*, which is a file in the *tar* format located on disk.

```
% tar uvf /dev/rmt0
```

where:

u means to write the specified directory, which is the default directory if none is given.

Refer to *tar*(1) in the *UNIX User's Reference Manual* for more details.

Even though disk drivers are controlled on the VMS level, you can use the UNIX disk usage *du* and disk free *df* commands to display information about disk usage. The *du* command shows the number of kilobytes contained in all files. The *df* command displays the amount of free disk space available on a filesystem. (Refer to the "Commands" section of the *UNIX User's Reference Manual* for more information about these commands.)

You can also perform a VMS backup of EUNICE executables.

NOTE: tar does not work with certain VMS files, such as SAVESET.

Batch Queues

Although the UNIX system does not have a batch queue, you can access the VMS batch queue in EUNICE by issuing either the VMS SUBMIT command or the EUNICE *at* command. The *at*(1) command, which executes commands at a specified later time, submits a job to the VMS batch queue. The *at* command syntax is as follows:

at [-c] [-s] [-m] time [day] [file]

where:

-c (for C Shell) and *-s* (for Bourne Shell) flags specify which shell executes the job. If no shell is specified, the current shell is used.

-m specifies that mail is to be sent to you after the job has run. If there are errors during job execution, a copy of the error diagnostics is sent. If not, a message informs you that no errors occurred.

time can be specified as one to four digits, with an optional *A* (for AM), *P* (PM), *N* (noon), or *M* (midnight). One- and two-digit numbers are interpreted as hours, and three and four digits as hours and minutes. If no letters follow the digits, a 24-hour clock time is assumed.

file is the name of the file that the shell is to run.

For example:

```
% at 0215 filename
```

submits the job *filename* to the VMS batch queue at 2:15 a.m.

Refer to the *at*(1) man page in the *UNIX User's Reference Manual* for more information.

REX—EUNICE's Runtime Library

The REX (Runtime EXecutive) Environment is the linked version of the EUNICE Runtime Library. It contains EUNICE-specific subroutines, the C Library subroutines (Section 3 commands), and system calls (Section 2). Although the system calls have been rewritten for EUNICE functions, at the user level they function the same as in native UNIX. The system service calls are under VMS.

The EUNICE control programs—*initgbl*, *clisetup*, and *vforkcli*—are part of the REX environment. *Initgbl* is run at EUNICE system startup to configure the EUNICE global section. *Clisetup*, which initializes global sections required for running EUNICE, maps the real VFORK code into P1 space and then transfers control to it. *Vforkcli* manipulates global sections, creates processes, and communicates with the VMS kernel and the Record Management System (RMS). Although *vforkcli* is the EUNICE command language interpreter, it is not one in the normal sense. EUNICE uses *vforkcli* to control the creation of subprocesses. (The command language interpreter for VMS is DCL.)

The calling sequence has been modified for several system calls. For example, EUNICE allows the specification of extra (optional) arguments to the *creat*(2)—create a new file—call. These options enable files to be created that are either UNIX or VMS text files, or a VMS mailbox file. (See the EUNICE notes in the System Calls(2) man pages for details.)

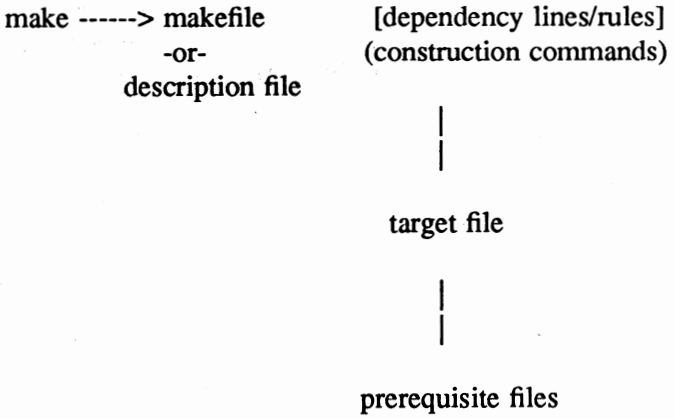
Note that, by default, the *cc* compiler loads executables with the REX environment's shareable library (TWG_LIBC_43.EXE).

Keeping Programs and Documents Current—Make

This section introduces the *make* utility, which is used in software development and document preparation. For more complete information on *make*, refer to *Make—A Program for Maintaining Computer Programs* in the *Programmer's Supplementary Document.*

The *make* program helps maintain up-to-date versions of programs by keeping track of differences in the modification times of the programs and the source programs upon which each depends. You can tell *make* the sequence of commands that creates certain files, as well as the list of files that requires other files to be current before the operations can be done. Then, whenever a change is made in any part of the program, *make* creates the proper files. The operation of the *make* utility depends on the ability to find the date and time that a file was last modified.

An overview of how *make* works follows. (Refer to the diagram below.) *Make* searches the dependency lines of the description file, which is usually called *makefile* or *Makefile.* The dependency lines show relationships between files and specify a target file that is dependent on one or more prerequisite files. If any of the prerequisite files have been modified more recently than their target file, *make* updates the target file based on the construction commands that follow the dependency lines.



The *makefile* (description) file consists of the following items:

- comments
- macro definitions
- rules
- dependencies
- commands

Commented lines begin with the number symbol (#).

Macro definitions define options in the *makefile* or description file. Macro definitions take the form of *string1* = *string2* where *string2* defines *string1*. If a macro is defined, it may be redefined in the command line. If *string2* is missing, it can be specified on the command line. (The sample *makefile* in Figure 5-2 shows an example of macro definitions.)

Make uses a table of suffixes and transformation rules to supply default dependency information and implied commands. *Rules* are specified in the description file with *dependency* information. Before a rule is performed, all its dependency information must be satisfied. If a rule does not have any dependency information, the command(s) following the rule is performed. Most rules have commands associated with them; *make* performs these commands after satisfying the dependency information.

The default suffix list is as follows:

object file = .o

C source file = .c

FORTRAN source file = .f

Assembler source file = .s

The format of the *make* command is as follows:

```
% make [options] [macro definition] [target-files]
```

where major options include:

-f file causes *file* to be used as input instead of *makefile*. For example:

```
% make -f file
```

-n causes *make* to display the commands that would be used to update the target files, but not execute them.

-t updates the modification times of target files so that if you issue the *ls* with the *-l* option, the latest modification date is given.

macro definitions define options in the description file.

target files refer to targets on dependency lines in *makefile*. If you do not specify a target file, *make* updates the target that appears on the first dependency line in *makefile*.

A Sample Makefile

Figure 5-2 shows a sample *makefile* that creates the *finger* executable.

```
#####################################################
#
#         FINGER Makefile
#
#####################################################

CFLAGS = -O
DESTDIR=
BINDIR=/usr/ucb

all:finger

install:all
        install -s finger $(DESTDIR)$(BINDIR)

clean:
        -/bin/rm -f finger *.o a.out core*

finger:  vmsfinger.o tcpfinger.o user.o
         cc -o finger ${CFLAGS} vmsfinger.o tcpfinger.o
         user.o -lvms -noshare -notraceback

vmsfinger.o:
         cc -c ${CFLAGS} vmsfinger.c

tcpfinger.o:
         cc -c ${CFLAGS} tcpfinger.c

user.o:
         cc -c ${CFLAGS} user.c
```

Figure 5-2. Sample Makefile

NOTE: When creating the makefile, use the Tab key to indent subordinate lines.

The *makefile* above has three macro definitions: *CFLAGS*, which is defined as *-O*, is used by the C compiler. *DESTDIR* can be specified on the command line, if necessary. *BINDIR*, which is defined as */usr/ucb*, can be written over from the command line.

Note that *rules* are separated from any *dependencies* by a colon. *Dependencies* can also be *rules* with their own dependencies (see *finger* in the above example).

The *clean*, *finger*, *vmsfinger.o*, *tcpfinger.o*, and *user.o* rules have commands associated with them.

Three examples of how the *makefile* works follow.

Example 1

In this example the *make* command is followed by the *all* rule.

```
% make all
```

1. *Make* searches the *Makefile* in the working directory until it finds the *all* rule, which is *all:finger*.
2. Since *all* is dependent upon *finger*, *make* searches for *finger*. As shown above, *finger* is dependent upon *vmsfinger.o*, *tcpfinger.o*, and *user.o*.
3. *Make* then searches for the first dependency, *vmsfinger.o*. Since *vmsfinger.o* has no dependencies, *make* executes the command *cc -c ${CFLAGS} vmsfinger.c*. It runs the C compiler (with the *-c* option and the options defined in the macro *CFLAGS*) on the file *vmsfinger.c*, creating the object file *vmsfinger.o*.
4. After completing the first dependency, *make* goes to the second dependency, *tcpfinger.o*. Since *tcpfinger.o* has no dependencies, *make* executes the C compiler command *cc -c ${CFLAGS} tcpfinger.c*. It runs the C compiler with the *-c* option and the *-O* option on the file *tcpfinger.c*, which creates the object file *tcpfinger.o*. After completing the second dependency, *make* goes to the third dependency in the *finger* rule line—*user.o*.
5. Since *user.o* has no dependencies, *make* performs the commands associated with it—*cc -c ${CFLAGS} user.c*. It runs the C compiler with the *-c* option and *CFLAGS (-O)* option on file *user.c*, creating the object file *user.o*. *Make* has now completed the three dependencies for the *finger* rule.

6. *Make* executes the command associated with *finger—cc -o finger ${CFLAGS} vmsfinger.o tcpfinger.o, user.o -lvms -noshare -notraceback.* The C compiler with the *-O* option creates the executable *finger* file. The three objects are used with the *-lvms*, *-noshare*, and *-notraceback* options for the loader (*ld*).
7. After completing the *finger* rule, *make* checks for additional dependencies or command associated with the *all* rule. Finding none, *make* exits.

Example 2

In this example the following command is entered:

```
% make install DESTDIR=/new_release
```

which invokes the *make* utility to run the *install* rule on *Makefile*. The *DESTDIR* macro is defined to be */new_release*. The *make* utility completes the following steps:

1. *Make* searches for the *makefile* in the current directory and after finding it, searches for the *install* rule. It finds *install:all.*
2. The *install* rule is dependent on the *all* rule. If *make all* has been run previously, the *all* rule is current. If *make all* has not been run, *make* performs the *all* as described in Example 1.
3. After the *all* rule is met, *make* performs the *install* command with the *-s* strip option on *finger* and stores the result in the directory specified by the macros DESTDIR and BINDIR. DESTDIR has been specified on the command line as *new_release.* BINDIR is defined as */usr/ucb. Finger* is thus installed in the directory */new_release/usr/ucb.*
4. *Make* exits after completing the *install* rule.

Example 3

The command line input for Example 3 is as follows:

```
% make clean
```

1. *Make* searches the *makefile* in the current working directory for the *clean* rule.
2. Since *clean* has no dependencies, *make* executes the command associated with it. It runs the *-/bin/rm* command, a general remove command that cleans out the directory, on the specified files: *finger*, all object files (*.o), *a.out*, and *core*.
3. After completing the *clean* rule, *make* exits.

Using the Debugger

With EUNICE you have access to the UNIX symbolic debugger—*dbx*. Symbolic debuggers enable you to work with the same names (symbols) as are in your source code. *Dbx* is a source-level rather than an assembly-level debugger, such as *adb*.

Dbx requires changes to the symbol tables generated by the various compilers. To compile programs for debugging, use the *-g* flag. For example, C programs should be compiled as follows to produce the right symbol tables:

```
% cc -g program.c -o program
```

Follow these steps to use the debugger:

1. Compile source code with the *-g* option. For example:

   % cc -g file.c -o test

2. Call the debugger by entering:

   % dbx test

3. Enter the *list* command, which causes the debugger to list your source code by the desired line number. Determine the line number at which you want to stop.

4. Enter the *stop at* command followed by the line number. For example, to stop at line 123, enter:

   stop at 123

5. Run the executable program with given options, for example:

   run -o

   causes *dbx* to stop at line 123 and to list the line's contents.

6. Enter the *next* command to step through the executable code. *Dbx* stops at the next line in the execution and lists the line number and its contents.

Note that the debugger command *print name* prints the contents of the given variable *name.*

Refer to the *dbx*(1) man page in the *UNIX User's Reference Manual* and *Debugging with dbx* in the *Programmer's Supplementary Documents* for additional information.

Notes

Chapter 6: Troubleshooting

This chapter provides a brief troubleshooting guide. Commands that may be troublesome are listed in alphabetical order. Symptoms associated with problems are described, and solutions are suggested in instances in which you can correct problems. If you need further assistance, please consult with your system administrator.

Processes Hang (go into MWAIT state)

symptom: processes go into *mwait*

solution: One must use *set notify* with the C Shell. To ensure that *set notify* is used by subprocesses, *set notify* should be put in the "*.cshrc*" file. (See Figure 2-4, The .cshrc Template.)

DECnet™ Errors

If you receive an error message that says *exceed quota, return to node vax (host)*, see your system administrator.

/etc/utmp Problem

The */etc/utmp* file is not updated for some terminal lines. See your system administrator.

Troubleshooting Problems Associated with Commands

ar(1)

symptom 1: phase error

solution: *ar(1)* archives UNIX-style object modules, not VMS. Make sure that the compiler, assembler, and loader are set to create UNIX objects. (See Chapter 5, "Selecting UNIX- or VMS-Style Object Code" for instructions.)

Note that the error message is just a warning; the *ar* command worked correctly.

symptom 2: table of contents out-of-date

solution: Use *ranlib(1)* to update the table of contents.

at(1)

symptom: output is lost, or the command does not work

solution: See your system administrator.

cc(1)

symptom 1: bad magic number

An attempt was made to load a VMS library with a UNIX object module.

solution: *cc(1)* or *f77(1)* uses the UNIX loader by default. UNIX objects cannot normally be linked with VMS objects. To get the special EUNICE 4.3 Software loader to create VMS objects with *cc(1)* or *f77(1)*, refer to Chapter 5, "Selecting UNIX- or VMS-Style Object Code," and the alias given in the *.login* file.

Example:

```
% vmsobj
% cc unix_first.obj unix_second.obj /usr/libvms/libm.olb
```

symptom 2: premature eof:

The UNIX loader is probably being used on a VMS object file. This would happen if *ld(1)* was used on objects created by *cc(1)* with *vmsobj* set or if *cc(1)* was used without setting *vmsobj* and linked with VMS objects.

solution: The *ld(1)* command does not check the environmental variables to determine whether VMS or UNIX objects are desired. If using the alias *vmsobj*, then use the *cc(1)* or *f77(1)* command. The assembler and loader for VMS style objects and executables will be used by them for both the compile and link phase.

cc(1)

symptom 3: ld: premature eof

unixobj and the *f77* compiler were used on VMS FORTRAN source code.

solution: Unlike the VMS FORTRAN compiler, *f77* is very particular about the filename extension used on the source file. In order for *f77* to function properly, use *.f*, not *.for*. Simply rename the file with the *.f* extension.

Example:

```
% mv test.for test.f
% f77 test.f
```

symptom 4: undefined symbols, __get_fderr :

UNIX C is calling VMS system calls.

solution: Make sure that UNIX_ASSEMBLER_CASE_CONVERT is undefined by entering:

```
% setenv UNIX_ASSEMBLER_CASE_CONVERT OFF
```

If code that manipulates EUNICE runtime system was created, make sure that */usr/include/eunice/eunice.h* was included.

symptom 5: ld: readit: cannot open

An attempt was made to run *f77(1)* or *cc(1)* on a file that does not have an extension.

solution: Unlike VMS compilers, UNIX compilers require an extension to be added to the filename.

For example, *f77(1)* requires a *.f* and *cc(1)* requires a *.c* extension. Be sure that proper extensions are used with a given compiler.

symptom 6: symbol table overflow

The compiler was used on large, complex programs.

solution: Any compiler feeds its symbol information into a table whose size is determined and hard-coded by the assembler. The solution is to break the program up into smaller modules.

cc(1)

symptom 7: multiply defined symbols: *vmserrno, errno, environ*

An attempt has been made to load UNIX object modules that manipulate the EUNICE runtime system (e.g., code with *#include <eunice/eunice.h>* in it).

solution: Since the UNIX loader will not produce an executable with multiply defined symbols, it is necessary to load the programs with the option *-noshare* set. This cancels the default, which loads the shareable *C* library and causes all routines to be loaded out of the standard *C* library.

symptom 8: multiple transfer address

An attempt was made to link VMS FORTRAN with UNIX C compiled under *vmsobj.*

solution: There is no transfer vector; use the options file supplied, which includes both the shareable C library, the standard C library, *crt0.obj,* and sets base=0.

$ LINK TWG$USR:[LIBVMS]CRT0.OBJ FILE1, FILE2,-
TWG$USR:[LIBVMS]:LIBC/LIB

Or, put **TWG$USR:[LIBVMS]:CRT0.OBJ** and */usr/libvms/libc.olb* in the VMS link:

$ LINK FILE1, FILE2, TWG$USR:[LIBVMS]SHARE/OPT

symptom 9: undefined symbol *abc_00f*

UNIX C was compiled with *vmsobj* calling a VMS FORTRAN subroutine, linking them in DCL with the VMS LINKER.

solution: Make sure the call to the subroutine is in lowercase. *C* distinguishes between lower and uppercase; VMS FORTRAN does not.

Define UNIX_ASSEMBLER_CASE_CONVERT OFF in DCL:

$ DEFINE/JOB UNIX_ASSEMBLER_CASE_CONVERT OFF

To use the VMS LINK command from the *csh* use:

% setenv UNIX_ASSEMBLER_CASE_CONVERT OFF

cc(1)

To test, do the following:

% vmsobj % cc -c test.c % strings test.o | more

With UNIX_ASSEMBLER_CASE_CONVERT turned on, those subroutines with uppercase or mixed case will be hashed to names such as *junk_0000f.*

symptom 10: linker -w illrectype in filename, record 1 is illegal

linker -w illrectype in filename, record 2 is illeg
linker -w illrectype in filename, no end of modu

An attempt was made to use the VMS LINKER with UNIX objects.

solution: Set *vmsobj* and recompile the UNIX sources.

symptom 11: linker -w illrectype in filename, record 1 is illegal

linker -w illrectype in filename, record 2 is illeg
linker -w illrectype in filename, no end of modu

" " illegal object language structure (13) should be 0 in module ....

The *unixobj* file type was specified, a *unixtovms* was done on that object file, and an attempt was made to link it with the VMS LINKER.

solution: *unixtovms*(1) only affects the type of the text file, not the object file. (Refer to the *unixtovms* man page in the *UNIX User's Reference Manual.*) The VMS LINKER expects both a VMS file type record (variable length) and a file compiled under *vmsobj.*

symptom 12: undefined _error, weak reference to main

An attempt was made to link VMS C or VMS FORTRAN with UNIX (*vmsobj*) C.

solution: Make sure */usr/libvms/crt0.obj* is the first module being linked in the VMS link command.

symptom 13: 512 byte record too large for buffer

An attempt was made to use a VMS compiler on a file with UNIX-style records. (512 byte blocks, line-feed, no carriage return.)

solution: Change the file type of the source to a VMS file with the command *unixtovms*(1). (See the *unixtovms*(1) man page in the *UNIX User's Reference Manual.*)

cd(1)

symptom: cd .. cannot find directory

solution: See your system administrator.

chmod(1)

symptom: *chmod* adds VMS delete protections to file

solution: EUNICE provides UNIX file protections without compromising VMS security. If umask in *.login* is used, files created under EUNICE follow the UNIX file protection conventions. If umask is not set in *.login*, files are created with VMS protections, which can be set in the LOGIN.COM file.

csh(1)

symptom: does not read the *.login* or *.cshrc* when logging in

solution: Verify that these files are in your VMS login directory and that they are owned by you. If they are, see your system administrator.

cu(1)

symptom: end-of-file not recognized when using *cu(1)* to transfer a file from System V to EUNICE 4.x

solution: VMS 4.x now implements CTRL J for line editing, but System V *cu(1)* expects CTRL Z. It is necessary to disable line editing for the terminal from the DCL by entering:

$ SET TERM/NOLINE

dd(1)

symptom: I/O error reading from tape

solution: Mount the tape from VMS with appropriate blocksize:

$ MOUNT/FOR/BLOCK=10240 MMA0:

delta(1)

symptoms: truncates files to zero, or cannot make *delta* .

solution: See your system administrator.

EUNICE.COM

symptom: global page table full

solution: See your system administrator.

eunlogin(1)

symptom: last login message says "*tty??*"

solution: See your system administrator.

@TWG$ADMIN:CSHELL.COM

symptoms: does not produce the last login message; *null no match* when accessing the *csh(1)*; no entry for you in *passwd* file

solution: See your system administrator.

f77(1)

symptom 1: symbol table overflow

solution: EUNICE 4.3 contains *f77* version 4.3c, which supports the -Nn option. This option enables you to use the layer static tables; use it when you compile code.

f77(1)

symptom 2. link w/illrectype illegal record type (32) in module # file *for*

Every record will show as an illegal record type. *f77(1)* is being used with *vmsobj* set. File names have '*.for*' extension.

solution: Rename the files with *.f* extension.

symptom 3: SYSTEM F *_opdec,opcode* reserved to DIGITAL fault at pc=00003000

```
"     " PSC=03C00C4,
trace-F-traceback symbolic
stack dump
module name     routine     line     Rel recs
crt0            $$C_text0
```

FORTRAN object modules have been created using the UNIX compiler with *vmsobj* and with the VMS linker (including the four necessary libraries).

solution: The problem is */usr/libvms/crt0.obj* expects to branch to C main. Create a simple C *main* routine that calls the FORTRAN program, such as:

```
main( )
{
/* The formal main FORTRAN program is now called fortran_prog_ *

        fortran_prog_( );
}
```

NOTE: The underscore appended after the driver name is essential. (Refer to the f77(1) man page in the UNIX User's Reference Manual for a discussion of the C/FORTRAN interface.)

The *link* command needs to bring in four libraries if UNIX *f77* object modules are being used. These libraries have hashed filenames and can be copied, not moved, to a standard VMS name.

$ LINK TWG$LIB:CRT0, [ ]DRIVC.O, [ ]FORPROG.O,-
TWG$USR:[LIB]LIB$f77/LIB,-
TWG$USR:[LIB]LIB$i77/LIB, TWG$LIB:LIBC/LIB

find(1)

symptom 1: "find / -name filename -print" doesn't work

solution: Use '/*' as the pathname. Also, be sure to specify either -print or -exec, or else *find(1)* executes but does not report anything back.

symptom 2: find: bad status < filename >

solution: This indicates that the user does not have read permission for the file, although the directory can still be read.

kill(1)

symptom: after a "kill %%" the process hangs

solution: *set notify* should be added to the *.cshrc*. If this is not done, processes can be put in *mwait* and the system will have to be rebooted to re-enable use of the process slot. (See Figure 2-4, The cshrc.csh Template.)

lint(1)

symptom: does not work

solution: See your system administrator.

lisp(1)

symptom 1: cannot open more than three files

solution: See your system administrator.

symptom 2: an image created by dumplisp may get an image activation error when run

solution: Run the VMS PATCH utility on the executable image, as follows:

```
$ PATCH/NONEW/ABSOLUTE 'image name'
 DEPOSIT/BYTE 10=1
 UPDATE
 EXIT
```

This utility can be used from a command file that accepts an image as a parameter.

lpr(1)

symptom: does not work, cannot create PRINTER:LPR

solution: See your system administrator.

mail(1)

symptom 1: mail does not get sent, but no error is returned

solution: See your system administrator.

symptom 2: instead of appending mail to the mbox, the mbox is written over

solution: *mail(1)* requires that EUNICE_1VERSION be turned ON. Purge the directory using the VMS PURGE command. (See Chapter 3, "List Contents of Directory" for instructions on setting version numbering.)

make(1)

symptom: does not work

solution: Requires that EUNICE_1VERSION be turned ON. Purge the directory using the VMS PURGE command. (See Chapter 3, "List Contents of Directory" for instructions on setting version numbering.)

man(1)

symptoms: 1) cannot unlink */tmp/catxx*; 2) cannot read */usr/lib/tmac/tmac.new* or *tab37*. The manual page is not piped to *more*(1); 3) the files */usr/man/cat** belong to the individual users who first run the *man* command on a particular entry. (This is only a problem if disk quotas are enabled.)

solution: See your system administrator.

mv(1)

symptom: does not move multiple versions correctly

solution: To emulate a native UNIX environment, use the VMS PURGE command to delete multiple file versions and turn on EUNICE_1VERSION in your LOGIN.COM file. (See Chapter 3, "List Contents of Directory" for instructions on setting version numbering.)

nroff(1)

symptom: file not found

solution: See your system administrator.

signal(2)

symptom: there is no SIGPIPE

solution: This is currently generated when a program writes to a pipe for which there is no reader.

spell(1)

symptom: cannot make pipe

solution: See your system administrator.

stty(1)

symptom 1: kill and erase not changed

solution: The VMS terminal driver, which EUNICE must use, does not have the ability to change the many terminal options supported by the UNIX terminal drivers.

symptom 2: *cbreak* does not work

solution: Try using raw mode.

symptom 3: returns message "is not a *tty*"

solution: See your system administrator.

tar(1)

symptom 1: *ls -l* shows a number instead of the owner's name

solution: These files were brought in from a real UNIX system. (This is a EUNICE feature.) You can change the owner (to the correct owner) from DCL as follows:

$ SET FILE/OWN=[LOGINAME] [...]*.*;*

symptom 2: cannot open */dev/rmt8*

solution: The tape must be mounted from VMS as a foreign tape with the correct block size. *Tar* tapes are frequently created with a blocking factor of 20, which is a block size of 10240.

$ MOUNT/FOR/BLOCK=10240 MTA0:

Tar reads from */dev/rmt8* by default. To use another drive such as */dev/mt0*, use:

% tar xv0

symptom 3: cannot read *tar* tape from native 4.3

solution: Filter through *dd(1)*, modifying the block size as appropriate. The raw device */dev/rmt0* is used for byte-by-byte transfer.

$ dd if=/dev/rmt0 ibs=10240 | tar xvf -

symptom 4: does not handle multiple versions correctly

solution: To emulate a UNIX environment, PURGE old versions of the file and turn on EUNICE_1VERSION. (See Chapter 3, "List Contents of Directory" for instructions on setting version numbering.)

touch(1)

symptom: file date not changed

solution: *touch(1)* works only on deletable UNIX format files. Use */usr/eun/vmstounix* to convert VMS files to UNIX format. (This also updates the modification date.)

tty(1)

symptom: returns message "*tty??*"

solution: See your system administrator.

uucp(1)

symptom: *uucp* will not log in to a VAX from a UNIX Version 7 System

solution: See your system administrator.

vi(1)

symptom 1: unable to create /tmp/ex00051

solution: This occurs when you are trying to write to a file and then exit *vi(1)* when there is not enough free disk space available. Check available free space with the *df(1)* command and delete any unnecessary files.

symptom 2: file not found

solution: See your system administrator.

vmsmail (1)

symptom: unable to send mail to VMS users

solution: Refer to the instructions given in Chapter 3, "Sending Mail to the VMS Mailer from UNIX." If the problem is not resolved, see your system administrator.

who(1)

symptom: does not work

solution: See your system administrator.

Notes

Notes

Index

D

E

F

G, H, I

K

L

M

N

O

P

R

S

T

U

V

W