EUNICE BSD

REFERENCE MANUAL

August 1988

THE WOLLONGONG GROUP, INC.
1129 San Antonio Road
Palo Alto, California 94303
(415) 962-7100
TWX 910-373-2085

EUNICE BSD

REFERENCE MANUAL

CONTENTS

LIST OF TABLES

Preface

This manual is intended as a reference for users of EUNICE™ BSD. The manual is divided into seven sections, as follows:

Section 1: Introduction

Explains the concept of EUNICE BSD as a MetaPort and the advantages of such a MetaPort.

Section 2: EUNICE BSD Command Availability

Contains a brief description of all of the 4.3 BSD commands, system calls, subroutines, special files, file formats, games and system maintenance programs supported, and not supported, under EUNICE BSD.

Section 3: EUNICE BSD Functional Description

This is of particular interest to programmers, as it explains how EUNICE BSD is implemented on VMS™. It includes information on UNIX® system calls (also incorporated as EUNICE NOTES to the *UNIX Programmer's Reference Manual [PRM]*) and internal EUNICE BSD calls.

Section 4: VMS/UNIX Interface

Explains how to interface VMS and UNIX modules by using the options available with the UNIX assembler and loader in order to produce VMS object code or link VMS shareable libraries with UNIX programs. This section also provides valuable hints for using the C and FORTRAN 77 compilers.

Section 5: Guide to Problem Solving

Contains a list (in alphabetical order) of the most frequently experienced customer problems reported when using EUNICE BSD commands. Many of these problems are the result of an incorrect VMS or UNIX environment setup. Solutions and clarifications to these problems are given. This is an excellent section to review when unexpected results are encountered. (*The EUNICE BSD Administrator's Guide* has a separate Appendix on installation problem solving.)

Section 6: Glossary

Section 7: Licensing Obligations

Explains the EUNICE BSD licensing agreements.

Document Conventions

The following conventions are used throughout this guide:

$	VMS system prompt.
%	UNIX system prompt.
[ ]	Enclose VMS device name. CTRL key
Bold	Literal commands to be entered exactly as shown. (Observe case sensitivity rules.)
Italics	1) Variable parameters, 2) EUNICE BSD commands, or 3) networking commands.
Regular	System response to a user's command.
<< >>	Enclose descriptions of site-specific information.

The VMS DCL (DIGITAL Command Language) converts all file name characters into upper-case letters before processing. UNIX, however, is case-sensitive, and the majority of the commands are in lower case. Examples and file names are given in VMS DCL environment format, except in those cases where the UNIX context is obvious to the reader.

Where VMS device names are to be entered, we have assumed a typical case: "DUA0:" for the disk, and "MUA0:" for the tape drive. Please change these command lines as necessary for your system configuration.

Related Publications

Other Wollongong documents in this product set are:

	EUNICE BSD User Guide
Vol I:	User's Reference Manual
Vol Ia:	SCCS Reference Manual
Vol II:	Programmer's Reference Manual
Vol III:	User's Supplementary Documents
Vol IV:	Programmer's Supplementary Documents
Vol V:	System Administrator's Guide

To order additional copies of the Wollongong publications listed here, write to:

The Wollongong Group, Inc.
Attn: Sales Dept.
1129 San Antonio Road
Palo Alto, CA 94303

or call:

(415) 962-7200

The following document contains additional information:

VAX/VMS Linker Reference Manual

EUNICE BSD

REFERENCE MANUAL

1. INTRODUCTION

This manual is intended to be a reference guide for users of EUNICE BSD. A separate *EUNICE BSD Administrator's Guide* covers installation, installation related problem solving, and EUNICE BSD administration.

1.1 THE METAPORT CONCEPT

The term "porting" refers to the process of transporting software to a particular hardware architecture. The UNIX operating system has been the object of much porting activity and is now available on many different hardware architectures. However, there is a major drawback to simply "porting" the UNIX operating system to an existing hardware architecture. Usually, such hardware already has a well established vendor supplied operating system and a large body of applications software. Since existing applications software cannot easily be ported to other environments, the cost of converting to UNIX is prohibitive for many users, even though it provides the environment desired for future growth. DEC™ VAX™ computers executing the VMS Operating System are a prime example of this dilemma.

The Wollongong Group's (Wollongong's) EUNICE BSD system provides transparent access to both VMS and UNIX environments. This concept of "coexistent environments" is called a MetaPort . The MetaPort system gives the user access to both VMS and the UNIX tools and utilities, so that previously written software and files can be executed or manipulated by either UNIX or VMS software facilities.

EUNICE BSD provides all the inherent features of a native port, plus the ability to concurrently execute both VMS and UNIX software on a single host CPU. Coexistent environments permit users to, at their option, use the UNIX environment, the VMS environment (including any other VMS software), or a hybrid environment, in which the features of *both* environments are available.

A further advantage of the EUNICE BSD MetaPort is that the user can take advantage of the excellent performance capabilities of the VMS Operating System.

1.2 FEATURES AND BENEFITS OF EUNICE BSD

Using EUNICE BSD, the user enjoys the following advantages:

A. A full UNIX environment

B. A full VMS environment

C. Intercommunication between the VMS and UNIX environments

D. A minimum of added overhead to the VMS Operating System

E. A minimum of required special privileges

F. Ability to use the standard VMS file system (RMS) (UNIX pathnames are transformed into VMS filenames)

G. Ability to run application programs written for Berkeley 4.3 BSD UNIX under VAX/VMS, with the following advantages:

1. Little or no modification to the source code is required
2. Access to a wealth of applications software implemented for UNIX
3. Use of VAX compatible hardware
4. Increased programmer productivity

2. EUNICE BSD COMMAND AVAILABILITY

The following tables outline the UNIX commands that are supported, EUNICE BSD-specific commands, and un-supported UNIX commands. Table 2-1 lists 4.3 BSD UNIX commands supported by EUNICE BSD; Table 2-2 lists EUNICE BSD specific commands; and Table 2-3 lists 4.3 BSD UNIX commands not supported by EUNICE BSD.

The *UNIX User's Reference Manual [URM]*, Section 1 contains EUNICE NOTES, with EUNICE BSD-specific information where appropriate.

Table 2-1. 4.3 BSD UNIX Commands Supported by EUNICE BSD

Section 1	Commands and Application Programs
intro	introduction to commands
adb	debugger
addbib	create or extend bibliographic database
apply	apply a command to a set of arguments
apropos	locate commands by keyword lookup
ar	archive and library maintainer
as	VAX-11 assembler
at	execute commands at a later time
awk	pattern scanning and processing language
basename	strip filename affixes
bc	arbitrary-precision arithmetic language
biff	be notified if mail arrives and who it is from
binmail	send or receive mail among users
cal	print calendar
calendar	reminder service
cat	catenate and print
cb	C program beautifier
cc	C compiler
cd	change working directory
checknr	check nroff/troff files
chmod	change mode
ci	check in RCS revisions
clear	clear terminal screen
cmp	compare two files
co	check out RCS revisions
col	filter reverse line feeds
colcrt	filter nroff output for CRT previewing

4.3 BSD UNIX Commands Supported by EUNICE BSD

Section 1 Commands and Application Programs

colrm	remove columns from a file
comm	select or reject lines common to two sorted files
compress	compress and expand data
cp	copy
crypt	encode/decode
csh	a shell (command interpreter) with C-like syntax
ctags	create a tags file
date	print and set the date
dbx	debugger
dc	desk calculator
dd	convert and copy a file
deroff	remove nroff, troff, tbl and eqn constructs
df	disk free
diction	print wordy sentences
diff	differential file and directory comparator
diff3	3-way differential file comparison
du	summarize disk usage
echo	echo arguments
ed	text editor
efl	Extended Fortran Language
eqn	typeset mathematics
error	analyze and disperse compiler error messages
ex	text editor
expand	expand tabs to spaces, and vice versa
expr	evaluate arguments as an expression
f77	Fortran 77 compiler
false	provide truth values
file	determine file type
find	find files
finger	user information lookup program
fmt	simple text formatter
fold	fold long lines for finite width output device
fp	Functional Programming language compiler/interpreter
fpr	print Fortran file
from	who is my mail from?
fsplit	split a multi-routine Fortran file into individual files
gprof	display call graph profile data
graph	draw a graph
grep	search a file for a pattern

4.3 BSD UNIX Commands Supported by EUNICE BSD

Section 1 Commands and Application Programs

groups	show group memberships
head	give first few lines
ident	identify files
indent	indent and format C program source
install	install binaries
join	relational database operator
kill	terminate a process with extreme prejudice
last	indicate last logins of users and teletypes
ld	link editor
learn	computer aided instruction about UNIX
leave	remind you when you have to leave
lex	generator of lexical analysis programs
lint	a C program verifier
lisp	lisp interpreter
liszt	compile a Franz Lisp program
ln	make links
lock	reserve a terminal
look	find lines in a sorted list
lookbib	build inverted index for a bibliography, find references in a bibliography
lorder	find ordering relation for an object library
lpr	off line print
ls	list contents of directory
lxref	lisp cross reference program
m4	macro processor
mail	send and receive mail
make	maintain program groups
man	find manual information by keywords print out the manual
merge	three-way file merge
mesg	permit or deny messages
mkdir	make a directory
mkstr	create an error message file by massaging C source
more	file perusal filter for crt viewing
mset	retrieve ASCII to IBM 3270 keyboard map
msgs	system messages and junk mail program
mt	magnetic tape manipulating program
mv	move or rename files
newaliases	rebuild the data base for the mail aliases file

4.3 BSD UNIX Commands Supported by EUNICE BSD

Section 1 Commands and Application Programs

nice	run a command at low priority (*sh* only)
nm	print name list
nroff	text formatting
od	octal, decimal, hex, ascii dump
pagesize	print system page size
passwd	change password file information
pc	Pascal compiler
pdx	pascal debugger
pi	Pascal interpreter code translator
pix	Pascal interpreter and executor
plot	graphics filters
pmerge	pascal file merger
pr	print file
printenv	print out the environment
prof	display profile data
ps	process status
ptx	permuted index
pwd	working directory name
px	Pascal interpreter
pxp	Pascal execution profiler
pxref	Pascal cross-reference program
ranlib	convert archives to random libraries
ratfor	rational Fortran dialect
rcs	change RCS file attributes
rcsdiff	compare RCS revisions
rcsmerge	merge RCS revisions
rdist	remote file distribution program
refer	find and insert literature references in documents
rev	reverse lines of a file
rlog	print log messages and other information about RCS files
rm	remove (unlink) files or directories
rmail	handle remote mail received via uucp
rmdir	remove (unlink) directories or files
roffbib	run off bibliographic database
sccs	front end for the SCCS subsystem
script	make typescript of terminal session
sed	stream editor
sendbug	mail a system bug report to 4bsd-bugs

4.3 BSD UNIX Commands Supported by EUNICE BSD

Section 1 Commands and Application Programs

sh	command language
size	size of an object file
sleep	suspend execution for an interval
soelim	eliminate ".so's" from nroff input
sort	sort or merge files
sortbib	sort bibliographic database
spell	find spelling errors
spline	interpolate smooth curve
split	split a file into pieces
strings	find the printable strings in an object, or other binary, file
strip	remove symbols and relocation bits
struct	structure Fortran programs
stty	set terminal options
style	analyze surface characteristics of a document
sum	sum and count blocks in a file
symorder	rearrange name list
tabs	set terminal tabs
tail	deliver the last part of a file
tar	tape archiver
tbl	format tables for nroff or troff
tc	photoypesetter simulator
tcopy	copy a mag tape
tee	pipe fitting
test	condition command
time	time a command
tip	connect to a remote system
tk	paginator for the Tektronix 4014
tn3270	full-screen remote login to IBM VM/CMS
touch	update date last modified of a file
tp	manipulate tape archive
tr	translate characters
troff	text formatting and typesetting
true	provide truth values

4.3 BSD UNIX Commands Supported by EUNICE BSD

Section 1 Commands and Application Programs

tset	terminal dependent initialization
tsort	topological sort
tty	get terminal name
ul	do underlining
unifdef	remove "ifdef'd" lines
uniq	report repeated lines in a file
units	conversion program
uptime	show how long system has been up
users	compact list of users who are on the system
uucp	unix to unix copy
uuencode	encode/decode a binary file for transmission via mail
uulog	display UUCP log files
uuname	list names of UUCP hosts
uuq	examine or manipulate the uucp queue
uusend	send a file to a remote host
uux	unix to unix command execution
vacation	return "I am on vacation" indication
vgrind	grind nice listings of programs
vi	screen oriented (visual) display editor based on ex
vlp	Format Lisp programs to be printed with nroff, vtroff, or troff
vwidth	make troff width table for a font
w	who is on the system and what they are doing
wait	await completion of process
wall	write to all users
wc	word count
what	show what versions of object modules were used to construct a file
whatis	describe what a command is
whereis	locate source, binary, and or manual for program
which	locate a program file including aliases and paths (*csh* only)
who	who is on the system
whoami	print effective current user id
write	write to another user
xsend	secret mail
xstr	extract strings from C programs to implement shared strings
yacc	yet another compiler-compiler
yes	be repetitively affirmative

4.3 BSD UNIX Commands Supported by EUNICE BSD

Section 2	System Calls
intro	introduction to system calls and error numbers
access	determine accessibility of file
acct	turn accounting on or off
adjtime	correct the time to allow synchronization of the system clock
brk	change data segment size
chdir	change current working directory
chmod	change mode of file
chown	change owner and group of a file
chroot	change root directory
close	delete a descriptor
creat	create a new file
dup	duplicate a descriptor
exec	execute a file
exit	terminate a process
fcntl	file control
flock	apply or remove an advisory lock on an open file
fork	create a new process
fsync	synchronize a file's in-core state with that on disk
getdtablesize	get descriptor table size
getgid	get group identity
getgroups	get group access list
getitimer	get/set value of interval timer
getpagesize	get system page size
getpgrp	get process group
getpid	get process identification
getpriority	get/set program scheduling priority
getrlimit	control maximum system resource consumption
getrusage	get information about resource utilization
gettimeofday	get/set date and time
getuid	get user identity
ioctl	control device
kill	send signal to a process
killpg	send signal to a process group

4.3 BSD UNIX Commands Supported by EUNICE BSD

Section 2	System Calls
link	make a hard link to a file
lseek	move read/write pointer
mkdir	make a directory file
open	open a file for reading or writing, or create a new file
pipe	create an interprocess communication channel
profil	execution time profile
ptrace	process trace
quota	manipulate disk quotas
read	read input
readlink	read value of a symbolic link
rename	change the name of a file
rmdir	remove a directory file
select	synchronize I/O multiplexing
setgroups	set group access list
setpgrp	set process group
setquota	enable/disable quotas on a file system
setregid	set real and effective group ID
setreuid	set real and effective user ID's
sigblock	block signals
sigpause	atomically release blocked signals and wait for interrupt
sigreturn	return from signal
sigsetmask	set current signal mask
sigvec	software signal facilities
stat	get file status
swapon	add a swap device for interleaved paging/swapping
symlink	make symbolic link to a file
syscall	indirect system call
truncate	truncate a file to a specified length
umask	set file creation mode mask
unlink	remove directory entry
utimes	set file times
vfork	spawn new process in a virtual memory efficient way
vhangup	virtually "hangup" the current control terminal
wait	wait for process to terminate
write	write output

4.3 BSD UNIX Commands Supported by EUNICE BSD

Section 3	Subroutines
intro	introduction to C library functions
abort	generate a fault
abs	integer absolute value
alarm	schedule signal after specified time
asinh	inverse hyperbolic functions
assert	program verification
atof	convert ASCII to numbers
bstring	bit and byte string operations
byteorder	correct values between host and network byte order
crypt	DES encryption
ctime	convert date and time to ASCII
ctype	character classification macros
curses	screen functions with "optimal" cursor motion
dbm	data base subroutines
directory	directory operations
ecvt	output conversion
end	last locations in program
erf	error functions
execl	execute a file
exit	terminate a process after flushing any pending output
exp	exponential, logarithm, power
fclose	close or flush a stream
ferror	stream status inquiries
floor	absolute value, floor, ceiling, and round-to-nearest functions
fopen	open a stream
fread	buffered binary input/output
frexp	split into mantissa and exponent
fseek	reposition a stream
getc	get character or word from stream
getdiskbyname	get disk description by its name
getenv	value for environment name
getfsent	get file system descriptor file entry
getgrent	get group file entry
getlogin	get login name
getopt	get option letter from argv
getpass	read a password
getpw	get name from uid

4.3 BSD UNIX Commands Supported by EUNICE BSD

Section 3	Subroutines
getpwent	get password file entry
gets	get a string from a stream
getttyent	get ttys file entry
getusershell	get legal user shells
getwd	get current working directory pathname
hypot	Euclidean distance, complex absolute value
ieee	copysign, remainder, exponent manipulations
infnan	signals invalid floating-point operations on a VAX (temporary)
initgroups	initialize group access list
insque	insert/remove element from a queue
j0	bessel functions
lgamma	log gamma function
lib2648	subroutines for the HP 2648 graphics terminal
malloc	memory allocator
math	introduction to mathematical library functions
mktemp	make a unique file name
monitor	prepare execution profile
mp	multiple precision integer arithmetic
ndbm	data base subroutines
nice	set program priority
nlist	get entries from name list
ns	Xerox NS address conversion routines
pause	stop until signal
perror	system error messages
plot	graphics interface
popen	initiate I/O to/from a process
printf	formatted output conversion
psignal	system signal messages
putc	put character or word on a stream
puts	put a string on a stream
qsort	quicker sort
rand	random number generator
random	better random number generator
regex	regular expression handler
scandir	scan a directory
scanf	formatted input conversion

4.3 BSD UNIX Commands Supported by EUNICE BSD

Section 3	Subroutines
setbuf	assign buffering to a stream
setjmp	non-local goto
setuid	set user and group ID
siginterrupt	allow signals to interrupt system calls
signal	simplified software signal facilities
sin	trigonometric functions and their inverses
sinh	hyperbolic functions
sleep	suspend execution for interval
sqrt	cube root, square root
stdio	standard buffered input/output package
string	string operations
stty	set and get terminal state (defunct)
swab	swap bytes
syslog	control system log
system	issue a shell command
termcap	terminal independent operation routines
time	get date and time
times	get process times
ttyname	find name of a terminal
ualarm	schedule signal after specified time
ungetc	push character back into input stream
usleep	suspend execution for interval
utime	set file times
valloc	aligned memory allocator
varargs	variable argument list
vlimit	control maximum system resource consumption
vtimes	get information about resource utilization
intro	introduction to FORTRAN library functions
abort	abnormal termination
access	determine accessibility of a file
alarm	execute a subroutine after a specified time
bessel	of two kinds, for integer orders
bit	and, or, xor, not, rshift, lshift bitwise functions
chdir	change default directory
chmod	change mode of a file
etime	return elapsed execution time
exit	terminate process with status
fdate	return date and time in an ASCII string
flmin	return extreme values
flush	flush output to a logical unit

4.3 BSD UNIX Commands Supported by EUNICE BSD

Section 3	Subroutines
fork	create a copy of this process
fseek	reposition a file on a logical unit
getarg	return command line arguments
getc	get a character from a logical unit
getcwd	get pathname of current working directory
getenv	get value of environment variables
getlog	get user's login name
getpid	get process id
getuid	get user or group ID of the caller
hostnm	get name of current host
idate	return date or time in numerical form
index	tell about character objects
ioinit	change f77 I/O initialization
kill	send a signal to a process
link	make a link to an existing file
loc	return the address of an object
long	integer object conversion
malloc	memory allocator
perror	get system error messages
plot	f77 library interface to plot (3X) libraries
putc	write a character to a fortran logical unit
qsort	quick sort
rand	return random values
random	better random number generator
rename	rename a file
signal	change the action for a signal
sleep	suspend execution for an interval
stat	get file status
system	execute a UNIX command
time	return system time
topen	f77 tape I/O
traper	trap arithmetic errors
trapov	trap and repair floating point overflow
trpfpe	trap and repair floating point faults
ttynam	find name of a terminal port
unlink	remove a directory entry
wait	wait for a process to terminate

4.3 BSD UNIX Commands Supported by EUNICE BSD

Section 4	Special Files
intro	introduction to special files and hardware support
cons	VAX-11 console interface
floppy	console floppy interface
mt	magnetic tape interface
null	data sink
printer	line printer interface
rta	DECnet virtual terminal interface
tty	general terminal interface
ttyp	TCP/IP virtual terminal interface (WIN/TCP™ option only)

4.3 BSD UNIX Commands Supported by EUNICE BSD

Section 5	File Formats
L-devices	UUCP device description file
L-dialcodes	UUCP phone number index file
L.aliases	UUCP hostname alias file
L.cmds	UUCP remote command permissions file
L.sys	UUCP remote host description file
USERFILE	UUCP pathname permissions file
a.out	assembler and link editor output
acct	execution accounting file
aliases	aliases file for sendmail
ar	archive (library) file format
core	format of memory image file
dbx	dbx symbol table information
dir	format of directories
disktab	disk description file
dump	incremental dump format
fs	format of file system volume
fstab	static information about the filesystems
gettytab	terminal configuration data base
group	group file
map3270	database for mapping ascii keystrokes into IBM 3270 keys
mtab	mounted file system table
passwd	password file
phones	remote host phone number database
plot	graphics interface
printcap	printer capability database
remote	remote host description file
rcsfile	format of rcsfile
stab	symbol table types
tar	tape archive file format
termcap	terminal capability database
tp	DEC/mag tape formats
ttys	terminal initialization data
types	primitive system data types
utmp	login records
uuencode	format of an encoded uuencode file
vfont	font formats for the Benson-Varian or Versatec
vgrindefs	vgrind's language definition database

4.3 BSD UNIX Commands Supported by EUNICE BSD

Section 6 Games

GAMES	are implemented, BUT NOT SUPPORTED

Section 7 Miscellaneous

intro	miscellaneous useful information pages
ascii	map of ASCII character set
environ	user environment
eqnchar	special character definitions for eqn
hier	file system hierarchy
mailaddr	mail addressing description
man	macros to typeset manual
me	macros for formatting papers
ms	text formatting macros
term	conventional names for terminals

Section 8 System Maintenance

intro	introduction to system maintenance and operation commands
catman	create the cat files for the manual
chown	change owner
cron	clock daemon
makekey	generate encryption key
mkpasswd	generate hashed password table
sendmail	send mail
uucico	transfer files queued by uucp or uux
uuclean	uucp spool directory clean-up
uupoll	poll a remote UUCP site
uusnap	show snapshot of the UUCP system
uuxqt	UUCP execution file interpreter
vipw	edit the password file

Table 2-2. EUNICE Specific Commands Supported by EUNICE BSD

Section 1	Commands and Applications Programs
cvtbackup	converts between VMS backup format and UNIX format
eunlogin	log into the EUNICE accounting
eunlogout	log out of the EUNICE accounting
filetype	provides information about the file type
mailinfo	tells the user that UNIX mail has been received
trpatch	trace patch
unixtovms	convert UNIX type file from fixed length to variable length VMS format
version	provides infomation about the EUNICE BSD version level
vms	call a VMS command from the shell
vmsas	create VMS object file from *cc(1)* assembler source
vmsld	link VMS object files into VMS executable image
vmsmail	mails to VMS mailbox
vmsname	give equivalent VMS file specification
vmstounix	convert variable length file to fixed length VMS format

Section 7	Miscellaneous
greek	graphics for extended TTY-37 type-box

Section 8	System Maintenance
cvtuaf	create */etc/passwd* from the authorize file
mketcgrp	set up groups for EUNICE
prtusers	provide information about EUNICE process accounting

Table 2-3. 4.3 BSD UNIX Commands Restricted or Not Supported by EUNICE BSD

Code Meaning

I - Implemented but not by this command name
N - Not implemented
R - Restricted by VMS
V - Done from VMS (gives some indication of where to look)
W - Separate Wollongong product - WIN/TCP

Section 1	Commands and Application Programs	code - comment
atq	print the queue of jobs waiting to be run	N - not implemented
atrm	remove jobs spooled by at	N - not implemented
chfn	change password file information	N - not implemented
chgrp	change group	N - not implemented
chsh	change password file information	N - not implemented
ftp	ARPANET file transfer program	W - WIN/TCP option only
gcore	get core images of running processes	V - use VMS utilities
hostid	set or print identifier of current host system	W - WIN/TCP option only
hostname	set or print name of current host system	W - WIN/TCP option only
iostat	report I/O statistics	R - restricted by VMS
jove	an interactive display-oriented text editor	N - not implemented User Contributed Software
lastcomm	show last commands executed in reverse order	V - accounting is not done; use VMS accountin
logger	make entries in the system log	N - not implemented
login	sign on	V - done by DCL, while eunlogin does EUNICE
lpq	spool queue examination program	N - not implemented
lprm	remove jobs from the line printer spooling queue	N - not implemented
lptest	generate lineprinter ripple pattern	N - not implemented
mh	Message Handler	N - not implemented User Contributed Software

Table 2-3. (Cont.) 4.3 BSD UNIX Commands Restricted or Not Supported by EUNICE BSD

Code Meaning

I - Implemented but not by this command name
N - Not implemented
R - Restricted by VMS
V - Done from VMS (gives some indication of where to look)
W - Separate Wollongong product - WIN/TCP

Section 1	Commands and Application Programs	code - comment
netstat	show network status	W - WIN/TCP option only
quota	display disc usage and limits	V - use VMS utilities
rcp	remote file copy	W - WIN/TCP option only
readnews	read news articles	N - not implemented User Contributed Software
rlogin	remote login	W - WIN/TCP option only
rsh	remote shell	W - WIN/TCP option only
ruptime	show host status of local machines	W - WIN/TCP option only
rwho	who's logged in on local machines	W - WIN/TCP option only
su	substitute user id temporarily	V - see SET UIC, epmi CAUTION: subprocess retains UIC
sysline	display system status on status line of a terminal	V - see VMS MONITOR utility
systat	display system statistics on a crt	V - see VMS MONITOR utility
talk	talk to another user	N - not implemented
telnet	user interface to the TELNET protocol	W - WIN/TCP option only
tftp	trivial file transfer program	W - WIN/TCP option only
vmstat	report virtual memory statistics	V - see VMS MONITOR utility
vnews	read news articles	N - not implemented User Contributed Software
whois	DARPA Internet user name directory service	W - WIN/TCP option only
window	window environment	N - not implemented

Table 2-3. (Cont.) 4.3 BSD UNIX Commands Restricted or Not Supported by EUNICE BSD

Code Meaning

I - Implemented but not this by command name
N - Not implemented
R - Restricted by VMS
V - Done from VMS (gives some indication of where to look)
W - Separate Wollongong product - WIN/TCP

Section 2	System Calls	code - comment
accept	accept a connection on a socket	W - WIN/TCP option only
bind	bind a name to a socket	W - WIN/TCP option only
connect	initiate a connection on a socket	W - WIN/TCP option only
gethostid	get/set unique identifier of current host	N - not implemented
gethostname	get/set name of current host	N - not implemented
getpeername	get name of connected peer	W - WIN/TCP option only
getsockname	get socket name	W - WIN/TCP option only
getsockopt	get and set options on sockets	W - WIN/TCP option only
listen	listen for connections on a socket	W - WIN/TCP option only
mknod	make a directory or a special file	R - no corresponding data structure in VMS
mount	mount or remove file system	V - see VMS MOUNT command
reboot	reboot system or halt processor	V - Use VMS command
recv	receive a message from a socket	W - WIN/TCP option only
send	send a message from a socket	W - WIN/TCP option only
sigstack	set and/or get signal stack context	N - not implemented
shutdown	shut down part of a full-duplex connection	N - not implemented
socket	create an endpoint for communication	W - WIN/TCP option only
socketpair	create a pair of connected sockets	W - WIN/TCP option only

Table 2-3. (Cont.) 4.3 BSD UNIX Commands Restricted or Not Supported by EUNICE BSD

Code Meaning

I - Implemented but not this by command name
N - Not implemented
R - Restricted by VMS
V - Done from VMS (gives some indication of where to look)
W - Separate Wollongong product - WIN/TCP

Section 3	Subroutines	code-comment
gethostbyname	get network host entry	W - WIN/TCP option only
getnetent	get network entry	W - WIN/TCP option only
getprotoent	get protocol entry	W - WIN/TCP option only
getservent	get service entry	W - WIN/TCP option only
inet	Internet address manipulation routines	W - WIN/TCP option only
rcmd	routines for returning a stream to a remote command	W - WIN/TCP option only
resolver	resolver routines	W - WIN/TCP option only
rexec	return stream to a remote command	W - WIN/TCP option only

Table 2-3. (Cont.) 4.3 BSD UNIX Commands Restricted or Not Supported by EUNICE BSD

Code Meaning

I - Implemented but not this by command name
N - Not implemented
R - Restricted by VMS
V - Done from VMS (gives some indication of where to look)
W - Separate Wollongong product - WIN/TCP

Section 4	Special Files	code-comment
acc	ACC LH/DH IMP interface	W - WIN/TCP option only
ad	Data Translation A/D converter	R - use VMS for drivers
arp	Address Resolution Protocol	W - WIN/TCP option only
autoconf	diagnostics from the autoconfiguration code	R - use VMS for drivers
bk	line discipline for machine-machine communication (obsolete)	R - use VMS for drivers
cons	VAX-11 console interface	R - use VMS for drivers
crl	VAX 8600 console RL02 interface	R - use VMS for drivers
css	DEC IMP-11A LH/DH IMP interface	R - use VMS for drivers
ct	phototypesetter interface	R - use VMS for drivers
ddn	DDN Standard Mode X.25 IMP interface	W - WIN/TCP option only (known as dda)
de	DEC DEUNA 10 Mb/s Ethernet interface	W - WIN/TCP option only
dh	DH-11/DM-11 communications multiplexer	R - use VMS for drivers
dhu	DHU-11 communications multiplexer	R - use VMS for drivers
dmc	DEC DMC-11/DMR-11 point-to-point communications device	W - WIN/TCP option only
dmf	DMF-32, terminal multiplexor	R - use VMS for drivers
dmz	DMZ-32 terminal multiplexor	R - use VMS for drivers
dn	DN-11 autocall unit interface	R - use VMS for drivers
drum	paging device	R - use VMS for drivers
dz	DZ-11 communications multiplexer	R - use VMS for drivers
ec	3Com 10 Mb/s Ethernet interface	R - use VMS for drivers
en	Xerox 3 Mb/s Ethernet interface	R - use VMS for drivers
ex	Excelan 10 Mb/s Ethernet interface	R - use VMS for drivers
fl	console floppy interface	R - use VMS for drivers
hdh	ACC IF-11/HDH IMP interface	W - WIN/TCP option only
hk	RK6-11/RK06 and RK07 moving head disk	R - use VMS for drivers
hp	MASSBUS disk interface	R - use VMS for drivers
ht	TM-03/TE-16,TU-45,TU-77 MASSBUS magtape interface	R - use VMS for drivers

Table 2-3. (Cont.) 4.3 BSD UNIX Commands Restricted or Not Supported by EUNICE BSD

Code Meaning

I - Implemented but not this by command name
N - Not implemented
R - Restricted by VMS
V - Done from VMS (gives some indication of where to look)
W - Separate Wollongong product - WIN/TCP

Section 4	Special Files	code - comment
hy	Network Systems Hyperchannel interface	W - WIN/TCP option only
icmp	Internet Control Message Protocol	R - use VMS for drivers
idp	Xerox Internet Datagram Protocol	R - use VMS for drivers
ik	Ikonas frame buffer, graphics device interface	R - use VMS for drivers
il	Interlan NI1010 10 Mb/s Ethernet interface	W - WIN/TCP option only
imp	1822 network interface	W - WIN/TCP option only
imp	IMP raw socket interface	W - WIN/TCP option only
inet	Internet protocol family	W - WIN/TCP option only
ip	Internet Protocol	W - WIN/TCP option only
ix	Interlan Np100 10 Mb/s Ethernet interface	R - use VMS for drivers
kg	KL-11/DL-11W line clock	R - use VMS for drivers
lo	software loopback network interface	W - WIN/TCP option only
lp	line printer	R - use VMS for drivers
mem	main memory	R - use VMS for drivers
mtio	UNIX magtape interface	R - use VMS for drivers
np	Interlan Np100 10 Mb/s Ethernet interface	R - use VMS for drivers
ns	Xerox Network Systems protocol family	R - use VMS for drivers
nsip	software network interface encapsulating ns packets in ip packets.	R - use VMS for drivers
pcl	DEC CSS PCL-11 B Network Interface	R - use VMS for drivers
ps	Evans and Sutherland Picture System 2 graphics device interface	R - use VMS for drivers
pty	pseudo terminal driver	W - WIN/TCP option only
qe	DEC DEQNA Q-bus 10 Mb/s Ethernet interface	W - WIN/TCP option only

Table 2-3. (Cont.) 4.3 BSD UNIX Commands Restricted or Not Supported by EUNICE BSD

Code Meaning

I - Implemented but not this by command name
N - Not implemented
R - Restricted by VMS
V - Done from VMS (gives some indication of where to look)
W - Separate Wollongong product - WIN/TCP

Section 4	Special Files	code - comment
rx	DEC RX02 floppy disk interface	R - use VMS for drivers
spp	Xerox Sequenced Packet Protocol	R - use VMS for drivers
tb	line discipline for digitizing devices	R - use VMS for drivers
tcp	Internet Transmission Control Protocol	W - WIN/TCP option only
tm	TM-11/TE-10 magtape interface	R - use VMS for drivers
tmscp	DEC TMSCP magtape interface	R - use VMS for drivers
ts	TS-11 magtape interface	R - use VMS for drivers
tu	VAX-11/730 and VAX-11/750 TU58 console cassette interface	R - use VMS for drivers
uda	UDA-50 disk controller interface	R - use VMS for drivers
udp	Internet User Datagram Protocol	W - WIN/TCP option only
up	unibus storage module controller/drives	R - use VMS for drivers
ut	UNIBUS TU45 tri-density tape drive interface	R - use VMS for drivers
uu	TU58/DECtape II UNIBUS cassette interface	R - use VMS for drivers
va	Benson-Varian interface	R - use VMS for drivers
vp	Versatec interface	R - use VMS for drivers
vv	Proteon proNET 10 Megabit ring	W - WIN/TCP option only

Table 2-3. (Cont.) 4.3 BSD UNIX Commands Restricted or Not Supported by EUNICE BSD

Code Meaning

I - Implemented but not this by command name
N - Not implemented
R - Restricted by VMS
V - Done from VMS (gives some indication of where to look)
W - Separate Wollongong product - WIN/TCP

Section 5	File Formats	code - comment
hosts	host name data base	W - WIN/TCP option only
networks	network name data base	W - WIN/TCP option only
protocols	protocol name data base	W - WIN/TCP option only
resolver	resolver configuration file	W - WIN/TCP option only
services	service name data base	W - WIN/TCP option only

Table 2-3. (Cont.) 4.3 BSD UNIX Commands Restricted or Not Supported by EUNICE BSD

Code Meaning

I - Implemented but not this by command name
N - Not implemented
R - Restricted by VMS
V - Done from VMS (gives some indication of where to look)
W - Separate Wollongong product - WIN/TCP

Section 6	Games	code - comment
aardvark	yet another exploration game	Implemented, Not supported
adventure	an exploration game	Implemented, Not supported
arithmetic	provide drill in number facts	Implemented, Not supported
backgammon	the game	Implemented, Not supported
banner	print large banner on printer	Implemented, Not supported
battlestar	a tropical adventure game	Implemented, Not supported
bcd	convert to antique media	Implemented, Not supported
boggle	play the game of boggle	Implemented, Not supported
canfield	the solitaire card game canfield	Implemented, Not supported
chess	the game of chess	Implemented, Not supported
ching	the book of changes and other cookies	Implemented, Not supported
cribbage	the card game cribbage	Implemented, Not supported
doctor	interact with a psychoanalyst	Implemented, Not supported
fish	play "Go Fish"	Implemented, Not supported
fortune	print a random, hopefully interesting, adage	Implemented, Not supported
hangman	Computer version of the game hangman	Implemented, Not supported
hunt	a multi-player multi-terminal game	Implemented, Not supported
mille	play Mille Bournes	Implemented, Not supported
monop	Monopoly game	Implemented, Not supported
number	convert Arabic numerals to English	Implemented, Not supported
quiz	test your knowledge	Implemented, Not supported
rain	animated raindrops display	Implemented, Not supported
robots	fight off villainous robots	Implemented, Not supported
rogue	Exploring The Dungeons of Doom	Implemented, Not supported
sail	multi-user wooden ships and iron men	Implemented, Not supported
snake	display chase game	Implemented, Not supported
trek	trekkie game	Implemented, Not supported
worm	Play the growing worm game	Implemented, Not supported
worms	animate worms on a display terminal	Implemented, Not supported
wump	the game of hunt-the-wumpus	Implemented, Not supported
zork	the game of dungeon	Implemented, Not supported

Table 2-3. (Cont.) 4.3 BSD UNIX Commands Restricted or Not Supported by EUNICE BSD

Code Meaning

I - Implemented but not by this command name
N - Not implemented
R - Restricted by VMS
V - Done from VMS (gives some indication of where to look)
W - Separate Wollongong product - WIN/TCP

Section 8	System Maintenance	code - comment
XNSrouted	XNS Routing Information Protocol daemon	N - not implemented
ac	login accounting	V - use VMS accounting
adduser	procedure for adding new users	N - not implemented
arff	archiver and copier for floppy	N - not implemented
arp	address resolution display and control	W - WIN/TCP option only
bad144	read/write dec standard 144 bad sector information	V- VMS uses RMS
badsect	create files to contain bad sectors	V - VMS uses RMS
bugfiler	file bug reports in folders automatically	N - Not implemented
clri	clear i-node	N - Not implemented
comsat	biff server	N - Not implemented
config	build system configuration files	W - WIN/TCP option only
crash	what happens when the system crashes	R - use VMS utilities
dcheck	file system directory consistency check	V - see VMS ANALYZE
diskpart	calculate default disk partition sizes	V - use VMS utilities
dmesg	collect system diagnostic messages to form error log	V - see errorlog in SYS$MANAGER
drtest	standalone disk test program	V - use VMS utilities
dump	incremental file system dump	V - use VMS BACKUP
dumpfs	dump file system information	V - use VMS utilities
edquota	edit user quotas	V - use VMS utilities
fastboot	reboot/halt the system without checking the disks	V - use VMS file SHUTDOWN.COM
fingerd	remote user information server	W - WIN/TCP option only
format	how to format disk packs	V - use VMS INITIALIZE
fsck	file system consistency check and interactive repair	N - not implemented
ftpd	DARPA Internet File Transfer Protocol server	W - WIN/TCP option only
gettable	get NIC format host tables from a host	W - WIN/TCP option only
getty	set terminal mode	N - not implemented
halt	stop the processor	V - use VMS utilities
htable	convert NIC standard format host tables	W - WIN/TCP option only
icheck	file system storage consistency check	V - use VMS ANALYZE or VERIFY utility
ifconfig	configure network interface parameters	W - WIN/TCP option only

Table 2-3. (Cont.) 4.3 BSD UNIX Commands Restricted or Not Supported by EUNICE BSD

Code Meaning

I - Implemented but not this by command name
N - Not implemented
R - Restricted by VMS
V - Done from VMS (gives some indication of where to look)
W - Separate Wollongong product - WIN/TCP

Section 8	System Maintenance	code - comment
implog	IMP log interpreter	W - WIN/TCP option only
implogd	IMP logger process	W - WIN/TCP option only
inetd	internet "super–server"	W - WIN/TCP option only
init	process control initialization	V - done by VMS
kgmon	generate a dump of the operating system's profile buffers	N - not implemented
lpc	line printer control program	N - not implemented
lpd	line printer daemon	N - not implemented
makedev	make system special files	N - not implemented
mkfs	construct a file system	V - done by VMS
mkhosts	generate hashed host table	N - not implemented
mklost+found	make a lost+found directory for *fsck*	V - use VMS ANALYZE
mknod	build a special file	R - no corresponding data structure in VMS
mkproto	construct a prototype file system	N - not implemented
mount	mount and dismount file system	V - use VMS MOUNT command
named	Internet domain name server	W - WIN/TCP option only
ncheck	generate names from i-numbers	R - No inodes in VMS
newfs	construct a new file system	V - use VMS utilities
pac	printer/plotter accounting information	V - use VMS utilities
ping	send ICMP ECHO_REQUEST packets to network hosts	W - WIN/TCP option only
pstat	print system facts	V - use VMS utilities
quot	summarize file system ownership	V - use VMS utilities
quotacheck	file system quota consistency checker	V - use VMS utilities
quotaon	turn file system quotas on and off	V - use VMS utilities
rc	command script for auto-reboot and daemons	I - done by STARTEUNICE.COM
rdump	file system dump across the network	N - not implemented
reboot	UNIX bootstrapping procedures	V - use VMS file SHUTDOWN.COM

Table 2-3. (Cont.) 4.3 BSD UNIX Commands Restricted or Not Supported by EUNICE BSD

Code Meaning

I - Implemented but not by this command name
N - Not implemented
R - Restricted by VMS
V - Done from VMS (gives some indication of where to look)
W - Separate Wollongong product - WIN/TCP

Section 8	System Maintenance	code - comment
renice	alter priority of running processes	N - not implemented
repquota	summarize quotas for a file system	V - use VMS utilities
restore	incremental file system restore	V - use VMS BACKUP
rexecd	remote execution server	N - not implemented
rlogind	remote login server	W - WIN/TCP option only
rmt	remote magtape protocol module	N - not implemented
route	manually manipulate the routing tables	W - WIN/TCP option only
routed	network routing daemon	W - WIN/TCP option only
rrestore	restore a file system dump across the network	N - not implemented
rshd	remote shell server	W - WIN/TCP option only
rwhod	system status server	W - WIN/TCP option only
rxformat	format floppy disks	V - use VMS utilities
sa	system accounting	V - use VMS accounting utilities
savecore	save a core dump of the operating system	V - use VMS utilities
shutdown	close down the system at a given given time	V - use VMS SYS$SYSTEM:SHUTDOWN
slattach	attach serial lines as network interfaces	N - not implemented
sticky	executable files with persistent text	R - see EUNICE SUCHMOD.COM
swapon	specify additional device for paging and swapping	R - VMS takes care of designating devices
sync	update the super block	R - under VMS control
syslogd	log systems messages	N - not implemented
talkd	remote user communication server	W - WIN/TCP option only
telnetd	DARPA TELNET protocol server	W - WIN/TCP option only
tftpd	DARPA Trivial File Transfer Protocol server	W - WIN/TCP option only
timed	time server daemon	W - WIN/TCP option only
timedc	timed control program	W - WIN/TCP option only
trpt	transliterate protocol trace	W - WIN/TCP option only
trsp	transliterate sequenced packet protocol trace	W - WIN/TCP option only
tunefs	tune up an existing file system	N - not implemented
update	periodically update the super-block	R - under VMS control

3. EUNICE BSD FUNCTIONAL DESCRIPTION

EUNICE BSD is a software environment which supports Berkeley 4.3 BSD UNIX software under VAX/VMS. It provides a UNIX system call emulation package which allows UNIX programs to run under VMS with little or no modification. This section describes some of the features and limitations of EUNICE BSD. With EUNICE BSD, users have the choice of operating in a UNIX environment using any UNIX shell, or in a VMS environment with the ability to use tools available under UNIX.

File name coercion in EUNICE BSD allows the user to enter file specifications in either UNIX or VMS format. Data coercion in EUNICE BSD allows UNIX programs to read and write both VMS files and UNIX files and to read VMS directory files as though they were UNIX directories. In order to provide a complete emulation of a UNIX environment, EUNICE BSD provides for the UNIX concept of ROOT ("/") and DEVICE (" */dev* ") directories.

Although EUNICE BSD was primarily designed to provide users with a UNIX environment, VMS users will find that they can conveniently make use of available UNIX tools without having to learn how to use UNIX. Furthermore, UNIX users can access VMS tools from the UNIX shells. In general, both VMS and UNIX file specifications may be used when specifying files to UNIX programs. Thus, a VMS user could use the UNIX *diff* program without having to know that it is a UNIX program.

3.1 FILE NAMES

EUNICE BSD uses the standard VMS file system. All references to file names in EUNICE BSD are passed transparently through an internal file name parsing routine. If the file name is a VMS file specification, it is passed unchanged to the VMS Record Management Services for processing. If the file name is a UNIX file specification, it is transformed into a VMS file name before being passed on to the VMS Record Management Services. Avoid using UNIX file names with extensions of numerical digits only, since VMS interprets more than one numerical extension as a file version number. For example, *filename.1* will be transformed correctly, but *filename.1.1* will not. With a name of *filename.1.1*, the second *.1* will be interpreted as a file version number. VMS logical names are used to mount UNIX file systems in the UNIX file hierarchy.[1] There is, as well, full support for "." and ".." in UNIX file names. To complete the UNIX file hierarchy, EUNICE BSD allows the system manager to set up pseudo root ("/") and device ("*/dev*") directories. A reference to */dev/device* will cause EUNICE BSD to look for a VMS logical name mapping *device* into its equivalent VMS device name.[2]

Since there are typically many terminals on a VAX, an excessive number of VMS Logical Names would ordinarily be required to map all the terminal devices from their UNIX names into their VMS names. In order to remedy this, EUNICE BSD first checks for the device logical name translation and, if this is not found, EUNICE BSD uses an internal algorithm to convert from a terminal name of the form */dev/ttyNNN* (where NNN is a decimal number) to a VMS terminal name (e.g., */dev/tty10* becomes *TTB2:*). Checking the logical name first allows sites to change names when necessary, but still provides a default for those sites which have no special requirements.

1. Creating a VMS logical name TWG$USR which translates to DBA6: will mount the entire file structure on DBA6: as /usr. EUNICE BSD will then translate the file specification /usr to DBA6:[000000] and /usr/src/cmd/file.ext to DBA6:[SRC.CMD]FILE.EXT. EUNICE BSD supports the full range of UNIX file names, including multiple extensions and case-sensitive file names.

2. Creating a VMS logical name NULL which translates into NLA0: will allow EUNICE BSD users to specify the null device as either NLA0: or /dev/null

Programs may make the root directory the working directory (using *chdir*) but may not create files in the root directory. Programs cannot currently make */dev* the working directory.

3.2 COERCION

EUNICE BSD performs many different kinds of coercion to make the VMS environment appear to be a UNIX environment to programs running under EUNICE BSD.

3.2.1 File I/O

Table 3-1 lists the many different forms in which VMS files may be stored.

Table 3-1. VAX/VMS File Formats

Format	Description
VMS text	Variable length records with implied new lines.
VMS data	Variable or fixed length records.
UNIX data	Fixed length 512 byte records (last record may be truncated).
UNIX text	This is a UNIX data file, but the "linefeed" character (octal 12) is interpreted by UNIX programs to mean "End of Line".
Directories	Variable length record VMS directory entries.
VMS I/O devices	Variable or fixed length records, depending on device characteristics.

When a program running under EUNICE BSD performs a *read* on a file, the data read from the file through the VMS Record Management Services is changed into a representation that UNIX programs will understand. VMS data records are passed unchanged to the program, although the read request will be truncated to the length of the record so that users may make use of record length information. VMS text records have a "linefeed" appended to them before being passed on to the program, so that UNIX programs can find the end of line. Reads on VMS text files will fill the entire user buffer unless an End of File is encountered during the read. This is consistent with the way in which files are read under UNIX. There is no distinction in UNIX between text and data files. Data in UNIX files is returned to programs unmodified. In all cases, data may be buffered internally to EUNICE BSD in order that the calling programs may view the data as a byte stream.

VMS directory records are converted into UNIX directory records before being passed on to the program. Thus, programs like *ls* (the UNIX directory listing program) run unmodified under EUNICE BSD. The VMS File Identification Number is treated by EUNICE as a UNIX *inode* number. This is provided to programs that request such UNIX specific information. The file name is converted to lower case and returned as the UNIX file name. The latest (highest) version of the file is returned with no version number and older (lower) versions of the file are returned as *filename.ext.version*.

Reads on VMS devices are coerced according to the device characteristics, although in most cases EUNICE BSD will not buffer the data coming in from the device. This is consistent with most UNIX raw devices and should not affect programs being ported to VMS via EUNICE BSD. The standard UNIX *ioctl* calls are supported by EUNICE BSD; facilities also exist for writing *ioctl* handlers for non-standard devices. The *ioctl* handlers are used to turn a UNIX *ioctl* request into its equivalent device-dependent request on VMS.

When a program running under EUNICE BSD performs a write on a file, the data will be appropriately coerced into the form expected in the file. UNIX files (which are the default created by

the *creat* call) are written with the data unchanged, with EUNICE BSD buffering data internally to give the appearance of a byte stream file. Unlike UNIX, directory files may not be written through EUNICE BSD. All attempts to *write* to directory files will return an appropriate error code. A *write* to a VMS data file creates a record boundary at the end of the data presented to the *write(2)* call, thus allowing users to control record length themselves (e.g., to create valid VMS object files). This is also the case with *writes* to most VMS devices, since it is usually desirable to not have EUNICE BSD perform any internal buffering on raw devices. A *write* to a VMS text file requires EUNICE BSD to search for "linefeed" characters and break the records up at end of line boundaries. In this case, internal buffering may be involved to make the file appear as a byte stream.

Since EUNICE BSD uses 512 byte buffers internally, no single line in a VMS text file may be longer than 511 characters (plus an end of line). If a line being buffered by EUNICE BSD exceeds the 512 byte buffer, EUNICE BSD proceeds to scan backwards from the end of the now full buffer for a word delimiter (space or tab) at which to break the line. If no such delimiter is found, the line is arbitrarily broken at the 512 byte boundary. No data is lost in this case, but extra "End of Line" characters may be seen when the data is read back by a program. This solution was considered preferable to having the program generate an exception condition and exit. In the few instances where programs have generated very long text lines (e.g., in Lisp), the technique of searching for the last word boundary has worked quite well.

Since the VMS Record Management Services are used for almost all I/O operations, EUNICE BSD will transparently support I/O over DECnet™. In fact, devices and file hierarchies on other DECnet nodes may be placed in the UNIX file hierarchy of a given machine by the standard EUNICE BSD method of mounting file structures through VMS logical names.

3.2.2 Pipes

UNIX pipes are implemented using VMS mailboxes (a virtual I/O device). The default mailbox size used by EUNICE BSD is 512 bytes. Therefore, writes to a pipe will block once there are 512 bytes waiting to be read from the pipe (in UNIX the quantity is 4096 bytes). It is extremely rare for a UNIX program to use the fact that it may write 4096 bytes without blocking. In this case, the user has the option of going into the EUNICE BSD data structures at run time and changing the default mailbox size to 4096 bytes.[3]

Since mailboxes provide a more general interprocess communication (IPC) mechanism than UNIX pipes, a non-UNIX IPC facility is also provided with EUNICE BSD. This facility allows for pipe-like objects on which the user can perform *read* and *write* operations identical to those on pipes. The *write* operation guarantees that the buffer written will be delivered through the IPC facility as a single atomic entity (as long as the buffer is no larger than the maximum message length specification for the mailbox -- which is under the control of the program creating the mailbox).

Unlike UNIX pipes, these objects need not be inherited from a parent process. They are given unique names which programs may use with the *open* call to gain access to a given IPC object. Access permission to these objects may be specified by the creating program in the same way that access permissions to files are specified. The creating program may also give these objects specific names which are recognized locally (to a specific group of processes) or globally (to the entire system); this greatly simplifies the task of creating and communicating with specialized server processes. Typically, a user might create a uniquely named IPC object, open a globally named IPC object for access to a

3. A file of "C" definitions of all internal EUNICE BSD data structures is included with EUNICE BSD so that users who require the ability to manipulate the internal state of EUNICE BSD or who wish to add/replace routines in EUNICE BSD may do so.

server, and then pass the name of its entity in a message to the server for a reply channel.

3.2.3 File Protection

EUNICE BSD translates file protection information and specifications. Tables 3-2 and 3-3 show the correspondence between VMS protection and UNIX protection:

Table 3-2. Protection Groups

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

NOTE: UNIX has no equivalent to the *system* protection group. EUNICE BSD does not display system protection information when protection information is returned. EUNICE BSD allows all access privileges to *system* (although in *creat* the default VMS protections are also applied) when protection is specified.

Table 3-3. Protection Codes

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

NOTE: UNIX provides no delete protection code. EUNICE BSD does not display delete protection information when protection information is returned. EUNICE BSD allows delete access to all users when protection is specified. Thus, delete protection is specified by the write protection of the directory in which the file resides; this is consistent with the way UNIX handles delete protection.

If *umask* is not used in the *.login* file, files EUNICE BSD creates use the default protection set on the VMS level. If you do specify *umask* in the *.login* file, files EUNICE BSD creates use the protection specified in *umask*. In addition, VMS delete bits for the system world, group, and user are also set.

VMS file protection options requested from within EUNICE BSD will supercede UNIX protections; therefore, if one wishes to apply more versatile protections to files on the system than those possible with UNIX, one can easily apply VMS protections by using the 'vms set prot=...' command. See the *UNIX User's Reference Manual [URM]*, Section 1, *vms(1)*.

3.2.4 Time

Binary time formats must also be translated. In UNIX, time is represented as the number of elapsed seconds since midnight January 1st, 1970 Greenwich Mean Time (GMT). In VMS, time is represented as 100 nanosecond units from midnight November 17, 1858 with no specific time zone as a reference. EUNICE BSD translates time formats internally for all system calls which deal with time. This makes time information and specification transparent to UNIX programs.

To deal with the lack of time zone information in VMS, EUNICE BSD uses two system-wide logical names for acquiring the time zone information required to translate between VMS and UNIX time representations. The logical name GMT_DSP specifies the displacement west of GMT of the current site (e.g., in San Francisco GMT_DSP should be assigned as "0-08:00:00", or 8 hours west of GMT). Negative values (e.g., "0-08:00:00") may be used to specify displacements east of GMT. The logical name DST_FLAG specifies whether or not daylight savings time is currently in effect. When daylight savings time is in effect DST_FLAG should be assigned as "ON". When daylight savings time is not in effect, the DST_FLAG should not exist. These two logical name assignments are generally placed in the site-specific VMS system startup command file.

3.2.5 Environment Variables

The Berkeley 4.3 BSD UNIX environment is correctly emulated by EUNICE BSD. Just as in native UNIX, environment variables are passed down to sub-processes. Additionally, the routine *getenv* checks the VMS logical name tables if a variable is not found in the environment. Thus, the VMS logical name facility supplements the normal 4.3 BSD UNIX environment. Note, VMS process logical name tables do not get passed to subprocesses.

3.2.6 stat and fstat

The *stat* and *fstat* file status calls also require that VMS dependent data be translated into a form acceptable to UNIX programs. The VMS File Identification Number becomes the UNIX *inode* number. File creation and modification times are translated to UNIX time format. The UNIX file access time is set to the same value as the modification time, as VMS does not keep track of read accesses to files. Device major and minor identification numbers are generated by EUNICE BSD such that the major device identification numbers are unique across VMS devices and the minor device identification number returns the device unit number. The file protection information is translated as specified above and file characteristics (i.e., directory file, block oriented device, character oriented device) are translated into their UNIX equivalent.

3.3 VFORKCLI

One of the big problems in producing a UNIX environment under VMS is that UNIX programs tend to create many sub-processes, while VMS creates sub-processes very slowly. To overcome this deficiency in VMS, EUNICE BSD does not release a sub-process once it has finished executing. Every time EUNICE BSD creates a new sub-process, it maps a special Command Language Interpreter known as *vforkcli* into the sub-process control region (P1 Space) so that *vforkcli* controls the execution of the sub-process. When the sub-process completes and tries to exit, *vforkcli* gains control; after returning all of its resources to VMS *vforkcli* puts the process into a "hybernation" state. Now, when EUNICE BSD requires another sub-process it can just resume this sub-process and tell the *vforkcli* to execute the program in the subprocess. This is considerably more efficient than creating a new VMS sub-process for every UNIX sub-process.

In order to minimize the impact of these "kept" processes on VMS system performance, *vforkcli* releases all the resources it can to VMS just before going to sleep. *vforkcli* also sets up a timer interrupt for five minutes from the time it goes to sleep. If EUNICE BSD requests a new sub-process from *vforkcli* before the five minute timer expires, then the timer is cancelled and *vforkcli* executes the sub-process request. If the timer expires, then *vforkcli* removes the VMS sub-process from the system. Thus, unneeded sub-processes will slowly disappear from the system and release all of their resources. In addition to its role in making EUNICE BSD sub-process creation more efficient, *vforkcli* is also used to implement other features of UNIX usually performed by the UNIX kernel.

3.3.1 Signals and Job Control

vforkcli plays a role in the delivery of certain UNIX signals.[4] Table 3-4 describes the various UNIX signals and the VMS exceptions or mechanisms which are used to implement them:

4. All UNIX signals may be sent to another process using the *kill* call.

Table 3-4. UNIX Signals

Signal	Description	VMS mechanism
SIGHUP	Terminal hang up	*** not a VMS concept***
SIGINT	Interrupt signal	VMS Control-C AST mechanism
SIGQUIT	Quit signal	VMS Control-C AST mechanism (If SIGINT=SIG_DFL)
SIGILL	Illegal Instruction	VMS Exception handling (Illegal instruction, Reserved Operand, Reserved Addr. Mode)
SIGTRAP	Trace Trap	VMS Exception handling (Trace Trap)
SIGIOT	IOT Instruction	*** no IOT on VAX ***
SIGEMT	EMT Instruction	VMS Exception handling (Compatibility Mode Trap)
SIGFPE	Float Point Exception	VMS Exception handling (Overflow/Underflow...)
SIGKILL	Kill	*** kill (External Signal) ***
SIGBUS	Bus Error	VMS Exception handling (Access violation)
SIGSEGV	Segmentation Violation	VMS Exception handling (Length Violation)
SIGSYS	System call error	*** not implemented ***
SIGPIPE	Broken Pipe	*** not yet implemented ***
SIGALRM	Alarm Clock	VMS Timer AST mechanism
SIGTERM	Software Terminate	*** kill (External Signal) ***
SIGSTOP	Stop Process	*** kill (External Signal) ***
SIGTSTP	Stop from *tty*	VMS ^Y AST mechanism (*vforkcli*)
SIGCONT	Continue Process	*** kill (External Signal) ***
SIGCHLD	Child Stop/Exit	VMS asynchronous MailBox I/O
SIGTTIN	BackGround Read	*** generated internally ***
SIGTTOU	BackGround Write	*** generated internally ***
SIGTINT	Input Character	VMS asynchronous *tty* I/O
SIGXCPU	CPU time limit	*** not implemented ***
SIGXFSZ	File size limit	*** not implemented ***

vforkcli generally deals with the signals involved in the Job Control System. *vforkcli* sets up and controls delivery of CTL-Y, CTL-C, and CTL-T Asynchronous System Traps (AST). The CTL-Y AST may then generate a SIGTSTP signal for the sub-process, causing it to call back *vforkcli* to perform a process stop. If the sub-process does not own the *tty* then both CTL-Y and CTL-C ASTs are suppressed. The CTL-T AST prints information about the process state. *vforkcli* also receives interrupts whenever the sub-process gains or loses ownership of the *tty*. It then generates an internal signal to the sub-process informing the sub-process of its current state of *tty* ownership (e.g., when the sub-process attempts to read from the *tty* it always checks for ownership first. If it does not own the *tty* then a SIGTTIN signal is generated). All signals marked as not implemented may, however, be generated from Programs via the kill UNIX System Call/Program.

3.3.2 UNIX Sub-Process Creation

EUNICE BSD currently implements both the *vfork* (Berkeley's Virtual Memory efficient version of fork) and *fork* system calls for process creation. *vfork* differs from *fork* in that when *vfork* is called, the parent process is put to sleep and the newly created child process executes in the same address space as the parent until an *exit* or *exec* system call is performed. On an *exec* system call, the child process gets its own address space and continues. The parent then resumes execution. In almost all

applications *vfork* can directly replace *fork*. There are, however, a few UNIX programs which rely on the address space copying effect of *fork* for their correct execution. *vforkcli* plays a special role in the *vfork* and *fork* system calls since it is *vforkcli* which sets up the sub-process address space according to the specifications of the *vfork* or *fork* system call.

The EUNICE BSD implementation of the *vfork* and *fork* calls correctly passes UNIX context (open files, environment, etc.) to the child process. VMS does not provide the ability to share file position pointers provided by UNIX. Thus, under EUNICE BSD, inherited UNIX file descriptors are not really shared with the parent. UNIX programs which rely on this feature will not run correctly on EUNICE BSD.[5]

EUNICE BSD emulates the most used form of file position pointer sharing, wherein a sub-process inherits a file descriptor with the file pointer at the end of file. The sub-process appends output to the end of the file and then returns control to the parent process which will then correctly see the new end of file position.

3.3.3 Debugging Aids

vforkcli is used to implement the UNIX *ptrace* and *core dump* debugging aids. *vforkcli* responds to *ptrace* requests from the parent process and performs the requested operation on the sub-process. *vforkcli* also intercepts signals in the traced sub-process and stops the sub-process, returning the signal information to the parent process. Thus, programs like *adb* and *dbx* (the UNIX debuggers) will run on EUNICE BSD. If an untraced program encounters one of the signals which, under UNIX, would normally cause a *core* file to be produced, *vforkcli* creates a *core* file which may then be examined with *adb* or *dbx*.

3.4 UNIX AND VMS OBJECT FILES

When using UNIX compilers under EUNICE BSD, the user has a choice of generating either VMS or UNIX object files. VMS object files may then be linked with other VMS object modules (from other VMS languages), while UNIX object files will make EUNICE BSD seem more like a UNIX environment, and used with UNIX programs which understand the format of UNIX object files. In order to use the UNIX *dbx* debugger you **must** generate UNIX object files, since they contain debugging information for *dbx*.[6] The UNIX loader has been modified to also accept VMS shareable libraries and link them into the resultant image, which allows users to include modules written in VMS languages by linking them as shareable libraries. The image produced by the UNIX loader is directly executable by VMS, but also contains a UNIX header (in the second block of the file).

Those sites with UNIX licenses will also be shipped a shareable "C" library which was produced by compiling all of the routines in *libc* and combining them into a single shareable library. Since most UNIX programs consist mainly of the various library routines (like the STDIO package) this results in a significant savings in both memory usage and disk space. By default, the UNIX loader will include this shareable image. Refer to Section 4.1 for more information.

5. Fortunately, there are very few UNIX programs which rely on this UNIX feature.
6. With UNIX object files you must also use the UNIX loader *ld*.

3.5 UNIX SYSTEM CALLS

NOTE: Some of these entries have been included as EUNICE NOTES in the *UNIX Programmer's Reference Manual [PRM].*

This section describes the UNIX system calls emulated under EUNICE BSD. EUNICE BSD follows the UNIX system call convention of having routines return a value greater than or equal to zero on success and a value of -1 on failure. The type of system call failure is stored in the global variable *errno*. If a VMS error code is also available, then it is stored in the global variable *vmserrno*. The following is a list and short description of the system call routines available through EUNICE BSD.

3.5.1 access

status = access (Filename, Accessmode);

access determines if *Filename* is accessible using the mode *Accessmode* (0=read,1=write,2=read and write). The *access* routine returns 0 if the file is accessible and -1 if it is not accessible. The reason for inaccessibility is stored in the global symbols *errno* (for UNIX error return status) and *vmserrno* (for VMS error return status).

3.5.2 alarm

old_time_to_go = alarm (New_time_to_go);

alarm is used in conjunction with the signal system to deliver a *SIGALRM* signal at a specified time from the time of the call to *alarm*. Specifying *New_time_to_go* as zero will cancel an outstanding alarm request. *alarm* returns the time remaining in any previous alarm call (which is overridden by the new value).

3.5.3 brk

status = brk (Address);

brk is used to directly set the new upper limit of a program's dynamically allocated data area to *Address*. If the call is successful *brk* returns 0.

Also see *sbrk*.

3.5.4 chdir

status = chdir (Directory_Name); *chdir* changes the calling program's current working directory to *Directory_Name*.[7] *Directory_Name* is first checked for accessibility before being set as the current working directory. Both UNIX and VMS directory specifications may be used in *Directory_Name*. *Status* is returned as zero if the *chdir* is successful or -1 if it is unsuccessful.

7. The directory "/dev" may not be used as an argument to *chdir*. If the directory "/dev" is used, *chdir* will return an error indicating that the directory is not accessible.

3.5.5 chmod

status = chmod (Filename, Mode);

chmod changes the mode of file *Filename* to *Mode*. The only modes recognized are the various protection modes for *user*, *group* and *other*. VMS delete access is turned on for all groups and all access permissions are turned on for the group SYSTEM.

3.5.6 chown

status = chown (Filename, User_ID, Group_ID);

chown changes the ownership of the file specified by *Filename* to the owner specified by *User_ID* and *Group_ID*. Use of the *chown* call is restricted to users with VMS *SYSPRV* privilege. Since the group ID is already contained within the user ID under EUNICE BSD, the *Group_ID* argument to *chown* is ignored.

3.5.7 close

status = close (File_Descriptor);

close is used to close an open UNIX file descriptor. If executed in *vfork* state, the file may not be truly closed until an *exit* or *exec* call is issued. If the file described by *File_Descriptor* has been "dup"ed, the file is not closed until all the file descriptors referring to the file have been closed. A *close* on a pipe which is open for write access sends an "End of File" message down the pipe, which is interpreted at the other end (by EUNICE BSD) to mean that the writer of the pipe has gone away and all further reads on the pipe should return 0. A *close* on a pipe which is not "owned" by a process does not send the "End of File" message. Ownership of a pipe is given to the first sub-process that inherits the pipe from the creating process.

3.5.8 creat

File_Descriptor = creat (Filename, Mode [,optional EUNICE BSD args....]);

creat creates a new file of name *Filename* and with access modes *Mode*. Only the protection information in *mode* is used and is "and"ed with both the *umask* and VMS default protections to produce the real protection for the file.

Unlike UNIX, which will truncate an existing file which is specified in a *creat* call, EUNICE BSD will always create a new version of the file (with a higher VMS version number). EUNICE BSD also allows the specification of extra (optional) arguments to a *creat* call, allowing the user to specify additional information to VMS and to take advantage of some EUNICE BSD features. The following optional arguments may be specified in any combination:[8]

ctg Create a contiguous file (this argument must be followed by an integer argument specifying the number of bytes to be allocated to the file).

8. By default (without the optional parameters) EUNICE BSD will create a UNIX file, which consists of fixed length 512 byte records and in which a "linefeed" means End of Line if the data it contains is to be interpreted as text.

txt Create a VMS text file. The file is given the attributes RAT=CR and variable length records. Writes to this file will be coerced to legal VMS text file writes.

var Create a VMS variable length data file. Data written to this file will not be changed when it is put into the file (i.e., no line splitting is done at "linefeed"s). This would be the kind of file specified if one were going to write a VMS object file.

tmd Create a temporary file which will be automatically delete when closed. (The same effect can be obtained using the unlink call on an opened file, but this guarantees that the file will really go away, even if the program bombs out).

1version Force *creat* to only allow one version of the file.

nversion Force *creat* to allow multiple versions of the file.

Thus, to specify the creation of a contiguous (1000 byte preallocated) VMS text file which is to be deleted on close and gives total access to all users, one would call *creat* as:

File_Descriptor = creat(file,0777,ctg,1000,txt,tmd);

To use the EUNICE BSD IPC, the optional parameter *ipc* is specified. The *creat* call then becomes:

File_Descriptor = creat(equivname,protection,ipc,maxmessagesize, prm[or, tmp], &unit);

The string *equivname* is used to specify a name for the IPC object. *protection* (same as the mode parameter) is used to put file access type protection on the IPC object. *maxmessagesize* specifies the longest length message (in bytes) which is to be guaranteed to be atomic. Messages longer than this length will be split into multiple messages which may arrive through the IPC facility mixed with other messages. The next argument is specified as either *prm* or *tmp* to indicate whether the IPC object is to be temporary (is automatically deleted when all channels to it are closed) and is to have **no** system-wide global *equivname* associated with it, or is to be permanent (remains when all channels to it are closed) and is to be known globally by the name *equivname*. *unit* specifies the address of a short integer to receive the unit number of the mailbox created by this call. The unique name for the IPC object can be generated by appending the ASCII form of the unit number to the string "MBA", and then appending a ":". Issuing a *creat* IPC call on a global object which already exists merely opens another channel to the object (the *open* call can be used for this as well, but an error will be returned if the object does not exist).

3.5.9 dup

New_File_Descriptor = dup [dup2] (Old_File_Descriptor);

Both *dup* and *dup2* create duplicate file descriptors. *dup2* differs from *dup*, in that the user specifies the file descriptor into which *Old_File_Descriptor* is to be duplicated. Within a program, *dup* ed file descriptors really access the same file byte stream (since, internally to EUNICE BSD, they both point to the same data structures for manipulating the file). The *dup* call selects the first free file descriptor for the result of the duplication.

3.5.10 execve

status = execve [exect] (Filename, Argv, Env);

execve is the root routine of a family of library routines used to execute a new program image. If *Filename* has no extension, both *Filename* and *Filename.exe* are checked for. If found, the VMS image activator is called to map the new image into memory for execution. The new image is checked to see whether or not it contains EUNICE BSD routines. If it does, the *Argv* and *Env* information is passed to the image. If it does not, the image is called as though DCL had started it. A DCL callback by the image to obtain the argument list will retrieve the first entry in *Argv*. When running in *vfork* state, an *exec* call causes a real sub-process to be invoked for the execution of the new image (using file descriptors and the supplied *Argv* and *Env* lists) and the parent process to be resumed. If *Filename* specifies a VMS text file with execute permission enabled, the *exec* call looks at the first line of the file. A line of the form "*#! filespec*" cause the *filespec* to be used as the shell which is to execute the file. Otherwise the UNIX error ENOEXEC is returned in *errno*.

exect is the same as *execve* but causes the sub-process to enter the stopped/traced state as soon as possible. This is used by the UNIX debuggers to start a traced sub-process.

3.5.11 exit

exit [_exit] (Status);

These two calls are used to terminate the execution of a program. *exit* also calls *_cleanup* to allow the Standard I/O package in the UNIX "C" library to clean up any internal buffers. If the *exit* or *_exit* calls are executed while in *vfork* state, the pseudo sub-process executing as the child is terminated and the parent resumed. The *status* returned to the parent process is the completion status of the child.

3.5.12 fork

child_pid = fork( );

fork creates a sub-process with an exact copy of the parent's address space. Currently, shared memory is not inherited by the sub-process (which gets a private copy of the shared memory at the time of the fork call). It should be replaced with *vfork* whenever possible.

3.5.13 stat and fstat

status = stat [fstat] (Filename, Stat_Buffer);

stat and *fstat* return information about files. The *stat* call tends to be somewhat inefficient since it must open the file to get the information to be returned. The only difference between the EUNICE BSD and UNIX versions of the *status* call is that EUNICE BSD returns the access time with the same value as the modified time, because VMS does not maintain access information for the file. File sizes for variable length record files will be slightly greater than the number of bytes actually returned when reading through the file, due to the fact that the file contains record control information which is counted in the file size.

3.5.14 time and ftime

Current_Time = time (&Timeloc);
ftime (&Ftime_Buffer);

time and *ftime* return, respectively, the current time (in seconds from 00:00:00 GMT January 1, 1970) and detailed information about the current time (time, milliseconds, timezone and daylight savings time flag).

3.5.15 getenv

string = getenv (Environment_String);

getenv looks in the environment and returns the value (a string) of the specified *Environment_String*. If the *Environment_String* is not found in the UNIX environment then the VMS logical name tables are searched.

3.5.16 getgid

Group_Id = getgid [getegid] ( );

getgid returns the VMS Group Identification Code of the process. *getegid* returns the effective Group Identification Code, which is always the same as the Group Identification Code returned by *getgid.*

3.5.17 getpgrp

Process_Group = getpgrp (Process_ID);

getpgrp returns the process group of the specified *Process_ID*. This is part of the Berkeley Job Control system and is used in identifying *tty* ownership.

3.5.18 getpid

Process_Id = getpid( );

getpid returns, as its value, the process identification number of the current process. Since VMS process identification numbers have a process slot re-use count in the high-order 16 bits, the returned value of *getpid* will usually be greater than 65535. UNIX programs which rely on the process identification number for generating temporary file names will have to mask off the high order 16-bits of the result of *getpid.*

3.5.19 getuid

User_Id = getuid [geteuid] ( );

getuid returns the process' VMS User Identification Code "or"ed with the VMS Group Identification Code (after the Group Identification Code has been shifted left by 8-bits), thus generating a unique number for each user on the system. *geteuid* returns the same value as *getuid.*

3.5.20 gtty

status = gtty [stty] (File_Descriptor, Buffer);

gtty and *stty* respectively retrieve and set teletype parameters if the specified *File_Descriptor* is connected to a terminal. If not, an error is returned. Some of the effects of *stty* are local to the process doing the *stty* call. *tty* modes like *ECHO* and *RAW* are global. Any UNIX programs which rely on *stty* changing terminal parameters globally may not run correctly. *stty* terminal parameter changes are always reflected in other file descriptors open to the same terminal in the same process.

3.5.21 ioctl

status = ioctl (File_Descriptor, Command, Buffer);

ioctl is used to provide device-dependent access to I/O devices. The standard UNIX *ioctl* commands are provided with EUNICE BSD. Additional device-dependent commands may be added by users.

3.5.22 kill

status = kill (Process_ID, Signal);

kill sends the signal number *Signal* to the process specified by *Process_ID*. *kill (0, Signal)* sends *Signal* to the process' Process Group.

3.5.23 killpg

status = killpg (Process_Group, Signal);

killpg sends the signal number *Signal* to the Process Group specified by *Process_Group*. *killpg (0, Signal)* sends *Signal* to the invoking process' Process Group.

3.5.24 link and unlink

status = link (source, destination);
status = unlink (Filename);

unlink is used to delete files. On UNIX, the *unlink* routine decrements a file reference count and deletes the file only when the file reference count reaches zero. Since VMS does not have reference counts associated with files, EUNICE BSD assumes that the link count is always one. This causes *unlink* to delete the file. *unlink* will also accept the name of an IPC object with the *prm* attribute and delete it.

Since VMS does not support reference counts, the *link* call cannot be as general under EUNICE BSD as it is under UNIX. EUNICE BSD emulates the interaction between *link* and *unlink* which is used in UNIX to perform the file rename function. The call to *link* does nothing but verify the *source* operand and remember the *destination* operand. If an *unlink* call is made and the *Filename* operand matches the *link source* operand, the file specified by *Filename* is renamed to the saved *destination* name.

3.5.25 lseek

new_position = lseek (File_Descriptor, Offset, Whence);

lseek positions the read/write pointer in an open file to the byte position specified by *Offset* and *Whence.*

Whence=0 → position = Offset
Whence=1 → position = current position + Offset
Whence=2 → position = End of File + Offset

On UNIX type files, all *lseek* operations work correctly. Doing an *lseek* past the end of file position causes enough zero bytes to be appended to the file to reach the new position. This has the same effect as the same operation on UNIX but will be much less efficient if many zero bytes need to be written. On all files with variable length records (VMS text files, directories and VMS data files with variable length records) *lseek* must read through the file to find the desired byte. Random *lseeks* to variable length record files can be very inefficient. The most efficient pattern of positioning in a variable length record file is to order the *lseek* calls to have increasing byte addresses in the file. *lseeks* beyond the end of file on variable length record files is illegal. Writes to variable length record files may only occur at the end of file position, attempts to write to any other position will cause an error.

3.5.26 nice

nice (Increment);

nice is used to change the priority of the current process. Every increment/decrement of 4 to *nice* becomes a decrement/increment of 1 to VMS. Sub-processes created by *fork* or *vfork* inherit the priority of the parent process. A sub-process can have its priority changed by *nice* when in the parent *vfork* state.

3.5.27 open

File_Descriptor = open (Filename, Mode);

This routine opens the file *Filename* for access with the mode *Mode* (Read=0, Write=1, Read and Write=2). The file read/write pointer is positioned at byte 0 of the file.

3.5.28 pause

pause( );

This routine places the process in a paused state, usually to await the arrival of a signal. The actual process state entered is the VMS hibernate state.

3.5.29 pipe

status = pipe (File_Descriptor_Array);

The *pipe* call creates a UNIX pipe and places a file descriptor open for reading in the first element of *File_Descriptor_Array*; a file descriptor open for writing is placed in the second element of *File_Descriptor_Array*. Currently, broken pipes are only detected when the process writing to the pipe closes it.

3.5.30 profil

profil (Buffer, Buffer_Size, PC_Offset, Scale);

profil controls profiling of program execution. It will produce a PC histogram of program execution in the buffer *Buffer* of size *Buffer_Size* starting from address *PC_Offset* using the scale *Scale*. Greater detail on this system call is available in the *UNIX Programmer's Reference Manual [PRM]*. The actual sampling interrupt is derived in a different manner than the UNIX sampling interrupt; however, sampling is still done every 1/60th of a second and produces the correct results.

3.5.31 ptrace

status = ptrace (Request, Process_ID, Address, Data);

ptrace is used by the UNIX debuggers to control the execution of a sub-process. User programs almost never use this system call. See the *UNIX Programmer's Reference Manual [PRM]* for more detail.

3.5.32 read

number_of_bytes = read (File_Descriptor, Buffer, Buffer_Size);

This routine performs a read operation on the file/device open on *File_Descriptor*. The maximum number of bytes to be transferred is *Buffer_Size*, while the number of bytes actually transferred is returned as the function value. Depending on the device or type of file being read, the number of bytes read may be less than the number of bytes requested. In most cases, the number of bytes read will be the same as the number of bytes requested (except at end of file).

3.5.33 sbrk

address = sbrk (Increment);

sbrk is used to dynamically expand/contract the highest address in *P0* space used by the program. *Increment* specifies the number of bytes to expand (if positive) or contract (if negative), rounded to a 1024 byte boundary. If *Increment* is zero, the current highest address in the program is returned with no change in *P0* space. Since EUNICE BSD uses the VMS *$EXPREG* system call to expand *P0* space, there is no guarantee that allocations using *sbrk* will be contiguous (both EUNICE BSD and VMS Record Management Services (RMS) allocate memory for internal data structures and buffers using the *$EXPREG* system call).

If you have a program which is sensitive to the state of *P0* space and requires contiguous allocations in *P0* space, link the program with *prealloc* in */lib/libc.a*. This will cause EUNICE BSD to use preallocated (static) data structures rather than dynamically allocating them at run time. The current version of *prealloc* in */lib/libc.a* is set up for 10 simultaneously open files. To ensure that RMS does not touch *P0* space, a linker options file is required with the specification:

iosegment=NNN,NOP0BUFS

Where *NNN* is the number of *P1* space pages to use as RMS buffer space. A number on the order of

100 to 250 is quite sufficient. NOP0BUFS is a linker option which ensures that RMS does not touch *P0* space.

3.5.34 setpgrp

status = setpgrp (Process_ID, Process_Group);

setpgrp is part of the Berkeley Job Control facility and is used to change the process group (and thus the *tty* ownership) of the specified process. If *Process_ID* is specified as 0, the current process is affected.

3.5.35 sighold

sighold (Signal_Number);

sighold (part of the new signal system) is used to cause the signal specified by *Signal_Number* to be held and not delivered until a *sigrelse* is done.

3.5.36 sigignore

sigignore (Signal_Number);

sigignore (part of the new signal system) is used to cause the signal specified by *Signal_Number* to be ignored.

3.5.37 signal

old_signal_value = signal (Signal_Number, New_Signal_Value)

signal is used to enable/disable the delivery of various flavors of UNIX interrupts and to specify the routines to handle them. A new signal value of SIG_DFL returns the interrupt action to its default value (cause a program exit). A new signal value of SIG_IFN causes the specified signal to be ignored. A signal value of SIG_HOLD causes the specified signal to be held. All other values specify the address of a routine to handle that signal. If the routine address is odd then the even address (1 lower) is used to specify the routine and the signal goes into the hold state after it has been delivered. Signal handling routines are called in the following way:

signal_handler(signal_number)

3.5.38 sigpause

sigpause (Signal_Number);

sigpause is part of the new signal system. It automatically un-holds *Signal_Number* and then places the process in a *paused* state.

3.5.39 sigrelse

sigrelse (Signal_Number);

sigrelse is part of the new signal system. It releases a previously held signal.

3.5.40 sigset

old_signal_value = sigset (Signal_Number, New_Signal_Value);

sigset is part of the new signal system. It has the same function as *signal*, but signal actions are permanent and are never reset upon delivery of specified signal.

3.5.41 times

times (Buffer);

times returns CPU time information about the current process and terminated children of the current process. The system time information is always set to zero (as VMS includes this in the user time figure).

3.5.42 umask

oldmask = umask (Complmode);

umask sets the *creat* file access mode mask. This mask is used to clear bits in the mode information supplied in the *creat* call to produce the true creation mode information. Note that in the *creat* call, the VMS default protection information is used to further restrict the access to the created file.

3.5.43 utime

utime (file,timep);

utime changes what the system thinks is the last accessed/modified time of a specified file. Since VAX/VMS does not keep track of accessed times, the accessed time is made the same as the modified time.

3.5.44 vfork

Process_Id = vfork( );

vfork is used to create a sub-process in a virtual-memory efficient way. Since the vast majority of *fork* calls almost immediately do an *exec*, it is very wasteful to copy a potentially large virtual address space to the child process only to have the child process immediately throw it away. To fix this problem, Berkeley implemented a call which is almost exactly the same as *fork* but avoids the address space copying. The *vfork* call creates a sub-process which shares the address space with its parent. On EUNICE BSD this is not a true sub-process, just a separate thread of execution. There are various flags for EUNICE BSD which indicate that the process is in *vfork* state. The parent is placed in a non-

running state until the sub-process does an *exit* or *exec* call. On the *exec* call, the sub-process gets its own address space (into which the new image is placed) and the parent is resumed. The value returned from *vfork* indicates which process is executing (zero means the sub-process is executing, non-zero -- the process ID of the created sub-process -- means the parent is executing). The vast majority of UNIX programs can use the *vfork* call as a direct substitute for the *fork* call.

3.5.45 vread

number_of_bytes = vread [vwrite](File_Descriptor,Buffer,Buffer_Size);

vread/vwrite is potentially more efficient when doing I/O to large files. *vread* maps data from a file into virtual memory and *vwrite* updates the file with only those pages which have been changed. See the *UNIX Programmer's Reference Manual [PRM]* for more details.

3.5.46 vtimes

vtimes (Parent_Info, Child_Info);

vtimes is similar to *times* but is used to get extended information about the parent and child processes (e.g., paging information).

3.5.47 wait

Process_Id = wait (&Status);

wait causes the process to suspend until one of its children processes (if any) has terminated.

3.5.48 wait3

Process_Id = wait3 (&Status, Options, Vtimep);

wait3 is similar to *wait* but includes options to return an error if no terminated child exists or to return information about stopped but untraced children. *Vtimep* is an optional pointer to a *vtime* structure to receive extended information about a terminated child process.

3.5.49 write

number_of_bytes = write (File_Descriptor, Buffer, Buffer_Size);

This routine performs a write operation on the file/device open on *File_Descriptor. Buffer_Size* is the number of bytes to be transferred, while the number of bytes actually transferred is returned as the function value. These two values should always agree, unless an error has occurred.

3.6 INTERNAL EUNICE BSD CALLS

This section describes internal EUNICE BSD calls available to the programmer who wishes to use EUNICE BSD internals. Though Wollongong does not recommend such a practice, these routines may be used for better program control.

3.6.1 cvt_unix_to_vms

vms_filename_size = cvt_unix_to_vms (UNIX_Filename,VMS_Filename)

cvt_unix_to_vms is a EUNICE BSD internal routine which converts a UNIX file name into its equivalent VMS file name. All the work of dealing with mounted UNIX file structures and converting */dev/device* specifications into the correct VMS device name is handled by this routine. The *UNIX_Filename* parameter is a standard null terminated "C" string. The *VMS_Filename* parameter is the address of a *char* array which will receive the translated file name. The size of this string is returned as the value of the function *Cvt_Unix_To_VMS*. To force a null terminated string use:

```
size=cvt_unix_to_vms(unix_filename, vms_filename);
vms_filename[size]='\0';
```

3.6.2 _$bbss

was_set = _$bbss (&Variable);

_$bbss performs an uninterruptable test and set on the low order bit of *Variable*. It returns a boolean value indicating whether or not the bit was set.

3.6.3 _$call_signal_routine

_$call_signal_routine (ClrAST_Flag, Signal_Number, PC_PSL, R0_R1);

_$call_signal_routine is the common routine for delivering a UNIX signal. *ClrAst_Flag*, if true, causes $CLRAST to be called to clear any VMS Asynchronous System Trap state. *Signal_Number* is the signal to be delivered. *PC_PSL* points to the PC/PSL pair and *R0_R1* points to the R0/R1 pair.

3.6.4 _$check_tty_ownership

value = _$check_tty_ownership( );

EUNICE BSD uses *_$check_tty_ownership* to call *vforkcli* to determine the state of *tty* ownership for the current process. *value* is returned with flags which indicate the conditions NOT_IN_TTY_PGROUP and LTOSTOP_SET. In addition, the internal flags used by EUNICE BSD to determine *tty* handling are set by the call to *_$check_tty_ownership*. This routine is not recommended for use outside the EUNICE BSD runtime system.

3.6.5 _$deliver_signal

_$deliver_signal (Signal_Number);

_$deliver_signal is used to deliver the signal number *Signal_Number*;

3.6.6 _$do_process_group

_$do_process_group (Process_Group, Process_Descriptors, Routine, Arg);

_$do_process_group is used to iterate through all of the processes in the current process' Process Group calling *Routine* to deal with each process based on the arguments in *arg*. It is also used to implement such things as *killpg*.

3.6.7 _$ediv

quotient = _$ediv (Divisor, Dividend, &Remainder);

_$ediv does a quadword divide (See VAX Architecture Handbook).

3.6.8 _$emul

_$emul (Multiplier, Multiplicand, Addend, Product);

_$emul does a quadword multiply (See VAX Architecture Handbook).

3.6.9 _$flush_subprocess

_$flush_subprocess (Process_ID, ForkComm_Range);

_$flush_subprocess is used to delete a bogus sub-process from the global data base. This is most often called by EUNICE BSD when an inconsistency is detected in the global data base. *ForkComm_Range* points to the global section used by the bogus sub-process.

3.6.10 _$fp

frame_pointer = _$fp( );

_$fp returns the hardware frame pointer for the current procedure.

3.6.11 _$free_fabrab

_$free_fabrab (FABRAB);

_$free_fabrab frees the specified FABRAB structure for re-use.

3.6.12 _$get_buffer

buffer = _$get_buffer( );

_$get_buffer is used to get a 512 byte buffer from EUNICE BSD.

3.6.13 _$get_data_segment

seg_no = _$get_data_segment (Size,Segment_Range,ForkComm_Range);

_$get_data_segment gets a shared memory data segment through which one can pass parameters to a sub-process. *seg_no* is the returned data segment number. The address range into which the data segment is mapped is in *Segment_Range*. *ForkComm_Range* points to the global section used by the process to which parameters will be passed via the shared memory segment.

3.6.14 _$get_defdev

device_name = _$get_defdev( );

_$get_defdev returns a character string which contains the device name of the current working directory.

3.6.15 _$get_defdir

device_name = _$get_defdir( );

_$get_defdir returns a character string containing the current working directory.

3.6.16 _$get_fabrab

fabrab = _$get_fabrab( );

_$get_fabrab is used to get a new FABRAB structure from EUNICE BSD on which a file may be opened. The returned FABRAB structure has already been initialized as much as possible; the caller need only fill in the FAB and RAB fields necessary to access the desired file.

3.6.17 _$get_gmt_info

seconds_from_gmt = _$get_gmt_info( );

_$get_gmt_info returns the number of seconds west of GMT and sets up the GMT Information cache.

3.6.18 _$get_subprocess

process_descriptor = _$get_subprocess (ForkComm_Range);

_$get_subprocess returns the *process_descriptor* of a sub-process which can then be used to complete a *fork* or *vfork* operation. *ForkComm_Range* points to the global section used by the sub-process which will be used to complete a *fork* or *vfork* operation.

3.6.19 _$Get_TTY_Pgrp

TTY_Pgrp = _$Get_TTY_Pgrp (ForkComm_Pointer);

_$Get_TTY_Pgrp returns the Process Group to which the *tty* belongs. *ForkComm_Pointer* is the address into which the FORKCOMM global section has been previously mapped.

3.6.20 _$get_tt_name

tty_name = _$get_tt_name( );

_$get_tt_name returns a character string containing the name of the controlling *tty* for this process.

3.6.21 _$hash_device_name

hash = _$hash_device_name (FABRAB);

_$hash_device_name looks at the device name in the supplied FABRAB structure and returns a 32-bit value with the low order 16-bits set to the major/minor ID number for the device and the high order 16-bits set to the device unit number.

3.6.22 _$initialize_process_info

_$initialize_process_info (Process_Descriptor);

_$initialize_process_info should be called **only** by the EUNICE BSD runtime system. It is used to initialize global information about a process which is using the global process (FORKCOMM) data base for the first time.

3.6.23 _$locc

nbytes = _$locc (Search_Character, String_Length, Address);

_$locc searches for *Search_Character* in the string specified by *String_Length* and *Address* and returns the number of characters left in the string. (See the VAX Architecture Handbook.)

3.6.24 _$LTOSTOP_Is_On

status = _$LTOSTOP_Is_On( );

_$LTOSTOP_Is_On returns a true or false value in *status* to indicate whether or not LTOSTOP mode is on for the controlling *tty* of this process.

3.6.25 _$lock_forkcomm

_$lock_forkcomm (ForkComm_Ptr);

_$lock_forkcomm locks FORKCOMM shared memory so that the caller may then update information there. *ForkComm_Ptr* should point to the place in memory where the FORKCOMM shared memory has been mapped. *_$lock_forkcomm* waits for the shared memory to be in an unlocked state before locking it. Various timeouts also exist to prevent resource lockout should someone lock the shared memory and never release it.

3.6.26 _$map_forkcomm

forkcomm_ptr = _$map_forkcomm (ForkComm_Range);

_$map_forkcomm maps the FORKCOMM shared memory into the *ForkComm_Range* specified area of memory. If *ForkComm_Range*[0] is 0, then the mapping is done at the current end of *P0* space using $EXPREG.

3.6.27 _$move

_$move (Length, Source, Destination);

_$move moves *Length* bytes from *Source* to *Destination.* See the VAX Architecture Handbook for the MOVC3 instruction.

3.6.28 _$release_data_segment

_$release_data_segment (Segment_Number, ForkComm_Range);

_$release_data_segment is used to give a global data segment back to the system for re-use.

3.6.29 _$release_forkcomm

_$release_forkcomm (ForkComm_Ptr);

_$release_forkcomm unlocks the FORKCOMM global memory.

3.6.30 _$release_subprocess

_$release_subprocess (Process_ID, ForkComm_Range);

_$release_subprocess is used to release the process specified by *Process_ID* once it has been allocated for a *fork* or *vfork* operation. *$release_subprocess* is usually used when an error occurs after a sub-process has been allocated. *ForkComm_Range* is a standard VMS Range Descriptor for the area in which the FORKCOMM global section has been mapped.

3.6.31 _$sbrk_prealloc

_$sbrk_prealloc (Npages);

_$sbrk_prealloc uses $EXPREG to preallocate *Npages* of virtual memory for use in the *sbrk* system call. It can be used to ensure contiguous *sbrk* allocation.

3.6.32 _$set_file_info

_$set_file_info (File_Descriptor, FABRAB);

_$set_file_info looks at the information in the supplied FABRAB structure for the file on *File_Descriptor* and sets up various flags inside the FABRAB structure to allow EUNICE BSD to correctly coerce data "coming from/going to" the file into a byte stream.

3.6.33 _$Set_TTY_Pgrp

_$Set_TTY_Pgrp (Process_Group);

_$Set_TTY_Pgrp changes the ownership of the process' controlling *tty* to *Process_Group.*

3.6.34 _$Setup_Term_MBX

_$Setup_Term_MBX( );

_$Setup_Term_MBX sets up the process termination mailbox so that child processes can report their termination to this process. EUNICE BSD uses *_$Setup_Term_MBX* to make sure that a termination mailbox exists before any sub-processes are created.

3.6.35 _$signal_init

_$signal_init( );

EUNICE BSD uses *_$signal_init* to initialize the signal sub-system.

3.6.36 _$Signal_UnInit

_$Signal_UnInit( );

_$Signal_UnInit is used to return the signal system to its uninitialized state. *_$signal_init* can then be used to re-initialize the signal system.

3.6.37 _$Stop_Process

_$Stop_Process (Signal_Number);

_$Stop_Process stops the current process and reports the signal number *Signal_Number* to the parent process. EUNICE BSD uses *_$Stop_Process* to stop a process when it tries to do *tty* I/O without owning the *tty*.

3.6.38 _$Startup_Unix / _$Startup_Vms

_$Startup_Unix ();
_$Startup_Vms();

These two routines are called by the "C" startup code (in *crt0*) to start the EUNICE BSD world from either from DCL or from a UNIX *exec* call.

3.6.39 _$Subtract_Quadword

_$Subtract_Quadword (Subtrahend, Difference);

_$Subtract_Quadword performs a quadword subtract of *Subtrahend* from *Difference.*

3.6.40 _$Unopen

_$Unopen (File_Descriptor);

_$Unopen causes the file on *File_Descriptor* to be closed in such a way that when any I/O operation is later done on *File_Descriptor* the file is automatically re-opened.

4. VMS/UNIX INTERFACE

EUNICE BSD allows a VMS program to call UNIX utilities or source code developed under UNIX. Conversely, EUNICE BSD allows a UNIX program to utilize VMS services.

This section describes how to generate object files which can be linked with VMS style objects into VMS executables or with UNIX style objects into UNIX executables. There is also a description of how to include the C library in a VMS program, and conversely, how to include VMS shareable executables in a UNIX C program. Information is also included about the shareable C library, which can be accessed either from UNIX style objects or from VMS style objects. Another section describes how objects from various source code environments can be combined together into UNIX or VMS style executables. Finally, this section presents some examples of simple cshell scripts which can be used as a template for creating different style objects and executables, and includes descriptions of errors and possible problem solving.

4.1 CREATING VMS AND UNIX OBJECT FILES

The C and Fortran compilers, *cc(1)* and *f77(1)* provided with EUNICE BSD can produce a choice of UNIX or VMS style objects, which allow C or Fortran programs created with EUNICE BSD to be linked with objects generated from either VMS or 4.3 BSD UNIX compilers. Separate assemblers and loaders exist for the two object styles. When invoked, the C and Fortran compiler reads two environmental variables (AS_IMAGE and LD_IMAGE) to determine which object style is desired, and starts the appropriate assembler and loader. The value of the variable AS_IMAGE determines which assembler will be started, while LD_IMAGE determines the loader to be used. The same source file can be used for either object type, since the C or Fortran syntax is the same as on a native Berkeley 4.3 BSD system.

Notice that the ability to determine which object style will be used is unique to *cc(1)* and *f77(1)* compilers. Other compilers like *pc(1)* (Pascal) or *lisp(1)* (Lisp) currently do not have this ability. If you need, for example, to link Pascal programs with the objects developed under VMS, see examples of Cshell scripts in Section 4.8.

4.1.1 VMS Objects and Executables

In order to link routines created under EUNICE BSD with VMS objects, a VMS style object file must be generated from the source files for the routines.

The following two environment variables must be set to create VMS objects:

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

The program *vmsas(1)* is a modified UNIX assembler which produces VMS object code while the program *vmsld(1)* converts a UNIX loader argument list into a valid argument list for the VMS linker. *cc(1)* and *f77(1)* then use these programs to create VMS style objects and to load VMS object code. Notice that requesting VMS objects results in the use of compilers provided with EUNICE BSD (not the VMS C compiler).

It is suggested that a *csh(1)* alias is used to set the environment variables to create VMS objects. Add the alias to the *.login* file (see the template file in */usr/skel* directory) and then use *vmsobj* to set the variables, as indicated below:

```
#Have compilers use VMS assembler and loader.
```

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

Note that *ld(1)* does not look at the environmental variables set by *unixobj* or *vmsobj* to determine whether UNIX or VMS objects are desired. Use *cc(1)* or *f77(1)* to do the load phase.

See Sections 4.8 and 4.9 for examples of simple Cshell scripts, which create VMS style objects and executables, and for an explanation of possible user's errors.

Should problems occur, use the debugging option *(-d)* with *cc(1)* or *f77(1)* compiler, which also displays the assembler and loader used with all accompanying files.

4.1.2 UNIX Objects and Executables

To link routines created under EUNICE BSD with UNIX objects, a UNIX style object file must be generated from the source files for the routines. The following two environmental variables must be unset:

```
unsetenv AS_IMAGE
unsetenv LD_IMAGE
```

When the environmental variables AS_IMAGE and LD_IMAGE are not set, *cc(1)* and *f77(1)* will use the standard */bin/as* assembler to create UNIX style objects and the standard loader */bin/ld* to load UNIX objects. The environmental variables AS_IMAGE and LD_IMAGE are not set up at EUNICE BSD system startup time; the UNIX assembler and loader are used by *cc(1)* or *f77(1)* as a default.

It is suggested that the following *csh(1)* alias be used to set the environment variables to create UNIX objects. Add the alias to the *.login* (see the template file in */usr/skel* for reference) and then use *unixobj* to unset the variables, as shown below:

```
#Have compilers use UNIX assembler and loader
alias unixobj 'unsetenv AS_IMAGE; unsetenv LD_IMAGE'
```

See Sections 4.8 and 4.9 for explanation of possible user's problems and simple Cshell scripts used during compilation.

4.2 UNIX AND VMS SYMBOL TABLES

The VMS and UNIX assemblers and linkers generate symbol tables which have different formats. Programs which need to use the symbol tables, such as the debugger, *adb* and *dbx* can be used only with UNIX symbol tables.

4.3 USING UPPER CASE WITH THE cc OR f77 COMPILER

Note that UNIX is case sensitive, while VMS is not case sensitive. This means that source code compiled in the UNIX environment which has upper and lower case variables will result in symbols with mixed cases. Whenever capital letters are to be used in source code (such as VMS system calls) be sure that the case conversion is turned off. This is true when UNIX or VMS objects are being created by the EUNICE BSD compilers or the VMS linker is used with VMS objects created by the EUNICE BSD compilers.

Normally, one of the EUNICE BSD system startup files (EUNICE.COM) sets this conversion (used by the EUNICE BSD hashing mechanism) "ON".

When the compilations are done in the UNIX environment using *csh*, do the following:

```
% setenv UNIX_ASSEMBLER_CASE_CONVERT OFF
```

Or if linking from the DCL environment, use the command:

```
$ DEFINE/JOB UNIX_ASSEMBLER_CASE_CONVERT OFF
```

which defines case conversion at the job level.

4.4 SHAREABLE C LIBRARY

One of the highly useful features of EUNICE BSD is the existence of the shareable C library in addition to the standard C library. If there are any changes in the contents of the shareable C library, the user does not need to relink all the programs which make calls to this library. Relinking of programs is always required when a change occurs in the standard C library.

4.5 UNIX LOADER OPTIONS

The option *-noshare* will cancel the (generally useful) default which loads the shareable "C" library and will cause all routines to be loaded from the standard "C" library. This produces an image with no dependencies on outside images and which can then be run on vanilla VMS systems. The *-noshare* option is also important if the user is linking code which manipulates the EUNICE BSD runtime system (e.g., code which contains *#include <eunice/eunice.h>*). It is possible that the module you are trying to link will generate multiple definition of other global symbols, causing the UNIX *ld* loader to report multiple definition errors for symbols such as *errno, vmserrno* and *environ.* It is not unusual for the VMS LINKER to generate multiple warning messages when object modules define global symbols more than once (e.g. two different files declare the variable *int i*; globally.) These error messages are only warnings; the VMS LINKER will still link things correctly. Ideally, programs should declare a variable as global only once and "extern" it everywhere else.

The options file **TWG$USR:[LIBVMS]SHARE.OPT** allows the VMS LINKER to obtain the shareable "C" library when linking VMS style objects, as follows:

```
$ LINK FILE1,FILE2,FILE3,TWG$USR:[LIBVMS]SHARE/OPT
```

The option *-nop0bufs* will set the NOP0BUFS flag in the VMS image header and will set the IMGIOCNT to 250, which will keep RMS from intruding on *P0* space in those programs which are sensitive to the state of *P0* space. Programs which do their own memory allocation and want to see contiguous *sbrks* are normally sensitive to the state of *P0*; very few UNIX programs (e.g., *adb* and *dd*) have this requirement. Including */lib/prealloc.o* for UNIX object files or */usr/libvms/prealloc.obj* for VMS object files in an *ld* command will keep EUNICE BSD from intruding on *P0* space by preallocating all of the internal EUNICE BSD data structures.

The option *-notraceback* will provide an image that has no traceback capability so that the image can be installed as a privileged program in VMS.

The flag *-vSHAREABLE_IMAGENAME* is responsible for including the shareable image SHAREABLE_IMAGENAME in the load. To get VMS shared libraries into programs generated by using UNIX object code, create a VMS shareable image with the VMS object modules and then link the UNIX object modules with this shareable image.

Global symbols defined by UNIX object modules will override global symbols in shareable images, allowing users to supply procedures with the same names as those in shareable images (in particular the shareable "C" library) without creating name conflicts.

4.6 FORTRAN 77 NOTES

4.6.1 f77 Compiler

The *f77(1)* Compiler automatically searches libraries in the following order: */lib/crt0.o*, */usr/lib/libF77.a*, */usr/lib/libI77.a* and */lib/libc.a*. If you need to specify these libraries in the load sequences, load them in this order. If using */usr/eun/vmsld*, load the corresponding VMS libraries, (e.g., */usr/libvms/libF77.olb*). VMS object libraries are found in */usr/libvms*, whereas the UNIX objects libraries are found in */usr/lib*.

4.6.2 f77 Vs VMS FORTRAN Compiler

EUNICE BSD supplies the *f77* (standard UNIX FORTRAN) compiler. Although there is no documentation in any of the accompanying UNIX manuals on standard ANSI FORTRAN, the article titled "A Portable FORTRAN 77 Compiler" (in the **UNIX Programmer's Supplementary Documents** [PS1]) lists the differences between standard ANSI (f66) and *f77*.

Although the VMS FORTRAN compiler offers more features than *f77*, such as parameter statements, variable name length (31 characters for VMS FORTRAN versus 6 characters for *f77*), and column size (80 characters versus 72), programs written using *f77* are much more portable than those written with the VMS FORTRAN compiler.

4.7 COMBINING OBJECTS FROM VARIOUS SOURCE LANGUAGES

4.7.1 Loading FORTRAN and C

When combining FORTRAN and C source code, use the *f77* compiler as the loader. The *f77* compiler is aware of the C compiler, but the C compiler is unaware of FORTRAN.

4.7.2 Loading Pascal and C

Note that the Pascal program does not initialize its variables to 0 at the beginning of runtime, which can occasionally cause problems if C subroutines are called from the Pascal programs and the Pascal variables are not set to 0. The problem can be solved by using a C *main()*; routine which calls the Pascal program. By using the C *main()*; routine, all variables will be set to 0 at runtime.

4.7.3 Loading VMS Module Using the UNIX C Compiler

If you create a VMS FORTRAN or Pascal module and want to load it under the UNIX C Compiler, create a C *main()*; routine which calls the Pascal or FORTRAN subroutine, because */usr/libvms/crt0.obj* expects to branch to C *main()*;. If a C *main()*; routine is not created, an error message will indicate a bad magic number.

4.8 EXAMPLES OF USEFUL SCRIPTS

Case 1. A Pascal program (or any other program except C or FORTRAN) needs to call VMS object.

Solution: Currently the *pc(1)* compiler does not have the ability to distinguish between UNIX and VMS style objects and variables. The following script presents a solution to this problem:

```
# Call Pascal compiler, but do not do any assembly
pc -S pascal_subroutine.p
# Set variables, so that VMS style assembler and loader
#     will be used by cc(1)
vmsobj
# Let C compiler to do the rest of the work
cc c_main_routine.c pascal_subroutine.s vms_called_by_pc.obj
```

Case 2. A UNIX C program uses a VMS system call.

Solution: Use the following script:

```
# Set case conversion off
setenv UNIX_ASSEMBLER_CASE_CONVERSION OFF
# Set variables, so that assembler and loader will produce
#     VMS style objects and executables
vmsobj
# Call C compiler
cc c_main.c c_subroutine1.c c_other.c
```

4.9 UNIX COMPILATION ERRORS

The following are examples of the errors which the user may see during compilation. If you are not able to find an answer to your problem among them, see the *UNIX User's Reference Manual [URM]*, Section 1 and *UNIX Programmmer's Supplementary Documents [PS1]* and *[PS2]* for references to the compiler used.

Symptom 1. load errors: undefine main___0003x :

situation: DEC object modules or programs calling UNIX C programs were loaded using *ld(1)* as the loader with *vmsobj* set. *ld(1)* does not recognize the setting of the environmental variable LD_IMAGE, which was done with the alias *vmsobj* (See Section 4.2). The load must be done by *cc(1)*, not *ld(1)*.

A *.obj* file may have been created with DEC compilers. Now these objects are to be loaded under EUNICE BSD, with some additional code created with *vmsobj*.

solution: *cc(1)* will automatically load *crt0* in as the first file in the load sequence. */usr/libvms/crt0.obj* expects to branch to C main. A C main program should be created calling the main DEC object program.

```
main( )
{
    /*** this is the pseudo C main program
    calling the DEC FORTRAN main program ***/

    dec_main_object( );
}
```

example

```
% vmsobj
% cc main.c dec_main_object.obj
```

Symptom 2. bad magic number:

situation: 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 BSD loader to create VMS objects with *cc(1)* or *f77(1)*, refer to Section 4.2 and the alias given in the login files in */usr/skel*.

Example

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

Symptom 3. premature eof:

situation: 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 doesn't 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.

Symptom 4. ld: premature eof

situation: *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 5. link w/illrectype illegal record type (32) in module # file *.for*

Every record will show as an illegal record type.

situation: *f77(1)* is being used with *vmsobj* set. File names have '*.for*' extension.

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

Symptom 6. undefined symbols, __get_fderr :

situation: UNIX C is calling VMS system calls.

solution: Make sure that UNIX_ASSEMBLER_CASE_CONVERT is undefined.

```
% setenv UNIX_ASSEMBLER_CASE_CONVERT OFF
```

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

Symptom 7. error: 0 byte record

situation: An attempt was made to run interactive VMS executables from inside the *csh*.

solution: Use *vms* to run interactive programs from the shell. Alternately, use a command file which includes the following:

```
$ DEFINE SYS$INPUT 'F$LOGICAL("SYS$OUTPUT")
$ 'P1 'P2
$ EXIT
```

For instance, the user may want to set up an alias and then refer to the alias to execute the program. Notice that special characters in the VMS file specification will have to be escaped. For example, to run 'myprog':

```
% alias myprog 'vms DRA1: \[dave\]myprog'
% myprog
```

Symptom 8. ld: readit: cannot open

situation: An attempt was made to run *f77(1)* or *cc(1)* on a file which 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 9. symbol table overflow

situation: 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.

Symptom 10. multiply defined symbols: *vmserrno, errno, environ*

situation: An attempt has been made to load UNIX object modules which manipulate the EUNICE BSD 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 will cancel the default which loads the shareable *C* library and cause all routines to be loaded out of the standard *C* library.

4.10 VMS LINK/RUN ERRORS

This section describes problems which can occur during link or runtime. (See the *UNIX User's Reference Manual [URM]*, Section 1 *vmsld(1)* or the *VAX/VMS Linker Reference Manual* for further information.)

Symptom 11. no transfer address

situation: 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 12. multiple transfer address

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

solution: See above solution for no transfer address.

Symptom 13. 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
```

situation: 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 which 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* Section in the *UNIX User's Reference Manual [URM]*, Section 1 concerning 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
```

Symptom 14. undefined symbol *abc_00f*

situation: 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 lower case. *C* distinguishes between lower and upper case; 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

To test, do the following:

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

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

Symptom 15. *link_w_userundef* undefined symbol *exit_* reference in

psect *$$c_text0*

Each DEC subroutine referenced in the *f77* program will show up as an undefined symbol with an underscore appended.

situation: Objects from *f77* were linked with *vmsobj* set, and linked VMS FORTRAN objects under VMS link. The *f77* program calls the VMS FORTRAN program.

solution: *f77* appends an underscore to the end of the subroutine name; UNIX C and VMS FORTRAN do not. Further, VMS FORTRAN does not allow subroutines to have an underscore, whereas UNIX C does. Therefore, a C interface must be written between the *f77* and VMS FORTRAN subroutine. Have *f77* call a pseudo-C subroutine which in turn calls the VMS FORTRAN subroutine.

example: *f77* program

```
program main
call vmsfortran( );   ! normally a call to DEC FORTRAN subroutine
```

C interface subroutine

```
vmsfortran_( )
{
return vmsfortran( );
}
```

VMS FORTRAN subroutine

```
subroutine vmsfortran( )
```

Symptom 16. linker -w illrectype in filename, record 1 is illegal
record 2 is illegal
no end of module.

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

solution: Set *vmsobj* and re-compile the UNIX sources.

Symptom 17. linker -w illrectype in filename, record 1 is illegal
record 2 is illegal
no end of module.

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

situation: 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 *UNIX User's Reference Manual [URM]*, Section 1.) The VMS LINKER expects both a VMS file type record (variable length) and a file compiled under *vmsobj*.

Symptom 18. undefined _error, weak reference to main

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

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

Symptom 19. 512 byte record too large for buffer.

situation: 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 *UNIX User's Reference Manual [URM]*, Section 1 for *unixtovms(1)*.)

Symptom 20. bad image header

situation: An attempt was made to run an incorrect executable type in VMS. If the created record is variable length with carriage return, it will complain of bad image header.

solution: Create the executable with a 512 fixed length record (use VMS FORTRAN compiler, linker).

Symptom 21. 0 byte record too large for user's buffers

situation: An attempt to run @TEST.EXE (where TEST.EXE is an executable) was made from C or FORTRAN. (It does not matter whether the program was UNIX or VMS.)

solution: The @ symbol is used to run a DCL .COM (command) file. Use RUN from VMS DCL to execute the compiled executables.

5. GUIDE TO OPERATIONS PROBLEM SOLVING

The following list contains solutions to problems that could possibly occur while operating in the EUNICE BSD environment. For further information, check Section 2, "EUNICE BSD Command Availability", of this manual, or the EUNICE NOTES in the *UNIX User's Reference Manual [URM]*, Section 1. Further information is contained in Appendix B, "Guide to Installation Problem Solving", in the *EUNICE BSD Administrator's Guide.*

ar(1)

symptom: phase error

solution: *ar(1)* archives UNIX objects, not VMS objects. Set the LD_IMAGE and AS_IMAGE environmental variables to produce UNIX objects by using the *unixobj* alias. The error message is just a warning; the *ar* command worked correctly.

ar(1)

symptom: table of contents out of date

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

If the system administrator suspects that several of the archives may have been touched, then he can avail himself of the EUNICE BSD utility *ranlib.csh*, which resides in */etc/eunice*. This shell-script will re-randomize all the *lib*.a* files in one run. (The script DOES, however, take a while to run.)

at(1)

symptom: output is lost

solution: The output of *at(1)* must be redirected or is lost. Also, the directory */usr/spool/at* must be writable to all. **Note:** There is no *atrun*. VMS SYS$BATCH takes over (or whichever VMS batch queue has been specified in RC.COM).

at(1)

symptom: does not work

solution: The output of at(1) must be redirected or is lost. Also, the directory */usr/spool/at* must be writable to all. **Note:** There is no atrun. VMS Batch Processing takes over. The logical for the batch queue used is TWG$ATQUEUE. This is normally assigned to SYS$BATCH.

cd(1)

symptom: cd .. cannot find directory

solution: Refer to the solution for *adduser.com* (*cvtuaf*) in Appendix B of the *EUNICE BSD Administrator's Guide.*

chmod(1)

symptom: *chmod* adds VMS delete protections to file

solution: EUNICE BSD provides UNIX file protections without compromising VMS security. Files created under EUNICE BSD adhere to the UNIX file protection conventions. In order to implement UNIX file protections using VMS protections, the VMS delete permission must be turned on; however, this does not violate file security.

To delete a file, VMS, like UNIX, requires the directory to have write permission. In addition, VMS requires the file to have delete permission. Therefore, even though *chmod(1)* adds the VMS delete permission to the file's group and world users, the file cannot be deleted from either EUNICE BSD or VMS unless both the write and delete permissions on the file and directory are turned on. EUNICE BSD is careful not to turn on write permission unless explicitly told to do so. File and directory permissions are determined by the *umask(1)* and the mode specified in the *chmod* command itself. This is consistent with native 4.3 BSD.

csh(1)

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

solution: Home directory listed in */etc/passwd* is incorrect.

csh(1)

symptom: unless an unprivileged account gives the full pathname, the command is not found.

solution: Check the RMS values. Lower the multi-block count to at least 16, or perhaps 8, and make the value of index, relative and disk multi buffer count at least 2.

cu(1)

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

solution: VMS 4.x now implements ^j for line editing, but System V *cu(1)* expects ^z. It is necessary to disable line editing for the terminal from the DCL with:

$ 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:

DECnet

symptom: exceed quota, return to *node vax* (host)

solution: The SYSGEN parameter, MAXBUF, needs to be increased on the host as well as the target machine. You will need to experiment with the lowest acceptable value; normally, doubling the original value will do the trick.

delta(1)

symptom: truncates files to zero

solution: Install *delta* with SYSPRV. The default protection should be S:RWED. See TWG$ADMIN:SUCHMOD.COM.

delta(1)

symptom: cannot make *delta*

solution: Requires emulation of native UNIX environment, so turn on EUNICE_1VERSION:

"% $ DEFINE/SYSTEM "EUNICE_1VERSION" "ON"

This is usually set system-wide.

EUNICE.COM

symptom: global page table full

solution: Increase the SYSGEN parameter GBLPAGES and reboot. Also edit MODPARAMS.DAT giving the new value. EUNICE BSD requires a minimum of 661 global pages.

eunlogin(1)

symptom: lastlogin message says "*tty??*".

solution: Refer to solution for *who(1)*.

@TWG$ADMIN:CSHELL

symptom: does not produce the last login message

solution: Permissions on files */usr/adm/wtmp* and */usr/adm/lastlog* must be W:R.

@TWG$ADMIN:CSHELL

symptom: null no match when accessing the *csh(1)*.

solution: Permissions on */etc/passwd* must be W:R. Or, the user may not have used the command file provided. */etc/cshell.com* should call TWG$ADMIN:EUNLOGIN and TWG$ADMIN:CSH.COM. It is a good idea to put a logical assignment for this command file in the system wide login file.

@TWG$ADMIN:CSHELL

symptom: no entry for you in password file

solution: ADDUSER.COM must be run every time new users are added to the SYSUAF.DAT. The protections on */etc/passwd* must be W:R.

/etc/utmp

symptom: not updated for some terminal lines.
solution: Refer to solution for *who(1)*.

f77(1)

symptom: allows use of -nN flag

solution: Most of the UNIX utilities provided with EUNICE BSD are from 4.3 BSD. However, *f77* is from a newer version, 4.3c. The documentation for *f77* is for the older version, so the -nN flag is not in the documents.

find(1)

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

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

find(1)

symptom: find: bad status < filename >

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

foreign(1)

symptom: arguments are passed as lower case.

solution: Put a caret (^) before any character to be passed as upper case.

get(1)

symptom: "get -e sccs/s.file" fails but "get sccs/s.file" works.

solution: Make the sccs directory writeable to allow creation of *p.files*.

kill(1)

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

solution: **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 re-booted to re-enable use of the port.

lavdriver

symptom: *w(1)*, *uptime(1)*, and ^T do not display the load average.

solution: Check to see if the following lines are in EUNICE.COM:

```
$ RUN SYS$SYSTEM:SYSGEN
CONN LAV0 /NOADAPTER/DRIVER=TWG$ADMIN:LAVDRIVER
$ ASSIGN/SYSTEM LAV0: $$VMS_LOAD_AVERAGE
```

ld(1)

symptom: *ld:/ms/src_lib/mx:* cannot open

solution: Verify that the named libraries are readable to the individuals (usually the world) using them.

lint(1)

symptom: does not work

solution: The */usr/tmp* directory is missing or is not accessible.

lisp(1)

symptom: cannot open more than three files

solution: There may not be enough unused global pages available. Use SYSGEN to increase the GBLPAGES, and add the number to SYS$SYSTEM:MODPARAMS.DAT and re-boot the system.

lisp(1)

symptom: an image created by dumplisp may get
an image activation failure 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
```

The above can be used from a command file which accepts an image as a parameter.

lisp(1)

symptom: premature eof

solution: See Section 4 of this manual.

liszt:

symptom: premature eof

solution: See Section 4 of this manual.

lpr(1)

symptom: does not work, cannot create PRINTER:LPR

solution: The printer may be the operator's console, or may be used as a terminal. If a user is logged into such a device, lpr(1) will not be able to access the device. Wait until the "printer" is free and try again.

lpr(1)

symptom: Can't print using lpr(1).

solution: Queues under VMS 4.X have access permissions associated with them. In the system start-up file, queue protections should be set to W:RWE before the print queue is started.

lpr(1)

symptom: cannot create LPA0:LPR

solution: The templates provided with EUNICE BSD reference LPA0: as the printer. If a terminal is used as the printer, refer to */etc/rc.com* for setting up the terminal as the print device. */etc/dev.com* should have the following logical assignment:

$ ASSIGN/SYSTEM/EXEC TXC7: PRINTER

NOTE: TXC7 stands for the device that is spooled as the printer. (Also, see below.)

lpr(1)

symptom: does not work, cannot create PRINTER:LPR

solution: Refer to */etc/rc.com* for an example of how the printer should be set up. The VMS logical name for PRINTER should translate to a hard device name with leading underscores. The translation should also be placed in */etc/dev.com*. The device must be spooled as follows:

$ SET DEV/SPOOLED TXA7:

If the device is a terminal, use the flag, "/TERM ".

$ INIT/QUE/TERM TXA7:
$ ASSIGN/SYSTEM TXA7: PRINTER

In VMS releases later than VMS 3.3, devices have access permissions associated with them. If

% ls -l /dev/printer

returns "*/dev/printer* not found", the appropriate access permissions must be set before starting the queue. With VMS OPER and LOGIO privileges, type:

$ SET PROT=(G:W,W:W)/DEV LPA0:

solution: When *lpr(1)* submits the job to the VMS printer queue it is still owned by the submitter. Users need sufficient disk quota on the system disk to enable them to write to the print queue. Alternately, use an alias to the VMS PRINT command.

mail(1)

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

solution: */etc/newaliase* has to be run after adding a user to EUNICE BSD. */etc/adduser.com* runs *newaliase* automatically in addition to updating the *passwd* and group files.

mail(1)

symptom: instead of appending mail to the mbox, the mbox is overwritten.

solution: *mail(1)* requires that EUNICE_1VERSION be turned ON. Purge the directory. (See TWG$ADMIN:EUNICE.COM.)

make(1)

symptom: does not work

solution: Requires that EUNICE_1VERSION (which can be turned on at the process level) be turned ON. The directory must also be PURGED.

man(1)

symptom: cannot unlink */tmp/catxx*

solution: The online version of the UNIX manual pages is unformatted in order to save disk space. When a page is requested, it is formatted and a copy is sent to the screen which has been filtered through *more(1)*. Another copy is sent to the reserve directories, */usr/man/cat[1-8]*, which directories must be writable to all. The 'cat' files will be owned by the first person requesting the manual page. To avoid disk quota problems, the system administrator should periodically change the ownership of these files to SYSTEM.

man(1)

symptom: cannot read */usr/lib/tmac/tmac.new* or *tab37*. The manual page is not piped to *more(1)*.

solution: Decrease the system RMS block value. Test for values ranging between 2 and 8.

```
$ SET RMS/SYSTEM/BLOCK=8
```

man(1)

symptom: 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: System should periodically run */etc/chown* on */usr/man/cat*, or use */etc/catman* during off peak hours to create all the *man* files.

```
% at 6pm mmdd /etc/catman 1234578
```

This will create all the sections (except Section 6 - games) starting at 6pm on the specified month and day.

metacharacters

symptom: neither shell can do metacharacter conversion

solution: The authorization file has set the user's device to a logical which is assigned to a rooted file specification. To the *.login* file add:

cd $cwd or cd

mkdir(1)

symptom: The EUNICE BSD command *mkdir testdir* returns:
cannot create directory testdir/
while, the DCL command shown below does work:

$ CREATE/DIRECTORY TESTDIR

solution: After your system has been running for some time, the VMS mail box (MBAs) numbers (used to implement *mkdir*) are reset. *mkdir* will work, once the system is rebooted. Alternately, use the VMS command (or an alias) to create directories until the system can be rebooted.

mv(1)

symptom: does not move multiple versions correctly

solution: To emulate a native UNIX environment, use PURGE to delete multiple file versions and turn on EUNICE_1VERSION.

mwait

symptom: processes go into *mwait*.

solution: One must use *set notify* with the *csh*. To ensure that *set notify* is used by any sub-process *csh's*, *set notify* should be put in the "*.cshrc*" file.

nroff(1)

symptom: file not found

solution: The VMS system RMS block value is too high. Try using values from 8 down to 2, as follows:

$ SET RMS/SYSTEM/BLOCK=8

nroff(1)

symptom: file not found

solution: There may not be enough unused global pages available. Use SYSGEN to increase GBLPAGES, and add the value to SYS$SYSTEM:MODPARAMS.DAT. The system must be re-booted for the new value to take effect.

printer

symptom: trying to have a program write to printer using */dev/lp*.

solution: */dev/lp* cannot be written to by a user program. As a spooled VMS device, it is allocated to another user, and can't be written to directly. Have the program write to a file and then spool it, or set up a non-spooled device specifically for this program.

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

symptom: cannot make pipe

solution: BYTLM, which affects the amount of data which can be re-directed or piped, is too low. Set it to 60000 or higher.

stty(1)

symptom: kill and erase not changed

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

stty(1)

symptom: *cbreak* does not work

solution: Try using raw mode.

stty(1)

symptom: returns "is not a *tty*".

solution: Refer to the solution for *who(1)*.

tar(1)

symptom: *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 BSD feature.) You can change the owner (to the correct owner) from DCL as follows:

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

tar(1)

symptom: cannot open */dev/rmt1*

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

$ MOUNT/FOR/BLOCK=10240 MTA0:

Tar reads from */dev/rmt1* by default. To use another drive such as /dev/mt0, use:

% tar xv0

tar(1)

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

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

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

tar(1)

symptom: does not handle multiple versions correctly

solution: To emulate a UNIX environment, PURGE old versions of the file and turn on EUNICE_1VERSION. (See TWG$ADMIN:EUNICE.COM.)

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 *tty??*

solution: Refer to solution for *who(1).*

uucp(1)

symptom: *uucp* will not login to a VAX from a UNIX Version 7 System.

solution: Most native UNIX systems use line feed, not carriage return to indicate end-of-record. VMS requires both carriage return and line feed. Thus, when a native UNIX Version 7 system tries to log in to a VAX machine, the VAX expects a carriage return, not a line feed, which will prevent *uucp* from logging on to the VAX.

Modify the native UNIX to send a carriage return when talking to the Vax machine via *uucp*. This modification must be done in the routine called *conn.c.*

vi(1)

symptom: /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 this with *df(1)* and delete any unnecessary files.

vi(1)

symptom: file not found

solution: From the SYSTEM account, decrease the RMS multi-block count. You will need to experiment with different values (usually between 8 and 2).

$ SET RMS/SYSTEM/BLOCK=8

vmsmail

symptom: mail to VMS mailboxes

solution: Aliases in */usr/lib/aliases* should be of the form:

user:" | /usr/eun/vmsmail user"

Run */etc/adduser.com* which will re-run *newaliase*. This should be run each time the */usr/lib/aliases* file is edited.

vpr(1)
vtroff(1)

symptom: not working under VMS 4.x.

solution: Can be used as a printer, but not with *troff(1).*

who(1)

symptom: does not work

solution: Verify the following:

1. Permissions on */etc/ttys and /etc/locations* must be W:R.

2. */etc/utmp, /usr/adm/wtmp* and */usr/adm/lastlog* must be W:RW. The intervening path should be W:RE.

3. Entries in */etc/eunice/dev.com and /etc/ttys*, which indicate the number of terminals available on the system must all reflect the same number.

4. Always make the logical assignments for devices by running */etc/eunice/dev.com.*

write(1)

symptom: permission denied.

solution: VMS OPERATOR privilege is needed to use *write(1).*

6. GLOSSARY

ASTLM

VMS user quota for asynchronous system trap creation.

BALSETCNT

VMS sysgen parameter. Maximum number of processes resident in memory at one time.

Berkeley 4.3 BSD

Version 4.3 of the Berkeley Software Distribution of UNIX software (from the University of California at Berkeley).

BYTLM

VMS user quota for buffered I/O byte count limit quota.

CLISETUP.EXE

One of the EUNICE BSD run-time executive files. Maps a special command language interpreter into the sub-process control region.

csh

Default *shell* in EUNICE BSD environment

.cshrc

File which is read by *csh* to perform various functions and set up the environment. The template for this file is in */usr/skel.*

cvtbackup

Allows tools produced by backup to be shipped over DECnet or ARPANET. The executable are converted to ASCII files and then should be unconverted after transfer, with *cvtbackup. cvtbackup* is located in */usr/eun.*

cvtfnames

A file created for EUNICE BSD 4.2 to convert files hashed under previous versions of EUNICE BSD to the new EUNICE BSD (VMS 4.x) hashing mechanism. Still present on EUNICE BSD 4.3.2. *cvtfnames* is located in */usr/eun.*

cvtuaf

Program to generate a UNIX *passwd* file from the VMS authorization file.

DCL

Digital Command Language. Command language interpreter on VMS.

DST_FLAG

Flag for daylight savings time.

EUNICE_FULL_FILENAMES

Logical name no longer used by EUNICE BSD.

EUNICE_1VERSION ON

Logical name to make file creation behave the same way as in the UNIX environment.

eunlogin

Programs to control UNIX access. Located in */etc/eunice*. See Section 4.2 in the *EUNICE BSD Administrator's Guide.*

.exrc

File used by the *ex* and *vi* editors to set up a minimum required environment for the editor. The template for this file is in */usr/skel.*

Fillm

VMS user quota for file and logical link limit.

forkdumy.exe

A EUNICE BSD run-time executive file. A dummy image used in *fork.*

forkdumy1.exe

A EUNICE BSD run-time executive file. A dummy image used in *fork.*

group file

UNIX group file located in */etc.* Created by *mketcgrp.*

KFILSTCNT

VMS parameter - known files count. Restricts the number of programs that can be installed as known images.

lastlog

Located in */usr/adm.* This file contains information about the last time each user logged in. It should be cleaned out periodically. When rebooting, *lastlog* is cleaned out by TWG$ADMIN:RC.COM

/lib

Location of UNIX libraries in VMS object format.

libc.a

The C library in UNIX format. *libc.a* resides in */usr/lib*.

libc.olb

The C library in VMS format. *libc.olb* resides in */usr/libvms*.

locations

File which describes the terminal locations. *locations* resides in */etc*.

.login

The *.login* file is executed by EUNICE BSD from shell file */etc/eunice/cshell.com*. Template for this file is in */usr/skel*.

login.com

File which is executed upon login into the VMS system. The template for this file is in */usr/skel*.

LPA0:

VMS device name for standard printer.

MAXPRO

VMS parameter MAXPROCESSCNT. The maximum process count.

motd

UNIX message of the day. *motd* resides in */etc*.

MTA0:

VMS device name for tape drive.

/dev/mt0

UNIX name for tape drive (logically assigned to MTA0:) in blocked mode. See RMT0:

NLA0:

Null device (VMS name).

OPA0:

Console terminal (VMS name).

passwd

UNIX *passwd* file located in */etc*. It is created by running *cvtuaf(1)*.

Prclm

VMS symbol for subprocess creation limit quota.

PRIO

VMS symbol for base priority quota.

prompt

The two UNIX prompts are '*%*' for the *csh* and '$' for the Bourne shell. The DCL prompt is also '$'. All three prompts can be changed if required.

RMS

Record Management System. The file system used by RMS.

/dev/rmt0

UNIX name for tape drive (logically assigned to MTA0:) in character mode. See */dev/mt0*.

root

Top level directory in native UNIX, denoted by /. In EUNICE BSD, the *root* directory is a VMS search list.

rooted file spec

One in which the last directory in the specification list is followed by a dot signalling that other directory specifications may be attached.

RTA1:

VMS device name for DECnet remote terminal.

sh

The UNIX shell (since Version 7) is also called the Bourne shell, after its author. The *csh* is from Berkeley UNIX. Both are available in EUNICE BSD.

snap.csh

Takes a snapshot of the EUNICE BSD and VMS environment. This is used in debugging site specific problems. It gets a listing of SYSGEN parameter values, runs AUTHORIZE on the SYSTEM and DEFAULT accounts. It also lists the contents of the files modified during the installation. (See Section 9.1 in the *EUNICE BSD Administrator's Guide*.) Located in */etc/eunice*.

SYSUAF.DAT

VMS system user authorization file.

termcap

Contains terminal entries, one per terminal type, based on specific capabilities. Located in */etc*.

tests.csh

This is a test shell script for the *csh*. *tests.csh* is located in */etc/eunice*.

tests.sh

This is a test shell script for the Bourne shell which is located in */etc/eunice*.

treewalk.com

Sets up proper protections on the EUNICE BSD directories and files. Many problems with EUNICE BSD are caused by the permissions on files or directories being altered from the permissions which were set at the time of installation. Located in */etc/eunice*.

/tmp

UNIX directory where temporary files are created by such utilities as editors, text processors, and the compilers.

TQELM

VMS user quota for timer queue entry limit.

trpatch

A utility created for EUNICE BSD 4.3.1 and subsequent releases of EUNICE BSD to change the header information on executable files for VMS 4.0, so they can be properly installed. Located in */usr/eunice*.

TWG_LIBC_43.EXE

Shareable C language runtime executable, containing some of the objects from the C library. It resides in SYS$SHARE.

unixtovms

Converts a UNIX file type to a VMS variable length, carriage return file. A manual page is provided in the *UNIX User's Reference Manual [URM]*, Section 1.

/usr/lib

Location of UNIX libraries in UNIX object format.

utmp

File to keep track of who is on the system. Located in */etc*.

uucp

Command used to transfer data between UNIX systems.

UUCP_HOST_NAME

UNIX system name, used by *uucp(1)*, *mail(1)*, and *who(1)*.

vforkcli.exe

One of the EUNICE BSD run-time executive files. Controls execution of the subprocess in which it resides.

VMS

Virtual Memory System. Standard operating system for VAX computers.

vmstounix

Converts a variable length file to a 512-byte file. A manual page is provided in the *UNIX User's Reference Manual [URM]*, Section 1.

vos

Directory containing the EUNICE BSD run-time files which create a Virtual Operating System.

wtmp

This file contains UNIX login/logout information. Located in */usr/adm*.

7. LICENSING OBLIGATIONS

The EUNICE BSD environment provides a transportation link which enables the use of software written for the UNIX operating system to run in a VMS environment. This ability is referred to as REX™ (Runtime Executive) by The Wollongong Group. REX provides a cost-effective solution for migrating your application to other VMS systems.

Porting UNIX-based software to run under VMS is a very powerful ability, and is the main reason why EUNICE BSD was purchased at some sites. However, it is important that the programmer using EUNICE BSD be aware of licensing obligations if these programs are to be moved to another VAX.

Both the UNIX and EUNICE BSD code are licensed on a per-CPU basis. This means that if any portion of the distributed software on the EUNICE BSD tape is moved to a machine other than the licensed CPU, a license agreement is necessary for the target machine. The licensed CPU serial number appears on your UNIX and/or EUNICE BSD license agreement(s).

The libraries which contain the Section 2 system calls, (see the *UNIX Programmer's Reference Manual [PRM]*, Section 2) are at the heart of what provides this transportation link. These libraries are:

SYS$SHARE:TWG_LIBC_43.EXE
TWG$USR:[LIBVMS]LIBC.OLB
/lib/libc.a

When programs which were originally written to run in the UNIX operating environment are compiled with these libraries, they can then be run under the VMS operating system. If a program is compiled by *cc(1)* or *f77(1)* with the alias *vmsobj* set, the code for the system calls included in the executable program will come from the VMS object format libraries. Because these libraries contain EUNICE BSD code, a REX license is required to move any portion of the code in *libc* or the shareable libraries to another machine.

In addition, if any of the following EUNICE BSD files are to be placed on a machine which does not have a EUNICE BSD license, a REX license is required.

CLISETUP.EXE	INITGBL
VFORKCLI.EXE	USERS.COM
FORKDUMY.EXE	ROOT.COM
FORKDUMY1.EXE	DEV.COM
SHELL	EUNICE.COM
	SUCHMOD.COM
	RC.COM
	STARTEUNICE.COM

Use of the UNIX executables is not included in the REX license. As a special case, a company can also obtain the right to use a selected list of the UNIX executables. For example, if commands such as *date*(1) or *csh*(1) are used in a program to be run on another VAX, a special agreement can be constructed which includes REX and the selected UNIX executables.

If the compilers provided with EUNICE BSD are only being used as a development tool, and no EUNICE BSD or UNIX code is being included in the executable, there is no need for a REX license for the target VAX. In this case, the compiled version for the target machine must be compiled with a non-UNIX compiler, such as the DEC C compiler. Note that if you choose to use the DEC C compiler, The Wollongong Group will not support the resultant code. It is important to recognize that no EUNICE BSD or UNIX proprietary code can legally be placed on a machine which does not have a valid license for that code.

Each REX distributorship is arranged individually with The Wollongong Group. If you have any questions regarding REX, please contact your sales representative.