System 8000™
ZEUS Reference Manual

03-3255-01

May, 1983

ZEUS REFERENCE MANUAL

Zilog Part Number Ø3-3255-Ø1

Software Release 3.1

AN INTRODUCTION TO THE ZEUS SOFTWARE DOCUMENTATION

Preface

This manual and the related manuals below provide the complete technical documentation for the System 8000 and the ZEUS operating system.

Title	Zilog Part Number
Zeus Software Documents:	
Zeus Languages/Programming Tools Manual	03-3249
Zeus Utilities Manual	03-3250
Zeus Administrator Documents:	
Zeus Administrator Manual (Model 11)	03-3254
Zeus Administrator Manual (Model 21/31)	03-3246
System 8000 Hardware Documents:	
System 8000 Hardware Reference Manual (Model 11)	03-3227
System 8000 Hardware Reference Manual (Model 21/31)	03-3237

Table of Contents

List of Illustrations

SECTION 1
INTRODUCTION TO ZEUS DOCUMENTATION

1.1. ZEUS Reference Manual

The following topics are described in the 9 sections of the ZEUS Reference Manual:

Section Ø	This introduction
Section 1	User commands
Section 2	System calls
Section 3	Library subroutines
Section 4	Special files
Section 5	File formats and conventions
Section 6	Games
Section 7	Program support files
Section M	System administration commands

1.1.1. Section Ø -- Introduction: This introduction is Section Ø of the Reference Manual. It outlines the ZEUS software document set, introduces the rest of the ZEUS Reference Manual, and explains the descriptions in each of the subsequent sections. It also provides an overview of the ZEUS operating system as a whole.

1.1.2. Section 1 -- User Commands: Section 1 of the ZEUS Reference Manual describes each of the more than 2ØØ ZEUS user commands that are entered at the keyboard and result in some action.

Most of these commands are located in the directory /bin or in /usr/bin.

1.1.3. Section 2 -- System Calls: System calls are issued from programs. They are like commands entered from the keyboard, but are written into a program and executed from within a running program. Section 2 explains each system call, its interface subroutine (shown for both the C language and Zilog's PLZ/ASM), the call's action, and errors returned.

Many of the system calls have corresponding command programs (Section 1).

1.1.4. Section 3 -- Subroutines: Section 3 describes the system library subroutines. These subroutines can only be linked with programs written in C. Other high-level languages provided with ZEUS have their own library facilities which are documented with the language.

1.1.5. Section 4 -- Special Files: Section 4 describes special files -- the I/O drivers. I/O functions of ZEUS are accomplished by reading or writing to a file that corresponds to an actual I/O device, like a port to a terminal, or a channel to a disk drive.

1.1.6. Section 5 -- Files and Conventions: Section 5 documents the structure of specific types of files, such as the output file format of the loader and assembler. Files that are used by only one command are excluded; for example, the intermediate assembler files are not described.

1.1.7. Section 6 -- Games: Section 6 documents the system's games, and other programs of a similar nature.

1.1.8. Section 7 -- Program Support Files: Section 7 documents specific files referred to by programs. These files include configuration tables, macro definition packages, and other online data. Section 7 also includes a summary of the entire ZEUS file hierarchy.

1.1.9. Section M -- System Administrator Commands: Section M contains information about commands used by the system administrator (login name "zeus"). These commands are privileged, and can only be executed by "zeus". They are found in the directory /etc.

1.1.1Ø. Command Syntax Descriptions: Each section in the ZEUS Reference Manual consists of independent alphabetical entries of a page or so each that describe a command function. The name of the entry being described and its section number are in the upper corners of the page.

COMMAND(1) Zilog COMMAND(1)

NAME
command.name - the name of the command typed into the terminal

SYNOPSIS
command.name [options] [arguments]

DESCRIPTION
This section of the page describes the workings of the command.

EXAMPLES
This section contains an example of the command exactly as it might be typed into the computer.

FILES
/path/file name of the file that contains the program

/path/file name of a related file

LIMITATIONS
As the name suggests, this section lists the limitations on the program (e.g., file size).

SEE ALSO
command(1), related.command(2), others(M).

Other documents related to the command are listed here.

1 Source 1

Figure 1-1 Sample Page from the ZEUS Reference Manual, Reduced

The pages for each entry are numbered independently within the section to simplify future revisions of this document. That is, each command starts on page 1.

1.1.11. Command Entry Descriptions: All entries are based on a common format, but some entries may not use all possible subsections.

- The **NAME** subsection lists the exact names of the commands and subroutines covered under the entry and gives a very short description of their purpose.

- The **SYNOPSIS** summarizes the use of the program being described. A few conventions are used, particularly in the description of commands.

  - **Boldface** words are considered literals, and are typed just as they appear.

  - Square brackets (" **[ ]** ") around an argument indicate that the argument is optional.

  - Ellipses -- three dots in a row -- (" **...** ") are used to show that the previous argument can be repeated.

  - An argument beginning with a minus sign (-) often means that the argument is an option, even if it appears in a position where a file name could appear. Therefore, it is unwise to have files names beginning with a minus sign.

- The **DESCRIPTION** section discusses the topic in detail.

- The **EXAMPLES** section provides one or two examples of the command's use.

- The **FILES** section gives the name of files that are built into the program, system call, or subroutine.

- A **SEE ALSO** section shows where related information is located.

- A **DIAGNOSTICS** section discusses the diagnostic indications that can be produced. Self-explanatory messages are not listed.

- The **LIMITATIONS** section gives known limitations (and sometimes deficiencies).

- In Section 2, a subsection called **ASSEMBLER** gives the assembly language required to implement the system call.

A permuted index points to each section by entry title and section number. The section number is important because some names are duplicated among the sections. Most of the duplications result from commands that exercise a particular system call.

1.2. Related Documents

ZEUS software documentation is supplied in four volumes.

The first volume is known as the ZEUS Reference Manual (ZRM). It is the basic user reference manual because it contains a description of all the basic user commands (commands that are typed at the terminal) available in the 3.1 ZEUS operating system.

The second volume of ZEUS documentation is the ZEUS Utilities Manual. It contains expanded instructions for 2Ø of the more complex ZEUS commands.

The third volume in the ZEUS set is the ZEUS Languages/Programming Tools Manual. It is a guide to the programming languages and related language aids provided with the ZEUS system.

The fourth volume in the ZEUS software documentation set is the System 8ØØØ ZEUS Administrator Manual. It contains information related to setting up and maintaining the System 8ØØØ. This manual is "model-specfic".

1.2.1. ZEUS Utilities Manual Summary: The ZEUS Utilities Manual contains articles that supplement the information in Section 1 of the ZEUS Reference Manual. The entries are:

Intro -- An Introduction to the ZEUS operating system
Acct -- The system accounting package
Awk -- A pattern scanning and processing language
Comm -- The Zeus communications package
Cshell -- The C Shell from UC Berkeley
Ed -- The basic line editor
Ex -- An expanded line editor
Fsck -- A File System Checking package
Learn -- Computer-aided learning program
Me -- The text formatting package from UC Berkeley
Ms -- The text formatting package from Bell

Laboratories
Nroff -- For terminal and line printer output
Sccs -- Source Code Control System package
Sed -- A non-interactive stream editor
Shell -- The Bourne Shell from Bell Laboratories
Tbl -- For formatting tables
Troff -- For CAT phototypesetter output
Uucp -- Unix to Unix Communication Package
Vi -- The visual mode of the Ex editor
Zeus for Beginners -- A basic introduction

Each expanded entry in the ZEUS Utilities Manual has a corresponding page in the ZEUS Reference Manual. Utilities entries are intended to provide the in-depth and tutorial information necessary for a complete understanding of the more complex and sophisticated commands in the ZEUS system.

1.2.2. ZEUS Languages/Programming Tools Manual Summary: The ZEUS Languages/Programming Tools Manual contains articles that supplement the information in the ZEUS Reference Manual.

Intro -- Introduction to ZEUS Languages/Programming Tools Manual
Adb -- A Tutorial Introduction to Adb (A De-Bugger)
As -- System 8ØØØ Assembly Language Reference Manual
C -- The C Programming Language
Call Conv -- System 8ØØØ Calling Conventions
C-Isam -- C Index Sequential Access Method
Curses -- Cursor Motion Package
Lex -- A Lexical Analyzer Generator
Lint -- A C Program Checker
Make -- A Control Program Programming Language
M4 -- The M4 Macro Processor
Pgmg -- ZEUS Programming
Plz/Asm -- Programming Language -- ZEUS/Assembly
Plz/Sys -- Programming Language -- ZEUS/System
Screen -- Screen Handling
Yacc -- Yet Another Compiler-Compiler

1.2.3. ZEUS Administrator Manual Summary: The ZEUS Administrator Manual discusses procedures that are intended for the administrator of a ZEUS installation. This document provides specialized knowledge required for system boot and recovery procedures. System maintenance commands are in this document, and can also be found in manM of the usr/man system directory.

The sections in the Administrator Manual are:

Introduction
Start-up and Shut-down Procedures
Restoring the System Disk
File System Management
System Generation
System Crashes and Other Problems
System Maintenance
An Overview of ZEUS
Redistributiing Files on the Disk

1.2.4. ZEUS and Related Document Part Numbers:

System 8ØØØ ZEUS Reference Manual	Ø3-3255
System 8ØØØ ZEUS Utilities Manual	Ø3-325Ø
System 8ØØØ ZEUS Languages/Programming Tools Manual	Ø3-3249
System 8ØØØ ZEUS System Administrator Manual	Ø3-3246
System 8ØØØ Model 11 Administrator Manual	Ø3-3254

Programming References:

PLZ/ASM Assembly Language Programming Manual	Ø3-3Ø55
Report on the Programming Language PLZ/SYS	Ø3-3Ø59
C Programming Language	Ø3-3161

Hardware References:

System 8ØØØ Hardware Reference Manual	Ø3-3237
System 8ØØØ Model 11 Hardware Reference Manual	Ø3-3227

UNIX References:

UNIX Reference Manual for System 3
UNIX Programmer's Manual for System 3

Available from:

AT&T Co.
P. O. Box 25ØØ
Greensborough, NC 27420

The Bell System Technical Journal,
Vol. 57, No. 6, Part 2

Available from:

Bell Laboratories
Circulation Group
Whippany Road
Whippany, NJ Ø7981

SECTION 2
SUMMARY OF SECTION 1 COMMANDS

2.1. Section 1 Commands

Entries in this section marked with an asterisk-U "*U" denote a longer entry for the command in the ZEUS Utilities Manual, entries marked with an asterisk-P "*P" symbolize a longer entry for the command in the ZEUS Languages/Programming Tools Manual.

The commands in section 1 fall into 5 basic catagories:

Files
Programming Languages
The Shell
The System, and
Users

2.2. Commands Related to Files

All the entries in this catagory are commands that act on files.

2.2.1. File Communications:

The entries in this catagory are commands that transfer information via files, or are a significant part of that transfer process.

cu - call up another ZEUS system
mail - send or read mail to and from users
rmail - send mail only to users (cannot read mail)
uucp - UNIX-to-UNIX / ZEUS-to-ZEUS copy *U
uuname - list the **uucp** names of known systems
uux - UNIX-to-UNIX / ZEUS-to-ZEUS command execution

2.2.2. File Compare: Commands in this section are used to compare differences or commonalities between files -- primarily text files. Commands that check the contents of a file against another file (like the **spell** program) are also included.

bdiff - find the differences in large files (**diff**)
cmp - compare the contents of two files
comm - select or reject lines common to two sorted files
diff - find the differences in files (**bdiff**)
diff3 - find the difference between three files (**diff**)
diffmk - mark differences between files (**diff**)
dircmp - compare the contents of two directories
sdiff - print **diff** results side-by-side
spell - find spelling errors in text files
spellin - add words to **spell** list (**spell**)
spellout - find words not in **spell** list (**spell**)
sort - sort and/or merge the lines in a file
tsort - topological sort of lines in a file
uniq - find repeated lines in a file

2.2.3. File Editors: Commands in this section include the actual system editors (**ed, edit, ex,** and **vi**) and commands that are used to change the contents of a file (usually at text file).

awk - pattern scanning and processing language *U
csplit - context split
cut - cut out selected fields of each line of a file
ed - basic editor for first-time users *U
edit - intermediate editor for casual users
ex - high-powered editor for advanced users *U
join - join together lines from two pre-sorted files
paste - merge same lines of several files
sed - non-interactive stream editor like **ed**
split - split a file into pieces
tr - translate characters from one class to another
vi - visual mode of the **ex** editor *U

2.2.4. File Formatter: The commands in this section are used to change raw input files (usually text and formatting commands) into formatted output files. Commands that are related to this process, like the **col** filter, are also included.

banner - print strings in large letters
col - filter escape characters out of a file
crypt - encode/decode a file
deroff - remove **nroff/troff, tbl,** and **eqn** constructs *U
eqn - **troff** pre-processor for typesetting equations
neqn- **nroff** pre-processor for printing equations
checkeq - check syntax of **eqn** input
expand - expand tabs to spaces
nl - add line numbers to each line of file
pr - format files for printer output

ptx - create and format the permuted index
tbl - table pre-processor for **nroff** or **troff** *U
troff - text formatter for CAT phototypesetter output *U
nroff - text formatter for printer output *U

2.2.5. File Examination:

The commands in the following section are used to produce or investigate the contents of files (usually text files).

cat - print (concatentate) the contents of a file
dog - print the contents of a file one screen at a time
grep - find a pattern in a file using **ed** syntax
egrep - **grep** using **ex** syntax
fgrep - fast **grep** using fixed strings
head - print the first 1Ø lines of a file
hyphen - find hyphenated words in a file
more - print the contents of a file one screen at a time
page - print the contents of a file one screen at a time
tail - print the last 1Ø lines of a file

2.2.6. File Moving: Commands in this section deal with moving and storing files.

ar - archive and library file maintainer
cp - copy a file to a new name and/or location
cpio - copy file archives in and out of their library
dd - convert and copy (dump) a file
getfile - transfer files from local to remote system
ln - link one filename to a file
mkdir - make a directory
mv - move or rename files and directories
pack - compress files to save disk space
pcat - **cat** compressed files without unpacking them
unpack - expand previously compressed files
putfile - transfer files from remote to local system
rm - remove a file
rmdir - remove a directory
tar - archive files for tape
tee - split output of a process into two destinations

2.2.7. File Protection: The commands in this section deal with the protection mode for files and directories. These commands control the read, write, and execute access for files.

chmod - change the protection mode of a file or directory
chown - change ownership of a file or directory
umask - set the default file protection mode for new files

2.2.8. File Source Code Control System: The commands in this section relate to the manipulation of the Source Code Control System (SCCS). Refer to the SCCS section of the ZEUS Utilities Manual for more information on this program.

admin - create and administer SCCS files *U
cdc - change the commentary in an SCCS file *U
chkdiff - list differences between versions of a source file
chkin - check in file to Zilog Source Control file
chkout - check out file from Zilog Source Control file
comb - combine SCCS changes *U
delta - make a delta (change) to an SCCS file *U
get - get a version of an SCCS file *U
prs - print an SCCS file *U
rmdel - remove a change from an SCCS file *U
sact - print current SCCS file editing activity *U
sccsdiff - compare two versions of an SCCS file *U
unget - undo a previous get of an SCCS file *U
val - validate SCCS file *U
what - identify SCCS files *U

2.2.9. File Status: Commands in this section are used to find, identify, investigate, or summarize files, their contents, or qualities.

chkwhat - print Zilog Source Control what strings
du - summarize disk space usage
file - determine file type
find - find files
ls - list files and directories in current directory
quot - summarize file system ownership
sum - sum and count blocks in a file
touch - update access and modification times of files
vls - "visually" list files and directories
wc - count the lines, words, and characters in a file
whereis - locate source, binary, and or manual for program

2.3. Languages and Programming Tools:

Commands in this section are used in conjunction with ZEUS programming languages. More information on the programming languages can be found in the ZEUS Languages/Programming Tools Manual.

adb - debugger *P
as - PLZ/ASM assembler *P
cas - invoke assembler
cb - C program beautifier
cc - System 8ØØØ C compiler *P
code - print characters with their octal equivalents
cref - make cross-reference listing *P
cxref - a simple C routine referencing program *P
error - analyze and disperse compiler error messages
expr - evaluate arguments as an expression
flow - flow analysis of C programs *P
ld - nonsegmented Z8ØØØ and 8-bit loader *P
lex - generate programs for simple lexical tasks *P
lint - a C program verifier *P
load - download to a Zilog Development Module
lorder - find ordering relation for an object library
m4 - macro processor *P
make - maintain, update, and regenerate groups of programs *P
mkstr - create an error message file by massaging C source *P
nm - print name list
objdu - dump for object and load modules
objhdr - object module header dump
objsu - object module underscore stripper
od hd - octal or hex dump
plz - plz/sys compiler driver *P
plzcg - plz/sys System 8ØØØ code generator *P
plzsys - plz/sys compiler *P
prof - display profile data
prom - prom programming utility
ranlib - convert archives to random libraries
regcmp - regular expression compile
scc - segmented C compiler *P
send - upload to the Zilog Development Module
size - size of an object file
sld - segmented Z8ØØØ loader
sprof - display profile data
strings - print strings in object or other binary file
strip - remove symbols and relocation bits and header
uimage - Zobj to a.out translator
xref - cross reference for C programs *P
xstr - extract strings from C programs
yacc - yet another compiler-compiler *P

2.4. The Shell Commands

The Shell is the primary interface between the user and the computer. The programs in this section relate to the functions of the shell. For more information on the shells refer

to CSH - The C Shell and SH - The Bourne Shell both in the ZEUS Utilities Manual

2.4.1. Shell Commands:

Shell commands control or report on the way the shell responds to all other commands typed into the terminal.

at - execute command or shell script file at a later time
cd - change working directory
env - set environment for command execution
kill - send a signal to a process
printenv - display environment variables
ps - process status
pwd - working directory name
script - record all terminal interactions
sleep - suspend execution for an interval
wait - await completion of background processes
xargs - construct argument list(s) and execute command

2.4.2. Shell Scripts:

Shell scripts are files of commands executed by the shell, the commands in this section relate to making and executing these shell scripts.

basename - truncate path name to filename
dirname - truncate path name to directory name
echo - print strings with carriage return
echo2 - print strings without carriage return
getopt - break up (parse) command line options
gets - get a string from the terminal input
line - read one line from the terminal input
nice - run a command at low system priority
nohup - run a command immune to phone hangups
test - test qualities of files and strings
true - provide the value "true"
false - provide the value "false"

2.4.3. The Shells as Commands: Since the shell is itself a command, it has its own program and manual entry. The three commands in this section are shells and can be used from the terminal.

csh - a command interpreter with C-like syntax *U
rsh - restricted shell (command interpreter)
sh - the Bourne Shell command interpreter *U

2.5. The System as a Whole

Some commands pertain to functions of the system as a whole, the on-line reference manual, the system calculator and clock, and the devices linked to the system.

The commands in this section deal with those programs that relate to these aspects of the system.

2.5.1. System Manual: The system contains an on-line manual. The commands in this section relate to that manual or other on-line sources of information.

apropos - locate commands by keyword lookup
getname - get NAME sections from manual source
help - on-line assistance
learn - computer-aided instruction program *U
look - find lines in a sorted dictionary list
man - print sections of the ZEUS Reference Manual
news - print news items
vnews - "visually" display the news items
whatis - describe a command

2.5.2. System Math: The system maintains an on-line calculator. The commands in this section relate to that function.

bc - better calculator (subset of **dc**)
dc - desk calculator
units - print formula to change one unit to another

2.5.3. System Devices: The system interfaces with a number of peripheral devices. The commands in this section regulate the way that interaction takes place.

3ØØ - handle special functions of DASI 3ØØ terminal
3ØØs - handle special functions of DASI 3ØØs terminal
45Ø - handle special functions of DASI 45Ø terminal
greek - select terminal filter
isrio - determine if terminal is a RIO System
local - return control to local system
lpr - line printer spooler
nq - print enqueing program
remote - transfer control to a remote ZEUS/UNIX system
reserv - tape drive reserving system
reset - reset terminal modes to default values
stty - set the options for a terminal
tabs - set tabs on a terminal
tty - get terminal name

vtzset - set up vtz terminal function keys
xq - examine or delete requests from the line printer

2.5.4. System Time: The system contains a clock and calendar mechanism. These commands relate to those functions:

cal - print calendar
calendar - reminder service
date - print the date and time
daytime - give the time to human-reasonable accuracy
time - time a command
timex - time a command and generate a system activity report

2.6. User-Oriented Commands

This last block of information relates to the fact that the ZEUS system is an interactive, multi-user system.

2.6.1. User Communications:

Users logged on at the same time can communicate with each other over the system with a variety of tools. The commands in this section relate to those communication tools.

mesg - permit or deny incoming user messages
talk - communicate with another user, character by character
write - communicate with another user, line by line

2.6.2. User Info: This group of commands provides information about other users on the system.

id - print user and group ID and names
logname - get login name
uname - print name of current ZEUS
users - compact list of users who are on the system
who - print a list of the users currently on the system
whoami - print current user login name
whodo - print current users and their process status
whois - access the user information database

2.6.3. User Manipulation: The commands in this group relate to the files and commands that provide an individual user with an account and a system "identity".

chgrp - change group
gpasswd - change group password

grpck - password/group file checkers
pwck - password/group file checkers
login - sign on
newgrp - log in to a new group
passwd - change login password
su - substitute user ID temporarily

SECTION 3
ZEUS FUNCTIONAL OVERVIEW

ZEUS is a powerful multi-user operating system for interactive use. It is developed as a super-set of UNIX, and has many tools for the development of operating systems, languages, and computer networks. ZEUS is also useful for document preparation.

ZEUS is installed on Zilog's System 8ØØØ 16-bit microcomputer. A standard video keyboard terminal provides console interaction.

This functional overview of ZEUS discusses the following topics:

- The File System
  - Ordinary Files
  - Directories
  - Special Files
  - Removable Files
- Protection
- ZEUS I/O
- Processes and Images
  - Process Creation
  - Program Execution
  - Process Parent/Child
  - Process Termination
- Process Synchronization
  - Signal
  - Pipe
  - Wait
- The Shell
  - Standard I/O
  - Filters
  - Pattern Matching
  - Multitasking
  - Command Files
- Shell Implementation

3.1. File System

One of the most important features provided by ZEUS is its file system. The file system is hierarchically structured, using the concepts of root directories, subdirectories, and path names to locate specific files. ZEUS provides three types of files: ordinary and directory files (the hierarchical file system), and special files, which are for system devices (I/O).

3.1.1. Ordinary Files: Ordinary disk files contain information placed there by a user. This information can be source programs, object programs, documents, or data bases. To the ZEUS operating system, a file is only a one-dimensional array of bytes with no implied structure. This means that any type of structure can be imposed on an ordinary file.

3.1.2. Directory Files: An ordinary file is located by reference to a directory file. Directory files provide the mapping between a file name and the file itself; this effectively induces a structure on the entire file system. Files can be grouped in subdirectories (subdirectories are created by the user) to any practical depth. The references to files organized in this manner, when graphically depicted as in Figure 3-1, look like a rooted tree, and provide a hierarchical structure. This structure is often called a directory tree.

ZEUS maintains several directories for its own use. Perhaps the most important of these is the root directory (/) which is the base of the entire file structure. All files in the system can be found by tracing a path from the root through a chain of directories, though it is not necessary for every path to start at the root.

The construction of the path through a chain of directories is called the path name. This path name to a file consists of directory names, separated by /, ending in the file name. Figure 3-1 illustrates a typical file structure. The same nondirectory file can appear in several directories under possibly different names. For instance, two files, file1 and file2 might be the same file. This feature is called linking; a directory entry for a file is sometimes called a link. (The link system call can be found in link(2).) In the ZEUS system, all links to a file have equal status. That is, a file does not exist within a particular directory; the directory entry for a file consists merely of its name and a pointer to the information actually describing

the file. Thus, a file exists independently of any directory entry, although in practice, a file disappears along with the last link to it.

At the base of the illustrated tree structure (Figure 3-1) is the system root, /. This root directory, like any directory, contains entries that point to any of the three types of files. The root directory illustrated has three entries, two of which point to other directory files, /dev and z, and one pointing to an ordinary file, me. The directory dev has entries for two special files, zd5 and ctl (I/O drivers for the disk and cartridge tape unit 1). The directory /z contains entries for two files, a and b, and a subdirectory, c. Under the directory /z/c, there are also two files named a and b. This poses no problems when the files are searched for because the full path name specifications are different (/z/a and /z/c/a).

All names in the ZEUS file system must be 14 or fewer characters. There is little other restriction imposed on names; for example, the naming of files beginning with a minus sign (-) is not recommended, and the C language compiler expects source files to end with .c. Other naming conventions can be found in the compiler descriptions.

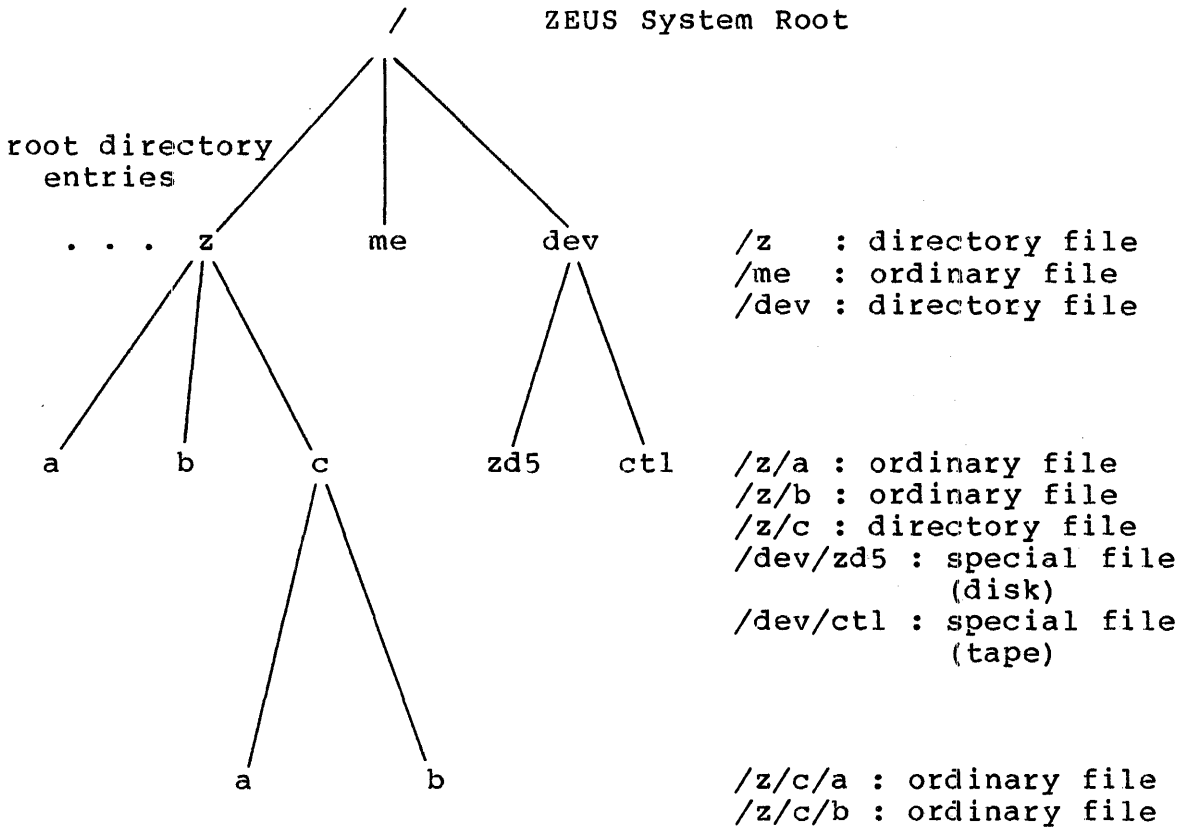

Figure 3-1 File System Representation

It is always possible to specify a complete path name for a file, in which the path name starts from the root. However, the unlimited depth allowed by the ZEUS directory structure can make such names inconveniently long. To remedy this, the file structure uses the concept of a current directory, also called the working directory. The system interprets path names not beginning with the root (nonrooted path names) as being relative to the current working directory. This current directory can be moved around the tree at will (unless the directory is read protected). To illustrate, suppose the current directory is placed at /z, shown in

Figure 3-1. Specifying nonrooted path name a refers to the file /z/a. If the current directory is moved to /z/c, the path name a refers to /z/c/a. In effect, the name of the current directory is prepended to nonrooted path names.

Normally, each directory contains a subtree of files and directories. The shape of these subtrees is free form, and subordinate directories can be created and destroyed at will. Directories can be structured according to the task to be done. At log in, the initial position of the working directory is typically set to the top of the subtree. This initial, current directory, specified as part of the login information, is contained in the file /etc/passwd, and is commonly called the home directory.

Each directory always has at least two entries. The name dot (.) in each directory refers to the directory itself. A program can read the current directory under the name . without knowing its complete path name. In Figure 3-1, the file a can be referred to by /z/a, or, if the current directory is /z, by a or ./a. The name .. refers to the parent of the directory in which it appears, that is, the directory in which it was created.

Except for the special entries . and .., each directory must appear as an entry in exactly one other directory, which is its parent. This simplifies the writing of programs that access subtrees of the directory structure, and more importantly, avoids the separation of portions of the hierarchy. If arbitrary links to directories were permitted, it would be difficult to detect when the last connection from the root to a directory was severed.

The difference between a directory and an ordinary file is that a directory can be written on only by privileged system programs. This is because directories impose a file structure. Write-permission on an ordinary file enables changes to be made to the contents of that file; write-permission on a directory (for all but privileged users) enables files to be added or deleted from that directory.

3.1.3. Special Files:

Each supported I/O device is associated with at least one special file. Special files are read and written just like ordinary disk files, but requests to read or write activate the associated device. An entry for each special file resides in the directory /dev. A link can be made to one of these files just as it can to an ordinary file. To write to the line printer, write to the file /dev/lpØ.

Special files exist for each communication line, disk file system, tape drive, line printer, or terminal, and for physical main memory. These files are generally created through the system generation process.

There are advantages in treating I/O devices this way. File and device I/O are as similar as possible; file and device names have the same syntax and meaning so that a program expecting a file name as a parameter can be passed a device name. Finally, special files are subject to the same protection mechanisms as regular files.

3.1.4. Removable File Systems:

When the system is generated, each disk on the generated system is partitioned into one or more unrelated but physically contiguous regions. A region can be thought of as a virtual disk associated with a file system. Although the root of the ZEUS file system is always stored on the same disk, the entire file system hierarchy does not have to reside on the same physical (virtual) disk.

The system mount request attaches an independent file system with its own directory hierarchy to the existing file system hierachy (tree). The mount request has two arguments: the name of an existing directory, and the name of a special file whose associated virtual disk has the independent file system and hierarchy. In effect, the mount request replaces a leaf or subtree (the directory) of the current hierarchy tree with a whole new subtree (the hierarchy stored on the virtual disk).

After a file is mounted, there is no distinction between files on the mounted virtual disk and those of the permanent file system. There is one exception to the rule of identical treatment of files mounted on different devices: no link can exist between one file system hierarchy and another. This restriction is enforced to avoid the elaborate system bookkeeping that would be required to ensure removal of any existing links whenever the removable file system is unmounted.

3.2. File Security Protection

The ZEUS system administrator, who is a person charged with the administration of the ZEUS installation, creates a unique individual user identification number and a group identification number for each new user. When the new user creates a file, it is marked with the assigned user and

group ID. This new file also has ten protection bits. Nine of these protection bits independently specify read (r), write (w), and execute (x) permission for each of three levels of access. The three levels of access protection are:

- owner--the user
- group--members of the user's group who have the same group ID
- public--all other users of the system

Example:

```
-rw-r--r-- 2 cpc  system    46 Feb 1Ø 18:43 addpath
drwxrwxr-x 2 cpc  system  2144 Apr 22 18:22 bin
```

This organization is the same for ordinary files and special files. For directories, the meaning of the access bits are modified. Read means the ability to read the directory as a file, that is, to discover all the names it contains. Execute means the ability to search a directory for a given name when it appears as part of a qualified name. Write means the ability to create and delete files in that directory. It is unrelated to writing of files in the directory.

If the tenth protection bit is on, the system temporarily changes the ID of the current user to that of the owner or creator of the file whenever the file is executed as a program. This change in user ID is effective only during execution of the program that calls for it. The set-user-ID feature allows for privileged programs that can use files inaccessible to other users. For example, a program may keep an accounting file that should neither be read nor changed except by the program itself. If the set-user-ID bit is on for an executing program, the file can be accessed (although this access might be forbidden to other programs invoked by the given program's user). Since the actual user ID of the user who invokes a program is always available, set-user-ID programs can take any measures required to check the invoker's credentials. This mechanism allows execution of carefully written commands that call privileged system entries. Because the set-user-ID bit can be set on one's own files, this mechanism is generally available without administrative intervention.

The system call that creates an empty directory can only be invoked by the super-user. Directories are expected to have . and .. entries. The command that creates a directory is owned by the super-user and has the set-user-ID bit set.

After ZEUS checks the invoker's authorization to create the specified directory, the directory is created and the entries for . and .. are made.

The system recognizes the super-user ID as able to access all files without regard to permissions. The super-user is also the only one permitted to make privileged system calls. Programs owned by the super-user with the set-user-ID bit set can be written to (such as dump and reload a file system) without unwarranted interference from the protection system.

The chmod system call and its corresponding command, which change the protection bits of a file, is only executable by the super-user or on files owned by the user.

3.3. Input/Output

Files in ZEUS are manipulated through Input/Output (I/O) system calls that create, delete, read, write, or seek into a file.

I/O system calls do not differentiate between various devices and styles of access methods. There is no distinction between random and sequential I/O, nor is there any concept of record. The size of an ordinary file is determined by the number of bytes written on it; no preallocation of disk area is needed since files grow dynamically as they are written.

To illustrate the essentials of I/O, some of the basic calls are summarized below. Each call to the system can result in an error return. (This is not represented in the examples.) To read or write a file assumed to exist already, it must be opened by the following system call:

filep = open (name, flag)

where name indicates the name (character string) of the file. An arbitrary path name can be given. The returned value filep is called the file descriptor. This is a small integer used to identify the file in subsequent calls that read, write, or otherwise manipulate the file. The flag argument indicates whether the file is to be read, written, or updated; that is, read and written simultaneously.

The flag argument also indicates the access privileges that other users have while the file is open. These open access privileges are distinct from the eleven protection bits discussed above. The protection bits are checked at open time

to see if the invoker has permission to access the file. Once this is done, if the file is already open, the open access privileges are checked to make sure the program that originally opened the file will allow others to open it. When a file is opened, the following permissions are specified for others attempting to open the file:

- Any executing program can open the file
- Any executing program can open the file as long as it only reads it
- No executing program can open the file

When the I/O is complete and the file is closed, any constraints imposed while the file was open are removed.

To create a new file or completely rewrite an old one, there is a create system call that creates the given file if it does not exist, or truncates it to zero length if it does exist; create also opens the new file for writing and, like open, returns a file descriptor. Such a call is defined as:

filep = creat(name, mode)

where name is the name of the file and filep is the file descriptor. The mode argument specifies the eleven protection bits that the file will have, as well as the open access privileges allowed others while the file is open.

Reading and writing are sequential (except as indicated below). This means that for any particular last byte written (or read) in the file, the next I/O call refers to the immediately following byte. For each open file there is a pointer, maintained inside the system. When n bytes are read or written, the pointer advances by n bytes.

Once a file is open, the following calls can be used:

n = read(filep, buffer, count)
n = write(filep, buffer, count)

Up to count bytes are transmitted between the file specified by filep and the byte array specified by buffer. The returned value n is the number of bytes actually transmitted. In the **write** case, n is the same as count, except under exceptional conditions, such as I/O errors or end of physical medium on special files. In a read, n can, without error, be less than count.

If the read pointer is so near the end of the file that reading count characters causes reading beyond the end of the file, bytes are transmitted to reach only to the end of the file. When a **read** call returns with n equal to zero, the end of the file has been reached. For disk files, this occurs when the read pointer becomes equal to the current size of the file.

Bytes written affect only those parts of a file pointed to by the position of the write pointer and the count; no other part of the file is changed. If the last byte lies beyond the current end of the file, the file expands as needed.

For random (direct access) I/O, the read or write pointer is moved to the appropriate location in the file; thus,

location = lseek(filep, offset, base)

The pointer associated with filep is moved to a position offset n bytes from the beginning of the file, from the current position of the pointer, or from the end of the file, depending on the base value x. Offset can be negative. For some devices (paper tape and video terminals), seek calls are ignored. The actual offset from the beginning of the file to which the pointer was moved is returned on location.

Other I/O and file system calls exist that are not discussed here. These are listed below with the corresponding command, if any.

close(2)	close file
stat(2)	change protection mode
chmod(2),chmod(1)	change owner
chown(2),chown(1)	create directory
mknod(2),mkdir(1)	create directory
link(2), Ln(1)	link existing file
unlink(2), rm(1)	delete file

The command counterparts to the system calls shown are generally more flexible and have more options than the system calls.

SECTION 4
PROCESSES AND IMAGES

An image can be thought of as a snapshot of the program execution environment. The image includes not only a memory image of the code and data of the program itself, but also the current state of the registers, the status of open files, and the current directory.

A process is the execution of an image, or, in other words, an executing program. While the CPU is executing on behalf of the process, the image must reside in main memory; during the execution of other processes, it remains in main memory unless the appearance of an active, higher-priority process forces it to be swapped out on disk.

The user has a virtual address space within which a process runs. This address space is either 64K bytes or 128K bytes, depending on how the program is compiled. (Compiling a program with the separate I&D (information and data) option allows for up to 64K of instruction space to 64K bytes of data space for a total of 128K.)

The user-memory of an image is divided into three logical groups or sections: the program text, data, and stack sections. These sections can share one or two contiguous memory areas. If the separate I&D compile option is used, then two memory areas are shared. In this case, the two physical memory areas are each 64K bytes--one for the shared write-protected text, and one for the data and stack. The text segment is write-protected during execution, and a single copy is shared among all processes executing the same program.

The program code, data, and stack reside in one memory area, up to 64K bytes long, when the separate I&D compile option is not used. In this case, the program code is not shared or write-protected. The stack, which is considered a part of the data since it shares the same memory address space, and the data are never shared among processes regardless of the compile option used.

The program text section begins at location Ø in the virtual address space. The data section also begins at location Ø in its own virtual 64K-byte address space if the program is compiled with the separate I&D option; otherwise, it follows the code. The data section can be extended in size by the brk and sbrk system calls. The stack section begins at the

highest address in the virtual address space and automatically grows downward as the stack pointer fluctuates.

4.1. Process Creation

A process can be put into execution from within an executing program by making the system call

```
processid = fork()
```

When fork is executed, the process splits and becomes two independently executing processes. The newly created process (child) is a copy of the original process (parent). The two processes have independent copies of the original memory image and share all open files. (If the parent process was executing from a read-only, sharable text segment, however, the child shares the text segment.) Copies of all the writable data sections are made for the child process.

Making a complete copy of a process with the fork is actually an effective way to communicate from the parent process to the child process; the child has access to the entire memory image of the parent. This avoids many structures and restrictions and allows arbitrary information transfer between parent and child.

Processes in ZEUS are inexpensive in terms of CPU time; a sharp contrast to many other operating systems. The forking of a medium-sized process requires only a few milliseconds. This low cost means that the fork feature is used extensively and provides the basis of shell interaction, in that (almost) every entered command is executed in a child process forked for the command. When the command completes, the child process terminates.

4.2. Execution of Programs

Programs are executed by invoking a form of the exec system call:

```
exec(file, arg1, arg2, ...,argn)
```

where exec is **execv, execl, execve,** or **execle.** The exec system call requests ZEUS to read into memory and execute the program file, passing it string arguments arg1, ..., argn.

All of the code and data in the process attached to exec are replaced by the code and data in file. Open files, current

directory position, and interprocess relationships are unaltered, because invoking exec does not change processes. The process attached to exec persists; it is just executing a different program file. A return to the calling process takes place only if the call to exec fails.

Consider interactive commands as an example of fork and exec usage. A process is interactively invoked from the keyboard by giving the name of the object file to be run. To copy a file temp1 in the current directory to a file temp2, a process is invoked with this keyboard command:

/bin/cp temp1 temp2

ZEUS can be given a single path name, such as cp, causing a search through a user-set path for the named executable file. The following command is most commonly entered:

cp temp1 temp2

The interactive command is interpreted by the shell to generate a process to perform the copy. The shell forks a copy of itself, and the copy of the shell searches for the executable program cp. When cp is found, it is invoked by an exec system call from the shell copy. The executing shell copy is the parent process of the cp command. In this way, the shell is not destroyed and can interpret subsequent commands when the child process, cp, finishes.

4.3. The Parent/Child Relationship

The new processes created by a fork differ only in that one is considered the parent process. In the parent, the returned processid actually identifies the child process and is never Ø, while for the child process, the returned value is always Ø. Because the values returned by fork in the parent and child process are different, each process can determine whether it is the parent or child.

If process A invokes processes B and C , process A is called a parent process, and processes B and C are its children. If process B invokes processes D and E, B is the parent of D and E. D and E would be the grandchildren of A or, more generally, descendents of A. If a parent process terminates, its descendents continue execution until they are finished. A descendent process's parent then becomes a ZEUS system process.

4.4. Process Termination

The system call

exit(status)

terminates a process, destroys its image, closes its open files, and generally removes it from the system.

A process can be terminated interactively (from the keyboard) by the command:

kill processid

This sends a signal (Section 2.2.5) to the process whose ID is processid. If the signal is not handled in some other way, the process is terminated. The command:

kill -9 processid

is a sure kill, and must be issued carefully.

The equivalent system call of the above command is:

status = kill
(processid, 9)

where **status** is an error status.

4.4.1. Process Synchronization: ZEUS provides mechanisms whereby a process can synchronize itself with either an external event or another process. These are discussed in this section.

4.5. The Signal Mechanism

A signal is generated by some abnormal event, or initiated at a console keyboard (quit, interrupt), by a program error (bus error, illegal instruction, etc.), or by a another program request (kill). Normally, all signals cause termination of the receiving process; however, a signal system call allows signals to be handled in one of three ways: signals can be ignored; they can cause termination of the process (reinstate the default); they can result in a call to a specified routine. The signal call looks like:

old_value = signal (sig,func)

where old value is a value that indicates how the signal was

handled previously. The variable sig is the event to be caught (such as a quit from the typewriter), func is an indication of what to do when the signal occurs, whether it is to terminate the process, ignore the signal, or call the process's routine func.

There are sixteen signals. The kill(1) command normally generates the signal SIGTERM (a constant equivalent to 15) which, when not caught, results in termination of the process. A process can choose to catch such signals to clean up temporary files before terminating. A process can also ignore such calls. If a process should be terminated, but ignores SIGTERM signals, the signal SIGKILL (a constant equivalent to 9) can be issued by entering:

kill -9 processid

This signal cannot be caught or ignored; it results in an automatic termination of the process associated with processid. (The kill command must be issued only for processes belonging to the user, unless the user is the super-user.)

It is possible to suspend program execution while waiting for a signal. The pause system call does this to prevent busy waiting.

If a process issues a signal call and then forks a child process, the child's signal is handled in the same way; the child inherits all signals.

4.6. The Pipe Mechanism

Processes can synchronize with related processes through the pipe mechanism. The pipe mechanism allows sending messages back and forth between processes using the same system read and write calls that are used for file system I/O. The system call:

Ireturn_value = pipe(fildes)

returns two file descriptors in array filep and creates an interprocess pipe. One file descriptor is used for reads, the other is used for writes. The command return value indicates whether or not the system call resulted in the successful creation of the pipe. A read, using a pipe file descriptor, waits until another process writes using the write file descriptor for the same pipe. The writing process can issue up to 4Ø96 bytes of data before it is suspended, waiting for a read from the pipe. Thus, data is passed between the images of the two processes. It does not

matter to either process that a pipe, rather than an ordinary file, is involved.

The pipe channel, like other open files, is passed from the parent to the child process image by the fork call.

4.7. The Wait Mechanism

Another process control system call:

processid = wait(status)

causes the parent process to suspend execution until one of its children completes execution. The command wait then returns the processid of the terminated process. An error return is taken if the calling process has no descendents. Certain status is available from the child process, such as a termination status.

As an example of the use of wait, the shell command line interpretor generally works as follows. When a command is entered, the shell forks a copy of itself. The child copy performs an exec, in effect becoming the process performing the requested command. Meanwhile, the parent process performs a wait and, when the child process finishes, interprets the next command.

A wait system call can also be interrupted by a signal mechanism.

4.7.1. The Shell: Most communication with ZEUS is through the shell, a command-line interpreter program that reads lines as requests to execute other programs. (The shell is described fully in the ZEUS Utilities Manual.) In simplest form, a command line consists of the command name followed by arguments to the command, all separated by spaces or tabs, as in

command arg1 arg2 ... argn

The shell splits the command name and the arguments into separate strings. Then a file with the name command is sought; command can be a path name including the / character to specify any file in the system. If command is found, it is brought into memory and executed. The arguments collected by the shell are accessible to the command. When the command is finished, the shell resumes its own execution, and indicates its readiness to accept another command by issuing a prompt character.

If the file command cannot be found, the shell usually prefixes a string such as /bin or /usr/bin to command and attempts again to find the file. (The path name, or sequence of directories to search can be changed by request.)

4.8. Standard I/O

The previous discussion of I/O implies that every file used by a program must be opened or created by the program to get that file's descriptor. Programs executed by the shell, however, start with three open files having the file descriptors Ø, 1, and 2. When such a program begins execution, file 1, called the standard output file, is open for writing. This file is the terminal, except under circumstances indicated in the following examples. File descriptor 1 is usually used to write program data. Conversely, file Ø starts open for reading, and programs that read entered messages read this file. File descriptor 2, another file open for writing, is similar to descriptor 1. By default, it is assigned to the terminal and usually used for standard error message output.

Many commands request information from the console keyboard. These commands issue a read system call using file descriptor Ø. No open is required of the command program.

The shell can change the standard assignments of these file descriptors from the terminal. If an argument to a command is prefixed by >, for the duration of the command file descriptor 1 refers to the file named after the >. For example:

```
ls
```

ordinarily lists, on the standard output, the names of the files in the current directory. The command:

```
ls > there
```

creates a file called there and places the listing there. The argument > there means place output on there. On the other hand:

```
ed
```

ordinarily enters the editor, which takes requests from the keyboard. The command

```
ed < script
```

interprets script as a file of editor commands; thus < script means take input from script.

Although the file name following < or > appears to be an argument to the command, it is interpreted completely by the shell and is not passed to the command at all. Thus, no special coding to handle I/O redirection is needed within each command; the command uses the standard file descriptors Ø and 1 where appropriate.

File descriptor 2, like file descriptor 1, is ordinarily associated with the terminal output stream. When an output-diversion request using > in the command argument is specified, file 2 remains attached to the terminal so that commands producing diagnostic messages do not silently place them in the redirected output file.

4.9. Filters

The output of one command can be directed to the input of another command by extending the concept of standard I/O.

A sequence of commands separated by a vertical bar (|) causes the shell to execute all the commands simultaneously and to arrange that the standard output of each command be delivered to the standard input of the next command in the sequence. This is called piping, since the output of one command is piped to the input of another. In the command line:

```
ls | pr -2 | opr
```

ls lists the names of the files in the current directory; its output is passed to pr, which paginates its input with dated headings. (The argument -2 requests double-column output.) Likewise, the output from pr is input to opr; this command invokes a hypothetical program that spools its input onto a file for off-line printing.

This procedure is inefficiently accomplished by:

```
ls > temp1
pr -2 < temp1 > temp2
opr < temp2
```

followed by removal of the temporary files. Without the ability to redirect output and input, a still clumsier method would require the ls command to accept requests to paginate its output, to print in multicolumn format, and to arrange that its output be delivered off-line. Actually, it

would be surprising, and in fact unwise for efficiency reasons, to expect authors of commands such as ls to provide such a wide variety of output options.

A program such as pr, which copies its standard input to its standard output (with processing), is called a filter. Some useful filters have been developed that perform character transliteration, selection of lines according to a pattern, and sorting of the input.

SECTION 5
PATTERN MATCHING AND OTHER TRICKS

The shell can generate a list of file names that match a pattern. These file names can then be used as input arguments to a command. In general, patterns are specified as follows:

* Matches any string of characters, including the null string. For example, the command

lpr /a/b/c/*

prints all the files in directory **/a/b/c** and

lpr /a/b/c/*.c

prints all the files whose names end in **.c** in that directory.

? Matches any single character.

[...]
Matches any one of the enclosed characters. A pair of characters separated by a minus sign matches any character lexically between the pair.

5.1. Command Separators and Multitasking

The shell supports multiple command entries on a single line. Commands need not be on different lines; instead they can be separated by semicolons.

ls; ed

first lists the contents of the current directory, then enters the editor.

A related feature is more interesting. If a command is followed by **&**, the shell executes the command in the background and does not wait for the command to finish before prompting again; instead, it is ready immediately to accept a new command. For example:

as source > output &

causes **source** to be assembled, with diagnostic output going

to **output;** no matter how long the assembly takes, the shell returns immediately. When the shell does not wait for the completion of a command, the identification number of the process running that command is printed. This process identification can be used to wait for the completion of the command or to terminate it (with kill). The & can be used several times in a line. The command

```
as source > output & ls > & files
```

does both the assembly and the listing in the background. In these examples, an output file other than the terminal was provided. If this had not been done, the outputs of the various commands would be intermingled at the video terminal.

The shell also allows parentheses in the above operations. For example:

```
(date; ls) > x &
```

writes the current date and time, followed by a list of the current directory, onto the file x. The shell also returns immediately for another request.

5.2. The Shell as a Command: Command Files

The shell is itself a command, and can be called recursively. Suppose file tryout contains the lines:

```
as source
mv a.out testprog
testprog
```

The mv command causes the file a.out to be renamed testprog. The file a.out is the (binary) output of the assembler, ready to be executed. Thus, if the three lines above were typed on the keyboard, source would be assembled, the resulting program renamed testprog, and testprog executed. When the lines are in tryout, the command:

```
sh < tryout
```

causes the shell sh to execute the commands sequentially. Furthermore, if the chmod command were used to change the protection bits of the file tryout so that the file became executable, the command:

```
tryout
```

would be equivalent to invoking the shell explicitly. This happens because the script file of shell commands has the execute permission bits turned on; the shell interprets the command as if it had been entered:

```
sh < tryout
```

The shell has further capabilities, including parameter substitution and the ability to construct argument lists from a specified subset of the file names in a directory. The shell also provides general conditional and looping constructions.

5.2.1. Implementation of the Shell: Most of the time, the shell is waiting for a command. When the new line character ending the line is typed, the shell's read call returns. The shell analyzes the command line, putting the arguments in a form appropriate for execute. Then fork is called. The child process, whose code is still that of the shell, attempts to perform an exec with the appropriate arguments. If successful, this brings in and starts execution of the program whose name was given. Meanwhile, the other process resulting from the fork, which is the parent process, waits for the child process to terminate. When this happens, the shell knows the command is finished, so it types its prompt and reads the keyboard to obtain another command.

Given this framework, the implementation of background processes is trivial; whenever a command line contains &, the shell merely refrains from waiting for the process that it created to execute the command.

This mechanism meshes well with the concept of standard input and output files. When a process is created by the fork system call, it inherits not only the memory image of its parent but also all the files currently open in its parent, including those with file descriptors Ø, 1, and 2. The shell, of course, uses these files to read command lines and to write its prompts and diagnostics. In the ordinary case, its children--the command programs--inherit these files automatically. When an argument with < or > is given, however, the offspring process, just before it performs exec, makes the standard I/O file descriptor (Ø or 1) refer to the named file. The smallest unused file descriptor is assigned when a new file is opened (or created); it is only necessary to close file Ø (or 1) and open the named file. Because the process in which the command program runs simply terminates when it is through, the association between a file specified after < or > and file descriptor Ø or 1 is

ended automatically when the process terminates. Therefore, the shell need not know the actual names of the files that are its own standard input and output, because it never needs to reopen them.

Filters are straightforward extensions of standard I/O redirection with pipes used instead of files.

In ordinary circumstances, the main loop of the shell never terminates. (The main loop includes the branch of the return from fork belonging to the parent process; that is, the branch that does a wait, then reads another command line.) One thing that causes the shell to terminate is discovering an end-of-file condition in its input file. When the shell is executed as a command with a given input file, as in:

sh < comfile

the commands in comfile are executed until the end of comfile is reached; then the shell process invoked by sh terminates. Because this shell process is the child of another invocation of the shell, the wait executed previously will return, and another command can then be processed.

NAME

INTRO - introduction to Section 1 commands

SYNOPSIS

INTRO is not a command, it is a manual entry, therefore there is no synopsis.

DESCRIPTION

The commands described in Section 1 of the ZEUS Reference Manual are available to all users and are typed into the computer via the terminal keyboard. The commands in each section of this manual are arranged alphabetically. Notations are made at the center of the foot of each command page indicating the engineer source of the command. These are:

Notation:	Source:
Bell	Bell Laboratories
UCB	University of California, Berkeley
Zilog	Zilog, Inc.

Reference to Section 1C as in the entry foreach(1C) refers to commands that are internal to the C Shell.

COMMAND SYNTAX

The description of each command (unless otherwise noted) uses the following conventions:

Bold Face Words in **bold** print are literal, they must be typed into the terminal [or they will appear on the terminal], just as they appear on paper.

Underlining Words that are underlined can or must be replaced by words of the user's choosing.

Brackets [ ] Words in brackets are options, they can appear in the command line, but may be omitted.

Ellipsis ... Three dots in a row -- an elipse -- is used to note that the preceding word can be repeated any number of times.

Quote Marks Double Quotes are used to isolate special characters from the body of the text.

Single Front Quotes are used to denote words with special meaning, though they may not be actual commands.

Single Back Quotes are used by the computer to substitute the output of a command for the command word itself.

SEE ALSO

Section Ø -- Introduction to ZEUS Software Documentation	
ZEUS Reference Manual	Ø3-3255
ZEUS Utilities Manual	Ø3-325Ø
ZEUS Languages/Programming Tools Manual	Ø3-3249
ZEUS Administrator Manual	Ø3-3246
Model 11 ZEUS Administrator Manual	Ø3-3254
PLZ/ASM Assembly Language Programming Manual	Ø3-3Ø55
Report on the Programming Language PLZ/SYS	Ø3-ØØ59
The C Programming Language	Ø3-3161

LIMITATIONS

Regretfully, many command descriptions do not adhere to the aformentioned conventions.

NAME

adb - debugger

SYNOPSIS

adb [**-w**] [objfil [corfil]]

DESCRIPTION

Adb is a general-purpose debugging program to examine files and provide a controlled environment for execution of ZEUS programs.

Objfil is normally an executable program file, preferably containing a symbol table; if not, the symbolic features of **adb** cannot be used, although the file can still be examined. The default for objfil is **a.out**. Corfil is assumed to be a core image file produced after executing objfil; the default for corfil is **core**.

Requests to **adb** are read from the standard input and responses are written to the standard output. **Adb** prompts for input with a question mark " ? ". If the -w flag is present, both objfil and corfil are created, if necessary, and opened for reading, modifying, and writing. **Adb** ignores QUIT; INTERRUPT causes return to the next **adb** command.

Requests to **adb** are of the form

[address] [,count] [command] [;]

If address is present, dot is set to address. Initially, dot is set to Ø. For most commands, count is an expression that specifies how many times the command will be executed (default is 1). Currently, address is considered an expression with an unspecifiable (default) segment number of Ø. It is expected that this will change.

The interpretation of an address depends on the context in which it is used. If a subprocess is being debugged, addresses are interpreted in the usual way in the address space of the subprocess. For further details of address mapping, see **ADDRESSES**.

EXPRESSIONS

. The value of dot

\+ The value of dot incremented by the current increment

^ The value of dot decremented by the current increment

" The last address typed

integer

An octal number if integer begins with a Ø; a hexadecimal number if preceded by %; otherwise, a decimal number

integer.fraction
A 32 bit floating point number

'cccc' An ASCII value of up to 4 characters; a backslash " \ " can be used to escape an apostrophe " ' ".

< name The value of name, which is either a variable name or a register name. **Adb** maintains a number of variables (see VARIABLES) named by single letters or digits. If name is a register name, the value of the register is obtained from the system header in corfil. The register names are **rØ ... r14 sp pc fcw**.

symbol A symbol is a sequence of upper or lowercase letters, underscores, or digits not starting with a digit. The value of the symbol is taken from the symbol table in objfil. An initial underbar " _ " or tilde " ~ " will be prepended to symbol if needed.

_ symbol
In C, the "true name" of an external symbol begins with _. It may be necessary to write this name to disinguish it from internal or hidden variables of a program.

routine.name
The address of the variable name in the specified C routine. Both routine and name are symbols. If name is omitted, the value is the address of the most recently activated C stack frame corresponding to routine. Currently, this expression type is not supported.

(exp) The value of the expression exp

Monadic Operators

*exp The contents of the location addressed by exp in corfil

@exp The contents of the location addressed by exp in objfil

-exp Integer negation

exp Bitwise complement

Dyadic Operators (left associative and less binding than monadic operators)

e1+e2 Integer addition

e1-e2 Integer subtraction

e1*e2 Integer multiplication

e1%e2 Integer division

e1&e2 Bitwise conjunction

e1|e2 Bitwise disjunction

e1#e2 E1 rounded up to the next multiple of e2

COMMANDS

Most commands consist of a verb followed by a modifier or list of modifiers. The following verbs are available. (The commands ? and / can be followed by an asterisk " * "; see ADDRESSES for further details.)

?f Locations starting at address in objfil are printed according to the format f

/f Locations starting at address in corfil are printed according to the format f

=f The value of address itself is printed in the styles indicated by the format f. (For i format, ? is printed for the parts of the instruction that reference subsequent words.)

A format consists of one or more characters that specify a style of printing. Each format character can be preceded by a decimal integer that is a repeat count for the format character. While stepping through a format, dot is incremented temporarily by the amount given for each format letter. If no format is given, the last format is used. The format letters available are as follows:

o 2 Print 2 bytes in octal. All octal numbers output by adb are preceded by Ø.

O 4 Print 4 bytes in octal.

d 2 Print in decimal.

D 4 Print long decimal.

x 2 Print 2 bytes in hexadecimal.

X 4 Print 4 bytes in hexadecimal.

u 2 Print as an unsigned decimal number.

U 4 Print long unsigned decimal.

f 4 Print the 32-bit value as a floating point number.

F 8 Print double floating point.

b 1 Print the addressed byte in octal.

c 1 Print the addressed character.

C 1 Print the addressed character using the following escape convention. Character values ØØØ to Ø4Ø are printed as @ followed by the corresponding character in the range Ø1ØØ to Ø14Ø. The character @ is printed as @@.

s n Print the addressed characters until a zero character is reached.

S n Print a string using the @ escape convention. n is the length of the string including its zero terminator.

Y 4 Print four bytes in date format (**ctime**(3)).

i n Print as Z8ØØØ instructions. n is the number of bytes occupied by the instruction. This style of printing causes variables 1 and 2 to be set to the offset parts of the source and destination respectively.

a Ø Print the value of dot in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below.

/	local or global data symbol
?	local or global text symbol
=	local or global absolute symbol

p 2
Print the addressed value in symbolic form using the same rules for symbol lookup as **a**.

t Ø
When preceded by an integer tabs to the next appropriate tab stop. For example, **8t** moves to the next 8-space tab stop.

r Ø
Print a space.

n Ø
Print a new line.

"..." Ø
Print the enclosed string.

^ Dot is decremented by the current increment. Nothing is printed.

\+ Dot is incremented by 1; nothing is printed.

\- Dot is decremented by 1; nothing is printed.

newline
If the previous command temporarily incremented dot, make the increment permanent. Repeat the previous command with a count of 1.

[?/]l value mask
Words starting at dot are masked with mask and compared with value until a match is found. If **L** is used, the match is for four bytes at a time instead of two. If no match is found, dot is unchanged; otherwise, dot is set to the matched location. If mask is omitted, -1 is used.

[?/]w value ...
Write the 2-byte value into the addressed location. If the command is **W**, write four bytes. Odd addresses are not allowed when writing to the subprocess address space.

[?/]m b1 e1 f1**[?/]**
New values for (b1, e1, f1) are recorded. If less than three expressions are given, the remaining map parameters are left unchanged. If the ? or / is followed by an asterisk " * " the second segment (b2,e2,f2) of the mapping is changed. If the list is terminated by ? or /, the file (objfil or corfil respectively) is used for subsequent requests. (For example, /m? causes / to refer to objfil.)

>name
Dot is assigned to the variable or register named.

! A shell is called to read the rest of the line following **!**.

$modifier

Miscellaneous commands. The available modifiers are:

$<f Read commands from the file f and return.

$>f Send output to the file f, which is created if it does not exist.

$r Print the general registers and the instruction addressed by **pc**. Dot is set to **pc**.

$f Print the floating registers.

$b Print all breakpoints and their associated counts and commands.

$C If $C is used, the 16 bit values for each active routine's stack frame are printed with the return address labeled. If count is given, only the first stack frames are printed If address is given, dot is temporarily moved to that address (i.e. pc is not altered) for perusal of instructions.

$e The names and values of external variables are printed.

$w Set the page width for output to address (default 72).

$s Set the limit for symbol matches to address (default 255).

$o All integers input are regarded as octal.

$x All integers input are regarded as hexadecimal.

$d Reset integer input as described in EXPRESSIONS.

$q Exit from **adb**.

$v Print all nonzero variables in hexadecimal.

$m Print the address map.

:modifier
Manage a subprocess. Available modifiers are:

:bc Set breakpoint at address. The breakpoint is executed count-1 times before causing a stop. Each time the breakpoint is encountered, the command :c is executed. If this command sets dot to zero, the breakpoint causes a stop.

:d Delete breakpoint at address.

:r Run objfil as a subprocess. If address is given explicitly, the program is entered at this point; otherwise, the program is entered at its standard entry point. Count specifies how many breakpoints are to be ignored before stopping. Arguments to the subprocess can be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command. All signals are turned on at entry to the subprocess.

:cs The subprocess is continued with signal s (signal (2)). If address is given, the subprocess is continued at this address. If no signal is specified, the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for r.

:ss As for **c**, except that the subprocess is single stepped count times. If there is no current subprocess, objfil is run as a subprocess as for **r**. In this case, no signal can be sent; the remainder of the line is treated as arguments to the subprocess.

:k The current subprocess, if any, is terminated.

VARIABLES

Adb provides a number of variables. Named variables are set initially by **adb** but are not used subsequently. Numbered variables are reserved for communication as follows:

Ø The last value printed.

1 The last offset part of an instruction source.

2 The previous value of variable 1.

On entry, the following are set from the system header in the corfil. If corfil does not appear to be a **core** file, these values are set from objfil.

b The base address of the data segment.

d The data segment size.

e The entry point.

m The magic number (ØxE6Ø7, ØxE611, ØxE6Ø5, ØxE7Ø7, ØxE711, ØxE7Ø5)

s The stack segment size.

t The text segment size.

ADDRESSES

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by two triples (b1, e1, f1) and (b2, e2, f2) and the file address corresponding to a written address is calculated as follows:

b1<address<e1 => file address=address+f1-b1

otherwise,

b2<address<e2 => file address=address+f2-b2

otherwise, the requested address is not legal. In some cases (for example, for programs with separated information and data space) the two segments for a file can overlap. If a question mark " ? " or a slash " / " is followed by an asterisk " * " only the second triple is used.

The initial setting of both mappings is suitable for normal **a.out** and **core** files. If either file is not of the kind expected, then, for that file, b1 is set to Ø, e1 is set to the maximum file size, and f1 is set to Ø; in this way the whole file can be examined with no address translation.

All appropriate values are kept as signed 32-bit integers so that **adb** can be used on large files.

FILES

/dev/mem
/dev/swap
a.out
core

SEE ALSO

ptrace(2), disasm(3), a.out(5), core(5)

DIAGNOSTICS

"Adb" when there is no current command or format. Comments about inaccessible files, syntax errors, abnormal termination of commands, etc. Exit status is Ø, unless last command failed or returned a non-zero status.

LIMITATIONS

A breakpoint set at the entry point is not effective on initial entry to the program.

When single stepping, system calls do not count as an

executed instruction.

Local variables whose names are the same as an external variable can cause errors in the accessing of the external.

A hexadecimal number specified for an address may be interpreted as a symbol if not preceded by a %.

NAME

admin - create and administer SCCS files

SYNOPSIS

admin [-**a**login]
[-**d**flag[flag-val]]
[-**e**login]
[-**f**flag[flag-val]]
[-**h**]
[-**i**[name]]
[-**m**[mrlist]]
[-**n**]
[-**r**rel]
[-**t**[name]]
[-**y**[comment]]
[-**z**]
files

DESCRIPTION

Admin is used to create new SCCS files and change parameters of existing ones. Arguments to **admin**, which may appear in any order, consist of keyletter arguments, which begin with -, and named files (note that SCCS file names must begin with the characters **s.**).

If a named file doesn't exist, it is created, and its parameters are initialized according to the specified keyletter arguments. Parameters not initialized by a keyletter argument are assigned a default value. If a named file does exist, parameters corresponding to specified keyletter arguments are changed, and other parameters are left as is.

If a directory is named, **admin** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored.

If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed since the effects of the arguments apply independently to each named file.

OPTIONS

-alogin
A login name, or numerical ZEUS group ID, to be added to the list of users which may make deltas (changes) to

the SCCS file. A group ID is equivalent to specifying all login names common to that group ID. Several **a** keyletters may be used on a single **admin** command line. As many logins, or numerical group ID, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas.

-dflag
Causes removal (deletion) of the specified flag from an SCCS file. The **-d** keyletter may be specified only when processing existing SCCS files. Several **-d** keyletters may be supplied on a single **admin** command. See the **-f** keyletter for allowable flag names.

-llist
A list of releases to be "unlocked". See the **-f** keyletter for a description of the **l** flag and the syntax of a list.

-elogin
A login name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all login names common to that group ID. Several **e** keyletters may be used on a single **admin** command line.

-fflag
This keyletter specifies a flag, and, possibly, a value for the flag, to be placed in the SCCS file. Several **f** keyletters may be supplied on a single **admin** command line. The allowable flags and their values are:

b Allows use of the **-b** keyletter on a **get**(1) command to create branch deltas.

cceil The highest release (i.e., "ceiling"), a number less than or equal to 9999, which may be retrieved by a **get** command for editing. The default value for an unspecified **c** flag is 9999.

dSID The default delta number (SID) to be used by a **get** command.

ffloor
The lowest release (i.e., "floor"), a number greater than Ø but less than 9999, which may be retrieved by a **get** command for editing. The default value for an unspecified **f** flag is 1.

i Causes the "No id keywords (ge6)" message issued by **get** or **delta**(1) to be treated as a fatal

error. In the absence of this flag, the message is only a warning. The message is issued if no SCCS identification keywords (see **get**(1)) are found in the text retrieved or stored in the SCCS file.

j Allows concurrent **get** commands for editing on the same SID of the SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.

llist A list of releases to which deltas can no longer be made (get -e against one of these "locked" release fails.). The list has the following syntax:

<list> ::= <range> | <list> , <range>
<range> ::= RELEASE NUMBER **| a**

The character **a** in the list is equivalent to specifying all releases for the named SCCS file.

mmod Module name of the SCCS file substituted for all occurrences of the %M% keyword in SCCS file text retrieved by **get**. If the **m** flag is not specified, the value assigned is the name of the SCCS file with the leading **s.** removed.

n Causes **delta** to create a "null" delta in each of those releases (if any) being skipped when a delta is made in a new release (e.g., in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas serve as "anchor points" so that branch deltas may later be created from them. The absence of this flag causes skipped releases to be non-existent in the SCCS file preventing branch deltas from being created from them in the future.

qtext User definable text substituted for all occurrences of the %Q% keyword in SCCS file text retrieved by **get**.

ttype Type of module in the SCCS file substituted for all occurrences of %Y% keyword in SCCS file text retrieved by **get**.

v[pgm]
Causes **delta** to prompt for Modification Request (MR) numbers as the reason for creating a delta. The optional value specifies the name of an MR number validity checking program (see **delta**(1)).

(If this flag is set when creating an SCCS file, the **m** keyletter must also be used even if its value is null).

-h Causes **admin** to check the structure of the SCCS file (see **sccsfile**(5)), and to compare a newly computed check-sum (the sum of all the characters in the SCCS file except those in the first line) with the check-sum that is stored in the first line of the SCCS file. Appropriate error diagnostics are produced.

This keyletter inhibits writing on the file, so that it nullifies the effect of any other keyletters supplied, and is, therefore, only meaningful when processing existing files.

-i[name]
The name of a file from which the text for a new SCCS file is to be taken. The text constitutes the first delta of the file (see **-r** keyletter for delta numbering scheme). If the **i** keyletter is used, but the file name is omitted, the text is obtained by reading the standard input until an end-of-file is encountered. If this keyletter is omitted, then the SCCS file is created empty. Only one SCCS file may be created by an **admin** command on which the **i** keyletter is supplied. Using a single **admin** to create two or more SCCS files require that they be created empty (no **-i** keyletter). Note that the **-i** keyletter implies the **-n** keyletter.

-m[mrlist]
The list of Modification Requests (MR) numbers is inserted into the SCCS file as the reason for creating the initial delta in a manner identical to **delta**. The **v** flag must be set and the MR numbers are validated if the **v** flag has a value (the name of an MR number validation program). Diagnostics will occur if the **v** flag is not set or MR validation fails.

-n This keyletter indicates that a new SCCS file is to be created.

-rrel
The release into which the initial delta is inserted. This keyletter may be used only if the **-i** keyletter is also used. If the **-r** keyletter is not used, the initial delta is inserted into release 1. The level of the initial delta is always 1 (by default initial deltas are named 1.1).

-t[name]
The name of a file from which descriptive text for the

SCCS file is to be taken. If the -t keyletter is used and **admin** is creating a new SCCS file (the **-n** and/or **-i** keyletters also used), the descriptive text file name must also be supplied. In the case of existing SCCS files: (1) a **-t** keyletter without a file name causes removal of descriptive text (if any) currently in the SCCS file, and (2) a **-t** keyletter with a file name causes text (if any) in the named file to replace the descriptive text (if any) currently in the SCCS file.

-y[comment]
The comment text is inserted into the SCCS file as a comment for the initial delta in a manner identical to that of **delta**. Omission of the **-y** keyletter results in a default comment line being inserted in the form:
date and time created YY/MM/DD HH:MM:SS by login
The **-y** keyletter is valid only if the **-i** and/or **-n** keyletters are specified (i.e., a new SCCS file is being created).

-z The SCCS file check-sum is recomputed and stored in the first line of the SCCS file (see **-h** , above).

Note: The use of this keyletter on a truly corrupted file may prevent future detection of the corruption.

FILES

The last component of all SCCS file names must be of the form **s.**file-name. New SCCS files are given mode 444 (see **chmod**(1)). Write permission in the pertinent directory is, of course, required to create a file. All writing done by admin is to a temporary x-file, called **x.**file-name, created with mode 444 if the **admin** command is creating a new SCCS file, or with the same mode as the SCCS file if it exists. After successful execution of **admin** , the SCCS file is removed (if it exists), and the x-file is renamed with the name of the SCCS file. This ensures that changes are made to the SCCS file only if no errors occurred.

It is recommended that directories containing SCCS files be mode 755 and that SCCS files themselves be mode 444. The mode of the directories allows only the owner to modify SCCS files contained in the directories. The mode of the SCCS files prevents any modification at all except by SCCS commands.

If it should be necessary to patch an SCCS file for any reason, the mode may be changed to 644 by the owner allowing use of **ed**(1). Care must be taken! The edited file should always be processed by an **admin -h** to check for corruption followed by an **admin -z** to generate a proper check-sum. Another **admin -h** is recommended to ensure the SCCS file is

valid.

Admin also makes use of a transient lock file (called **z.**file-name), which is used to prevent simultaneous updates to the SCCS file by different users. See **get**(1) for further information.

SEE ALSO

delta(1), ed(1), get(1), help(1), prs(1), what(1), sccsfile(5).
Source Code Control System User's Guide in the ZEUS Utilites Manual

DIAGNOSTICS

Use **help** (1) for explanations.

NAME

alias - substitute a word for a command or command string

SYNOPSIS

alias word command-string

DESCRIPTION

An **alias** is an abreviation for a longer command. The shell maintains a list of aliases which can be established, displayed, modified, and removed by the **alias** and **unalias** commands.

EXAMPLE

The command for extracting a name from the /etc/passwd file:

```
grep user.name /etc/passwd
```

can be aliased to the word "lookup" with the command:

```
alias lookup 'grep  \!* /etc/passwd'
```

the command:

```
lookup bill
```

is the same as the command:

```
grep bill /etc/passwd
```

The alias command can be used in any of 4 ways:

alias
prints all aliases.

alias name
prints the alias for name.

alias name command.string
assigns the specified command.string as the alias of name. Name is not allowed to be **alias** or **unalias**

unalias pattern
All aliases whose names match the specified pattern are discarded. Thus all aliases are removed by unalias *.

LIMITATIONS

Alias limits substititutions on a single line to 20; meta-characters must be escaped with a backslash "\".

SEE ALSO

csh(1C), set(1C).
The C Shell in the ZEUS Utilities Manual

NAME

apropos - locate commands by keyword lookup

SYNOPSIS

apropos word ...

DESCRIPTION

Apropos shows which manual sections contain instances of any of the given keywords in their title. Each word is considered separately and case of letters is ignored. Words which are part of other words are considered. Thus looking for compile will hit all instances of compiler also.

If the line starts `name (section) ...' you can do `man section name' to get the documentation for it. Try `apropos format' and then `man 5 core' to get the manual write-up on core.

EXAMPLES

```
% apropos password
getpass (3)              - read a password
getpwent(3) ...          - get password file entry
gpasswd (1)              - change group password
passwd (1)               - change login password
passwd (5)               - password file
pwck, grpck (1)          - password/group file checkers
```

FILES

/usr/lib/whatis data base

SEE ALSO

makewhatis(1), man(1), whatis(1)

NAME

ar - archive and library maintainer

SYNOPSIS

ar key [posname] afile name ...

DESCRIPTION

Ar maintains groups of files combined into a single archive file. It creates and updates library files as used by the loader.

OPTIONS

c Create. Normally **ar** creates afile when it needs to. The create option suppresses the normal message that is produced when afile is created.

l Local. Normally **ar** places its temporary files in the directory /tmp. This option causes them to be placed in the local directory.

v Verbose. **ar** gives a file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with t, it gives a long listing of all information about the files. When used with p, it precedes each file with a name.

KEYS

Key is one character from the set **dmpqrtx**, optionally concatenated with one or more of the options, abciluv. Afile is the archive file. The names are constituent files in the archive file. The meanings of the key characters are as follows:

d Delete the named files from the archive file.

m Move the named files to the end of the archive. If a positioning character is present, then the posname argument must be present and, as in r, specifies where the files are to be moved.

p Print the named files in the archive.

q Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive. Useful only to avoid quadratic behavior when creating a large archive piece-by-piece.

r Replace the named files in the archive file. If the optional character u is used with r, then only those files with modified dates later than the archive files

are replaced. If an optional positioning character from the set abi is used, then the posname argument must be present and specifies that new files are to be placed after (a) or before (b or i) posname. Otherwise, new files are placed at the end.

t Print a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.

x Extract the named files. If no names are given, all files in the archive are extracted. In neither case does x alter the archive file.

FILES

/tmp/v* temporaries

SEE ALSO

ld(1), ranlib(1), ar(5)

LIMITATIONS

If the same file is mentioned twice in an argument list, it may be put in the archive twice.

NAME

as - PLZ/ASM assembler

SYNOPSIS

as [option ...] file

DESCRIPTION

As assembles the named file.

OPTIONS

-f Allow assembly of floating point instructions.

-l Produce a listing containing object code and locations. For input file file**.s**, the listing is produced in file file**.l** in the current directory.

-o objfile
The output of the assembly is left on objfile. If this option is omitted, the output is left on the file a.out.

-p Produce a listing to standard output.

-u Treat all undefined references as externals.

-z Produce Zobj object format for MCZ compatible systems. When this option is specified, the default output file becomes t.out instead of a.out.

FILES

/lib/as2	pass 2 of the assembler
/lib/as2d	pass 2 data
/lib/asz2	pass 2 for Zobj output
/lib/asz2d	pass 2 data for Zobj output
/tmp/T_*H	temporary
/tmp/T_*I	temporary
a.out	object

SEE ALSO

cas(1), ld(1), nm(1), objdu(1), objhdr(1), a.out(5).
PLZ/ASM Assembler User Guide in the ZEUS Languages / Programming Tools Manual

DIAGNOSTICS

When syntactic or semantic errors occur, the offending line is printed followed by an error number. The errors are described in the user guide.

NAME

at - execute command or shell script file at a later time

SYNOPSIS

at time [day] file

DESCRIPTION

At makes a copy of the named shell script file and executes it at the specified time.

At checks the file to see if it is a C-Shell or Bourne-Shell script, inserts a **cd**(1) command (directing the shell to the proper file) and the appropriate shell variables (see **setenv** in **sh(1)**) and places a copy of the file in /usr/spool/at/yy.ddd.hhhh.uu.

At the specified time atrun checks the file to determine its shell type and invokes the appropriate shell which executes the **cd** command, sets the environment variables and executes the shell script commands.

When the file is run, it has the access privileges of its owner and group. The program /usr/lib/atrun insures that the file was placed on the spool by **at** and runs the program

The time is one to four digits, with an optional following A, P, N, or M, for AM, PM, noon or midnight. One and two digit numbers are taken to be hours, three and four digits to be hours and minutes. If no letters follow the digits, a 24 hour clock time is understood.

The optional day is either a month name followed by a day number, or a day of the week; if the word week follows, execution of the file is moved seven days further off. Names of months and days can be recognizably truncated.

The mode of the shell script file need not be marked executable.

At programs are executed by periodic execution of the command /usr/lib/atrun from **cron**(M). The frequency of **at** depends upon how often atrun is executed.

Standard output or error output is lost unless redirected.

EXAMPLES

Examples of legitimate commands are:

at 8:30am file

executes file at 8:30 on the current day

at 8am jan 24 file

executes file at 8:00 on jan 24

at 1530 fr week file

executes file at 3:30 a week from this friday

FILES

/usr/spool/at/yy.ddd.hhhh.uu copy of shell program
/usr/spool/at/lasttimedone time of last execution
/usr/spool/at/past dir of activities in progress
/usr/lib/atrun deamon which executes due files

SEE ALSO

calendar(1), cd(1), pwd(1), setenv -- in csh(1), cron(M).

DIAGNOSTICS

Complains about various syntax errors and times out of range.

LIMITATIONS

Due to the periodic execution of /usr/lib/atrun by cron(M), accurate scheduling of tightly synchronized shell scripts is not possible.

NAME

awk - pattern scanning and processing language

SYNOPSIS

awk [**-F** c] [prog] [file] ...
or
awk [**-F** c] [**-f** progfile] [file] ...

DESCRIPTION

Awk scans each input file for lines that match any of a set of patterns specified in prog. With each pattern in prog there can be an associated action that will be performed when a line of a file matches the pattern. The set of patterns may appear literally as prog, or in a file specified as **-f** file.

Files are read in order; if there are no files, the standard input is read. The file name `-' means the standard input. Each line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

An input line is made up of fields separated by white space. (This default can be changed by using FS, vide infra.) The fields are denoted $1, $2, ... ; $Ø refers to the entire line.

A pattern-action statement has the form

pattern { action }

A missing { action } means print the line; a missing pattern always matches.

An action is a sequence of statements. A statement can be one of the following:

```
if ( conditional ) statement [ else statement ]
while ( conditional ) statement
for ( expression ; conditional ; expression ) statement
break
continue
{ [ statement ] ... }
variable = expression
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
next # skip remaining patterns on this input line
exit # skip the rest of the input
```

Statements are terminated by semicolons, newlines or right braces. An empty expression-list stands for the whole line. Expressions take on string or numeric values as appropriate,

and are built using the operators +, -, *, /, %, and concatenation (indicated by a blank). The C operators ++, --, +=, -=, *=, /=, and %= are also available in expressions. Variables may be scalars, array elements (denoted x[i]) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. String constants are quoted "...".

The print statement prints its arguments on the standard output (or on a file if >file is present), separated by the current output field separator, and terminated by the output record separator. The printf statement formats its expression list according to the format (see **printf**(3)).

The built-in function length returns the length of its argument taken as a string, or of the whole line if no argument. There are also built-in functions exp, log, sqrt, and int. The last truncates its argument to an integer. substr(s, m, n) returns the n-character substring of s that begins at position m. The function sprintf(fmt, expr, expr, ...) formats the expressions according to the **printf**(3) format given by fmt and returns the resulting string.

Patterns are arbitrary Boolean combinations (!, ||, &&, and parentheses) of regular expressions and relational expressions. Regular expressions must be surrounded by slashes and are as in egrep. Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions.

A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second.

A relational expression is one of the following:

```
expression matchop regular-expression
expression relop expression
```

where a relop is any of the six relational operators in C, and a matchop is either ~ (for contains) or !~ (for does not contain). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special patterns BEGIN and END may be used to capture control before the first input line is read and after the last. BEGIN must be the first pattern, END the last.

A single character c may be used to separate the fields by starting the program with

```
BEGIN { FS = "c" }
```

or by using the **-F**c option.

Other variable names with special meanings include NF, the number of fields in the current record; NR, the ordinal number of the current record; FILENAME, the name of the current input file; OFS, the output field separator (default blank); ORS, the output record separator (default newline); and OFMT, the output format for numbers (default "%.6g").

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Add up first column, print sum and average:

```
      { s += $1 }
END   { print "sum is", s, " average is", s/NR }
```

Print fields in reverse order:

```
{ for (i = NF; i > 0; --i) print $i }
```

Print all lines between start/stop pairs:

```
/start/, /stop/
```

Print all lines whose first field is different from previous one:

```
$1 != prev { print; prev = $1 }
```

SEE ALSO

lex(1), sed(1).
Awk - a pattern scanning and processing language
In the ZEUS Utilities Manual.

LIMITATIONS

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate "" to it.

When a set of pattern-actions appears literally as prog, it is generally necessary to enclose it in single quotes to prevent interpretation of special characters by the shell.

For example:

```
awk '{print $2,$1}' test
```

will print the first two fields of each entry in test in reverse order.

```
awk {print $2,$1} test
```

or

```
awk "{print $2,$1}" test
```

will not.

NAME

banner - make posters

SYNOPSIS

banner strings

DESCRIPTION

Banner prints its arguments (each up to 10 characters long) in large letters on the standard output.

EXAMPLES

banner "hi there"
banner hello world
banner "happy" "birthday"

LIMITATIONS

In order to get any of the special symbols, or "hard" spaces, the symbol or space must be preceded by a backslash " \ ". Banner accepts only 10 characters.

NAME

basename, dirname - deliver portions of path names

SYNOPSIS

basename string [suffix]
dirname string

DESCRIPTION

Basename deletes any prefix ending in a slant " / " and the suffix, (if it is present in the string) from string, and prints the result on the standard output.

Basename can be used inside of command substitution marks (` `) in the body of shell scripts.

Dirname delivers all but the last level of the path name in string.

EXAMPLES

The command:

```
basename /usr/spool/file.c
```

returns the string:

```
file.c
```

the command:

```
basename /usr/spool/file.c .c
```

returns the string:

```
file
```

The command:

```
set NAME=`dirname /usr/src/cmd/cat.c`
```

sets the shell variable **NAME** to

```
/usr/src/cmd
```

SEE ALSO

csh(1), sh(1) and,
The C Shell in the ZEUS Utilities Manual

NAME

bc - arbitrary-precision arithmetic language

SYNOPSIS

bc [**-cl**] [file ...]

DESCRIPTION

Bc is an interactive translator for a language which resembles C but provides unlimited precision arithmetic. It receives input from any files given and then reads the standard input.

Comments

are enclosed in /* and */.

Names

simple variables: letters a-z
array elements: letter [expression]
The words 'ibase', 'obase', and 'scale'

Other operands

arbitrarily long numbers with optional sign and decimal point.
(expression)
sqrt (expression)
length (expression) number of significant decimal digits
scale (expression) number of digits right of decimal point
letter (expression , ... , expression)

Operators

```
+  -  *  /  %  ^
    (% is remainder; ^ is power)
++   --
    (prefix and postfix; apply to names)
==  <=  >=  !=  <  >
=  =+  =-  =*  =/  =%  =^
```

Statements

expression
{ statement ; ... ; statement }
if (expression) statement
while (expression) statement
for (expression ; expression ; expression) statement
null statement
break
quit

Function definitions

```
define letter ( letter ,..., letter ) {
     auto letter, ... , letter
```

```
        statement; ... statement
        return ( expression )
    }
```

Functions in -l math library

```
    s(x) sine
    c(x) cosine
    e(x) exponential
    l(x) log
    a(x) arctangent
    j(n,x)    Bessel function
```

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or newlines may separate statements. Assignment to scale influences the number of digits to be retained on arithmetic operations in the manner of **dc**(1). Assignments to ibase or obase set the input and output number radix respectively.

The same letter may be used as an array, a function, and a simple variable simultaneously. All variables are global to the program. 'Auto' variables are pushed down during function calls. When arrays are used as function arguments or defined as automatic variables, empty square brackets must follow the array name.

Bc is actually a preprocessor for **dc**(1). **Bc** automatically runs **dc** and opens a pipe to it.

OPTIONS

-c Only compiles, does not run **dc** and the **dc** input appears on **bc** 's standard output.

-l Defines a math function library.

EXAMPLES

```
scale = 20
define e(x){
     auto a, b, c, i, s
     a = 1
     b = 1
     s = 1
     for(i=1; 1==1; i++){
          a = a*x
          b = b*i
          c = a/b
          if(c == 0) return(s)
          s = s+c
     }
```

```
}
```

defines a function to compute an approximate value of the exponential function and

```
for(i=1; i<=10; i++) e(i)
```

prints approximate values of the exponential function of the first ten integers.

FILES

/usr/lib/lib.b mathematical library
dc(1) desk calculator proper

LIMITATIONS

No &&, ||, or ! operators.

For statement must have all three expression's.

Quit is interpreted when read, not when executed.

NAME

bdiff - the diff program for very large files

SYNOPSIS

bdiff file1 file2 [n] [**-s**]

DESCRIPTION

Bdiff is used in a manner analogous to **diff**(1) to find which lines must be changed in two files to bring them into agreement. Its purpose is to allow processing of files which are too large for **diff**.

Bdiff ignores lines common to the beginning of both files, splits the remainder of each file into n-line segments, and invokes **diff** upon corresponding segments. The value of n is 3500 by default. If the optional third argument is given, and it is numeric, it is used as the value for n. This is useful in those cases in which 3500-line segments are too large for **diff**, causing it to fail.

If file1 (file2) is -, the standard input is read.

The optional **-s** (silent) argument specifies that no diagnostics are to be printed by **bdiff** (note, however, that this does not suppress possible exclamations by **diff**. If both optional arguments are specified, they must appear in the order indicated above.

The output of **bdiff** is exactly that of **diff**, with line numbers adjusted to account for the segmenting of the files (that is, to make it look as if the files had been processed whole).

Note: because of the segmenting of the files, **bdiff** does not necessarily find a smallest sufficient set of file differences.

FILES

/tmp/bd?????

SEE ALSO

diff(1).

DIAGNOSTICS

Use **help**(1) for explanations.

NAME

break -- C-Shell flow control interrupt statement

SYNOPSIS

break

DESCRIPTION

Break causes execution to resume after the **end** of the nearest enclosing **foreach** or **while** loop. The remaining commands on the current line are executed. Multi-level breaks are thus possible by writing them all on one line.

The built-in command **continue** can be used to continue the loop prematurely.

EXAMPLE

```
# test.script
while(1)
   echo -n 'enter x: '
   set x = 'gets'
   if( $x == 'a') then
        break
   else
        echo 'it didn't break'
   endif
end
echo 'it broke'
```

SEE ALSO

foreach(1C), while(1C), breaksw(1C), end(1C), continue(1C) and
The C Shell in the ZEUS Utilities Manual

NAME

breaksw - C-Shell flow control interrupt statement

SYNOPSIS

breaksw

DESCRIPTION

Breaksw causes a break from a **switch**, resuming after the **endsw**. The command **breaksw** causes execution to continue after the **endsw**.

EXAMPLE

```
# test
foreach i ( * )
     switch ( $i )
          case ????
               echo "$i is a 4 character filename"
          breaksw

          case ?????
               echo "$i is a 5 character filename"
          breaksw

          case ??????
               echo "$i is a 6 character filename"
          breaksw

          default
               echo "$i is not 4, 5, or 6, characters"
     endswitch
end
```

SEE ALSO

foreach(1C), switch(1C), case(1C), echo(1C), default(1C), endswitch(1C), and end(1C).

The C Shell in the ZEUS Utilities Manual

NAME

cal - print calendar

SYNOPSIS

cal [month] year

DESCRIPTION

Cal prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. Year can be between 1 and 9999. The month is a number between 1 and 12. The calendar produced is that for England and her colonies.

EXAMPLES

INCORRECT	CORRECT
cal September 1752	cal 9 1752
cal '42	cal 1942

LIMITATIONS

The year is always considered to start in January even though this is historically naive.

Beware that "cal 78" refers to the early Christian era, not the 20th century.

NAME

calendar - reminder service

SYNOPSIS

calendar [-]

DESCRIPTION

Calendar consults the file "calendar" in the current directory and prints out lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates such as "Dec. 7," "december 7," "12/7," etc., are recognized, but not "7 December" or "7/12". On weekends, "tomorrow" extends through Monday.

When an argument is present, **calendar** does its job for every user who has a file **calendar** in their login directory and sends any positive results by **mail**(1). Normally this is done daily after midnight under control of **cron**(M).

EXAMPLE

The file **calendar** in the home directory can have the following lines:

```
3/3  meeting, 10:00 am conference room B
3/3  call hank re schedule
3/5  dinner with carol -- pm.
```

the command:

```
(cd; calendar)
```

will always execute the **calendar** file in the home directory, regardless of the current working directory.

FILES

/usr/lib/calprog to figure out today's and tomorrow's dates
/etc/passwd
/tmp/cal*
/usr/lib/crontab

SEE ALSO

egrep(1), sed(1), at(1), mail(1), cron(M)

LIMITATIONS

The calendar must be public information to get reminder service.
Calendar's extended idea of tomorrow does not account for holidays.

NAME

cas - invoke assembler

SYNOPSIS

cas [**-oud**] file

DESCRIPTION

Cas assembles the named file written in the assembly language described in the ZEUS Languages/Programming Tools Manual. It will not assemble files written in PLZ/ASM assembly language.

OPTIONS

-o objfile The output of the assembly is left on objfile. If this option is omitted, the output is left on the file a.out.

-u Treat all undefined references as externals.

-d Include internal labels in the a.out symbol table.

FILES

/bin/cas	assembler
/tmp/as*	temporary
a.out	object

SEE ALSO

ld(1), nm(1), objdu(1), strip(1), a.out(5).

ZEUS Languages/Programming Tools Manual.

DIAGNOSTICS

The assembler produces error messages to standard error if an error occurs during the assembly process. If errors prevent further assembly, the assembler aborts, closes all files, and prints a message to standard error. If the assembler is interrupted during assembly, the assembler aborts and closes all files.

LIMITATIONS

The following features are not implemented.

- Floating point numbers, constants and conversion operators (^F, ^FD, ^FS, .quad, .extend).
- Absolute sections and common sections (.asec and .csec).
- Listing facilities.
- Error recovery (currently the assembler terminates on most errors).
- Program sectioning (.psec) in nonsegmented mode.

NAME

cat - concatenate and print files

SYNOPSIS

cat [**-u**] [**-s**] file ...

DESCRIPTION

Cat reads each file in sequence and writes it on the standard output. The input file may not be the same as the output file unless it is a special (device) file.

OPTIONS

- Reads from the standard output; same as if no output file is given.

-s Makes **cat** silent about non-existent files.

-u Does not buffer the output in 512-byte blocks as usual.

EXAMPLES

```
cat file
```

prints the file, and:

```
cat file1 file2 > file3
```

concatenates the first two files and places the result in the third file. The third file is created if it does not exist.

SEE ALSO

cp(1), pr(1).

NAME

cb - C program beautifier

SYNOPSIS

cb < file.c

DESCRIPTION

Cb reformats a C program file, providing the spacing and indentation to display the structure of the program.

EXAMPLE

The raw file named test.c containing the line:

```
main() { printf("hello, world\n"); }
```

when reformatted with the command:

```
cb < test.c
```

produces the output:

```
main() {
     printf("hello, world\n");
}
```

NAME

cc - S8000 C compiler

SYNOPSIS

cc [option] file

DESCRIPTION

Cc is the portable C compiler modified to produce Z8000 code. Depending on the options, a single **cc** call can compile; compile and assemble; compile, assemble, and link; or do any of these combinations with an optional global optimization pass.

The **cc** compiler provides an unsigned char data type, initialized bit fields, the ZEUS version 7 features of C (structure assignments and enumeration types) and the new ZEUS System III additions to the C language, the "void" data type and unique identification of names of structure and union members.

The default is to compile, assemble, and link, using internal calls to the Z8000 assembler, **cas**, and S8000 linker, **ld**. File names ending in .c are taken to be C source files to be compiled. The **-Ol** option causes the optional global optimization pass to be invoked in order that loop optimization be applied to the code. The **-Or** option invokes global optimization so that both loop optimization and register allocation be applied to the code. The assembly language code produced by the compiler can be (peephole) optimized with the **-O** option before being passed to the assembler. (The **-Ol** and **-Or** options also invoke the peephole optimizer.) The **-S** option saves the Z8000 assembly language code in .s files and suppresses further processing. By default, the code is assembled and then passed to the linker. The **-c** option saves the assembled code in .o files and suppresses further processing. By default, the linker then links the code to produce an executable Z8000 program.

File names ending with .s are taken to be Z8000 assembly language. By default, the .s files are assembled to produce .o files and then linked. The compilation step is skipped with .s files.

Other file names are taken to be names of C-compatible object programs (typically produced by an earlier **cc** run), or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are linked (in the order given) to produce an executable load module with name a.out.

Options on the **cc** call can be for **cc** or for ld.

OPTIONS

-c Compile and assemble the named C source files but suppress the linking step. Force an object file .o to be produced even if only one source file is compiled. If a number of C source files are specified, the .o files are saved.

-Dname
-Dname=def
Define name to the preprocessor, as if by #define. If no definition is given, name is defined as 1.

-E Run only the macro preprocessor and send the result to the standard output.

-Idir Bring in a directory of #include files. Names that do not begin with / are always sought first in the same directory as the source file, then in directories named in **-I** options, then in directories on a standard list.

-Ol Invoke the C global optimizer to apply loop optimization.

-Or Invoke the C global optimizer to apply loop optimization and register allocation.

-O Invoke the C peephole optimizer for Z8000 code.

-p Arrange for the compiler to produce code that counts the number of times each routine is called; also, if loading takes place, replace the standard startup routine by one which automatically calls **monitor**(3) at the start and arranges to write out a mon.out file at normal termination of execution of the object program. An execution profile can then be generated by use of **prof**(1).

-P Run only the macro preprocessor and place the result for each .c file in a corresponding .i file with no # lines in it.

-S[l] Compile the named C source files but suppress the assembly and link step. Leave the assembly language code on corresponding files named .s. If l is specified, make the original C source lines appear as assembly language commments preceding the code produced for them.

-Uname Remove any initial definition of name.

Other options can be specified on the **cc** call and are passed to the linker ld. No options are passed to **as** from the **cc** call, but any internal call to the assembler uses the **-u** and **-f** options and the **-o** option with a name consisting of the original name and .o. The internal call to the linker specifies the options **-X** and **-e** with entry name start, and adds the library name /lib/libc.a to the end of the list of object module names. See the description of **ld**.

FILES

file.c	source file
file.o	object file
a.out	load module
/tmp/cc.?	temporaries
/lib/cpp	preprocessor
/lib/cparse	compiler pass1
/lib/gopt	optional global optimizer
/lib/codgen	compiler pass3
/lib/popt	optional peephole optimizer
/lib/clist	optional listing pass
/lib/libc.a	standard library
/lib/mcrt0.o	optional startup routine for profiling

SEE ALSO

as(1), ld(1).

The C Programming Language (by B. W. Kernighan and D. M. Ritchie, Prentice-Hall, 1978),

C in the ZEUS Languages/Programming Tools Manual.

DIAGNOSTICS

The diagnostics produced by the compiler, assembler, or linker are self-explanatory.

IMPLEMENTATION

The **cc** compiler has the following characteristics:

- As many as seven register declarations can be honored. The Z8000 registers r8 through r14 can be used for register variables.
- The **cc** compiler produces object code that conforms to the S8000 calling conventions.

NAME

cd - change working directory

SYNOPSIS

cd directory

DESCRIPTION

Directory becomes the new working directory. The process must have execute (search) permission in directory.

Cd is recognized and executed by the shell. A new process is created to execute each command, and **cd** would be ineffective if it were written as a normal command.

SEE ALSO

csh(1), pwd(1), sh(1), chdir(2).
The C Shell in the ZEUS Utilities Manual

NAME

cdc - change the delta commentary of an SCCS delta

SYNOPSIS

cdc -r*SID* [**-m**[*mrlist*]] [**-y**[*comment*]] *files*

DESCRIPTION

Cdc changes the *delta commentary*, for the SID specified by the **-r** keyletter, of each named SCCS file.

Delta commentary is defined to be the Modification Request (**MR**) and comment information normally specified via the **delta**(1) command (**-m** and **-y** keyletters).

If a directory is named, **cdc** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of - is given, the standard input is read (see **WARNINGS**); each line of the standard input is taken to be the name of an SCCS file to be processed.

Arguments to **cdc**, which may appear in any order, consist of *keyletter* arguments, and file names.

All the described *keyletter* arguments apply independently to each named file:

-m[*mrlist*] If the SCCS file has the **v** flag set (see **admin**(1)) then a list of **MR** numbers to be added and/or deleted in the delta commentary of the *SID* specified by the **-r** keyletter *may* be supplied. A null **MR** list has no effect.

MR entries are added to the list of **MR** in the same manner as that of **delta**(1). In order to delete an **MR**, precede the MR number with the character **!** (see **EXAMPLES**). If the **MR** to be deleted is currently in the list of **MR** s, it is removed and changed into a "comment" line. A list of all deleted **MR** s is placed in the comment section of the delta commentary and preceded by a comment line stating that they were deleted.

If **-m** is not used and the standard input is a terminal, the prompt **MRs?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The **MRs?** prompt always precedes the **comments?** prompt (see **-y** keyletter).

MR s in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the

MR list.

Note that if the **v** flag has a value (see **admin** (1)), it is taken to be the name of a program (or shell procedure) which validates the correctness of the **MR** numbers. If a non-zero exit status is returned from the MR number validation program, **cdc** terminates and the delta commentary remains unchanged.

-r SID Used to specify the SCCS ID entification (SID) string of a delta for which the delta commentary is to be changed.

-y [comment] Arbitrary text used to replace the comment (s) already existing for the delta specified by the **-r** keyletter. The previous comments are kept and preceded by a comment line stating that they were changed. A null comment has no effect.

If **-y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read. If the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.

The exact permissions necessary to modify the SCCS file are documented in the Source Code Control System User's Guide. Simply stated, they are either (1) if you made the delta, you can change its delta commentary; or (2) if you own the file and directory you can modify the delta commentary.

EXAMPLES

```
cdc -r1 .6 -m"bl78-12345 \!bl77-54321 bl79-00001"
-ytrouble s.file
```

adds bl78-12345 and bl79-00001 to the **MR** list, removes bl77-54321 from the **MR** list, and adds the comment **trouble** to delta 1 . 6 of s . file.

```
cdc -r1 . 6 s.file
MRs? !bl77-54321 bl78-12345 bl79-00001
comments? trouble
```

does the same thing.

WARNINGS

If SCCS file names are supplied to the **cdc** command via the standard input (- on the command line), then the **-m** and **-y** keyletters must also be used.

FILES

x-file (see **delta**(1))

z-file (see **delta**(1))

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(5).
Source Code Control System User's Guide in the ZEUS Utilities Manual

DIAGNOSTICS

Use **help**(1) for explanations.

NAME

chgrp - change group

SYNOPSIS

chgrp group file ...

DESCRIPTION

Chgrp changes the group-ID of the files to group. The group may be either a decimal group-ID number or a group name found in the group-ID file.

FILES

/etc/chgrp
/etc/passwd
/etc/group

SEE ALSO

chown(1), chown(2), passwd(5), group(5), chmog(M)

100 * (original - combined) / original

It is recommended that before any SCCS files are actually combined, this option should be used to determine exactly how much space is saved by the combining process.

If no keyletter arguments are specified, **comb** preserves only leaf deltas and the minimal number of ancestors needed to preserve the tree.

FILES

s.COMB The name of the reconstructed SCCS file.

comb????? Temporary.

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(5). Source Code Control System User's Guide in the ZEUS Utilities Manual

DIAGNOSTICS

Use **help**(1) for explanations.

LIMITATIONS

Comb can rearrange the shape of the tree of deltas. It may not save any space; in fact, it is possible for the reconstructed file to actually be larger than the original.

NAME

chgrp - change group

SYNOPSIS

chgrp group file ...

DESCRIPTION

Chgrp changes the group-ID of the files to group. The group may be either a decimal group-ID number or a group name found in the group-ID file.

FILES

/etc/chgrp
/etc/passwd
/etc/group

SEE ALSO

chown(1), chown(2), passwd(5), group(5), chmog(M)

NAME

chkdiff - list differences between versions of a source file

SYNOPSIS

chkdiff [**-h**] [**-v** *rel.lev*] [**-v** *rel.lev*] *file*

DESCRIPTION

Chkdiff lists the differences between a source code file and another version of it in its source control file; chkdiff can also list the differences between two versions in the source control file. The differences are described in the form used by **diff**(1).

The filename argument must be the name of a source file, not its control file. By default, the differences listed are those between the source file and the last version in the control file.

OPTIONS

-h Invokes the "halfhearted" version of **diff**(1).

-v Lists the differences between the source file and the specified version if used once. If the -v option is used twice, the differences listed are those between the two specified versions. Each -v option is followed by an argument of the form *rel.lev*, where *rel* is the release number and *lev* is the release level number.

FILES

file+ -- control file for *file*
/tmp/chkdiffXXX -- temporarily holds one of the versions

SEE ALSO

chkin(1), chkout(1), chkwhat(1), zsc(5), diff(1)

DIAGNOSTICS

corrupt chkfile: the convention specified in zsc(5) has been violated

diff: files too big, try -h: version differences cannot be calculated because the file is too big

version rel.lev not found: use chkout -h to find correct version numbers

LIMITATIONS

If the files are too big for the differences to be calculated, an version entry is made showing *no* differences.

NAME

chkin - check in file to Zilog Source Control file

SYNOPSIS

chkin [-r] [-b] [-c comment] [-d dir] file ...

DESCRIPTION

Chkin checks in a source file to its Zilog Source Control file (see **zsc**(5)). If the control file does not exist, it is created, and the entire contents of the source file entered. If the source file does exist, the differences between the source file and the last version in the control file are found using **diff**(1) and these differences entered.

If the control file exists, chkin looks for a lock file. The existence of the lock file indicates that the source file was previously checked out for editing by **chkout**(1). If the lock file is missing, chkin will not check in the file.

The filename argument can be either the name of the source file (must not end with +) or the control file name (the local name of the source file ending with a +). If the control file is specified and the -d option is not used, the source file is assumed to be in the working directory even if the control file is not.

By default, chkin assigns the new version the same major version number as the last version and a minor version number one greater than the last version's. Also by default, chkin prompts for comments and reads its standard input until an EOF (control-d on terminal input). This input is inserted in the control file as comment lines (see **zsc**(5)). Each comment line (which is terminated by a carriage return) is limited to 256 characters.

Note that the interrupt key typed after the comment lines have been entered can cause an inconsistent control file. After the new version entry is added to the control file, chkin removes (if it has one) its lock file. The source file is replaced by a checked-out read-only version of itself: see **chkout**(1).

OPTIONS

-b Bump the release number: the major version number for this version is one greater than that of the previous version and the minor version number is 1.

-c comment
Insert comment as a comment line enclosed in double quotes; don't prompt for a comment; ignore the standard input.

-d *dir*
Get the source from directory *dir* instead of the working directory.

-r The source file is just removed and not replaced by a read-only file.

FILES

file+	-- control file for *file*
file-	-- lock file for *file*
/tmp/chkinXXXXXX	-- latest version restored here
chkinXXXXXX	-- temporary for keyword substitution
/bin/diff	-- program which derives version differences

SEE ALSO
chkout(1), chkdiff(1), chkwhat(1), zsc(5)

DIAGNOSTICS
no lock file for xxxxx: either the lock file has been removed or you are trying to checkin a new source file to an existing control file

corrupt chkfile: the convention specified in zsc(5) has been violated

LIMITATIONS
It is possible to fool **chkin** into accepting a file that was not previously checked out. This is done at your own risk.

NAME

chkout - check out file from Zilog Source Control file

SYNOPSIS

chkout [**-d** dir] [-e] [-h] [-p] [**-v** rel.lev] file ...

DESCRIPTION

Chkout reconstructs (checks out) any version of a source file using the information contained in a Zilog Source Control file (see **zsc**(5)). For each file checked out, the version number and number of lines in the file are listed.

The filename argument can be either the name of the source file (which cannot end with a +) or the name of the control file (which is the local name of the source file with a + appended). If the control file is specified and the -d option is not used, the source file is created in the working directory, even if the control file is elsewhere.

Used without options, chkout checks out the last version as a read-only file: the source file has no write permission and has keywords substituted. (Keywords are described in **zsc**(5).) Use read-only files for listing, compilation, or other program input. A read-only source file must not be modified or checked back in to the control file.

OPTIONS

-d dir
Create the source file in the directory dir instead of the working directory.

-e Check out the version as an editable file: the source file has the usual mode, the keywords are left alone, and a lock file is created. The lock file prevents additional checkouts for this file; it must be present if the edited editable file is to be checked back in. See **chkin**(1).

-h For each version in the control file, list version number, date checked in, comments, and who checked in this version. No version is checked out.

-p List the version on the standard output. Substitute keywords.

-v rel.lev
Check out the specified version instead of the last version.

FILES

file+ -- control file for file
file- -- lock file for (editable) file

SEE ALSO

chkin(1), chkdiff(1), chkwhat(1), zsc(5)

DIAGNOSTICS

xxxxx checked out by xxxxx at xxxxx: the file is checked out for editing; it can not be checked out again until the new version is checked back in or the editing copy and its lock file are removed.

writable xxxxx exists: checking out the file would overwrite a file which is not apparently a checked out file

corrupt chkfile: the convention specified in zsc(5) has been violated

version rel.lev not found: use chkout -h to find correct version numbers

LIMITATIONS

No editor checks for the presence of a lock file, so be careful not to edit read-only files: your mistake will not become apparent until the editor refuses to overwrite the file.

NAME

chkwhat - print Zilog Source Control what strings

SYNOPSIS

chkwhat [-w] file ...

DESCRIPTION

Chkwhat searches the specified files for "what strings" and lists the identifying portion of these strings. If the -w option is used the entire "what string" will be printed. This option can be helpful when placing "what strings" in an archive. The file need not be a text file.

A "what string" shows which version of a source file the specified file is or is associated with. It is defined as beginning with the four-character sequence @[$] and ending with a null, newline, ", or '. Chkwhat lists the string from after the @[$] to before the terminating character.

SEE ALSO

zsc(5), chkin(1), chkout(1)

LIMITATIONS

The "what string" was presumably created when the file or the source code it was generated from was properly checked out of a Zilog Source Control file, but there is no guarantee of this.

NAME

chmod - change mode

SYNOPSIS

chmod mode file ...

DESCRIPTION

The file protection mode controls the read, write, and execute permissions for the owner of a file, the owner's group, and other users. The file protection mode is changed according to mode, which can be an absolute number, or a symbolic set of letters.

The protection mode of a file is shown with the **ls -l** command as in the example below:

```
-rwxrw-r-- 1 owner group  2268 Mar  3 12:42 filename
```

The mode portion of the command usually takes the form of a 3-digit number. The first digit controls the permission bits for the owner of the file, the second digit controls the permission bits for the members of the same group, and the third digit controls the permissions for everyone else.

A leading fourth digit controls special access codes to set a new user or group identification on execution of the file.

1000	tricky bit (chmod (2))
2000	set group ID on execution
4000	set user ID on execution

The mode is a three-digit number constructed from the following numbers:

Number:	Bits:	Meaning:
0	---	no permissions
1	--x	execute (search in directory) only
2	-w-	write only
3	-wx	write and execute (search)
4	r--	read only
5	r-x	read and execute (search)
6	rw-	read and write
7	rwx	read, write and execute (search)

Thus the command:

chmod 750 file

changes the protection mode of file such that the owner has

read, write, and execute permission; members of the owner's group have read and execute permission, and all others are excluded from any access to the file.

The file permission bits will look like the following:

```
-rwxr-x--- 1 owner group 2268 Mar  3 12:42 filename
```

Note that the first character in the string refers to the nature of the file ("-" if it is a regular file, "d" if it is a directory file and "p" if it is a named pipe. For special device files, a "c" refers to a "character" file, and "b" refers a "block" file).

The first set of 3 bits refers to the permissions of the owner, the second set of 3 bits refers to the permissions of those in the owner's group, and the last set of 3 bits refers to the permissions for everyone else.

```
File  User  Group Others
file   7     5     Ø
-     rwx   r-x   ---    1 owner group 2268 Mar  3
12:42 filename
```

A command using the symbolic mode has the form:

chmod who operator permission file

Where who is one or more of the following letters:

Letter:	who:	Bits affected:
u	user	-rwx------
g	group	----rwx---
o	others	-------rwx
a	all	-rwxrwxrwx

If who is omitted, the default is all but the setting of the file creation mask (umask(2)) is taken into account.

The operator can be any of the following:

```
+    to add permission to the file's mode,
-    to take away permission
=    to assign permission absolutely.
```

All other bits are reset.

Permission is any combination of the letters

Letter:	Bit:	Meaning:
r	r--	read
w	-w-	write
x	--x	execute
s		set owner or group ID
t		save text - sticky bit

Only the owner of a file (or the super-user) can change its mode.

EXAMPLES

The command:

chmod 740 filename

produces the following file protections:

```
-rwxr----- 1 owner group 2268 Mar  3 12:42 filename
```

The owner can read, write on, and execute the file; members of the owner's group can only read the file, and everyone else has no access at all.

The command:

chmod 551 filename

produces the following file protections:

```
-r-xr-x--x 1 owner group 2268 Mar  3 12:42 filename
```

The owner has read and execute permission, members of the owner's group also have read and execute permission, while everyone else is restricted to execute permission.

The command:

chmod 765 filename

produces the following file protections:

```
-rwxrw-r-x 1 owner group 2268 Mar  3 12:42 filename
```

The owner can read, write on, and execute the file, members of the owner's group can read and write on the file (but they cannot execute it), and all others can read the the file and execute it, but they cannot write on it.

SEE ALSO

ls(1), umask(1), chown(1), chmod(2), stat(2), umask(2), chmog(M), chown(M).

NAME

chown - change the owner-name of a file

SYNOPSIS

chown owner file

DESCRIPTION

Chown changes the owner of the files to owner. The owner may be either a decimal user-ID number or a login name found in the /etc/passwd file.

EXAMPLES

The command:

chown bill test.c

changes the ownership of the file test.c to bill (assuming bill is a valid user-name in the /etc/passwd file).

FILES

/etc/chown
/etc/passwd

SEE ALSO

chown(2), chgrp(1), passwd(5), group(5), chmog(M).

LIMITATIONS

Only the owner or the super-user should be able to change the ownership of a file.

NAME

cmp - compare two files

SYNOPSIS

cmp [**-l**] [**-s**] file1 file2

DESCRIPTION

The two files are compared. If file1 is a minus sign "-", the standard input is used. Under default options, **cmp** makes no comment if the files are the same; if they differ, it announces the byte and line number at which the difference occurred. If one file is an initial subsequence of the other, that fact is noted.

OPTIONS

-l Print the byte number (decimal) and the differing bytes (octal) for each difference.

-s Print nothing for differing files; return codes only.

SEE ALSO

diff(1), comm(1)

DIAGNOSTICS

Exit code Ø is returned for identical files, 1 for different files, and 2 for an inaccessible or missing argument.

NAME

code - print characters with their octal equivalents.

SYNOPSIS

code [< file]

DESCRIPTION

Code reads from standard input or from a file (with the "less than" redirect symbol) and lists each character with its hex equivalent on the standard output.

Code runs in `raw mode' and terminates on the DEL character (hex 7F).

EXAMPLES

Without argument:

```
code
a       61
b       62
c       63
<del>   7f
```

Input from file

```
code < file
a       61
b       62
c       63
```

NAME

col - nroff post-processing filter for printer output

SYNOPSIS

col [**-bfx**] [< file]
cat file | **col**

DESCRIPTION

This filter removes halfline and reverse-line motions generated by **nroff**(1).

To display equations, tables, or multi-column formats on a device that lacks reverse-line capability, such as a line printer or video terminal, run **nroff** without the -T option and filter the output with **col.**

Col is also used to optimize output to a TTY37. The -f option should be used in this case.

These are **col**'s primary features:

- **Col** transforms input containing the TTY37 sequences for reverse-line (ESC-7), reverse-halfline (ESC-8), and forward-halfline (ESC-9) to a form suitable for a device without these capabilities. It does this by overlaying the text lines on an internal buffer the same way the TTY37 overlays the physical lines on the output paper. VT (^K) is also assumed to mean reverse-line.

- Where possible, **col** changes blanks to equivalent tabs. Tab stops are assumed for the ninth columns and every 8 columns thereafter.

- The TTY37 uses SI (^N) and SO (^O) to shift to and from its Greek character set. **Nroff** generates a SI and a SO for every Greek letter. **Col** eliminates the unnecessary shift-out-shift-in sequences. **Col** can be used to optimize any use of SI and SO to indicate an alternate character set.

- **Col** eliminates all control characters except for ESC (escape, hexadecimal 1b) when followed by a 7, 8, or 9 character,

  SP (space, 2Ø),
  BS (backspace, Ø8),
  HT (tab, Ø9),
  CR (return, Ød),
  NL (newline, Øa),
  SI (shift in, Øf),
  SO (shift out, Øe),

OPTIONS

-b Generate output suitable for a device that cannot backspace. In a series of overstruck characters, only the last is output.

-f Eliminate all reverse motion but permit halfline-forward (ESC-9) sequences. Useful to optimize output to TTY37.

-x Do not generate new tab characters.

SEE ALSO

troff(1), tbl(1), eqn(1), ascii(7)

LIMITATIONS

Can't back up more than 128 lines.

Permits no more than 800 characters, including backspaces, on a line.

NAME

comb - combine SCCS deltas

SYNOPSIS

comb [-c*list* **-o** -p*SID* **-s**] *files*

DESCRIPTION

Comb generates a shell procedure (see **sh**(1)) which reconstructs the given SCCS files. The reconstructed files will be smaller than the original files. The arguments can be specified in any order, but all keyletter arguments apply to all named SCCS files.

If a directory is named, **comb** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are ignored.

If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are ignored.

The generated shell procedure is written on the standard output.

OPTIONS

The keyletter arguments are as follows. Each is explained as if only one named file is to be processed, but the effects of any keyletter argument apply independently to each named file.

-c*list* A *list* (see **get**(1) for the syntax of a *list*) of deltas to be preserved. All other deltas are discarded.

-o Causes the reconstructed file to be accessed at the release of the created delta for each **get -e** generated; otherwise, the reconstructed file is accessed at the most recent ancestor. Use of the **-o** keyletter can decrease the size of the reconstructed SCCS file. It can also alter the shape of the delta tree of the original file.

-p*SID* The *SCCS IDentification* string (*SID*) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file.

-s Causes **comb** to generate a shell procedure which produces a report for each file giving: the file name, size (in blocks) after combining, original size, and percentage change computed by:

100 * (original - combined) / original

It is recommended that before any SCCS files are actually combined, this option should be used to determine exactly how much space is saved by the combining process.

If no keyletter arguments are specified, **comb** preserves only leaf deltas and the minimal number of ancestors needed to preserve the tree.

FILES

s.COMB The name of the reconstructed SCCS file.

comb????? Temporary.

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(5). Source Code Control System User's Guide in the ZEUS Utilities Manual

DIAGNOSTICS

Use **help**(1) for explanations.

LIMITATIONS

Comb can rearrange the shape of the tree of deltas. It may not save any space; in fact, it is possible for the reconstructed file to actually be larger than the original.

NAME

comm - select or reject lines common to two sorted files

SYNOPSIS

comm [- 123] file1 file2

DESCRIPTION

Comm reads file1 and file2, (which must be ordered in ASCII collating sequence, see **sort**(1)) and produces three-column output.

column 1 contains those lines only in file1;
column 2 contains lines only in file2;
column 3 contains lines in both files.

The minus sign "-" means the standard input.

Flags 1, 2, or 3 suppress printing of the corresponding column. Thus the command:

comm -12 file1 file2

prints only the lines in column 3 -- the lines common to the two files;

The command:

comm -23 file1 file2

prints only lines in column 1 -- the lines only in the first file

The command:

comm -123 file1 file2

is not valid.

EXAMPLES

The results for the examples were obtained using the following two lists.

```
% cat templ
a
b
c
d
e

% cat temp2
c
d
```

```
e
f
g
h

% comm temp1 temp2
a
b
            c
            d
            e
        f
        g
        h

% comm -23 temp1 temp2
a
b

% comm -13 temp1 temp2
f
g
h

% comm -12 temp1 temp2
c
d
e
```

SEE ALSO

cmp(1), diff(1), sort(1), uniq(1).

NAME

continue - C Shell flow control statement

SYNOPSIS

continue

DESCRIPTION

Continue execution of the nearest enclosing **while** or **foreach.** The rest of the commands on the current line are executed.

EXAMPLE

```
# test
while ( 1 )
     echo -n "enter x:"
     set x=`gets`

     if ( $x == 'a' ) then
          echo "it continued"
          continue
     endif

     echo "it didnt continue"
     exit
end
```

This shell script prompts for input and sets the variable x to whatever is input at the terminal. If the input is **"a"** the **continue** statement moves control of the program back to the **while** statement at the top and the process repeats.

If the input is not the letter **"a"** control drops through the loop and **exits.**

SEE ALSO

foreach(1C), while(1C).
The C Shell in the ZEUS Utilities Manual

NAME

cp - copy a file into another or into a directory

SYNOPSIS

cp file1 file2
cp file directory

DESCRIPTION

file1 is copied onto file2. The mode and owner of file2 are preserved if it already existed; the mode of the source file is used otherwise.

In the second form, one or more files are copied into the directory with their original file names.

Cp refuses to copy a file onto itself.

SEE ALSO

cat(1), pr(1), mv(1).

NAME

cpio - copy file archives in and out

SYNOPSIS

cpio -o [**acBv**]

cpio -i [**Bdmrtuvs6**] [patterns]

cpio -p [**adlmuv**] directory

DESCRIPTION

Cpio -o (copy out) reads the standard input to obtain a list of path names and copies those files onto the standard output together with path name and status information.

Cpio -i (copy in) extracts from the standard input (which is assumed to be the product of a previous **cpio -o**) the names of files selected by zero or more patterns given in the name-generating notation of **sh**(1). In patterns, meta-characters **?**, *****, and **[...]** match the slash **/** character. The default for patterns is ***** (i.e., select all files).

Cpio -p (pass) copies out and in in a single operation. Destination path names are interpreted relative to the named directory.

OPTIONS

a Reset access times of input files after they have been copied.

B Input/output is to be blocked 5,12Ø bytes to the record (does not apply to the pass option; meaningful only with data directed to or from tape devices).

c Write header information in ASCII character form for portability.

d Directories are to be created as needed.

l Whenever possible, link files rather than copying them. Usable only with the **-p** option.

m Retain previous file modification time. This option is ineffective on directories that are being copied.

r Interactively rename files. If the user types a null

line, the file is skipped.

s This option swaps the bytes of a words as they are read.

t Print a table of contents of the input. No files are created.

u Copy unconditionally (normally, an older file will not replace a newer file with the same name).

v Verbose: causes a list of file names to be printed. When used with the **t** option, the table of contents looks like the output of an **ls -l** command (see **ls**(1)).

6 Process an old (i.e., UNIX Sixth Edition format) file. Only useful with **-i** (copy in).

EXAMPLES

The first example below copies the contents of a directory into an archive; the second duplicates a directory hierarchy:

```
ls | cpio -o >/dev/ctØ

cd olddir
find . -print | cpio -pdl newdir
```

The command:

```
``find . -print | cpio -oB >/dev/rctØ''
```

can be handled more efficiently by:

```
find . -exec cpio -oB > /dev/rctØ
```

SEE ALSO

ar(1), find(1), cpio(5).

LIMITATIONS

Path names are restricted to 128 characters. If there are too many unique linked files, the program runs out of memory to keep track of them and, thereafter, linking information is lost. Only the super-user can copy special files. Archive files created with the 'co' option can't be read back with the 'ci' option.

NAME

cref - make cross-reference listing

SYNOPSIS

cref [-ilnostux123] files

DESCRIPTION

Cref makes a cross-reference listing of C programs; files are searched for symbols in the appropriate syntax.

The output report is in four columns:

1. symbol;
2. file name;
3. see below;
4. text as it appears in the file.

Cref uses either an ignore file or an only file. Ignore and only files are lists of symbols separated by new-lines. All symbols in an ignore file are ignored in columns 1 and 3 of the output. If an only file is given, only symbols in that file will appear in column 1. Only one of these options may be given. C keywords are ignored.

In C the current symbol is the current function name. This file is created and is not removed at the end of the process.

OPTIONS

i The next argument is taken to be an ignore file (see FILES below).

l Put line number in column 3 (instead of current symbol).

n Omit column 4 (no context).

o The next argument is taken to be an only file.

s Current symbol in column 3 (default).

t Causes the next available argument to be used as the name of the intermediate file (instead of the temporary file **/tmp/crt??**).

u print only symbols that occur exactly once.

x Print only C external symbols.

1 Sort output on column 1 (default).

2 Sort output on column 2.

3 Sort output on column 3.

FILES

/tmp/crt?? temporaries
/usr/lib/cref/cign
default C ignore file
/usr/lib/cref/ctab
grammar table for C files
/usr/lib/cref/crpost
post-processor
/usr/lib/cref/upost
post-processor for **-u** option

SEE ALSO

cc(1), sort(1),

LIMITATIONS

Cref inserts an ASCII **DEL** character into the intermediate file after the eighth character of each name that is eight or more characters long in the source file.

NAME

crypt - encode/decode

SYNOPSIS

crypt [password] < in.file > out.file
crypt [password] < out.file > in.file

DESCRIPTION

Crypt reads from files (or from the standard input) and writes to the standard output (or an output file). The password is a key that determines the particular transformation. If no password is given, **crypt** demands a key from the terminal and turns off printing (echoing the characters on the terminal) while the key is being typed in. **Crypt** encrypts and decrypts with the same key; thus the command:

crypt key < file > encrypted

encrypts file with the password key and puts the encrypted output in the file encrypted. The command:

crypt key < encrypted | pr

decodes the encrypted file with the same password key and prints the file.

Files encrypted by **crypt** are compatible with those treated by the editor **ed** in encryption mode.

The security of encrypted files depends on three factors: the fundamental method must be hard to solve; direct search of the key space must be infeasible; and "sneak paths" by which keys or cleartext can become visible must be minimized.

Crypt implements a one-rotor machine designed along the lines of the German Enigma, but with a 256-element rotor. Debug methods on such machines require a large amount of work.

The transformation of a key into the internal settings of the machine is deliberately designed to take a substantial fraction of a second to compute. However, if keys are restricted to three lowercase letters, for example, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

Since the key is an argument to the **crypt** command, it is potentially visible to **ps**(1) or a derivative. To minimize this possibility, **crypt** destroys any record of the key immediately upon entry. The choice of keys and key security is the most vulnerable aspect of **crypt**.

FILES

/dev/tty for typed key

SEE ALSO

ed(1), makekey(M).

LIMITATIONS

There is no warranty, either expressed or implied, about the accuracy of the enclosed materials or their suitability for any particular purpose. Accordingly, Zilog assumes no responsibility for their use by the recipient. Further, Zilog assumes no obligation to furnish any assistance of any kind whatsoever, or to furnish any additional information or documentation.

NAME

csh, - a command interpreter with C-like syntax

SYNOPSIS

csh[**-cefinstvVxX**] [arg ...]

DESCRIPTION

A **csh** command script can be interpreted by entering

csh script ...

where script is the name of the file containing a number of **csh** commands and ... is replaced by a sequence of arguments. The C shell places these arguments in the variable argv and then begins to read commands from the script. These parameters are then available through the same mechanisms used to reference any other C shell variables. When a login shell terminates, it executes commands from the file .logout in the home directory.

The shell then repeatedly performs the following actions: a line of command input is read and broken into words. This sequence of words is placed on the command history list and parsed. Finally, each command in the current line is executed.

OPTIONS

The flag arguments are interpreted as follows:

-c Commands are read from the (single) following argument that must be present. Any remaining arguments are placed in argv.

-e The shell exits if any invoked command terminates abnormally or yields a nonzero exit status.

-f The shell starts faster, because it neither searches for nor executes commands from the file .cshrc in the invoker's home directory.

-i The shell is interactive and prompts for its top-level input, even if it appears to not be a terminal. Shells are interactive without this option if their inputs and outputs are terminals.

-n Commands are parsed, but not executed. This aids in syntactic checking of shell scripts.

-s Command input is taken from the standard input.

-t A single line of input is read and executed. A \ can be used to escape the new line at the end of

this line and continue onto another line.

-v Causes the verbose variable to be set, with the effect that command input is echoed after history substitution.

-x Causes the echo variable to be set, so that commands are echoed immediately before execution.

-V Causes the verbose variable to be set even before .cshrc is executed.

-X Is to **-x** as **-V** is to **-v**.

After processing of flag arguments, if arguments remain but none of the **-c**, **-i**, **-s**, or **-t** options was given, the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by $Ø.

FILES

~/.cshrc	Read at beginning of execution by each shell
~/.login	Read by login shell, after .cshrc at login
~/.logout	Read by login shell, at logout
/etc/cshprofile	Read by login shell, before .cshrc file. This is a system profile file for the csh. See cshrc(5).
/bin/sh	Standard shell, for shell scripts not starting with a #
/tmp/sh*	Temporary file for <<
/dev/null	Source of empty file
/etc/passwd	Source of home directories for ~name.

SEE ALSO

alias(1C), break(1C), breaksw(1C), cd(1C), continue(1C), echo(1C), exit(1C), foreach(1C), glob(1C), history(1C), if(1C), logout(1C), nice(1C), nohup(1C), onintr(1C), repeat(1C), set(1C), setenv(1C), source(1C), switch(1C), time(1C), umask(1C), wait(1C), while(1C), access(2), exec(2), fork(2), pipe(2), signal(2), umask(2), wait(2), a.out(5), environ(5).

The C shell in the ZEUS Utilities Manual

LIMITATIONS

Control structure should be parsed rather than being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with |, and to be used with & and ; metasyntax.

Commands within loops, prompted for by ?, are not placed in the history list.

It should be possible to use the : modifiers on the output of command substitutions. All and more than one : modifier should be allowed on $ substitutions.

Words can be no longer than 512 characters. The number of characters in an argument list or a ccommand substitution is limited to 512Ø characters. The number of arguments to a command that involves file name expansion is limited to 853.

To detect looping, the shell restricts the number of **alias**(1C) substitutions on a single line to 2Ø.

NAME

csplit - split file according to contextual arguments

SYNOPSIS

csplit [**-s**] [**-k**] [**-f** *prefix*] *file* *arg1* [... *argn*]

DESCRIPTION

Csplit reads *file* and separates it into n+1 sections, defined by the arguments *arg1*... *argn*. By default the sections are placed in **xxØØ** ... **xxn** (*n* may not be greater than 99). These sections get the following pieces of *file*:

ØØ: From the start of *file* up to (but not including) the line referenced by *arg1*.

Ø1: From the line referenced by *arg1* up to the line referenced by *arg2*.

n+1: From the line referenced by *argn* to the end of *file*.

OPTIONS

-s **Csplit** normally prints the character counts for each file created. If the **-s** option is present, **csplit** suppresses the printing of all character counts.

-k **Csplit** normally removes created files if an error occurs. If the **-k** option is present, **csplit** leaves previously created files intact.

-f *prefix* If the **-f** option is used, the created files are named *prefix***ØØ** ... *prefix**n*. The default is **xxØØ** ... **xx***n*.

ARGUMENTS

/*rexp*/ A file is to be created for the section from the current line up to (but not including) the line containing the regular expression *rexp*. The current line becomes the line containing *rexp*. This argument may be followed by an optional **+** or **-** some number of lines (e.g., **/Page/-5**).

%*rexp*% This argument is the same as /*rexp*/, except that no file is created for the section.

lnno A file is to be created from the current line up to (but not including) *lnno*. The current line becomes *lnno*.

{*num*} Repeat argument. This argument may follow any of the above arguments. If it follows a *rexp*

type argument, that argument is applied num more times. If it follows lnno, the file will be split every lnno lines (num times) from that point.

Enclose all rexp type arguments that contain blanks or other characters meaningful to the Shell in the appropriate quotes. Regular expressions may not contain embedded newlines. **Csplit** does not affect the original file; it is the users responsibility to remove it.

EXAMPLES

```
csplit -f cobol file '/procedure division/' /par5./ /par16./
```

This example creates four files, **cobol00** ... **cobol03**. After editing the "split" files, they can be recombined as follows:

```
cat cobol0[0-3] > file
```

Note that this example overwrites the original file.

```
csplit -k file 100 {99}
```

This example would split the file at every 100 lines, up to 10,000 lines. The **-k** option causes the created files to be retained if there are more than 10,000 lines; however, an error message would still be printed.

```
csplit -k prog.c '%main(%' '/^}/+1' {20}
```

Assuming that **prog.c** follows the normal **C** coding convention of ending routines with a } at the beginning of the line, this example will create a file containing each separate **C** routine (up to 21) in **prog.c**.

SEE ALSO

ed(1), sh(1), regexp(7).

DIAGNOSTICS

Self explanatory except for:

arg - out of range

which means that the given argument did not reference a line between the current position and the end of the file.

NAME

ctags - maintain a tags file for C or Fortran programs

SYNOPSIS

ctags [**-auw**] file ...

DESCRIPTION

Ctags makes a tags file for **ex**(1) from the specified C or Fortran programs. Files ending in ".f" are assumed to be Fortran source files and all others are assumed to be C source files. A tags file gives the locations of specified objects (in this case functions) in a group of files. Each line of the tags file contains the function name, the file in which it is defined, and a scanning pattern used to find the function definition. These are given in separate fields on the line, separated by blanks or tabs.

OPTIONS

-a Causes the output to be appended to the tags file instead of rewriting it.

-u Causes the specified files to be updated in tags, that is, all references to them are deleted, and the new values are appended to the file. The tags file is then piped through **sort**(1) to restore the ordering required by **ex**(1). This option implies the **-a** option.

-w Suppresses warning diagnostics.

The tag main is treated specially in C programs. The tag formed is created by prepending M to the name of the file, with a trailing .c removed, if any, and leading pathname components also removed. This makes use of ctags practical in directories with more than one program. Fortran programs are not named main, so this is not necessary.

FILES

tags output tags file

SEE ALSO

ex(1), vi(1).

LIMITATIONS

In a directory with only one main, the tag main will still not be created.

In Fortran 77, the program statement is optional, and if missing, no entry will be made in tags.

NAME

cu - call another ZEUS system

SYNOPSIS

cu [-s*speed*] [-a*acu*] [-l*line*] [-h] [-o|-e] *telno* | *dir*

DESCRIPTION

Cu calls up another ZEUS system, a terminal, or possibly a non-ZEUS system. It manages an interactive conversation with possible transfers of ASCII files.

Cu will try each line listed in the file **/usr/lib/uucp/L-devices** until it finds an available line with appropriate attributes or runs out of entries. The *L-devices* file must be set up using:

CONNECTION line speed

where *connection* is either **DIR** (for a direct connection to another system) or **ACU** (where the line is connected to an ACU) (see **acu**(4)). *Line* is the tty line which will serve to make the connection. The form of *line* is **tty***X* where *X* is the tty number. *Speed* is the baud rate for the connection. 300 baud is the default.

After making the connection, **cu** runs as two processes: the *transmit* process reads data from the standard input and, except for lines beginning with ~, passes it to the remote system; the *receive* process accepts data from the remote system and, except for lines beginning with ~, passes it to the standard output. Normally, an automatic DC3/DC1 protocol is used to control input from the remote system so the buffer is not overrun. Lines beginning with ~ have special meanings.

The *transmit* process interprets the following:

~.	terminate the conversation.
~!	escape to an interactive shell on the local system. The shell is taken from the SHELL environment variable. If it is not set, /bin/sh is used.
~!*cmd*...	run *cmd* on the local system. The shell to be used is taken from the SHELL environment variable. If it is not set, /bin/sh is used.
~$*cmd*...	run *cmd* locally and send its output to the remote system.

~%take *from* [*to*] copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places.

~%put *from* [*to*] copy file *from* (on local system) to file *to* on remote system. If *to* is omitted, the *from* argument is used in both places.

~~... send the line ~... to the remote system.

~nostop turn off the DC3/DC1 input control protocol for the remainder of the session. This is useful in case the remote system is one which does not respond properly to the DC3 and DC1 characters.

The *receive* process normally copies data from the remote system to its standard output. A line from the remote system that begins with **~>** initiates an output diversion to a file. The complete sequence is:

~>[>]:*file*
zero or more lines to be written to *file*
~>

Data from the remote system is diverted (or appended, if **>>** is used) to file. The trailing **~>** terminates the diversion.

The use of **~%put** requires **stty**(1) and **cat**(1) on the remote side. It also requires that the current erase and kill characters on the remote system be identical to the current ones on the local system. Backslashes are inserted at appropriate places.

The use of **~%take** requires the existence of **echo**(1) and **cat**(1) on the remote system. Also, **stty tabs** mode should be set on the remote system if tabs are to be copied without expansion.

For example, in order to call another system on a direct line, you would first have to know the terminal name and the default speed for that line. Then you would type (assuming that the direct line is tty3):

```
cu -ltty3 dir
```

OPTIONS

-e(-o)
Designate that even (odd) parity is to be generated for data sent to the remote system.

-h Emulate local echo, supporting calls to other computer systems which expect terminals to be in half-duplex mode.

-a(-l)
Specify a device name for the ACU and communications line device. They can be used to override searching for the first available ACU with the right speed.

-sspeed
Give the transmission speed (110, 150, 300, 1200, 4800, 9600); 300 is the default value. Most of our modems restrict us to choose between 300 and 1200. Directly connected lines may be set to other speeds.

Telno
is the telephone number. Character sequences to dial using the Hayes Microcomputer Products modems as well as the Ven-Tel 212 Plus modem may be included in telno. **dir** Must be used for directly connected lines, and implies a null ACU. When the string **dir** is specified the **-l** option must also be specified.

FILES
/usr/lib/uucp/L-devices
/usr/spool/uucp/LCK..(tty-device)
/dev/null

SEE ALSO
cat(1), echo(1), stty(1), uucp(1), tty(4).

DIAGNOSTICS
Exit code is zero for normal exit, non-zero (various values) otherwise.

LIMITATIONS
There is an artificial slowing of transmission by **cu** during the **~%put** operation so that loss of data is unlikely.

NAME

cut - cut out selected fields of each line of a file

SYNOPSIS

cut **-c**list [file1 file2 ...]
cut **-f**list [**-d**char] [**-s**] [file1 file2 ...]

DESCRIPTION

Use **cut** to cut out columns from a table or fields from each line of a file; in data base parlance, it implements the projection of a relation. The fields as specified by list can be fixed length, i.e., character positions as on a punched card (**-c** option), or the length can vary from line to line and be marked with a field delimiter character like tab (**-f** option). **Cut** can be used as a filter; if no files are given, the standard input is used.

OPTIONS

list — A comma-separated list of integer field numbers (in increasing order), with optional - to indicate ranges as in the **-o** option of nroff/troff for page ranges; e.g., **1,4,7**; **1-3,8**; **-5,1Ø** (short for **1-5,1Ø**); or **3-** (short for third through last field).

-clist — The list following **-c** (no space) specifies character positions (e.g., **-c1-72** would pass the first 72 characters of each line).

-dchar — The character following **-d** is the field delimiter (**-f** option only). Default is tab. Space or other characters with special meaning to the shell must be quoted.

-flist — The list following **-f** is a list of fields assumed to be separated in the file by a delimiter character (see **-d**); e.g., **-f1,7** copies the first and seventh field only. Lines with no field delimiters will be passed through intact (useful for table subheadings), unless **-s** is specified.

-s — Suppresses lines with no delimiter characters in case of **-f** option. Unless specified, lines with no delimiters will be passed through untouched.

Either the **-c** or **-f** option must be specified.

HINTS

Use **grep**(1) to make horizontal ``cuts'' (by context) through a file, or **paste**(1) to put files together column-wise (i.e., horizontally). To reorder columns in a table, use **cut** and **paste**.

EXAMPLES

```
cut -d: -f1,5 /etc/passwd
                        mapping of user IDs to names

name=`who am i | cut -f1 -d" "`
                        to set name to current login name.
```

DIAGNOSTICS

line too long
: A line can have no more than 511 characters or fields.

bad list for c/f option
: Missing **-c** or **-f** option or incorrectly specified *list*. No error occurs if a line has fewer fields than the *list* calls for.

no fields
: The *list* is empty.

SEE ALSO

grep(1), paste(1).

NAME

cxref - a simple C routine referencing program

SYNOPSIS

cxref file ...

DESCRIPTION

Cxref is a simple shell script which uses **grep**(1) and **ex**(1) and **sort**(1) to make a listing of the routines in the specified C program files and the lines on which they are defined. It is useful as a summary when prowling in a large program.

LIMITATIONS

Cxref assumes that routines begin in the first column of lines, and that type names are given on different lines than the routine names. If you have a program which is in a different format than this, **cxref** will fail miserably. The operating system, C compiler, **ex** editor, etc. all work with **cxref.**

NAME

date - print the date and time

SYNOPSIS

date [-u]

DESCRIPTION

The current date and time are printed, including the day of week, month, day of month, time (hh:mm:ss), time zone, and year.

If the -u option is given, GMT time is printed.

EXAMPLE

```
% date
Thu Nov 11 14:44:32 PST 1982
```

SEE ALSO

time(2), ctime(3), datem(M).

NAME

daytime - give the time to human-reasonable accuracy

SYNOPSIS

daytime

DESCRIPTION

Daytime prints out in English the current time of day, accurate to the nearest five minutes.

EXAMPLE

```
% daytime
Five after four
```

LIMITATIONS

Daytime depends on the user to determine whether it's currently night or day.

NAME

dc - desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

Dc is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. The overall structure of **dc** is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. The following constructions are recognized:

c All values on the stack are popped.

d The top value on the stack is duplicated.

f All values on the stack and in registers are printed.

i The top value on the stack is popped and used as the number radix for further input.

I Pushes the input base on the top of the stack.

k The top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.

lx The value in register x is pushed on the stack. The register x is not altered. All registers start with zero value. If the **l** is capitalized, register x is treated as a stack and its top value is popped onto the main stack.

o The top value on the stack is popped and used as the number radix for further output.

O Pushes the output base on the top of the stack.

p The top value on the stack is printed. The top value remains unchanged. **P** interprets the top of the stack as an ascii string, removes it, and prints it.

q Exits the program. If executing a string, the recursion level is popped by two. If **q** is capitalized, the top value on the stack is popped and the string execution level is popped by that value.

sx The top of the stack is popped and stored into a register named x, where x may be any character. If the **s** is capitalized, x is treated as a stack and the value is pushed on it.

v Replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.

x Treats the top element of the stack as a character string and executes it as a string of dc commands.

X Replaces the number on the top of the stack with its scale factor.

z The stack level is pushed onto the stack.

Z Replaces the number on the top of the stack with its length.

number
The value of the number is pushed on the stack. A number is an unbroken string of the digits Ø-9. It may be preceded by an underscore (_) to input a negative number. Numbers may contain decimal points.

+ - / * % ^
The top two values on the stack are added (+), subtracted (-), multiplied (*), divided (/), remaindered (%), or exponentiated (^). The two entries are popped off the stack; the result is pushed on the stack in their place. Any fractional part of an exponent is ignored.

[...]
Puts the bracketed ascii string onto the top of the stack.

<x **>**x **=**x
The top two elements of the stack are popped and compared. Register x is executed if they obey the stated relation.

! Interprets the rest of the line as a ZEUS command.

? A line of input is taken from the input source (usually the terminal) and executed.

; : Are used by **bc** for array operations.

EXAMPLES

```
$ dc
2.Ø5
156.35
+ p       (add and print the answer)
158.4Ø
2.Ø5
+ p
16Ø.45
6
* p       (multiply and print the answer)
962.7Ø
14
/ p       (divide and print the answer)
68
```

An example which prints the first ten values of n! is

```
[la1+dsa*pla1Ø>y]sy Øsal lyx
```

SEE ALSO

bc(1), which is a preprocessor for dc providing infix notation and a C-like syntax which implements functions and reasonable control structures for programs.

DIAGNOSTICS

`x is unimplemented' where x is an octal number.

`stack empty' for not enough elements on the stack to do what was asked.

`Out of space' when the free list is exhausted (too many digits).

`Out of headers' for too many numbers being kept around.

`Out of pushdown' for too many items on the stack.

`Nesting Depth' for too many levels of nested execution.

NAME

dd - convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

Dd copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block size can be specified to take advantage of raw physical I/O.

Where sizes are specified, a number of bytes is expected. A number may end with **k**, **b**, or **w** to specify multiplication by 1024, 512, or 2 respectively; a pair of numbers may be separated by **x** to indicate a product.

Cbs is used only if ascii or ebcdic conversion is specified. In the former case cbs characters are placed into the conversion buffer, converted to ASCII, and trailing blanks trimmed and new-line added before sending the line to the output. In the latter case ASCII characters are read into the conversion buffer, converted to EBCDIC, and blanks added to make up an output record of size cbs.

After completion, **dd** reports the number of whole and partial input and output blocks.

OPTIONS

option	values
bs=n	set both input and output block size, superseding ibs and obs; also, if no conversion is specified, it is particularly efficient since no in-core copy need be done
cbs=n	conversion buffer size
conv=ascii	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
lcase	map alphabetics to lower case
noerror	do not stop processing on an error
swab	swap every pair of bytes
sync	pad every input record to ibs

ucase map alphabetics to upper case

... , ... several comma-separated conversions

count=n copy only n input records

files=n skip n files on (tape) input before starting copy **seek=**n seek n records from beginning of output file before copying

ibs=n input block size n bytes (default 512)

if=file input file name; standard input is default

obs=n output block size (default 512)

of=file output file name; standard output is default

seek=n seek n records from beginning of output file before copying

skip=n skip n input records before starting copy

EXAMPLE

This command will read an EBCDIC tape blocked ten 8Ø-byte EBCDIC card images per record into the ASCII file **x**:

```
dd if=/dev/rmtØ of=x ibs=8ØØ cbs=8Ø conv=ascii,lcase
```

Note the use of raw magtape. **Dd** is especially suited to I/O on the raw physical devices because it allows reading and writing in arbitrary record sizes.

SEE ALSO

cp(1).

DIAGNOSTICS

f+p records in(out)
numbers of full and partial records read(written)

LIMITATIONS

The ASCII/EBCDIC conversion tables are taken from the 256 character standard in the CACM Nov, 1968. The ibm conversion, while less blessed as a standard, corresponds better to certain IBM print train conventions. There is no universal solution.

New-lines are inserted only on conversion to ASCII; padding is done only on conversion to EBCDIC. These should be separate options.

NAME

delta - make a delta (change) to an SCCS file

SYNOPSIS

delta [**-r**SID]
[**-s**n]
[**-g**list]
[**-m**[mrlist]]
[**-y**[comment]]
[**-p**]
files

DESCRIPTION

Delta is used to permanently introduce into the named SCCS file changes that were made to the file retrieved by **get**(1) (called the g-file, or generated file).

Delta makes a delta to each named SCCS file. If a directory is named, **delta** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of **-** is given, the standard input is read (see **WARNINGS**); each line of the standard input is taken to be the name of an SCCS file to be processed.

Delta may issue prompts on the standard output depending upon certain keyletters specified and flags (see **admin**(1)) that may be present in the SCCS file (see **-m** and **-y** keyletters below).

Keyletter arguments apply independently to each named file.

-rSID
Uniquely identifies which delta is to be made to the SCCS file. The use of this keyletter is necessary only if two or more outstanding **gets** for editing (**get -e**) on the same SCCS file were done by the same person (login name). The SID value specified with the **-r** keyletter can be either the SID specified on the **get** command line or the SID to be made as reported by the **get** command. A diagnostic results if the specified SID is ambiguous, or, if necessary and omitted on the command line.

-s Suppresses the issue, on the standard output, of the created delta's SID, as well as the number of lines inserted, deleted and unchanged in the SCCS file.

-n Specifies retention of the edited g-file (normally removed at completion of delta processing).

-glist
Specifies a list for the definition of list) of deltas which are to be ignored when the file is accessed at the change level (SID) created by this delta.

-m[mrlist]
If the SCCS file has the **v** flag set (see **admin**(1)) then a Modification Request (**MRs**) number must be supplied as the reason for creating the new delta.

If **-m** is not used and the standard input is a terminal, the prompt **MRs?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The **MRs?** prompt always precedes the **comments?** prompt (see **-y** keyletter).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the **MRs** list.

Note that if the **v** flag has a value it is taken to be the name of a program (or shell procedure) which will validate the correctness of the **MRs** numbers. If a non-zero exit status is returned from **MRs** number validation program, **delta** terminates (it is assumed that the **MRs** numbers were not all valid).

-y[comment]
Arbitrary text used to describe the reason for making the delta. A null string is considered a valid comment.

If **-y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.

-p Causes **delta** to print (on the standard output) the SCCS file differences before and after the delta is applied in a **diff**(1) format.

FILES
All files of the form ?-file are explained in the Source Code Control System User's Guide. The naming convention for these files is also described there.

g-file
Existed before the execution of **delta**; removed after completion of **delta**.

p-file
Existed before the execution of **delta**; may exist after completion of **delta**.

q-file
Created during the execution of **delta**; removed after completion of **delta**.

x-file
Created during the execution of **delta**; renamed to SCCS file after completion of **delta**.

z-file
Created during the execution of **delta**; removed during the execution of **delta**.

d-file
Created during the execution of **delta**; removed after completion of **delta**.

/usr/bin/bdiff
Program to compute differences between the ``gotten'' file and the g-file.

WARNINGS

Lines beginning with an **SOH** ASCII character (binary ØØ1) cannot be placed in the SCCS file unless the **SOH** is escaped. This character has special meaning to SCCS (see **sccsfile**(5)) and will cause an error.

A **get** of many SCCS files, followed by a **delta** of those files, should be avoided when the **get** generates a large amount of data. Instead, multiple **get**/**delta** sequences should be used.

If the standard input (-) is specified on the **delta** command line, the **-m** (if necessary) and **-y** keyletters must also be present. Omission of these keyletters causes an error to occur.

SEE ALSO

admin(1), bdiff(1), get(1), help(1), prs(1), sccsfile(5).
Source Code Control System User's Guide in the ZEUS Utilities Manual

DIAGNOSTICS

Use help(1) for explanations.

NAME

deroff - remove nroff/troff, tbl, and eqn constructs

SYNOPSIS

deroff [**-mx**] [**-w**] [files]

DESCRIPTION

Deroff reads each of the files in sequence and removes all **troff**(1) requests, macro calls, backslash constructs, **eqn**(1) constructs (between .EQ and .EN lines, and between delimiters), and **tbl**(1) descriptions, and writes the remainder of the file on the standard output. **Deroff** follows chains of included files (.so and .nx **troff** commands); if a file has already been included, a .so naming that file is ignored and a .nx naming that file terminates execution. If no input file is given, **deroff** reads the standard input.

OPTIONS

-m may be followed by an **m**, **s**, or **l**. The resulting **-mm** or **-ms** option causes the **mm** or **ms** macros to be interpreted so that only running text is output (i.e., no text from macro lines). The **-ml** option forces the **-mm** option and also causes deletion of lists associated with the **mm** macros.

-w the output is a word list, one "word" per line, with all other characters deleted. Otherwise, the output follows the original, with the deletions mentioned above. In text, a "word" is any string that contains at least two letters and is composed of letters, digits, ampersands (&), and apostrophes ('); in a macro call, however, a "word" is a string that begins with at least two letters and contains a total of at least three letters. Delimiters are any characters other than letters, digits, apostrophes, and ampersands. Trailing apostrophes and ampersands are removed from "words."

EXAMPLES

```
% cat example.file
.PP
This is a sample file with nroff commands.
It will be used with deroff:
.RS
deroff example.file
.RE
to remove the nroff commands for reading purposes.

% deroff example.file

This is a sample file with nroff commands.
It will be used with deroff:
```

deroff example.file

to remove the nroff commands for reading purposes.

```
% deroff -w example.file
This
is
a
sample
file
with
nroff
commands
It
will
be
used
with
deroff
deroff
example
file
to
remove
the
nroff
commands
for
reading
purposes
```

SEE ALSO

eqn(1), tbl(1), troff(1).

LIMITATIONS

Deroff is not a complete **troff** interpreter, so it can be confused by subtle constructs. Most such errors result in too much rather than too little output.

The **-ml** option does not handle nested lists correctly.

NAME

diff - differential file comparer

SYNOPSIS

diff [-befh] file1 file2

DESCRIPTION

Diff tells what lines must be changed in two files to bring them into agreement. If file1 (file2) is -, the standard input is used. If file1 (file2) is a directory, then a file in that directory whose file-name is the same as the file-name of file2 (file1) is used. The normal output contains lines of these forms:

n1 a n3,n4
n1,n2 d n3
n1,n2 c n3,n4

These lines resemble **ed**(1) commands to convert file1 into file2. The numbers after the letters pertain to file2. By exchanging a for d and reading backward, conversion of file2 into file1 is given. As in ed, identical pairs where n1 = n2 or n3 = n4 are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by <, then all the lines that are affected in the second file flagged by >.

Except in rare circumstances, **diff** finds a smallest sufficient set of file differences.

OPTIONS

-b Cause trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

-e Produce a script of a, c, and d commands for the editor ed, which recreates file2 from file1.

-f Produce a similar script, not useful with ed, in the opposite order.

In connection with **-e**, the following shell program can help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version ed scripts ($2,$3,...) made by diff need be on hand. A latest version appears on the standard output.

```
(shift; cat $*; echo '1,$p') | ed - $1
```

-h Does a fast, but incomplete job. It works only when changed parts are short and well separated. Options **-e** and **-f** are unavailable with **-h**.

EXAMPLES

```
% cat list1
boat
boathouse
boatload
boatman
boatmen
boatyard
```

```
% cat list2
boa
boar
board
boardinghouse
boast
boat
boathouse
boatload
boatman
boatmen
```

```
% diff list1 list2
0a1,5
> boa
> boar
> board
> boardinghouse
> boast
6d10
< boatyard
```

FILES

/tmp/d?????
/usr/lib/diffh for **-h**

SEE ALSO

cmp(1), comm(1), ed(1)

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some, 2 for trouble.

LIMITATIONS

Editing scripts produced under the **-e** or **-f** option are unable to create lines consisting of a single period (.). The script file is produced, but it will produce incorrect results if run under ed.

NAME

diff3 - 3-way differential file comparison

SYNOPSIS

diff3 [**-ex3**] file1 file2 file3

DESCRIPTION

Diff3 compares three versions of a file and publishes disagreeing ranges of text flagged with these codes:

====	all three files differ
====1	file1 is different
====2	file2 is different
====3	file3 is different

The type of change needed to convert a given range of a given file to some other is indicated in one of these ways:

f : n1 **a**	Text is to be appended after line number n1 in file f, where f = 1, 2, or 3.
f : n1 , n2 **c**	Text is to be changed in the range line n1 to line n2. If n1 = n2, the range may be abbreviated to n1.

The original contents of the range follows immediately after a "c" indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

OPTIONS

-e **diff3** publishes a script (to standard output) for the editor **ed**(1) that will incorporate into file1 all changes between file2 and file3, that is, the changes that normally would be flagged "====" and "====3".

-x (-3)
produces a script to incorporate only changes flagged "====" ("====3").

EXAMPLES

The following command will apply the resulting script to file1:

```
(cat script; echo '1,$p'; echo w) | ed - file1
```

FILES

/tmp/d3*
/usr/lib/diff3prog

SEE ALSO
diff(1), ed(1)

LIMITATIONS
Text lines that consist of a single "." will defeat the -e option.

Files longer than 64K bytes won't work.

NAME

diffmk - mark differences between files

SYNOPSIS

diffmk name1 name2 name3

DESCRIPTION

Diffmk compares two versions of a file and creates a third file that includes "change mark" commands for **nroff**(1) or **troff**(1). name1 and name2 are the old and new versions of the file. **Diffmk** generates name3, which contains the lines of name2 plus inserted formatter "change mark" (**.mc**) requests. When name3 is formatted, changed or inserted text is shown by | at the right margin of each line. The position of deleted text is shown by a single *.

Diffmk can be used to produce listings of C (or other) programs with changes marked. A typical command line for such use is:

```
diffmk old.c new.c tmp; nroff macs tmp | pr
```

where the file **macs** contains:

```
.pl 1
.ll 77
.nf
.eo
.nc
```

The **.ll** request might specify a different line length, depending on the nature of the program being printed. The **.eo** and **.nc** requests are probably needed only for C programs.

If the characters | and * are inappropriate, a copy of **diffmk** can be edited to change them (diffmk is a shell procedure).

SEE ALSO

diff(1), nroff(1).

BUGS

Aesthetic considerations may dictate manual adjustment of some output. File differences involving only formatting requests may produce undesirable output, i.e., replacing **.sp** by **.sp 2** will produce a "change mark" on the preceding or following line of output.

NAME

dircmp - directory comparison

SYNOPSIS

dircmp dir1 dir2

DESCRIPTION

Dircmp examines dir1 and dir2 and generates various tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated in addition to a list that indicates whether the files common to both directories have the same contents.

SEE ALSO

cmp(1), diff(1).

NAME

dog - controlled output flow filter for CRT previewing

SYNOPSIS

dog [file...]

DESCRIPTION

Dog is a filter that allows examination of a continuous text on a soft-copy terminal. **Dog** pauses after the first 23 lines of output. If RETURN is pressed, 23 more lines are displayed.

If CTRL-D is pressed, 11 more lines are displayed so that the file is scrolled. It is also possible to type positive numbers to **dog** causing that many lines to be printed, or negative numbers causing that many lines to be discarded followed by a scroll.

The terminal is set to noecho mode by this program so that the output can be continuous. The numbers and carriage returns therefore do not show on the terminal.

SEE ALSO

cat(1), more(1), pr(1).

NAME

du - summarize disk usage

SYNOPSIS

du [**-ars**] [files]

DESCRIPTION

Du gives the number of blocks contained in all files and (recursively) directories within each directory and file specified by the names argument. The block count includes the indirect blocks of the file. If names is missing, . is used.

Absence of an option causes an entry to be generated for each directory only. **Du** is normally silent about directories that cannot be read, files that cannot be opened, etc.

A file with two or more links is only counted once.

OPTIONS

-a Cause an entry to be generated for each file.

-r Cause **du** to generate messages in cases where files and directories cannot be opened or read.

-s Cause only the grand total (for each of the specified names) to be given.

LIMITATIONS

If the **-a** option is not used, non-directories given as arguments are not listed.

If there are too many distinct linked files, **du** will count the excess files more than once.

Files with holes in them will get an incorrect block count.

NAME

echo - echo (print) arguments to the standard output (terminal)

SYNOPSIS

echo [-n] [arg] ...

DESCRIPTION

Echo writes its arguments separated by blanks and terminated by a new-line on the standard output. It also understands C-like escape conventions; beware of conflicts with the shell's use of \:

\b	backspace
\c	print line without new-line
\f	form-feed
\n	new-line
\r	carriage return
\t	tab
****	backslash
\n	the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number n, which must start with a zero.

Echo is useful for producing diagnostics in command files and for sending known data into a pipe.

OPTIONS

-n No newline is added to the output.

EXAMPLE

```
% echo this is using the echo command
this is using the echo command
```

SEE ALSO

csh(1), echo(1C), echo2(1).

NAME

echo - echo (print) arguments to the standard output (terminal)

SYNOPSIS

echo [-n] [arg] ...

DESCRIPTION

Echo is both an internal shell command, and an external program, it writes its arguments separated by blanks and terminated by a new-line on the standard output. It also understands C-like escape conventions; beware of conflicts with the shell's use of ****:

\b	backspace
\c	print line without new-line
\f	form-feed
\n	new-line
\r	carriage return
\t	tab
****	backslash
****n	the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number n, which must start with a zero.

Echo is useful for producing diagnostics in command files and for sending known data into a pipe.

OPTIONS

-n No newline is added to the output.

EXAMPLE

```
% echo this is using the echo command
this is using the echo command
```

SEE ALSO

csh(1), echo(1), echo2(1).
The C Shell in the ZEUS Utilities Manual

NAME

echo2 - echo (print) arguments to standard error

SYNOPSIS

echo2 [**-n**] [arguments ...]

DESCRIPTION

Echo2 writes its arguments separated by blanks and terminated by a newline on the standard error.

OPTIONS

-n No newline is added to the output.

SEE ALSO

NAME

ed - text editor

SYNOPSIS

ed [-] [**-x**] [file]

DESCRIPTION

Ed is the standard text editor.

If a file argument is given, **ed** simulates an e command on the named file; that is, the file is read into **ed's** buffer for editing.

Ed operates on a copy of any file it is editing; changes made in the copy have no effect on the file until a w (write) command is given. The copy of the text being edited resides in a temporary file called the buffer.

Commands to **ed** have a simple and regular structure: zero or more addresses followed by a single character command, possibly followed by parameters to the command. These addresses specify one or more lines in the buffer. Missing addresses are supplied by default. Only one command can appear on a line.

Certain commands allow the addition of text to the buffer. While **ed** is accepting text, it is said to be in input mode. In this mode, no commands are recognized; all input is merely collected. Input mode is left by typing a period (.) alone at the beginning of a line.

Ed supports a limited form of regular expression notation. A regular expression specifies a set of strings of characters. A member of this set of strings is said to be matched by the regular expression. In the following specification for regular expressions the word "character" means any character but new line.

1. Any character except a special character matches itself. Special characters are the regular expression delimiter plus \,**[**, and sometimes ^, *, $.

2. A . matches any character.

3. A \ followed by any character except a digit or () matches that character.

4. A nonempty string s bracketed [s] (or [^s]) matches any character in (or not in) s. In s, \ has no special meaning, and] can appear only as the first letter. A substring a-b, with a and b

in ascending ASCII order, stands for the inclusive range of ASCII characters.

5. A regular expression of form 1-4 followed by * matches a sequence of Ø or more matches of the regular expression.

6. A regular expression, x, of form 1-8, bracketed \(x\) matches what x matches.

7. A \ followed by a digit n matches a copy of the string that was matched by the bracketed regular expression beginning with the nth \(.

8. A regular expression of form 1-8, x, followed by a regular expression of form 1-7, y matches a match for x followed by a match for y, with the x match being as long as possible while still permitting a y match.

9. A regular expression of form 1-8 preceded by ^ (or followed by $), is constrained to matches that begin at the left (or end at the right) end of a line.

1Ø. A regular expression of form 1-9 picks out the longest among the leftmost matches in a line.

11. An empty regular expression stands for a copy of the last regular expression encountered.

Regular expressions are used in addresses to specify lines and in the substitution command to specify a portion of a line which is to be replaced. To use one of the regular expression metacharacters as an ordinary character, precede that character by \. This also applies to the character bounding the regular expression (often /) and to \ itself.

To understand addressing in **ed**, it is necessary to know that at any time there is a current line. Generally speaking, the current line is the last line affected by a command; however, the effect on the current line is discussed under the description of the command. Addresses are constructed according to the following rules:

1. The dot (.) addresses the current line.

2. The ($) addresses the last line of the buffer.

3. A decimal number n addresses the n-th line of the buffer.

4. 'x addresses the line marked with the name x, which must be a lowercase letter. Lines are marked with the k command described below.

5. A regular expression enclosed in slashes (/) addresses the line found by searching forward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary, the search wraps around to the beginning of the buffer.

6. A regular expression enclosed in queries (?) addresses the line found by searching backward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary, the search wraps around to the end of the buffer.

7. An address followed by a plus sign (+) or a minus sign (-) followed by a decimal number specifies that address plus (or minus) the indicated number of lines. The plus sign can be omitted.

8. If an address begins with + or - the addition or subtraction is taken with respect to the current line; for example, -5 is understood to mean .-5.

9. If an address ends with + (or -), then 1 is added (or subtracted). As a consequence of this rule and rule 8, the address - refers to the line before the current line. Trailing + and - characters have cumulative effect, so -- refers to the current line less 2.

10. To maintain compatibility with earlier versions of the editor, the character ^ in addresses is equivalent to -.

Commands can require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when insufficient ones are given. If more addresses are given than such a command requires, the last one or two (depending on what is accepted) are used.

Addresses are separated from each other by a comma (,). They can also be separated by a semicolon (;). In this case the current line, dot (.) is set to the previous address before the next address is interpreted. This feature can be used to determine the starting line for forward and backward searches (/ or ?). The second address of any two-address

sequence must correspond to a line following the line corresponding to the first address.

In the following list of **ed** commands, the default addresses are shown in parentheses; the parentheses are not part of the address.

It is generally illegal for more than one command to appear on a line. However, most commands can be suffixed by p or by l, in which case the current line is either printed or listed respectively in the following way:

(.)a
<text>
.

The append command reads the given text and appends it after the addressed line. Dot is left on the last line input, if there were any, otherwise at the addressed line. Address Ø is legal for this command; text is placed at the beginning of the buffer.

(., .)c
<text>
.

The change command deletes the addressed lines, then accepts input text that replaces these lines. Dot is left at the last line input; if there were none, it is left at the line preceding the deleted lines.

(., .)d

The delete command deletes the addressed lines from the buffer. The line originally after the last line deleted becomes the current line; if the lines deleted were originally at the end, the new last line becomes the current line.

e filename

The edit command causes the entire contents of the buffer to be deleted, and then the named file to be read in. Dot is set to the last line of the buffer. The number of characters read is typed. Filename is retained for possible use as a default file name in a subsequent r or w command. If filename is missing, the retained name is used.

E filename

This command is the same as e, except that no diagnostic results when no w has been given since the last buffer alteration.

f filename

The filename command prints the currently retained file

name. If filename is given, the currently retained file name is changed to filename.

(1,$)g/regular expression/command list
In the global command, the first step is to mark every line that matches the given regular expression. Then for every such line, the given command list is executed with dot initially set to that line. A single command or the first of multiple commands appears on the same line with the global command. All lines of a multi-line list except the last line must be ended with \. A, i, and c commands and associated input are permitted; the dot terminating the input mode can be omitted if it is on the last line of the command list. The commands g and v are not permitted in the command list.

(.)i
<text>
.
This command inserts the given text before the addressed line. Dot is left at the last line input, or, if there were none, at the line before the addressed line. This command differs from the a command only in the placement of the text.

(., .+1)j
This command joins the addressed lines into a single line; intermediate newlines simply disappear. Dot is left at the resulting line.

(.)kx
The mark command marks the addressed line with name x, which must be a lowercase letter. The address form 'x addresses this line.

(., .)l
The list command prints the addressed lines in an unambiguous way: nongraphic characters are printed in two-digit octal, and long lines are folded. The l command can be placed on the same line after any non-I/O command.

(., .)ma
The move command repositions the addressed lines after the line addressed by a. The last of the moved lines becomes the current line.

(., .)p
The print command prints the addressed lines. Dot is left at the last line printed. The p command can be placed on the same line after any non-I/O command.

(., .)P
This command is a synonym for p.

q The quit command causes ed to exit. No automatic write of a file is done.

Q This command is similar to q, but no diagnostic results when no w has been given since the last buffer alteration.

($)r filename
The read command reads in the given file after the addressed line. If no file name is given, the retained file name, if any, is used (e and f commands). The file name is retained if there was no retained file name already. Address Ø is legal for r and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed. Dot is left at the last line read in from the file.

(., .)s/regular expression/replacement/ or,
(., .)s/regular expression/replacement/g
The substitute command searches each addressed line for an occurrence of the specified regular expression. On each line in which a match is found, all matched strings are replaced by the replacement specified, if the global replacement indicator g appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. It is an error for the substitution to fail on all addressed lines. Any character other than space or newline can be used instead of / to delimit the regular expression and the replacement. Dot is left at the last line substituted.

An ampersand (&) appearing in the replacement is replaced by the string matching the regular expression. The special meaning of & in this context can be suppressed by preceding it by \. The characters \n, where n is a digit, are replaced by the text matched by the n-th regular subexpression enclosed between \(and \). When nested subexpressions in parentheses are present, n is determined by counting occurrences of \(starting from the left.

Lines can be split by substituting new line characters into them. The new line in the replacement string must be escaped by preceding it by \.

(., .)ta
This command acts like the m command, except that a

copy of the addressed lines is placed after address a that can be Ø. Dot is left on the last line of the copy.

(., .)u
The undo command restores the preceding contents of the current line, which must be the last line in which a substitution was made.

(1, $)v/regular expression/command list
This command is the same as the global command g except that the command list is executed g with dot initially set to every line except those matching the regular expression.

(1, $)w filename
The write command writes the addressed lines onto the given file. If the file does not exist, it is created mode 666 (readable and writable by everyone). The file name is retained if there was no retained file name already. If no file name is given, the retained file name, if any, is used (e and f commands). Dot is unchanged. If the command is successful, the number of characters written is printed.

(1,$)W filename
This command is the same as w, except that the addressed lines are appended to the file.

x A key string is demanded from the standard input. Later r, e and w commands encrypt and decrypt the text with this key by the algorithm of **crypt**(1). An explicitly empty key turns off encryption.

($)= The line number of the addressed line is typed. Dot is unchanged by this command.

!<shell command>
The remainder of the line after the ! is sent to **sh**(1) to be interpreted as a command. Dot is unchanged.

(.+1)<newline>
An address alone on a line causes the addressed line to be printed. A blank line alone is equivalent to .+1p; it is useful for stepping through text.

If an interrupt signal (ASCII DEL) is sent, **ed** prints a ? and returns to its command level.

Some size limitations are 512 characters per line, 256 characters per global command list, 64 characters per file name, and 128K characters in the temporary file. The limit on the

number of lines depends on the amount of core: each line takes 1 word.

A line of text in append ("a"), change ("c") or insert ("i") commands should not exceed 256 characters.

When reading a file, **ed** discards ASCII NUL characters and all characters after the last newline. It does not read files containing non-ASCII characters.

OPTIONS

-x An x command is simulated first to handle an encrypted file.

\- Suppress the printing of character counts by e, r, and w commands.

FILES

/tmp/e*
ed.hup: work is saved here if terminal hangs up

SEE ALSO

The Zeus Line-Oriented text editor ED
The EX Reference Manual both in the ZEUS Utilities Manual.
edit(1), ex(1), vi(1), sed(1).

DIAGNOSTICS

"?name" for inaccessible file; "?" for errors in commands

"?TMP" for temporary file overflow.

A q or e command is in error, unless a w has occurred since the last buffer change. A second q or e is always obeyed.

LIMITATIONS

The l command mishandles DEL.

A ! command cannot be subject to a g command.

Because 0 is an illegal address for a w command, it is not possible to create an empty file with **ed**.

NAME

edit - text editor for new or casual users

SYNOPSIS

edit [**-r**] name ...

DESCRIPTION

Edit is a variant of the text editor **ex**(1) and is recommended for new or casual users who wish to use a command-oriented editor. The following is a brief introduction to **edit.** A more complete basic introduction is provided by the EX Reference Manual. See **ex**(1) for other useful documents; for example, documents on **vi**(1) for easily manipulating text on a CRT terminal.

BRIEF INTRODUCTION

To edit the contents of an existing file, begin with the command:

edit file

Edit makes a copy of the file and returns with the number of lines and characters in the file. To create a new file, make up a name for the file and run **edit** on it. This causes an error diagnostic, but allows the edit to procede.

Edit prompts for commands with the character :. Editing an existing file means there are some lines in **edit's** buffer (its name for the copy of the file being edited). Most commands to **edit** use its current line if a line number is not specified. (The current line is the last line affected by a command.) Thus, to **print** the current line, enter p, and press RETURN. To delete the current line, enter the delete command (d). **Edit** deletes the line and prints the new current line (usually the line following the deleted line). When the last line is deleted, the new last line becomes the new current line.

To start with an empty file or to add some new lines, use the **append** (a) command. **Edit** reads lines from the terminal until it receives a line consisting of just a dot, then places these lines after the current line. The last line typed then becomes the current line. The command **insert** (i) is like **append**, but places the lines before the current line.

Edit sequentially numbers the lines in the buffer, starting with 1 for the first line. The command 1 causes **edit** to type this first line.

Change text within the current line by using the **substitute** (s) command. Enter s/old/new/ where old is the characters

to be replaced and new is the new characters.

The **file** command (f) tells how many lines there are in the buffer being edited and says "[Modified]" if that number has changed. After modifying a file in the buffer, replace the original (unmodified) file with it by entering a **write** (w) command. Leave the editor by issuing a **quit** (q) command. If **edit** is run on a file, but it is not changed, it is not necessary to **write** the file. Trying to **quit** from **edit** after modifying the buffer without writing it prints a warning that there has been "No **write** since last change" and **edit** awaits another command. Issuing another **quit** command causes the buffer to be irretrievably discarded, and there is a return to the shell.

By using the **delete** and **append** commands and giving line numbers to see lines in the file, any changes can be made. The following commands are handy, however, if **edit** is used more than a few times.

The **change** command (c) changes the current line to a sequence of lines supplied. To **change** more than one line, give the line numbers of the lines to be changed; for example, "3,5change." Lines can be printed this way too; for example, "1,23p" prints the first 23 lines of the file.

The **undo** command (u) reverses the effect of the last command that changed the buffer. Thus, if a **substitute** command does not produce the desired results, enter u and the old contents of the line will be restored. An **undo** command can also act on itself. **Edit** gives a warning message when commands affect more than one line of the buffer. If the amount of change seems unreasonable, issue an undo and see what happened. If the change is ok, then enter undo again to get back the changes made before the first undo. Commands such as write and quit cannot be undone.

To look at the next line in the buffer, press RETURN. To look at a number of lines, press ^D (control key and, while it is held down, D key, rather than carriage return). This displays a half screen of lines on a CRT or 12 lines on a hardcopy terminal. The text around the current location can be scanned by giving the z. command. The current line is then the last line printed. To get back to the line before the z command, type ". The z command can also be given other following characters: z- prints a screen of text (or 24 lines) ending at the current line; z+ prints the next screen. A number of lines can be specified with the z command, for example "z.12" prints 12 lines. This method of giving counts works in general; thus, five 5 lines of text starting with the current line can be deleted with the command d5.

Search the file for strings by giving commands of the form **/text/** to search forward for text or **?text?** to search backward for text. If a search reaches the end of the file without finding the text, it wraps around and continues to search to the line where the search command was issued. A useful feature here is a search of the form **/^text/** which searches for text at the beginning of a line. Similarly **/text$/** searches for text at the end of a line. The trailing / or ? can be omitted in these commands.

The current line has a symbolic name dot (.); this is most useful in a range of lines, as in .,$print that prints the rest of the lines in the file. To get to the last line in the file, use its symbolic name $. Thus, the command $ delete or $d deletes the last line in the file. Arithmetic with line references is also possible. Thus, the line $-5 is the fifth before the last line, and .+2Ø is 2Ø lines after the present line.

The current line is printed in response to a .= entry. This is useful to move or copy a section of text within a file or between files. Find out the first and last line numbers to copy or move (say 1Ø to 2Ø). For a move, enter 1Ø,2Ømove "a which deletes these lines from the file and places them in a buffer named a. **Edit** has 26 such buffers named a through z. Enter "a move . to put the contents of buffer a after the current line. To move or copy these lines between files, give an **edit** (e) command after copying the lines, following it with the name of the other file to edit; for example, edit chapter2. By changing move to copy, a pattern can be established for copying lines. If the text to move or copy is all within one file, enter 1Ø,2Ømove $. It is not necessary to use named buffers in this case.

OPTIONS

-r Recover named files after an editor or system crash; the last saved version is retrieved.

SEE ALSO

ed(1), ex(1), vi(1), sed(1).
EX Reference Manual in the ZEUS Utilities Manual

LIMITATIONS

See **ex**(1).

NAME

env - set environment for command execution

SYNOPSIS

env [-] [name=value] ... [command args]

DESCRIPTION

Env obtains the current environment, modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form name=value are merged into the inherited environment before the command is executed.

If no command is specified, the resulting environment is printed, one name=value pair per line.

OPTIONS

- Cause the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments.

SEE ALSO

sh(1), csh(1), exec(2), environ(5).

NAME

eqn, neqn, checkeq - typeset mathematics

SYNOPSIS

eqn [**-dxy**] [**-pn**] [**-sn**] [**-fn**] [file] ...
neqn [**-dxy**] [**-pn**] [**-sn**] [**-fn**] [file] ...
checkeq [file] ...

DESCRIPTION

Eqn is a troff(1) preprocessor for typesetting mathematics on a Graphic Systems phototypesetter, **neqn** on terminals. Usage is almost always

```
eqn file ... | troff
neqn file ... | nroff
```

If no files are specified, these programs read from the standard input. A line beginning with `.EQ' marks the start of an equation; the end of an equation is marked by a line beginning with `.EN'. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to set two characters as `delimiters'; subsequent text between delimiters is also treated as **eqn** input. Delimiters may be set to characters x and y with the command-line argument **-dxy** or (more commonly) with `delim xy' between .EQ and .EN. The left and right delimiters may be identical. Delimiters are turned off by `delim off'. All text that is neither between delimiters nor between .EQ and .EN is passed through untouched.

The program **checkeq** reports missing or unbalanced delimiters and .EQ/.EN pairs.

Tokens within **eqn** are separated by spaces, tabs, newlines, braces, double quotes, tildes or circumflexes. Braces {} are used for grouping; generally speaking, anywhere a single character like x could appear, a complicated construction enclosed in braces may be used instead. Tilde ~ represents a full space in the output, circumflex ^ half as much.

Subscripts and superscripts are produced with the keywords **sub** and **sup.**

Thus x sub i makes:

x_i

a sub i sup 2 produces:

$$a_i^2$$

and e sup {x sup 2 + y sup 2} gives

$$e^{x^2+y^2}$$

Fractions are made with **over**: a over b yields:

$$\frac{a}{b}$$

sqrt makes square roots: 1 over sqrt {ax sup 2 +bx+c} results in:

$$\frac{1}{\sqrt{ax^2+bx+c}}$$

The keywords **from** and **to** introduce lower and upper limits on arbitrary things:

$$\lim_{n \to \infty \emptyset} \sum_{i}^{n} Rx$$

is made with lim from {n-> inf } sum from ∅ to n x sub i.

Left and right brackets, braces, etc., of the right height are made with **left** and **right**: left [x sup 2 + y sup 2 over alpha right] ~=~1 produces:

$$\left[x^2 + \frac{y^2}{A} \right] = 1$$

The **right** clause is optional. Legal characters after **left** and **right** are braces, brackets, bars, c and f for ceiling and floor, and "" for nothing at all (useful for a right-side-only bracket).

Vertical piles of things are made with **pile**, **lpile**, **cpile**, and **rpile**: pile {a above b above c} produces:

a
b
c

There can be an arbitrary number of elements in a pile. **lpile** left-justifies, **pile** and **cpile** center, with different vertical spacing, and **rpile** right justifies.

Matrices are made with **matrix**: matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } } produces:

$$\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$$

In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot**, **dotdot**, **hat**, **tilde**, **bar**, **vec**, **dyad**, and **under**: x dot = f(t) bar is:

$$\dot{x} = \overline{f(t)}$$

y dotdot bar ~=~ n under is:

$$\underline{\ddot{\bar{y}}} = \underline{n}$$

and x vec ~=~ y dyad is:

$$\vec{x} = \overleftrightarrow{y}$$

Sizes and font can be changed with **size** n or **size** +n, **roman**, **italic**, **bold**, and **font** n. Size and fonts can be changed globally in a document by **gsize** n and **gfont** n, or by the command-line arguments **-s**n and **-f**n.

Normally subscripts and superscripts are reduced by 3 point sizes from the previous size; this may be changed by the command-line argument **-p**n.

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with **define**: define thing % replacement % defines a new token called thing which will be replaced by replacement whenever it appears thereafter. The % may be any character that does not occur in replacement.

Keywords like sum (sum) int (int) inf (inf) and shorthands like >= (>=) -> (->), and != (!=) are recognized. Greek letters are spelled out in the desired case, as in alpha or GAMMA. Mathematical words like sin, cos, log are made Roman automatically. Troff(1) four-character escapes like \(bs can be used anywhere. Strings enclosed in double quotes "..." are passed through untouched; this permits keywords to be entered as text, and can be used to

communicate with troff when all else fails.

SEE ALSO

troff(1), tbl(1), ms(7), eqnchar(7).
NROFF/TROFF User's Manual in the ZEUS Utilities Manual

BUGS

To embolden digits, parens, etc., it is necessary to quote them, as in `bold "12.3"'.

NAME

error - analyze and disperse compiler error messages

SYNOPSIS

error [**-I** ignorefile] [**-n**] [**-q**] [**-s**] [**-t** suffixlist] [**-v**] [name]

DESCRIPTION

Error analyzes and optionally disperses the diagnostic error messages produced by a number of compilers and language processors to the source file and line where the errors occurred. It can replace the painful, traditional methods of scribbling abbreviations of errors on paper, and permits error messages and source code to be viewed simultaneously without machinations of multiple windows in a screen editor.

Error looks at the error messages, either from the specified file name or from the standard input, and attempts to determine which language processor produced each error message, determines the source file and line number to which the error message refers, determines if the error message is to be ignored or not, and inserts the (possibly slightly modified) error message into the source file as a comment on the line preceeding to which the line the error message refers.

Error messages which can't be categorized by language processor or content are not inserted into any file, but are sent to the standard output. **Error** touches source files only after all input has been read. By specifying the **-q** query option, the user is asked to confirm any potentially dangerous (such as touching a file) or verbose action. Otherwise **error** proceeds on its merry business. If the **-t** touch option and associated suffix list is given, **error** will restrict itself to touch only those files with suffices in the suffix list.

Error can be asked (by specifying **-v)** to invoke **vi**(1) on the files in which error messages were inserted; this obviates the need to remember the names of the files with errors.

Error is intended to be run with its standard input connected via a pipe to the error message source. Some language processors put error messages on their standard error file; others put their messages on the standard output. Hence, both error sources should be piped together into **error.** For example, when using the csh syntax,

```
make -s lint |& error -q -v
```

will analyze all the error messages produced by whatever programs **make**(1) runs when making lint.

Error knows about the error messages produced by: **make, cc,** cpp, ccom, **as, lint,** and **f77. Error** knows a standard format for error messages produced by the language processors, so is sensitive to changes in these formats. Error messages are restricted to be on one line. Some error messages refer to more than one line in more than one file; **error** will duplicate the error message and insert it at all of the places referenced.

Error will do one of six things with error messages.

synchronize
: Some language processors produce short errors describing which file it is processing. **Error** uses these to determine the file name for languages that don't include the file name in each error message. These synchronization messages are consumed entirely by **error.**

discard
: Error messages from **lint**(1) that refer to one of the two **lint**(1) libraries, /usr/lib/llib-lc and /usr/lib/llib-port are discarded, to prevent accidently touching these libraries. Again, these error messages are consumed entirely by **error.**

nullify
: Error messages from **lint**(1) can be nullified if they refer to a specific function, which is known to generate diagnostics which are not interesting. Nullified error messages are not inserted into the source file, but are written to the standard output. The names of functions to ignore are taken from either the file named .errorrc in the user's home directory, or from the file named by the **-I** option. If the file does not exist, no error messages are nullified. If the file does exist, there must be one function name per line.

not file specific
: Error messages that can't be intuited are grouped together, and written to the standard output before any files are touched. They will not be inserted into any source file.

file specific
: Error message that refer to a specific file, but to no specific line, are written to the standard output when that file is touched.

true errors
: Error messages that can be intuited are candidates for insertion into the file to which they refer.

Only true error messages are candidates for inserting into the file they refer to. Other error messages are consumed entirely by **error** or are written to the standard output. **Error** inserts the error messages into the source file on the line preceeding the line the language processor found in error. Each error message is turned into a one line comment for the language, and is internally flagged with the string "###" at the beginning of the error, and "%%%" at the end of the error. This makes pattern searching for errors easier with an editor, and allows the messages to be easily removed.

In addition, each error message contains the source line number for the line the message refers to. A reasonably formatted source program can be recompiled with the error messages still in it, without having the error messages themselves cause future errors. For poorly formatted source programs in free format languages, such as C, it is possible to insert a comment into another comment, which can wreck havoc with a future compilation.

To avoid this, format the source program so there are no language statements on the same line as the end of a comment.

Error catches interrupt and terminate signals, and if in the insertion phase, will orderly terminate what it is doing.

OPTIONS

-I ignorefile
: Names file containing the names of the functions to ignore.

-n
: Do not touch any files; all error messages are sent to the standard output.

-q
: The user is queried whether s/he wants to touch the file. A "y" or "n" to the question is necessary to continue. Absence of the **-q** option implies that all referenced files (except those referring to discarded error messages) are to be touched.

-s
: Print out statistics regarding the error categorization. Not too useful.

-t
: Take the following argument as a suffix list. Files whose suffixes do not appear in the suffix list are not touched. The suffix list is dot separated, and "*" wildcards work. Thus the suffix list:

```
".c.y.foo*.h"
```

allows **error** to touch files ending with ".c", ".y", ".foo*" and ".h".

-v After all files have been touched, overlay the visual editor **vi**(1) to edit all files touched, and positioned in the first touched file at the first error. If **vi**(1) can't be found, try **ex**(1) or **ed**(1) from standard places.

FILES

~/.errorrc	function names to ignore for lint error messages
/dev/tty	user's teletype

LIMITATIONS

Opens the teletype directly to do user querying.

Source files with links make a new copy of the file with only one link to it.

Changing a language processor's format of error messages may cause **error** to not understand the error message.

Error, since it is purely mechanical, will not filter out subsequent errors caused by `floodgating' initiated by one syntactically trivial error. Humans are still much better at discarding these related errors.

Error was designed for work on CRTs at reasonably high speed. It is less pleasant on slow speed terminals, and has never been used on hardcopy terminals.

NAME

ex, edit - text editor

SYNOPSIS

ex
[**-**]
[**-v**]
[**-t**tag]
[**-r**]
[**-R**]
[**+**[command]]
[**-l**]
file ...

edit [ex options]

DESCRIPTION

Ex is the root of a family of editors: **edit, ex,** and **vi. Ex** is a superset of **ed,** with the most notable extension being a display editing facility. Display-based editing is the focus of **vi.**

The available options are:

- This option suppresses all editor prompts and printing of character counts output by e, r, and w commands; this option is also useful in processing editor scripts in command files.

-v This option causes vi to be invoked instead of ex.

-t tag
The cursor will be positioned at the definition of tag immediately after ex is entered.

-r This option is used to recover named files after an editor or system crash; the last saved version is retrieved.

-R This option is used to invoke a "read only" version of ex.

+[command]
The editor begins by executing the command, command; if command is omitted, then the editor begins with the cursor positioned at the last line of the file.

-l This sets up ex for LISP editing; that is, the editing options, showmatch and lisp are set.

file Name of the file(s) to be edited.

The editor edit is convenient for casual users; it avoids some of the complexities of ex.

To use a display-based editor on a CRT terminal, see **vi** (1), a command that focuses on the display editing portion of **ex.**

DOCUMENTATION

The document The ZEUS Line-Oriented Text Editor -- Ed provides a comprehensive introduction to **edit** assuming no previous knowledge of computers or the ZEUS system.

The Ex Reference Manual is a comprehensive and complete manual for the command mode features of **ex,** but is not a tutorial to learn from.

Introduction to Display Editing with vi introduces the display editor **vi** and provides reference material on **vi.**

These documents can be found in the ZEUS Utilities Manual.

FILES

/usr/lib/ex?.?strings	error messages
/usr/lib/ex?.?recover	recover command
/usr/lib/ex?.?preserve	preserve command
/etc/termcap	describes capabilities of terminals
~/.exrc	editor startup file
/tmp/Exnnnnn	editor temporary
/tmp/Rxnnnnn	named buffer temporary
/usr/preserve	preservation directory

SEE ALSO

ed(1), grep(1), sed(1), vi(1), termcap(5), environ(5), termlist(7)

LIMITATIONS

The undo command causes all marks to be lost on lines changed, then restored if the marked lines were changed.

Undo never clears the buffer modified condition.

The z command prints a number of logical rather than physical lines. More than a screen full of output results if long lines are present.

File input/output errors do not print a name if the command line option - is used.

There is no easy way to do a single-scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files, and cannot appear in resultant files.

NAME

exit - exit a shell

SYNOPSIS

exit
exit (expression)

DESCRIPTION

The shell exits either with the value of the status variable (first form) or with the value of the specified expression (second form). This command is ignored if the ignoreexit variable is set.

SEE ALSO

break(1C), logout(1C).
The C Shell in the ZEUS Utilities Manual.

NAME

expand - expand tabs to spaces

SYNOPSIS

expand [-tabstop] [-tab1,tab2,...,tabn] [file ...]

DESCRIPTION

Expand processes the named files or the standard input writing the standard output with tabs changed into blanks. Backspace characters are preserved into the output and decrement the column count for tab calculations. **Expand** is useful for pre-processing character files (before sorting, looking at specific columns, etc.) that contain tabs.

If a single tabstop argument is given then tabs are set tabstop spaces apart instead of the default 8. If multiple tabstops are given then the tabs are set at those specific columns.

NAME

expr - evaluate arguments as an expression

SYNOPSIS

expr arg ...

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Each token of the expression is a separate argument.

The operators and keywords are listed here. The list is in order of increasing precedence, with equal precedence operators grouped.

expr | expr
yields the first **expr** if it is neither null nor Ø; otherwise, yields the second **expr**

expr & expr
yields the first **expr** if neither **expr** is null or Ø; otherwise, yields Ø

expr relop **expr** where relop is one of <, <=, =, !=, >=, or >, it yields 1 if the indicated comparison is true, "Ø" if false. The comparison is numeric if both **expr** are integers, otherwise, it is lexicographic

expr + expr

expr - expr
addition or subtraction of the arguments

expr * expr

expr / expr

expr % expr
multiplication, division, or remainder of the arguments

expr : expr
The matching operator compares the string first argument with the regular expression second argument; regular expression syntax is the same as that of **ed**(1). The **\(...\)** pattern symbols can be used to select a portion of the first argument; otherwise, the matching operator yields the number of characters matched and returns Ø on failure

substr string start length
yields the substring of **length** characters of **string** starting at the **start** (numeric index, 1 is the first

character of **string**)

length string
yields the **length** in bytes of **string**

index string target
yields the index of the first occurrence in **string** of any one of the characters in **target.** The characters of **target** are not treated as a string; rather, they are treated as individual characters.

(expr)
parentheses for grouping

Examples:

To add 1 to the shell variable a, enter:

```
a=`expr $a + 1`
```

To find the filename part (least significant part) of the path name stored in variable a, that can contain /:

```
expr $a : ".*/\(.*\)" "|" $a
```

Note the quoted shell metacharacters.

SEE ALSO
csh(1), ed(1), sh(1), test(1)

DIAGNOSTICS
Expr returns the following exit codes:

```
Ø    if the expression is neither null nor Ø
1    if the expression is null or Ø,
2    for invalid expressions
```

LIMITATIONS
substr does not follow the system standard of counting from Ø.

NAME

file - determine file type

SYNOPSIS

file [-f] file ...

DESCRIPTION

File performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, **file** examines the first 512 bytes and tries to guess its language. If the -f option is given, the next argument is taken to be a file containing the names of the files to be examined.

LIMITATIONS

It often makes mistakes. In particular, it often suggests that command files are C programs. It also has trouble distinguishing between PLZ/ASM and PLZ/SYS programs.

NAME

find - find files

SYNOPSIS

find path-name-list expression

DESCRIPTION

Find recursively descends the directory hierarchy for each path name in the path-name-list (i.e., one or more path names) seeking files that match a Boolean expression written in the primaries given below. In the descriptions, the argument n is used as a decimal integer where +n means more than n, -n means less than n, and n means exactly n.

The primaries can be combined using the following operators (in order of decreasing precedence):

1) The negation of a primary (! is the unary not operator).

2) Concatenation of primaries (the and operation is implied by the juxtaposition of two primaries).

3) Alternation of primaries (**o** is the or operator).

OPTIONS

-atime n — True if the file has been accessed in n days.

-cpio device — Write the current file on device in **cpio**(5) format (5120 byte records).

-ctime n — True if the file has been changed in n days.

-exec cmd — True if the executed cmd returns a zero value as exit status. The end of cmd must be punctuated by an escaped semicolon. A command argument **{}** is replaced by the current path name.

-group gname — True if the file belongs to the group gname. If gname is numeric and does not appear in the **/etc/group** file, it is taken as a group ID.

-links n — True if the file has n links.

-mtime n — True if the file has been modified in n days.

-name file — True if file matches the current file name. Normal shell argument syntax may be used if

escaped (watch out for [,? and *).

-newer file — True if the current file has been modified more recently than the argument file.

-ok cmd — Like **-exec** except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing **y**.

-perm onum — True if the file permission flags exactly match the octal number onum (see **chmod**(1)). If onum is prefixed by a minus sign, more flag bits (Ø17777, see **stat**(2)) become significant and the flags are compared:
(flags&onum)==onum

-print — Always true; causes the current path name to be printed.

-size n — True if the file is n blocks long (512 bytes per block).

-type c — True if the type of the file is c, where c is **b**, **c**, **d**, **p**, or **f** for block special file, character special file, directory, fifo (a.k.a named pipe), or plain file.

-user uname — True if the file belongs to the user uname. If uname is numeric and does not appear as a login name in the **/etc/passwd** file, it is taken as a user ID.

(expression **)** — True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).

EXAMPLE

To find all files named **a.out** that have not been accessed for a week:

```
find / -name a.out -atime +7 -print
```

FILES

/etc/passwd, /etc/group

SEE ALSO

cpio(1), sh(1), test(1), stat(2), cpio(5), fs(5).

NAME

flow - flow analysis of C programs

SYNOPSIS

flow [**-bcors**] [output-suffix] files ...

DESCRIPTION

Flow performs a flow analysis on the named files producing 3 tables: which functions call which; which functions are called by which; which functions reside in which source files. These tables are called flow.CALLS, flow.CALLEDBY and flow.RESIDES respectively.

To create these tables, first the C source files must be converted into trace files. Then any trace files specified on the command line as well as any trace files produced are processed to create the needed tables. In order to facilitate the use of flow with make(1) the `trace files' may be saved.

The `files' specified on the command line can be any mixture of C source files and trace files. Remember that trace files are name `file.t'.

OPTIONS

-b Generate the `CALLEDBY' file.

-c Generate the `CALLS' table.

-o Rather than using the suffix `flow' for naming the output tables, use the suffix supplied by the user, which is the next argument.

-r Generate the `RESIDES' file.
The default for the `table' flags are to produce all tables.

-s Save the trace files. Trace file names are the `root-name' of the C source file, with a `.t' appended. For example, if the source file is adb.c, its trace file would be adb.t.

FILES

/tmp/flow[ABCDE] temporary processing files
file.i Files produced by the C pre-processor
file.t Trace files

SEE ALSO

cc(1)

DIAGNOTICS

Diagnostics are intended to be self-explanatory. However

when processing interrupts, misleading diagnostics can occur.

LIMITATIONS

Since the output tables are produced with the help of tbl(1) and nroff(1), the process is slow.

NAME

foreach - C Shell flow control loop initiation

SYNOPSIS

foreach name (list)
command
end

DESCRIPTION

The variable name is successively set to each member of wordlist and the sequence of commands between this command and the matching **end** are executed. Both **foreach** and **end** must appear alone on separate lines.

When this command is read from the terminal, the loop is read up once prompting with ? before any statements in the loop are executed. A typing mistake in a loop at the terminal can be rubbed out.

FILES

/bin/csh

SEE ALSO

break(1C), breaksw(1C), continue(1C), exit(1C), if(1C), nohup(1C), onintr(1C), switch(1C), while(1C).
The C Shell in the ZEUS Utilities Manual

NAME

get - get a version of an SCCS file

SYNOPSIS

get [**-r**SID]
[**-c**cutoff]
[**-e**]
[**-b**]
[**-i**list]
[**-x**list]
[**-k**]
[[**-l**[p]]
[**-pmnsbgt**]
[**-a**seq-no.] file

DESCRIPTION

Get generates an ASCII text file from each named SCCS file according to the specifications given by its keyletter arguments, which begin with **-**. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, **get** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of **-** is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the g-file whose name is derived from the SCCS file name by simply removing the leading **s.**; (see also FILES, below).

Each of the keyletter arguments is explained below as though only one SCCS file is to be processed, but the effects of any keyletter argument applies independently to each named file.

-rSID — The SCCS IDentification string (SID) of the version (delta) of an SCCS file to be retrieved. Table~1 below shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by **delta**(1) if the **-e** keyletter is also used), as a function of the SID specified.

-ccutoff — Cutoff date-time, in the form:

YY[MM[DD[HH[MM[SS]]]]]

No changes (deltas) to the SCCS file which were created after the specified cutoff date-time are

included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, **-c75Ø2** is equivalent to **-c75Ø228235959**. Any number of non-numeric characters may separate the various 2 digit pieces of the cutoff date-time. This feature allows one to specify a cutoff date in the form: **"-c77/2/2 9:22:25"**. Note that this implies that one may use the %E% and %U% identification keywords (see below) for nested gets within, say the input to a **send**(1) command:

```
~!get "-c%E% %U%" s.file
```

-e Indicates that the get is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of **delta**(1). The **-e** keyletter used in a get for a particular version (SID) of the SCCS file prevents further gets for editing on the same SID until delta is executed or the **j** (joint edit) flag is set in the SCCS file (see **admin**(1)). Concurrent use of **get -e** for different SIDs is always allowed.

If the g-file generated by get with an **-e** keyletter is accidentally ruined in the process of editing it, it may be regenerated by re-executing the get command with the **-k** keyletter in place of the **-e** keyletter.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file (see **admin**(1)) are enforced when the **-e** keyletter is used.

-b Used with the **-e** keyletter to indicate that the new delta should have an SID in a new branch as shown in Table 1. This keyletter is ignored if the **b** flag is not present in the file (see **admin**(1)) or if the retrieved delta is not a leaf delta. (A leaf delta is one that has no successors on the SCCS file tree.)
Note: A branch delta may always be created from a non-leaf delta.

-ilist A list of deltas to be included (forced to be applied) in the creation of the generated file. The list has the following syntax:

```
<list> ::= <range> | <list> , <range>
<range> ::= SID | SID - SID
```

SID, the SCCS Identification of a delta, may be in any form shown in the ``SID Specified'' column of Table 1. Partial SIDs are interpreted as shown in the ``SID Retrieved'' column of Table 1.

-x_list_ A _list_ of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the **-i** keyletter for the _list_ format.

-k Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The **-k** keyletter is implied by the **-e** keyletter.

-l[p] Causes a delta summary to be written into an _l-file_. If **-lp** is used then an _l-file_ is not created; the delta summary is written on the standard output instead. See _FILES_ for the format of the _l-file_.

-p Causes the text retrieved from the SCCS file to be written on the standard output. No _g-file_ is created. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the **-s** keyletter is used, in which case it disappears.

-s Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.

-m Causes each text line retrieved from the SCCS file to be preceded by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.

-n Causes each generated text line to be preceded with the %M% identification keyword value (see below). The format is: %M% value, followed by a horizontal tab, followed by the text line. When both the **-m** and **-n** keyletters are used, the format is: %M% value, followed by a horizontal tab, followed by the **-m** keyletter generated format.

-g Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an _l-file_, or to verify the existence of a particular SID.

-t Used to access the most recently created (``top'') delta in a given release (e.g., **-r1**),

or release and level (e.g., **-rl.2**).

-aseq-no. The delta sequence number of the SCCS file delta (version) to be retrieved (see **sccsfile**(5)). This keyletter is used by the **comb**(1) command; it is not a generally useful keyletter, and users should not use it. If both the **-r** and **-a** keyletters are specified, the **-a** keyletter is used. Care should be taken when using the **-a** keyletter in conjunction with the **-e** keyletter, as the SID of the delta to be created may not be what one expects. The **-r** keyletter can be used with the **-a** and **-e** keyletters to control the naming of the SID of the delta to be created.

For each file processed, get responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the **-e** keyletter is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a new-line) before it is processed. If the **-i** keyletter is used included deltas are listed following the notation ``Included''; if the **-x** keyletter is used, excluded deltas are listed following the notation ``Excluded''.

TABLE 1. Determination of SCCS Identification String

SID* Spec.	-b Key Used	Other Conditions	SID Retrieved	SID of Delta to be Created
none	no	R defaults to mR	mR.mL	mR.(mL+1)
none	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	-	R < mR and R does not exist	hR.mL**	hR.mL.(mB+1).1
R	-	Trunk succ.# in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk succ.	R.L	R.(L+1)
R.L	yes	No trunk succ.	R.L	R.L.(mB+1).1
R.L	-	Trunk succ. in release > R	R.mL	R.mL.(mB+1).1

R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch succ.	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB+1).1
R.L.B.S	-	Branch succ.	R.L.B.S	R.L.(mB+1).1

Code

R = release
L = level
B = branch
S = sequence
m = maximum.

Thus, for example,

R.mL = the maximum level number within release R;

R.L.(mB+1).1
= the first sequence number on the new branch (i.e., maximum branch number plus one) of level L within release R.

Note that if the SID specified is of the form R.L, R.L.B, or R.L.B.S, each of the specified components must exist.

** ``hR'' is the highest existing release that is lower than the specified, nonexistent, release R.

*** This is used to force creation of the first delta in a new release.

Successor.

+ The **-b** keyletter is effective only if the **b** flag (see **admin**(1)) is present in the file. An entry of - means ``irrelevant''.

= This case applies if the **d** (default SID) flag is not present in the file. If the **d** flag is present in the file, then the SID obtained from the **d** flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

IDENTIFICATION KEYWORDS

Identifying information is inserted into the text retrieved from the SCCS file by replacing identification keywords with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

Keyword	Value
%M%	Module name: either the value of the **m** flag in the file (see **admin**(1)), or if absent, the name of the SCCS file with the leading **s.** removed.
%I%	SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text.
%R%	Release.
%L%	Level.
%B%	Branch.
%S%	Sequence.
%D%	Current date (YY/MM/DD).
%H%	Current date (MM/DD/YY).
%T%	Current time (HH:MM:SS).
%E%	Date newest applied delta was created (YY/MM/DD).
%G%	Date newest applied delta was created (MM/DD/YY).
%U%	Time newest applied delta was created (HH:MM:SS).
%Y%	Module type: value of the **t** flag in the SCCS file (see **admin**(1)).
%F%	SCCS file name.
%P%	Fully qualified SCCS file name.
%Q%	The value of the **q** flag in the file (see **admin**(1)).
%C%	Current line number. This keyword is intended for identifying messages output by the program such as ``this shouldn't have happened'' type errors. It is not intended to be used on every line to provide sequence numbers.
%Z%	The 4-character string **@(#)** recognizable by **what**(1).
%W%	A shorthand notation for constructing **what**(1) strings for ZEUS program files. %W% = %Z%%M%<horizontal-tab>%I%
%A%	Another shorthand notation for constructing **what**(1) strings for non-ZEUS program files. %A% = %Z%%Y% %M% %I%%Z%

FILES

Several auxiliary files may be created by get. These files are known generically as the g-file, l-file, p-file, and z-file. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form **s.**module-name, the auxiliary files are named by replacing the leading **s** with the tag. The g-file is an exception to this scheme: the g-file is named by removing the **s.** prefix. For example, **s.xyz.c**, the auxiliary file names would be **xyz.c**, **l.xyz.c**, **p.xyz.c**, and **z.xyz.c**, respectively.

The g-file, which contains the generated text, is created in the current directory (unless the **-p** keyletter is used). A g-file is created in all cases, whether or not any lines of text were generated by the get. It is owned by the real

user. If the **-k** keyletter is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

The l-file contains a table showing which deltas were applied in generating the retrieved text. The l-file is created in the current directory if the **-l** keyletter is used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the l-file have the following format:

a. A blank character if the delta was applied;
   * otherwise.
b. A blank character if the delta was applied or wasn't applied and ignored;
   * if the delta wasn't applied and wasn't ignored.
c. A code indicating a ``special'' reason why the delta was or was not applied:
   ``I'': Included.
   ``X'': Excluded.
   ``C'': Cut off (by a **-c** keyletter).
d. Blank.
e. SCCS identification (SID).
f. Tab character.
g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.
h. Blank.
i. Login name of person who created delta.

The comments and **MR** data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The p-file is used to pass information resulting from a get with an **-e** keyletter along to delta. Its contents are also used to prevent a subsequent execution of get with an **-e** keyletter for the same SID until delta is executed or the joint edit flag, **j**, (see **admin**(1)) is set in the SCCS file. The p-file is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the p-file is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the get was executed, followed by a blank and the **-i** keyletter argument if it was present, followed by a blank and the **-x** keyletter argument if it was present, followed by a new-line. There can be an arbitrary number of lines in the p-file at any time; no two

lines can have the same new delta SID.

The z-file serves as a lock-out mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (i.e., get) that created it. The z-file is created in the directory containing the SCCS file for the duration of get. The same protection restrictions as those for the p-file apply for the z-file. The z-file is created mode 444.

SEE ALSO

admin(1), delta(1), help(1), prs(1), what(1), sccsfile(5). Source Code Control System User's Guide in the ZEUS Utilities Manual

DIAGNOSTICS

Use **help**(1) for explanations.

LIMITATIONS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user doesn't, then only one file may be named when the **-e** keyletter is used.

NAME

getfile - transfer files from local to remote system

SYNOPSIS

getfile [**-qfBb**] file1 [[**-b**] file2 ...]

DESCRIPTION

Getfile uploads one or more files to ZEUS from another ZEUS system or a RIO System running the file transfer software of the S-8000 Communication Package. This program is invoked from the remote system; therefore, **remote**(1) must be executed first. Files are transferred one record at a time along with a checksum to ensure the accuracy of the data. Transfer of one file can be terminated by entering a control-x. The entire transfer can be aborted by entering an escape.

OPTIONS

-b The next file is considered to be a binary. Carriage returns are not replaced by new lines.

-B All file names on the line are treated as if they were preceded by a -b. This is usually desirable for ZEUS-to-ZEUS transfers.

-f The program suppresses all nonfatal error messages.

-q The program prints a query before replacing an existing file of the same name as the one being transferred.

SEE ALSO

putfile(1), local(1), remote(1).

DIAGNOSTICS

"checksum error ... retry" Printed if the computed checksum does not match the transmitted checksum.

"<filename> ... transfer aborted" Printed after a specific number of retries, if a control-x or an escape is entered, or if the transfer failed (due to lack of space or bad media).

"getfile: <n1> successful transfers <n2> unsuccessful transfers" Printed at program termination.

"<filename> ... unable to open file" Printed if the file cannot be opened on either system.

The program outputs a single dot (.) after each successful transfer of a record. If the transfer appears to halt, allow 20 seconds for a retry.

NAME

getNAME - get NAME sections of manual for whatis/apropos data base

SYNOPSIS

getNAME name ...

DESCRIPTION

Getname reads the first few lines of each of the argument manual section sources, and finds the .TH entry and the .SH entry writing them to the standard output in a form suitable for making into the whatis/apropos data base.

FILES

/usr/lib/whatis Eventual data base

SEE ALSO

apropos(1), makewhatis(1), whatis(1).

NAME

getopt - parse command options

SYNOPSIS

set -- getopt optstring **$***

DESCRIPTION

Getopt is used to break up options in command lines for easy parsing by shell procedures, and to check for legal options.

Optstring is a string of recognized option letters (see getopt(3)); if a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by white space.

Getopt will place -- in the arguments at the end of the options, or recognize it if used explicitly. The shell arguments ($1 $2 . . .) are reset so that each option is preceded by a - and in its own shell argument; each option argument is also in its own shell argument.

OPTIONS

-- Delimit the end of the options.

DIAGNOSTICS

Getopt prints an error message on the standard error when it encounters an option letter not included in optstring.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the options **a** and **b**, and the option **o**, which requires an argument.

```
set -- getopt abo: $*
if [ $? != Ø ]
then
     echo $USAGE
     exit 2
fi
for i in $*
do
    case $i in
    -a | -b)  FLAG=$i; shift;;
    -o)       OARG=$2;  shift; shift;;
    --)       shift;    break;;
    esac
done
```

This code will accept any of the following as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
```

```
cmd -oarg -a file file
cmd -a -oarg -- file file
```

SEE ALSO

sh(1), getopt(3).

LIMITATIONS

Getopt is not useful when typed at the prompt. Its use is within the body of a shell script. Although it is not a built-in feature of the C Shell, it is included in the 1C section of the reference manual because of its use.

NAME

gets - get a string from standard input

SYNOPSIS

gets [default]

DESCRIPTION

Gets can be used with **csh**(1) to read a string from the standard input. If a default is given it is used if an error occurs. The resultant string (either the default or as read from the standard input) is written to the standard output. If no default is given and an error occurs, **gets** exits with exit status 1.

EXAMPLE

The following shell script will set the variable a to whatever is typed into the terminal at the prompt, and then echo the contents of the variable "a".

```
# example shell script

echo -n "Enter a letter:"

set a=`gets`

echo $a
```

LIMITATIONS

The **gets** command is used from within the body of a shell script, it is not useful from the prompt. For that reason it is in the "1C" section of the manual.

SEE ALSO

csh(1), line(1).
The C Shell in the ZEUS Utilities Manual.

NAME

glob - print strings on the terminal without spaces

SYNOPSIS

glob wordlist

DESCRIPTION

Like **echo**, but no \ escapes are recognized and words are delimited by null characters in the output. Useful for programs which use the shell to expand a list of words.

SEE ALSO

echo(1C), echo2(1).
The C Shell in the ZEUS Utilities Manual

NAME

gpasswd - change group password

SYNOPSIS

gpasswd [name]

DESCRIPTION

This command changes (or installs) a password associated with the group name (your own group by default).

The program prompts for the old password and then for the new one. The caller must supply both. The new password must be typed twice to forestall mistakes.

New passwords must be at least four characters long if they use a sufficiently rich alphabet and at least six characters long if monocase.

Only the owner of the name or the super-user can change a password; the owner must prove he knows the old password.

Once the password has been changed, a notice is sent to all members of the group.

FILES

/etc/passwd
/etc/gtmp*

SEE ALSO

login(1), crypt(3), passwd(5).

LIMITATIONS

Under certain conditions the group file will not be updated. In these situations the new file resides in /etc/gtmp*.

NAME

greek - select terminal filter

SYNOPSIS

greek [**-T**terminal]

DESCRIPTION

Greek is a filter that reinterprets the extended character set, as well as the reverse and half-line motions, of a 128-character TELETYPE(Reg.) Model 37 terminal (which is the **nroff**(1) default terminal) for certain other terminals. Special characters are simulated by overstriking. If the argument is omitted, **greek** attempts to use the environment variable **$TERM** (see **environ**(7)). The following terminals are currently recognized:

hp	Hewlett-Packard 2621, 2640, and 2645.
tek	Tektronix 4014.
300	DASI 300.
300-12	DASI 300 in 12-pitch.
300s	DASI 300s.
300s-12	DASI 300s in 12-pitch.
450	DASI 450.
450-12	DASI 450 in 12-pitch.
1620	Diablo 1620 (alias DASI 450).
1620-12	Diablo 1620 (alias DASI 450) in 12-pitch.
2621	Hewlett-Packard 2621, 2640, and 2645.
2640	Hewlett-Packard 2621, 2640, and 2645.
2645	Hewlett-Packard 2621, 2640, and 2645.
4014	Tektronix 4014.

FILES

/usr/bin/300
/usr/bin/300s
/usr/bin/450

SEE ALSO

300(1), 4014(1), 450(1), eqn(1), greek(7), nroff(1), environ(5), term(7).

NAME

grep, egrep, fgrep - search a file for a pattern

SYNOPSIS

grep [options] expression [files]

egrep [options] [expression] [files]

fgrep [options] [strings] [files]

DESCRIPTION

Commands of the **grep** family search the input files (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. **Grep** patterns are limited regular expressions in the style of **ed**(1); it uses a compact non-deterministic algorithm. **Egrep** patterns are full regular expressions; it uses a fast deterministic algorithm that sometimes needs exponential space. **Fgrep** patterns are fixed strings; it is fast and compact.

In all cases, the file name is output if there is more than one input file. Care should be taken when using the characters **$**, *****, **[**, **^**, **|**, **(**, **)**, and **** in expression, because they are also meaningful to the shell. It is safest to enclose the entire expression argument in single quotes **'...'**.

Fgrep searches for lines that contain one of the strings separated by new-lines.

Egrep accepts regular expressions as in **ed**(1), except for **\(** and **\)**, with the addition of:

1. A regular expression followed by **+** matches one or more occurrences of the regular expression.
2. A regular expression followed by **?** matches Ø or 1 occurrences of the regular expression.
3. Two regular expressions separated by **|** or by a new-line match strings that are matched by either.
4. A regular expression may be enclosed in parentheses **()** for grouping.

The order of precedence of operators is **[]**, then ***?+**, then concatenation, then **|** and new-line.

OPTIONS

-b Each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.

-c Only a count of matching lines is printed.

-e expression

Same as a simple expression argument, but useful when the expression begins with a - (does not work with **grep**).

-f file
The regular expression or strings list (fgrep) is taken from the file.

-h Do not print filename headers with output lines.

-l Only the names of files with matching lines are listed (once), separated by new-lines.

-n Each line is preceded by its relative line number in the file.

-s The error messages produced for nonexistent or unreadable files are suppressed (grep only).

-v All lines but those matching are printed.

-x (Exact) only lines matched in their entirety are printed (fgrep only).

SEE ALSO

csh(1), ed(1), sed(1), sh(1).

DIAGNOSTICS

Exit status is Ø if any matches are found, 1 if none, 2 for syntax errors or inaccessible files.

LIMITATIONS

Ideally there should be only one **grep**, but we don't know a single algorithm that spans a wide enough range of space-time tradeoffs.

Lines are limited to 256 characters; longer lines are truncated.

Egrep does not recognize ranges, such as **[a-z]**, in character classes.

NAME

head - give first few lines of a stream

SYNOPSIS

head [-count] [file ...]

DESCRIPTION

This filter gives the first count lines of each of the specified files, or of the standard input. If count is omitted it defaults to 10.

SEE ALSO

cat(1), dog(1), more(1), tail(1).

NAME

help - ask for help

SYNOPSIS

help [args]

DESCRIPTION

Help finds information to explain a message from a command or explain the use of a command. Zero or more arguments may be supplied. If no arguments are given, **help** will prompt for one.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

type 1 Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (e.g., **ge6**, for message 6 from the **get** command).

type 2 Does not contain numerics (as a command, such as **get**)

type 3 Is all numeric (e.g., **212**)

The response of the program will be the explanatory information related to the argument, if there is any.

When all else fails, try "help stuck."

FILES

/usr/lib/help directory containing files of message text

DIAGNOSTICS

Use **help**(1) for explanations.

SEE ALSO

man(1), INTRO(1), apropos(1).

NAME

history - print list of previous commands

SYNOPSIS

history
set history=N

DESCRIPTION

Commands input from the teminal are numbered sequentially from one and are saved on the history list, whose size is controlled by the history variable.

The contents of the history list is seen with the command:

history

The size of the history list is determined by setting the history shell variable with the command:

set history=N

where N is the desired size of the history list (15 is a recommended number).

These saved commands (also called events) are referred to in the following ways:

!n	event number n
!-m	the desired even is m events prior to the current event
!p	prefix of a command in an event
!?string?	string in an event argument; trailing ? can be omitted if nothing follows
!!	immediately previous event

A history reference can be given without an event specification; for example, !$. In this case, the reference is to the previous command unless a previous history reference occurred on the same line in which case this form repeats the previous reference.

SEE ALSO

set(1C), setenv(1C).
The C Shell in the ZEUS Utilities Manual

NAME

hyphen - find hyphenated words

SYNOPSIS

hyphen files

DESCRIPTION

Hyphen finds all the hyphenated words in files and prints them on the standard output. If no arguments are given, the standard input is used. **Hyphen** can be used as a filter.

LIMITATIONS

Hyphen can't cope with hyphenated italic (i.e., underlined) words; it often misses them completely, or mangles them.

NAME

id - print user and group IDs and names

SYNOPSIS

id

DESCRIPTION

Id writes a message on the standard output giving the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs do not match, both are printed.

SEE ALSO

logname(1), getuid(2), getgid(2).

NAME

if - C Shell flow control branch statement

SYNOPSIS

if (expression.1) **then**
command.1
else if (expression.2) **then**
command.2
else
command.3
endif

DESCRIPTION

If the specified expression.1 is true, the commands to the first **else** are executed; else if expression.2 is true, the commands to the second else are executed, etc. Any number of **else-if** pairs are possible; only one **endif** is needed. The **else** part is likewise optional. The words **else** and **endif** must appear at the beginning of input lines; command must be a simple command, not a pipeline, a command list, or a command list within parentheses.

SEE ALSO

foreach(1C), while(1C).
The C Shell in the ZEUS Utilities Manual.

NAME

isrio - determine if terminal is a RIO System

SYNOPSIS

isrio

DESCRIPTION

Isrio outputs a character sequence ("ESC?") that causes a RIO System to return another character sequence. **Isrio** prints a "y" if the terminal is a RIO System, an "n" if it is not. Thus, the program can be used significantly in initialization files, as in the following example:

```
if (`isrio` == "y") setenv TERM mcz
```

SEE ALSO

getfile(1), local(1), putfile(1).

DIAGNOSTICS

Non-RIO terminals that recognize the sequence "ESC?" will execute that terminal function first and after a timeout, **isrio** returns "y".

NAME

join - relational database operator

SYNOPSIS

join [options] file1 file2

DESCRIPTION

Join forms, on the standard output, a join of the two relations specified by the lines of file1 and file2. If file1 is minus, the standard input is used.

File1 and file2 must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first field in each line.

There is one line in the output for each pair of lines in file1 and file2 that have identical join fields. The output line normally consists of the common field, then the rest of the line from file1, then the rest of the line from file2.

Fields are normally separated by blank, tab, or new line. In this case, multiple separators count as one, and leading separators are discarded.

OPTIONS

-an In addition to the normal output, produce a line for each unpairable line in file n, where n is 1 or 2.

-e s Replace empty output fields by string s.

-jn m
Join on the mth field of file n. If n is missing, use the mth field in each file.

-o list
Each output line comprises the fields specifed in list, each element of which has the form n.m, where n is a file number and m is a field number.

-tc Use character c as a separator (tab character). Every appearance of c in a line is significant.

EXAMPLES

```
% cat list1          % cat list2
boat                 boa
boathouse            boar
boatload             board
boatman              boardinghouse
boatmen              boast
boatyard             boat
                     boathouse
                     boatload
```

boatman
boatmen

% join list1 list2
boat
boathouse
boatload
boatman
boatmen

% join -a2 list1 list2
boa
boar
board
boardinghouse
boast
boat
boathouse
boatload
boatman
boatmen

% join -a1 list1 list2
boat
boathouse
boatload
boatman
boatmen
boatyard

SEE ALSO

sort(1), comm(1)

LIMITATIONS

With default field separation, the collating sequence is that of **sort -b**; with **-t**, the sequence is that of a plain sort.

The conventions of join, sort, comm, uniq, look (1) are incongruous.

NAME

kill - send a signal to a process

SYNOPSIS

kill [-signo] processid ...

DESCRIPTION

Kill sends a signal (15 by default to terminate) to the specified processes. If a signal number preceded by a minus is given as the first argument, that signal is sent instead of terminate (signal (2)). This kills processes that do not catch the signal; "kill -9 ..." is a sure kill.

If process number 0 is specified, all members in the process group (processes resulting from the current login) are signaled.

The killed processes must belong to the current user unless the super-user executes it. To shut the system down and bring it up, the super-user uses "kill -1 1"; init(M).

The process number of an asynchronous process started with an & is reported by the shell. Process numbers are also found by using ps(1).

EXAMPLE

```
% nroff -man intro.03 > INTRO &
1028

% ps
   PID TTY TIME CMD
    42 8   0:12 -csh
  1028 8   0:02 nroff -man intro.03
  1029 8   0:02 ps

% kill -9 1028
1028: nroff: Killed

% ps
   PID TTY TIME CMD
    42 8   0:12 -csh
  1038 8   0:02 ps
```

SEE ALSO

ps(1), kill(2), signal(2), init(M).

NAME

ld - nonsegmented Z8000 and 8-bit loader

SYNOPSIS

ld [option **]** file **...**

DESCRIPTION

Ld creates load modules for execution under ZEUS and downloading to target hardware.

Ld combines several object files into one load module file. In the process, it resolves external references and searches libraries. In the simplest case, several object files are given and **ld** combines them to produce an executable load module. An object module can also be produced and used as input to a subsequent **ld** run, in which case the **-r** option must be given to preserve the relocation bits.

The output of **ld** is left on the file a.out(5). If the **-o** option is used, the name so specified is used instead a.out; the file has the same format.

If no errors occur during the link, the output file is marked executable.

The argument routines are concatenated in the order specified. In absence of the **-e** option, the entry point of the output is the beginning of the first routine.

If any argument is a library, it is searched once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are linked.

If a routine from a library references another routine in the library, and the library has not been processed by **ranlib**(1),**the**referenced routine must appear after the referencing routine in the library. Thus, the order of modules within libraries is important. If the first member of a library is named __.SYMDEF, then it is understood to be a dictionary for the library, such as one produced by **ranlib**. The dictionary is repeatedly searched to satisfy all possible references.

The symbols _etext, _edata, and _end in assembly language code (etext, edata, and end in C programs) are reserved and cannot be redefined by the user. The symbol _etext refers to the first location above the text section, or the start of the data section. The symbol _edata refers to the first location above initialized data, or the start of the bss section. The symbol _end refers to the first location above all data, and can be used as a starting location for a

dynamic allocation area managed by the user.

Except for **-l**, the options must appear before the file names. The **-l** option must follow the names of any routines which reference it.

To link a program for downloading to a Z8000 development module, the **-o** option must be used to specify an uppercase file name, and the **-b** option must be used to specify a starting address that is 0x4600 or higher. No special name or address is required to link a program for downloading to a Z8 development module.

OPTIONS

-b addr

-bx addr
Set the bottom location for the program, or for the specified section if x is specified. X can be one of **t**, **d**, or b for text, data, and bss, respectively. The address can be specified in decimal, hex, or octal using the standard C language conventions: a leading zero indicates octal, and a leading 0x indicates hex. The address specified must be a multiple of 256. If no section is selected, the bottom applies to all three sections if the program is combined instruction and data, or to data and bss if separate instruction and data. Only one **-t** or **-b** option per section can be specified. Errors can result if sections overlap, or the bottom address causes a section to wrap around.

-d Force definition of common storage even if the **-r** flag is present.

-e name
Take the following argument as the name of the entry point of the loaded program. The link address of the text section is the default. **-i** Separate the program text and data (also called instruction and data) areas when the output file is executed.

-lx Search the named library. The library /lib/libx.a, is sought, if this is inaccessible or missing /usr/lib/libx.a, is searched followed by /z/bin/libx.a. The specifed library file is used as though its full name had been used instead of the **-l** option. A library is searched when its name is encountered, so the placement of a **-l** option is significant.

-o name
Change the name of the **ld** output file to the name specified. The form of the load module remains the

same as described in a.out(5).

-r Generate relocation bits in the output file so that it can be used in a subsequent **ld** run. This flag overrides the **-t** and **-b** options. It also prevents final definitions from being given to common symbols, and suppresses the undefined symbol diagnostics. The **-r** flag may not be used to generate 8-bit object files.

-s Strip the output: remove the symbol table and relocation bits to save space.

-t addr

-tx addr
Set the highest location of the program or section to the hex, octal, or decimal number specified. X can be one of t, **d**, or b for text, data, and bss respectively. This option is similar to that of **-b** except a top address is specified instead of a bottom one. The link location of the program or section is justified to meet the specified top. The low address of the section or program is always a multiple of 256. In most cases, the top is within 256 bytes of the specified address, and is padded with zeros to meet the address.

-u Take the following argument as a symbol and enter it as undefined in the symbol table. This is useful for linking wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the link of the first routine.

-w Suppress the symbol redefinition warning. This warning is produced while searching archives. If an archive contains a module that defines a symbol that is already defined, a redefinition warning is produced.

-x Enter only external and global symbols and do not preserve local symbols in the output symbol table. This option saves some space in the output file.

-X Save local symbols except for section name entries in the symbol table and for those whose names begin with L. This option is used by the C compiler to discard internally generated labels while retaining symbols local to routines.

FILES

/lib, /usr/lib libraries for -l option
a.out output file

SEE ALSO

load(1), ar(1), ranlib(1), a.out(5).

DIAGNOSTICS

Any undefined references cause the special symbol "_end" to be reported as undefined. The "redefinition" warning occurs if, while searching an archive, a symbol that is already defined is defined in an archive member. This warning can be suppressed with the **-w** option. It does not occur if the symbol name begins with two underscores (one underscore in C).

LIMITATIONS

The "redefinition" warning should not occur on symbols that occur twice within the same archive. The double underscore exception is a kludge.

NAME

learn - on-line computer-aided instruction

SYNOPSIS

learn [subject [lesson]]

DESCRIPTION

Learn gives practice in the use of ZEUS by providing a number of CAI courses on the system. To get started, simply type "learn" and follow the instructions. The strength of the **learn** facility is learning by doing, rather than by reading about the system.

To go directly to a specific subject, specify the subject name on the command line, or specify both the subject and lesson number. The subjects are:

ftuser (first-time user)
files (intro to file usage)
morefiles (more detail)
editor (line-oriented editor ed)
C (programming in C)
macros (-ms package for text formatting)

Other subjects can be supplied on the system. Each site can prepare and provide local courses that serve a particular audience. For a local course, the site can install a new directory containing lessons for that subject.

For debugging lessons, there are a few additional options. If the lesson number is minus (-), **learn** prompts for each lesson. Also, the first option to **learn** can be -directory, followed by subject and lesson, in which case a lesson script can be exercised anywhere.

The special command bye terminates a **learn** session prematurely.

FILES

/usr/lib/learn and all subdirectories such as subjects (ftuser, files, morefiles, etc.) and play, which contain subdirectories for individual learn sessions

NAME

lex - generate programs for simple lexical tasks

SYNOPSIS

lex [**-tvfn**] [file] ...

DESCRIPTION

Lex generates programs to be used in simple lexical analysis of text.

The input files (standard input default) contain strings and expressions to be searched for, and C text to be executed when strings are found.

A C source program lex.yy.c is generated, to be compiled thus:

```
cc -u _main lex.yy.c -ll
```

This program copies the input to the output except when a string specified in the file is found; then the corresponding program text is executed. The actual string matched is left in yytext, an external character array.

Matching is done in order of the strings in the file. The strings may contain square brackets to indicate character classes, as in [abx-z] to indicate a a, b, x, y and z and the operators *, +, and ? mean respectively any non-negative number of, any positive number of, and either zero or one occurrences of, the previous character or character class. The character . is the class of all ASCII characters except new-line. Parentheses for grouping and vertical bar for alternation are also supported.

The notation r { d , e } in a rule indicates between d and e instances of regular expression r. It has higher precedence than |, but lower than *, ?, +, and concatenation. The character ^ at the beginning of an expression permits a successful match only immediately after a new-line, and the character $ at the end of an expression requires a trailing new-line. The character / in an expression indicates trailing context; only the part of the expression up to the slash is returned in yytext, but the remainder of the expression must follow in the input stream. An operator character may be used as an ordinary symbol if it is within " symbols or preceded by \. Thus [a-zA-Z]+ matches a string of letters.

Three subroutines defined as macros are expected: **input()** to read a character; **unput(c)** to replace a character read; and **output(c)** to place an output character. They are defined in terms of the standard streams, but you can override them.

The program generated is named **yylex()**, and the library contains a **main()** which calls it. The action REJECT on the right side of the rule causes this match to be rejected and the next suitable match executed; the function **yymore()** accumulates additional characters into the same yytext; and the function **yyless(p)** pushes back the portion of the string matched beginning at p, which should be between yytext and yytext+yyleng. The macros input and output use files **yyin** and **yyout** to read from and write to, defaulted to **stdin** and **stdout**, respectively.

Any line beginning with a blank is assumed to contain only C text and is copied; if it precedes **%%** it is copied into the external definition area of the lex.yy.c file. All rules should follow a **%%**, as in YACC. Lines preceding **%%** which begin with a non-blank character define the string on the left to be the remainder of the line; it can be called out later by surrounding it with **{}**.

Note that curly brackets do not imply parentheses; only string substitution is done.

EXAMPLES

```
D       [Ø-9]
%%
if      printf("IF statement\n");
[a-z]+  printf("tag, value %s\n",yytext);
Ø{D}+   printf("octal number %s\n",yytext);
{D}+    printf("decimal number %s\n",yytext);
"++"    printf("unary op\n");
"+"     printf("binary op\n");
"/*"    {       loop:
                while (input() != '*');
                switch (input())
                        {
                        case '/': break;
                        case '*': unput('*');
                        default: go to loop;
                        }
                }
```

The following **lex** program converts upper case to lower, removes blanks at the end of lines, and replaces multiple blanks by single blanks.

```
%%
[A-Z] putchar(yytext[Ø]+'a'-'A');
[ ]+$ ;
[ ]+  putchar(' ');
```

The external names generated by **lex** all begin with the prefix **yy** or **YY**.

The options have the following meanings.

-t Place the result on the standard output instead of in file lex.yy.c .

-v Print a one-line summary of statistics of the generated analyzer.

-n Opposite of **-v**; **-n** is default;

-f `Faster' compilation: don't bother to pack the resulting tables; limited to small programs.

Multiple files are treated as a single file. If no files are specified, standard input is used.

Certain table sizes for the resulting finite state machine can be set in the definitions section:

%p n number of positions is n (default 2000)

%n n number of states is n (500)

%t n number of parse tree nodes is n (1000)

%a n number of transitions is n (3000)

The use of one or more of the above automatically implies the **-v** option, unless the **-n** option is used.

SEE ALSO

yacc(1).
LEX - Lexical Analyzer Generator in the ZEUS Languages / Programming Tools Manual

NAME

line - read one line from the terminal

SYNOPSIS

line

DESCRIPTION

Line copies one line (up to a new-line) from the standard input and writes it on the standard output. It returns an exit code of 1 on **EOF** and always prints at least a new-line. It is often used within shell files to read from the user's terminal.

SEE ALSO

gets(1C), sh(1), read(2).

NAME

lint - a C program verifier

SYNOPSIS

lint [**-abchnpuvx**] file ...

DESCRIPTION

Lint detects C program bugs and checks the type usage of the program more strictly than the compilers. Among the things which are currently found are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions which return values in some places and not in others, functions called with varying numbers of arguments, and functions whose values are not used.

Lint assumed that all files are loaded together; they are checked for mutual compatibility. Function definitions for certain libraries are available to **lint**; these libraries are referred to by a conventional name, such as `-lm', in the style of **ld**(1).

Exit(2) and other functions which do not return are not understood; this causes various lies.

Certain conventional comments in the C source will change the behavior of **lint:**

/*NOTREACHED*/
: at appropriate points stops comments about unreachable code.

/*VARARGSn*/
: suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first n arguments are checked; a missing n is taken to be 0.

/*NOSTRICT*/
: shuts off strict type checking in the next expression.

/*ARGSUSED*/
: turns on the **-v** option for the next function.

/*LINTLIBRARY*/
: at the beginning of a file shuts off complaints about unused functions in this file.

OPTIONS

Any number of the options in the following list may be used. The **-D**, **-U**, and **-I** options of **cc**(1) are also recognized as

separate arguments.

a Report assignments of long values to int variables.

b Report **break** statements that cannot be reached. (This is not the default because, unfortunately, most **lex** and many **yacc** outputs produce dozens of such comments.)

c Complain about casts which have questionable portability.

h Apply a number of heuristic tests to intuit bugs, improve style, and reduce waste.

n Do not check compatibility against the standard library.

p Attempt to check portability to the IBM and GCOS dialects of C.

u Do not complain about functions and variables used and not defined, or defined and not used (this is suitable for running **lint** on a subset of files out of a larger program).

v Suppress complaints about unused arguments in functions.

x Report variables referred to by extern declarations, but never used.

FILES

/usr/lib/lint[12] programs
/usr/lib/llib-lc declarations for standard functions
/usr/lib/llib-port declarations for portable functions

SEE ALSO

cc(1).
Lint, a C Program Checker in the ZEUS Languages / Programming Tools Manual

NAME

ln - link a filename to an actual file

SYNOPSIS

ln name1 name2

DESCRIPTION

A link is a directory entry referring to a file; the same file (together with its size, all its protection information, etc.) can have several links to it. There is no way to distinguish a link to a file from its original directory entry; any changes in the file are effective independently of the name by which the file is known.

A link is created from name2 to name1.

It is forbidden to link to a directory or to link across file systems.

SEE ALSO

rm(1).

NAME

LOAD - Download to Z8000 or Z8 Development Module

SYNOPSIS

LOAD file

DESCRIPTION

LOAD takes an executable Z8000 or Z8 load module and downloads the text, data, and bss sections into the Z8000 or Z8 Development Module (DM). The hardware link is described fully in the appropriate DM manual. The magic number of the load module must be N_MAGIC1, X_MAGIC1, or X_MAGIC3 (a.out (5)),. The execute permission bit is checked to ensure that the load module contains no errors.

LOAD determines the load points of the text and data by inspecting the header and the segment table, which contains the upper eight bits of each load point. Therefore, the text, data, and bss sections are loaded only on 256-byte boundaries. The load module must be specified to link at location 4600 hex or higher if the target system is a non-segmented Z8000 DM or at location 5000 hex or higher if the target system is a segmented Z8000 DM. These restrictions do not apply if the target system is a Z8 DM.

The bss section of Z8 programs refers to the REGISTER memory of the Z8. LOAD does not download bss section of a Z8 program. If a Z8 program has combined instruction and data (I and D) areas the data section is downloaded following the code section; the data section is considered to be part of program memory. If a Z8 program has separate I and D areas, the data section is not downloaded; the data section refers to external data memory.

If a file is loaded to a Z8000 DM without error, the DM announces the entry point of the program; if a file is loaded to a Z8 DM without error, the DM monitor simply prints a prompt. To start the program jump to the entry point with the monitor JUMP command. If the program runs and terminates normally, the exit routine linked into the program returns the user to the monitor. The Z8 DM provides no exit routine: for details on program execution consult the Z8 Development Module User's Manual.

LOAD is invoked from the DM monitor. On a Z8000 DM, lowercase characters can be entered at the terminal but are translated to uppercase by the monitor. The monitor sends only uppercase characters to ZEUS for compatibility with Zilog's MCZ/ZDS RIO system. Therefore, the file to be downloaded is not found unless the the load module name is all in uppercase. Also, unless the path name is all in uppercase, the user must use **LOAD** only on a load module file in

the current working directory.

The Z8 DM does not require uppercase input and the monitor makes no translation. No special care need be taken naming files or running the program from the Z8 DM.

FILES

a.out load module

SEE ALSO

ld(1), SEND(1), a.out(5)
Z8000 Development Module Hardware Reference Manual, 03-0394-01
Z8000 Development Module Hardware Reference Manual Errata, E3-0394-01
Z8000 Development Module Monitor Program Listing, 03-3094-01
Z8000 Development Module Monitor Program Listing Errata, E3-3094-01
Z8 Development Module User's Manual, 03-3157

DIAGNOSTICS

The DM monitor displays any error messages received from LOAD. The error messages are the same as for LOAD in the DM manual.

DOWNLOAD PROCEDURE FOR THE Z8000 DM

1. Install the Zilog Z8000 DM between the terminal and the System 8000. The line from the System 8000 attaches to the channel A RS-232C connection of the DM. The terminal attaches to channel B of the DM.

2. Power up the DM and turn on the terminal.

3. Press DM RESET switch. The DM monitor prints its < prompt.

4. Press DM NMI switch. The DM monitor prints NMI and its < prompt.

5. Enter quit. This command allows transparent mode for communication with ZEUS.

6. Press the RETURN key again, to get a response from ZEUS. Login to the ZEUS system if you aren't already correctly logged in.

7. Create a Z8000 program for download. The following simple program can be used for a download example:

```
main()
```

```
{
printf("DOWNLOAD OK\n");
}
```

8. Compile, assemble, and link the program. The C compiler might be run to do this with a command such as:

   cc -o TEST -b Øx46ØØ test.c

9. Press the DM NMI switch to return control to the monitor. The DM monitor responds with NMI and its prompt <.

1Ø. Enter

   LOAD TEST

   This downloads the program.

11. Wait for the DM monitor to print ENTRY POINT 4DBA and the < prompt. Enter

   jump 4dba

   This jumps to the entry point of the program.

12. The program executes. The monitor prints DOWNLOAD OK and its prompt <.

At this point, enter other monitor commands or go back to ZEUS by entering the quit command.

DOWNLOAD PROCEDURE FOR THE Z8 DM

1. Install the Zilog Z8 DM between the terminal and the System 8ØØØ. The line from the System 8ØØØ attaches to the DM RS-232C connection marked "computer." The terminal attaches to the DM connection marked "terminal."

2. Power up the DM and turn on the terminal.

3. Press DM RESET switch. The DM monitor prints its @ prompt.

4. Enter quit. This command allows transparent mode for communication with ZEUS.

5. Press the RETURN key again, to get a response from ZEUS. Login to the ZEUS system if you aren't already correctly logged in.

6. Create a Z8 program for download. The following simple program can be used for a download example:

```
test module

global
     p1 procedure
     entry
          srp #%1Ø
     11:  jr 11
     end p1

end test
```

7. Assemble and link the program. The Z8 cross-assembler and the loader would be run with commands such as:

```
z8as -o test.o test.s
ld -o test test.o
```

8. Press DM RESET and MODE switches. The monitor prompts @.

9. Download the program with the command:

```
LOAD test
```

1Ø. Wait for the DM monitor prompt, @. Enter

```
G Ø
```

This jumps to the entry point of the program.

11. The program executes and enters an infinite loop. Enter H to return to the monitor.

12. Enter R. The register pointer has been set to %1Ø.

At this point, enter other monitor commands or enter quit to return to ZEUS.

NAME

local - return control to local system

SYNOPSIS

local [-l]

DESCRIPTION

Local returns control to the local system which is running the ZLAB-8000 Communication Package file transfer software.

OPTIONS

-l A "logout" is given to the remote system before returning to the local system.

FILES

/usr/src/local.c C source program for local

SEE ALSO

getfile(1), putfile(1), remote(1).
COMM -- The ZEUS Communications Package in the ZEUS Utilities Manual

NAME

login - sign on to the computer

DESCRIPTION

Login can no longer be invoked explicitly, but is invoked by the system when a connection is established.

Echoing is turned off (if possible) during the typing of the password, so it does not appear on the written record of the session.

If password aging has been invoked by the super-user the password may have expired. In this case, **passwd**(1) is invoked to change it.

A dial-up **login** attempt must complete the login within one minute or the connection is broken.

After a successful login, accounting files are updated, the user is informed of the existence of mail, and the message-of-the-day file is displayed (**motd**(5)). **Login** initializes the environment variable TERM (describing the login terminal type), the user and group IDs, and the working directory, then executes a command interpreter (usually **csh**(1)) according to specifications found in a password file. Argument Ø of the command interpreter is -csh.

The environment (see **environ**(5)) is initialized to:

HOME=login-directory
PATH=:/bin:/usr/bin
LOGNAME=login-name

OPTIONS

- Turns off the display of message-of-the-day file.

FILES

/etc/utmp	accounting
/usr/adm/wtmp	accounting
/usr/spool/mail/*	mail
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/profile	system profile
/etc/cshrc	system cshrc
/etc/ttytype	terminal type file

SEE ALSO

newgrp(1), mail(1), passwd(1), motd(5), passwd(5), ttytype(5), init(M), getty(M).
ZEUS for Beginners in the ZEUS Utilities Manual

DIAGNOSTICS

"Login incorrect," if the name or the password is bad.

"No shell," if the shell is missing.

"Cannot open password file," if the password file is missing.

"No directory," cannot find the home directory (usually due to unmounted file systems).

NAME

logname - get login name

SYNOPSIS

logname

DESCRIPTION

Logname returns the contents of the environment variable **$LOGNAME**, which is set when a user logs into the system.

FILES

/etc/profile

SEE ALSO

env(1), login(1), logname(3), environ(5).

NAME

logout - terminate the current login session

SYNOPSIS

logout

DESCRIPTION

Logout terminates a login shell. Especially useful if ignoreeof is set.

LIMITATIONS

Logout will only execute from the login shell.

the logout process does not recognize interupts

SEE ALSO

exit(1C), kill(1), onintr(1C).
The C Shell in the ZEUS Utilities Manual

NAME

look - find lines in a sorted list

SYNOPSIS

look [**-df**] *string* [*file*]

DESCRIPTION

Look consults a sorted *file* and prints all lines that begin with *string*. It uses binary search. If no *file* is specified, */usr/dict/words* is assumed with collating sequence **-df**.

OPTIONS

The options **-d** and **-f** affect comparisons as in **sort**(1):

-d Dictionary order: only letters, digits, tabs, and blanks are compared.

-f Fold. Uppercase letters compare equal to lowercase.

FILES

/usr/dict/words

SEE ALSO

sort(1), grep(1).

NAME

lorder - find ordering relation for an object library

SYNOPSIS

lorder file ...

DESCRIPTION

The input is one or more object or library archive (see **ar**(1)) files. The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by **tsort**(1) to find an ordering of a library suitable for one-pass access by **ld**(1).

This brash one-liner intends to build a new library from existing `.o' files.

```
ar cr library `lorder *.o | tsort`
```

FILES

*symref, *symdef

SEE ALSO

ar(1), ld(1), tsort(1).

LIMITATIONS

The names of object files, in and out of libraries, must end with `.o'; nonsense results otherwise.

NAME

lpr - line printer spooler

SYNOPSIS

lpr [option] ... [file] ...

DESCRIPTION

Lpr is now linked to **nq(1).** Please refer to this page in the manual for more information.

NAME

ls - list the contents of a directory

SYNOPSIS

ls [-aAcCdDfFgilmnqrRstuxl] file ...

DESCRIPTION

For each file argument, **ls** repeats its name and any other information requested; for each directory argument, **ls** lists the contents of the directory. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

The mode information printed under the -l option contains 10 characters. The first character is interpreted as follows:

d	directory
b	block-type special file
c	character-type special file
p	named pipe
-	plain file

The next nine characters are interpreted as three sets of permission bits. The first set is permissions for the owner of the file or directory; the next set is for others in the same user-group; and the last set is for all other users. Within each set, the three characters indicate permission respectively to read, write on, or execute the file as a program. For a directory, execute permission is interpreted to mean permission to search the directory for a specified file. The permissions are as follows:

r	readable
w	writable
x	executable
-	permission is not granted

The group-execute permission character is given as **s** if the file has set-group-ID mode; likewise the user-execute permission character is given as **s** if the file has set-user-ID mode.

The last character of the mode (normally x or -) is **t** if the 1000 bit of the mode is on. See **chmod**(1) for the meaning of this mode.

There are three possible listing formats. The format chosen depends on whether the output is going to a terminal, and can also be controlled by option flags. (1) The default format for a terminal is to list the contents of directories

in multicolumn format, with the entries sorted down the columns (changed with the **-x** option). (2) If the standard output is not a terminal, the default format is to list one entry per line. (3) Finally, there is a stream output format in which files are listed across the page, separated by commas (changed with the **-m** option). Files which are not the contents of a directory being interpreted are always sorted across the page rather than down the page in columns. This is because the individual file names are arbitrarily long.

OPTIONS

-a List all entries. Without this option, files beginning with . are omitted.

-A Like **-a**, but . and .. are suppressed. For the superuser, this option is on by default, and **-A** suppresses . listings.

-c Use time of last modification to i-node (mode, etc.) instead of last modification to file for sorting (**-t**) or printing (**-l**).

-C Force multicolumn output (default if output device is a terminal).

-d If argument is a directory, list only its name, not its contents (mostly used with **-l** to get status on directory).

-D List only directories.

-f Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **-l**, **-t**, **-s**, and **-r**, and turns on **-a**; the order is the order in which entries appear in the directory.

-F Indicate directories by appending a / to the filename in the listing, and executable files by appending a ***.**

-g Give group ID instead of owner ID in long listing; use only with the **-l** function.

-i Print i-number in first column of the report for each file listed.

-l List in long format, giving permission mode bits, number of links, owner, group, size in bytes, and time of last modification for each file. The year is printed (instead of the time in hours and minutes) for files older than six months. For a directory, the

total count of blocks (including indirect blocks) is printed. For a special file, the size field instead contains the major and minor device numbers.

-m Force stream output format.

-n In long listings, give numeric uid or group id instead of name of user or group.

-q Force printing of nongraphic characters in file names as the character ? (default if output device is a terminal).

-r Reverse the order of sort to get reverse alphabetic or oldest first.

-R Recursively list the contents of each directory found.

-s Give size in blocks, including indirect blocks, for each entry.

-t Sort by time modified (latest first) instead of by name, as is normal.

-u Use time of last access instead of last modification for sorting (**-t**) or printing (**-l**).

-x Force columnar printing to be sorted across rather than down the page (default if the last character of the name the program is invoked with is not an l or an s).

-1 Force one entry per line output format (default if output is redirected or piped).

FILES

/etc/passwd to get user ID's for "ls -l"
/etc/group to get group ID's for "ls -lg"

SEE ALSO

file(1), find(1), chmod(1), chown(1), umask(1C), chmog(M), chown(M).

LIMITATIONS

New line and tab are considered printing characters in file names.

NAME

m4 - macro processor

SYNOPSIS

m4 [files]

DESCRIPTION

M4 is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no arguments, or if an argument is `-', the standard input is read. The processed text is written on the standard output.

Macro calls have the form

name(arg1,arg2, . . . , argn)

The `(' must immediately follow the name of the macro. If a defined macro name is not followed by a `(', it is deemed to have no arguments. Leading unquoted blanks, tabs, and newlines are ignored while collecting arguments. Potential macro names consist of alphabetic letters, digits, and underscore `_', where the first character is not a digit.

Left and right single quotes (``') are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

M4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

define — The second argument is installed as the value of the macro whose name is the first argument. Each occurrence of $n in the replacement text, where n is a digit, is replaced by the n-th argument. Argument Ø is the name of the macro; missing arguments are replaced by the null string.

undefine — Removes the definition of the macro named in its argument.

ifdef — If the first argument is defined, the value is the

second argument, otherwise the third. If there is no third argument, the value is null. The word unix is predefined on UNIX versions of m4.

changequote
Change quote characters to the first and second arguments. Changequote without arguments restores the original values (i.e., `').

divert M4 maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The divert macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.

undivert Causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.

divnum Returns the value of the current output stream.

dnl Reads and discards characters up to and including the next newline.

ifelse Has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.

incr Returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

eval Evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, -, *, /, %, ^ (exponentiation); relationals; parentheses.

len Returns the number of characters in its argument.

index Returns the position in its first argument where the second argument begins (zero origin), or -1 if the second argument does not occur.

substr Returns a substring of its first argument. The

second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.

translit Transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.

include Returns the contents of the file named in the argument.

sinclude Is identical to include, except that it says nothing if the file is inaccessible.

syscmd Executes the UNIX command given in the first argument. No value is returned.

maketemp Fills in a string of XXXXX in its argument with the current process id.

errprint Prints its argument on the diagnostic output file.

dumpdef Prints current names and definitions, for the named items, or for all if no arguments are given.

SEE ALSO

The M4 Macro Processor in the ZEUS Languages / Programming Tools Manual

NAME

mail, rmail - send and receive mail among users

SYNOPSIS

mail [**-pqr**] [**-f** file]

mail persons

rmail persons

DESCRIPTION

Mail without arguments prints a user's mail, message-by-message, in last-in, first-out order. For each message, the user is prompted with a **?**, and a line is read from the standard input to determine the disposition of the message:

d Delete message and go on to next message.

EOT (control-D)
Put undeleted mail back in the mailfile and stop.

m[persons]
Mail the message to the named persons (yourself is default).

<new-line> Go on to next message.

p Print message again.

q Same as **EOT.** and stop.

s[file] Save message in the named files (**mbox** is default).

w[files] Save message, without its header, in the named files (**mbox** is default).

x Put all mail back in the mailfile unchanged and stop.

+ Same as <new-line>.

- Go back to previous message.

!command Escape to the shell to do command.

***** Print a command summary.

When persons are named, **mail** takes the standard input up to an end-of-file (or a line consisting of just a .) and adds it to each person's mailfile. The message is preceded by the sender's name and a postmark. Lines that look like

postmarks in the message, (i.e., "From...") are preceded with a **>**. A person is usually a user name recognized by **login**(1). If a person being sent mail is not recognized, or if **mail** is interrupted during input, the **dead.letter** will be saved to allow editing and resending. It will be placed in the current working directory.

To denote a recipient on a remote system, prefix person by the system name and exclamation mark (see **uucp**(1)). Everything after the first exclamation mark in persons is interpreted by the remote system. In particular, if persons contains additional exclamation marks, it can denote a sequence of machines through which the message is to be sent on the way to its ultimate destination. For example, specifying **a!b!cde** as a recipient's name causes the message to be sent to user **b!cde** on system **a**. System **a** will interpret that destination as a request to send the message to user **cde** on system **b**. This is useful if the sending system can access system **a** but not system **b**, and system **a** has access to system **b**.

The mailfile can be manipulated in two ways to alter the function of **mail**. The other permissions of the file may be read-write, read-only, or neither read nor write to allow different levels of privacy. If changed to other than the default, the file will be preserved even when empty to perpetuate the desired permissions. The file can also contain the first line:

Forward to person

which causes all mail sent to the owner of the mailfile to be forwarded to person. This is especially useful to forward all of a person's mail to one machine in a multiple machine environment.

Rmail only permits the sending of mail; **uucp**(1) uses **rmail** as a security precaution.

When a user logs in he is informed of the presence of mail, if any.

OPTIONS

-ffile
: causes **mail** to use file (e.g., **mbox**) instead of the default mailfile.

-p
: causes all mail to be printed without prompting for disposition.

-q
: causes **mail** to terminate after interrupts. Normally an interrupt only causes the termination of the message

being printed.

-r causes messages to be printed in first-in, first-out order.

FILES

/etc/passwd	to identify persons
/usr/spool/mail/*	incoming mail for user *
$HOME/mbox	saved mail
$MAIL	mailfile
/tmp/ma*	temporary file
/usr/spool/mail/*.lock	lock for mail directory
dead.letter	unmailable text

SEE ALSO

login(1), uucp(1), write(1).

LIMITATIONS

Race conditions sometimes result in a failure to remove a lock file.
After an interrupt, the next message may not be printed; printing may be forced by typing a **p**.

NAME

make - maintain, update, and regenerate groups of programs

SYNOPSIS

make [**-bdeikmnpqrst**] [-f makefile] [names]

DESCRIPTION

Make executes commands in makefile to update one or more target names. Name is typically a program. If no -f option is present, **makefile**, **Makefile**, **makefile+** and **Makefile+** are tried in order. If makefile is -, the standard input is taken. More than one -f makefile argument pair may appear.

Make updates a target only if it depends on files that are newer than the target. All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out of date.

Makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, then a **:**, then a (possibly null) list of prerequisite files or dependencies. Text following a **;** and all following lines that begin with a tab are shell commands to be executed to update the target. The first line that does not begin with a tab or **#** begins a new dependency or macro definition. Shell commands may be continued across lines with the <backslash><new-line> sequence. Sharp (**#**) and new-line surround comments.

The following makefile says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) and a common file **incl.h**:

```
pgm: a.o b.o
     cc a.o b.o -o pgm
a.o: incl.h a.c
     cc -c a.c
b.o: incl.h b.c
     cc -c b.c
```

Command lines are executed one at a time, each by its own shell. A line is printed when it is executed unless the **-s** option is present, or the entry **.SILENT:** is in makefile, or unless the first character of the command is **@**. The **-n** option specifies printing without execution; however, if the command line has the string **$(MAKE)** in it, the line is always executed (see discussion of the **MAKEFLAGS** macro under Environment). The **-t** (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate **make**. If the **-i** option is present, or the entry **.IGNORE:** appears in makefile, or if the line specifying the command begins with <tab><hyphen>, the error is ignored. If the **-k** option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The -b option allows old makefiles (those written for the old version of **make**) to run without errors. The difference between the old version of **make** and this version is that this version requires all dependency lines to have a (possibly null) command associated with them. The previous version of **make** assumed if no command was specified explicitly that the command was null.

Interrupt and quit cause the target to be deleted unless the target depends on the special name **.PRECIOUS.**

OPTIONS

-b Compatibility mode for old makefiles.

-d Debug mode. Print out detailed information on files and times examined.

-e Environment variables override assignments within makefiles.

-f makefile
Description file name. Makefile is assumed to be the name of a description file. A file name of - denotes the standard input. The contents of makefile override the built-in rules if they are present.

-i Ignore error codes returned by invoked commands. This mode is entered if the fake target name **.IGNORE** appears in the description file.

-k Abandon work on the current entry, but continue on other branches that do not depend on that entry.

-n No execute mode. Print commands, but do not execute them. Even lines beginning with an @ are printed.

-p Print out the complete set of macro definitions and target descriptions.

-q Question. The **make** command returns a zero or non-zero status code depending on whether the target file is or is not up-to-date.

-r Do not use the built-in rules.

-s Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name **.SILENT** appears in the description file.

-t Touch the target files (causing them to be up-to-date) rather than issue the usual commands.

.DEFAULT
If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name **.DEFAULT** are used if it exists.

.IGNORE
Same effect as the **-i** option.

.PRECIOUS
Dependents of this target will not be removed when quit or interrupt are hit.

.SILENT
Same effect as the **-s** option.

ENVIRONMENT

The environment is read by make. All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any makefile and after the internal rules; thus, macro assignments in a makefile override environment variables. The **-e** option causes the environment to override the macro assignments in a makefile.

The **MAKEFLAGS** environment variable is processed by **make** as containing any legal input option (except **-f**, **-p**, and **-d**) defined for the command line. Further, upon invocation, **make** "invents" the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, **MAKEFLAGS** always contains the current input options. This proves very useful for "supermakes". In fact, as noted above, when the **-n** option is used, the command **$(MAKE)** is executed anyway; hence, one can perform a **make -n** recursively on a whole software system to see what would have been executed. This is because the **-n** is put in **MAKEFLAGS** and passed to further invocations of **$(MAKE)**. This is one way of debugging all of the makefiles for a software project without actually doing anything.

Macros

Entries of the form string1 = string2 are macro definitions. Subsequent appearances of $(string1[:subst1=[subst2]]) are replaced by string2. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional :subst1=subst2 is a substitute sequence. If it is specified, all non-overlapping

occurrences of subst1 in the named macro are replaced by subst2. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. An example of the use of the substitute sequence is shown under Libraries.

Internal Macros

There are five internally maintained macros which are useful for writing rules for building targets.

$*** The macro **$* stands for the file name part of the current dependent with the suffix deleted. It is evaluated only for inference rules.

$@** The **$@ macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.

$<** The **$< macro is only evaluated for inference rules or the **.DEFAULT** rule. It is the module which is out of date with respect to the target (i.e., the "manufactured" dependent file name). Thus, in the **.c.o** rule, the **$<** macro would evaluate to the **.c** file. An example for making optimized **.o** files from **.c** files is:

```
.c.o:
        cc -c -O $*.c
```

or:

```
.c.o:
        cc -c -O $<
```

$?** The **$? macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are out of date with respect to the target; essentially, those modules which must be rebuilt.

$% The **$%** macro is only evaluated when the target is an archive library member of the form **lib(file.o)**. In this case, **$@** evaluates to **lib** and **$%** evaluates to the library member, **file.o**.

Four of the five macros can have alternative forms. When an upper case **D** or **F** is appended to any of the four macros the meaning is changed to "directory part" for **D** and "file part" for **F**. Thus, **$(@D)** refers to the directory part of the string **$@**. If there is no directory part, **./** is generated. The only macro excluded from this alternative form is **$?**. The reasons for this are debatable.

Suffixes
Certain names (for instance, those ending with **.o**) have inferable prerequisites such as **.c**, **.s**, etc. If no update commands for such a file appear in makefile, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, **make** has inference rules which allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. The current default inference rules are:

```
.c .c+ .f .f+ .sh .sh+ .c.o .c+.o .c+.c .f.o .f+.o
.f+.f .s.o .s+.o .s+.s .y.o .y+.o .y+.c .y.c .c.a .c+.a
.s+.a
```

To print out the rules compiled into the **make** on any machine, the following command is used:

```
make -fp - 2>/dev/null </dev/null
```

The only peculiarity in this output is the **(null)** string which **printf**(3) prints when handed a null string. **(Cshell**(1) users will have to redirect the standard output to a file and ignore the errors printed out on the terminal.)

A rule with only one suffix (i.e. **.c:**) is the definition of how to build x from x **.c.** In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., shell procedures, simple C programs).

Additional suffixes are given as the dependency list for **.SUFFIXES.** Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite.

The default list is:

```
.SUFFIXES: .o .c .c+ .y .y+ .f .f+ .p .p+ .s .s+ .sh
.sh+ .h .h+
```

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; **.SUFFIXES:** with no dependencies clears the list of suffixes.

Inference Rules
The first example can be done more briefly:

```
pgm: a.o b.o
     cc a.o b.o -o pgm
a.o b.o: incl.h
```

This is because **make** has a set of internal rules for building files. The user may add rules to this list by simply putting them in the makefile .

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, **CFLAGS** and **YFLAGS** are used for compiler options to **cc**(1) and **yacc**(1) respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix **.o** from a file with suffix **.c** is specified as an entry with **.c.o:** as the target and no dependents. Shell commands associated with the target define the rule for making a **.o** file from a **.c** file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

Libraries

If a target or dependency name contains parenthesis, it is assumed to be an archive library, the string within parenthesis referring to a member within the library. Thus **lib(file.o)** and **$(LIB)(file.o)** both refer to an archive library which contains **file.o**. (This assumes the **LIB** macro has been previously defined.) The expression **$(LIB)(file1.o file2.o)** is not legal. Rules pertaining to archive libraries have the form **.XX.a** where the XX is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires the XX to be different from the suffix of the archive member. Thus, one cannot have **lib(file.o)** depend upon **file.o** explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib: lib(file1.o) lib(file2.o) lib(file3.o)
     @echo lib is now up to date
.c.a:
     $(CC) -c $(CFLAGS) $<
     ar rv $@ $*.o
     rm -f $*.o
```

In fact, the **.c.a** rule listed above is built into **make** and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib: lib(file1.o) lib(file2.o) lib(file3.o)
     $(CC) -c $(CFLAGS) $(?:.o=.c)
     ar rv lib $?
     rm $?
     @echo lib is now up to date
```

```
.c.a:;
```

Here the substitution mode of the macro expansions is used. The **$?** list is defined to be the set of object file names (inside **lib**) whose C source files are out of date. The substitution mode translates the **.o** to **.c**. Note also, the disabling of the **.c.a:** rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

FILES

[Mm]akefile
[Mm]akefile+

SEE ALSO

chkin(1), chkout(1), csh(1), sh(1), touch(1).
Make - A Program for Maintaining Computer Programs in the ZEUS Languages / Programming Tools Manual

LIMITATIONS

Some commands return non-zero status inappropriately; use **-i** to overcome the difficulty. Commands that are directly executed by the shell, notably **cd**(1), are ineffectual across new-lines in **make**. The syntax **lib(file1.o file2.o file3.o)** is illegal. You cannot build **lib(file.o)** from **file.o**.

NAME

man - print sections of this manual

SYNOPSIS

man [**option** ...] [chapter] title ...

DESCRIPTION

Man locates and prints the section of this manual named title in the specified chapter. (In this context, the word "page" is often used as a synonym for "section.") The title, is entered in lowercase. The chapter number does not need a letter suffix. If no chapter is specified, the whole manual is searched for title and all occurrences of it are printed.

OPTIONS

-e Appended or prefixed to any of the above causes the manual section to be preprocessed by **neqn** or **eqn**(1); **-e** alone means **-te**.

-h Send the manual entry to the output device using the program "cat" rather than the default "more." This is intended primarily for hardcopy terminals.

-n Print the section on the standard output using **nroff**(1).

-t Phototypeset the section using **troff**(1).

-v Send the manual entry to the output device using the program "view" rather than the default "more."

-w Print the path names of the manual sections, but do not print the sections themselves.

(default)
Copy an already formatted manual section to the terminal, or, if none is available, act as **-n**. It may be necessary to use a filter to adapt the output to the particular terminal's characteristics.

Further options, for example, to specify the kind of terminal you have, are passed on to **troff**(1) or **nroff**(1). Options and chapter can be changed before each title.

EXAMPLES

man man

reproduces this section, as well as any other sections named **man** that can exist in other chapters of the manual, (**man**(7)).

```
man 1 man
```

reproduces only this section.

FILES

/usr/man/man?/*
/usr/man/cat?/*

SEE ALSO

INTRO(Ø), apropos(1), help(1), nroff(1), eqn(1), man(7).

LIMITATIONS

The manual is supposed to be reproducible either on a phototypesetter or on a terminal. However, on a terminal some information is lost.

NAME

mesg - permit or deny messages

SYNOPSIS

mesg [**n**] [**y**]

DESCRIPTION

With no arguments, **mesg** reports the current state without changing it.

OPTIONS

n Forbids messages via **write**(1) or **talk**(1), by revoking non-user write permission on the user's terminal.

y Reinstates permission.

FILES

/dev/tty*
/dev

SEE ALSO

talk(1), write(1).

DIAGNOSTICS

Exit status is Ø if messages are receivable, 1 if not, 2 on error.

NAME

mkdir - make a directory

SYNOPSIS

mkdir dirname ...

DESCRIPTION

Mkdir creates specified directories in mode 777 joined with logical and to the complement of the current **umask** value. The **umask** value is controlled by the **umask** shell command. Standard entries (. for the directory itself, and .. for its parent), are made automatically.

Mkdir requires write permission in the parent directory.

SEE ALSO

csh(1), rm(1), sh(1), umask(2), chmod(1).

DIAGNOSTICS

Mkdir returns exit code Ø if all directories were successfully made. Otherwise, it prints a diagnostic and returns nonzero.

NAME

mknod - build special file

SYNOPSIS

/etc/mknod name [**c**] [**b**] major minor
/etc/mknod name **p**

DESCRIPTION

Mknod makes a directory entry and corresponding i-node for a special file.

Mknod can also be used to create fifo's (a.k.a named pipes) (second case in **SYNOPSIS** above). This use of **Mknod** can be used by any user. The first case can be used only by members of the 'system' group. It is used to create special device files.

The first argument is the name of the entry. In the first case, the second is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the major device type and the minor device (e.g. unit, drive, or line number), which may be either decimal or octal. A leading Ø for the major and minor device numbers mean that they are in octal.

The assignment of major device numbers depends on the position of the driver in dispatch tables in the kernel. The major device numbers for current drivers is as follows:

Device	Character Dev. Major Number	Block Dev. Major Number
ZD	Ø	Ø
CT	1	1
SMD	2	8
MT	3	9
MD	4	1Ø
ERR	5	--
MEM	6	--
TTY	7	--
SIO	8	--
LP	9, 1Ø	--
PTC (unet)	11	--
PTS (unet)	12	--
UP (unet)	13	--
UU (unet)	14	--
UD (unet)	15	--

Device (cont.)	Character Dev. Major Number	Block Dev. Major Number
U1 (user defined dev)	16	2
U2 (user defined dev)	17	3
U3 (user defined dev)	18	4
U4 (user defined dev)	19	5
U5 (user defined dev)	20	6
U6 (user defined dev)	21	7

The minor device number is device dependent. For disks, the minor device number is the number of the file system. The first digit (decimal) corresponds to the drive number and the second digit (decimal) is the order of the file system on the disk. For ttys, the minor device number is the number of the port. For other devices the numbers represent options passed to the drivers (eg. no-rewind for tapes).

SEE ALSO

mknod(M), mknod(2).

NAME

mkstr - create an error message file by massaging C source

SYNOPSIS

mkstr [-] messagefile prefix file ...

DESCRIPTION

Mkstr is used to create files of error messages. Its use can make programs with large numbers of error diagnostics much smaller, and reduce system overhead in running the program as the error messages do not have to be constantly swapped in and out.

Mkstr will process each of the specified files, placing a massaged version of the input file in a file whose name consists of the specified prefix and the original name. A typical usage of **mkstr** would be:

mkstr pistrings xx *.c

This command would cause all the error messages from the C source files in the current directory to be placed in the file pistrings and processed copies of the source for these files to be placed in files whose names are prefixed with xx.

To process the error messages in the source to the message file **mkstr** keys on the string `error("' in the input stream. Each time it occurs, the C string starting at the `"' is placed in the message file followed by a null character and a new-line character; the null character terminates the message so it can be easily used when retrieved, the new-line character makes it possible to sensibly **cat** the error message file to see its contents. The massaged copy of the input file then contains a **lseek** pointer into the file which can be used to retrieve the message, i.e.:

```
char efilname[] =  "/usr/lib/pi_strings";
int  efil = -1;

error(a1, a2, a3, a4)
{
     char buf[256];

     if (efil < 0) {
          efil = open(efilname, 0);
          if (efil < 0) {
oops:
               perror(efilname);
               exit(1);
          }
```

```
        }
        if (lseek(efil, (long) a1, 0) || read(efil, buf, 256) <= 0)
         goto oops;
        printf(buf, a2, a3, a4);
    }
```

OPTIONS

- Causes the error messages to be placed at the end of the specified message file for recompiling part of a large **mkstr**ed program.

SEE ALSO

lseek(2), xstr(1).

LIMITATIONS

All the arguments except the name of the file to be processed are unnecessary.

NAME

mm - the MM macro package for formatting documents

SYNOPSIS

mm [options] [files]

nroff -mm [options] [files]

nroff -cm [options] [files]

mmt [options] [files]

troff -mm [options] [files]

troff -cm [options] [files]

DESCRIPTION

This package provides a formatting capability for a wide variety of documents. It is the standard package used by the BTL typing pools and documentation centers. The manner in which a document is entered and edited is essentially independent of whether the document is to be eventually formatted at a terminal or phototypeset. See the references below for further details.

OPTIONS

-cm Causes nroff(1) and troff(1) to use the compacted version of the macro package, thus speeding up the process of loading.

-mm Results in the use of the non-compacted version of the macro package.

FILES

/usr/lib/tmac/tmac.m	pointer to the non-compacted version of the package
/usr/lib/macros/mm[nt]	non-compacted version of the package
/usr/lib/macros/cmp.[nt].[dt].m	compacted version of the package
/usr/lib/macros/ucmp.[nt].m	initializers for the compacted version of the package

SEE ALSO

troff(1).

NAME

more, page - file perusal filter for crt viewing

SYNOPSIS

more [-dfl*n*] [+*line*] [+/*pat*] [*file* ...]

page [-dfl*n*] [+*line*] [+/*pat*] [*file* ...]

DESCRIPTION

More allows examination of a file on a terminal one screenful at a time. It normally pauses after each screenful, printing:

--More--

at the bottom of the screen. A carriage return displays one more line, a space displays another screenful.

If the -l option is not given, **more** will pause after any line that contains a control-L "^L", as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen will be cleared before the file is printed.

If the program is invoked as **page**, then the screen is cleared before each screenful is printed (but only if a full screenful is being printed), and **k** - 1 rather than **k** - 2 lines are printed in each screenful, where **k** is the number of lines the terminal can display.

More looks in the file /etc/termcap to determine terminal characteristics, and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

If **more** is reading from a file, rather than a pipe, then a percentage is displayed along with the --More-- prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences which may be typed when **more** pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1) :

i<space>
display *i* more lines, (or another screenful if no argument is given)

^D Control-D
display 11 more lines (a ``scroll''). If *i* is given, then the scroll size is set to *i*.

d same as ^D (control-D)

*i***z** same as typing a space except that *i*, if present, becomes the new window size.

*i***s** skip *i* lines and print a screenful of lines

*i***f** skip *i* screenfuls and print a screenful of lines

q or **Q**
Exit from **more.**

= Display the current line number.

v Start up the editor **vi** at the current line.

h Help command; give a description of all the **more** commands.

i/*expression*
search for the *i*-th occurrence of the regular expression *expression*. If there are less than *i* occurrences of *expression*, and the input is a file (rather than a pipe), then the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.

*i***n** search for the *i*-th occurrence of the last regular expression entered.

' (single quote) Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!*command*
invoke a shell with *command*. The characters `%' and `!' in "command" are replaced with the current file name and the previous shell command respectively. If there is no current file name, `%' is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.

*i***:n** skip to the *i*-th next file given in the command line (skips to last file if *i* doesn't make sense)

*i***:p** skip to the *i*-th previous file given in the command line. If this command is given in the middle of printing out a file, then **more** goes back to the beginning of

the file. If i doesn't make sense, **more** skips back to the first file. If **more** is not reading from a file, the bell is rung and nothing else happens.

:f display the current file name and line number.

:q or **:Q**
exit from **more** (same as q or Q).

. Dot
(dot) repeat the previous command.

The commands take effect immediately, i.e., it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may hit the line kill character to cancel the numerical argument being formed. In addition, the user may hit the erase character to redisplay the --More--(xx%) message.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\). **More** will stop sending output, and will display the usual --More-- prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

The terminal is set to noecho mode by this program so that the output can be continuous. What you type will thus not show on your terminal, except for the / and ! commands.

If the standard output is not a teletype, then **more** acts just like **cat**, except that a header is printed before each file (if there is more than one).

A sample usage of **more** in previewing **nroff** output would be

```
nroff -ms +2 doc.n | more
```

OPTIONS

-d **More** will prompt the user with the message "Hit space to continue, Rubout to abort" at the end of each screenful. This is useful if **more** is being used as a filter in some setting, such as a class, where many users may be unsophisticated.

-f This causes **more** to count logical, rather than screen lines. That is, long lines are not folded.

-l Do not treat ^L (form feed) specially.

-n An integer which is the size (in lines) of the window which **more** will use instead of the default.

+line
Start up at line.

+/pat
Start up two lines before the line containing the regular expression pattern.

FILES

/etc/termcap	Terminal data base
/usr/lib/more.help	Help file

SEE ALSO

cat(1), dog(1), head(1), tail(1), pr(1).

NAME

mv - move or rename files and directories

SYNOPSIS

mv file1 file2

mv file ... directory

DESCRIPTION

Mv moves (changes the name of) file1 to file2.

If file2 already exists, it is removed before file1 is moved. If file2 has a mode which forbids writing, **mv** prints the mode (see **chmod**(2)) and reads the standard input to obtain a line; if the line begins with **y**, the move takes place; if not, **mv** exits.

In the second form, one or more files are moved to the directory with their original file-names.

Mv refuses to move a file onto itself.

DIAGNOSTICS

"cannot link /dir2/dir1 to /dir1" Printed if not superuser.

SEE ALSO

cat(1), cp(1), chmod(2).

LIMITATIONS

If file1 and file2 lie on different file systems, **mv** must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

NAME

newgrp - log in to a new group

SYNOPSIS

newgrp [group]

DESCRIPTION

Newgrp changes the group identification of its caller, analogously to **login**(1). The same person remains logged in, and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new group ID.

A password is demanded if the group has a password and the user does not.

When most users log in, they are members of the group named other. **Newgrp** is known to the shell, that executes it directly without a fork if the shell is a login shell.

You can always **newgrp** back to your default group at login by typing: **newgrp.** The super-user can **newgrp** to any group.

FILES

/etc/group, /etc/passwd

SEE ALSO

csh(1), login(1), sh(1), group(5).

NAME

news - print news items

SYNOPSIS

news [**-ans**] [items]

DESCRIPTION

News is used to keep the user informed of current events. By convention, these events are described by files in the directory /usr/lib/news.

When invoked without arguments, **news** prints the contents of all current files in /usr/lib/news, most recent first, with each preceded by an appropriate header. **News** stores the "currency" time as the modification date of a file named .news_time in the user's home directory (the identity of this directory is determined by the environment variable $HOME);

If a delete is typed during the printing of a news item, printing stops and the next item is started. Another delete within one second of the first causes the program to terminate. Only files more recent than this currency time are considered "current".

OPTIONS

-a Prints all items, regardless of currency; the stored time is not changed.

-n Reports the names of the current items without printing their contents, and without changing the stored time.

-s Reports how many current items exist, without printing their names or contents, and without changing the stored time. It is useful to include such an invocation of **news** in one's .login file.

All other arguments are assumed to be specific news items that are to be printed.

FILES

/usr/lib/news/*
$HOME/.news_time
/etc/motd

SEE ALSO

environ(5).

NAME

nice, nohup - run a command at low priority

SYNOPSIS

nice [-number] command [arguments]

nohup command [arguments]

DESCRIPTION

Nice executes command with low scheduling priority. If the number argument is present, the priority is incremented (higher numbers mean lower priorities) by that amount up to a limit of 20. The default number is 7.

The super-user can run commands with priority higher than normal by using a negative priority, for example --10.

Nohup executes command immune to hangup and terminate signals from the controlling terminal. The priority is incremented by 5. **Nohup** should be invoked from the shell with & to prevent it from responding to interrupts by or stealing the input from the next person who logs in on the same terminal.

NOTE: The C shell executes these commands internally. The syntax and features differ somewhat from the commands described here, so C shell users should refer directly to **nice**(1C), and **nohup**(1C).

FILES

nohup.out standard output and standard error file under **nohup**

SEE ALSO

csh(1), nice (1C), nice(2).

DIAGNOSTICS

Nice returns the exit status of the subject command.

NAME

nice - set the priority of a command

SYNOPSIS

nice [number] command

DESCRIPTION

Nice without an argument, increments the nice value for this shell by seven. With a number argument, **nice** increments the nice number of the current shell by the given number (e.g., 'nice -+8' and 'nice -8' have the same effect, that is , incrementing the priority by 8).

With a command argument, nice increments the nice value of the given command by number and by seven if no number is given.

The super-user can specify negative niceness by using the "nice --number ..." command. The command is always executed in a subshell, and the restrictions placed on commands in simple **if** statements apply.

SEE ALSO

nice(1), nohup(1C).
The C Shell in the ZEUS Utilities Manual

NAME

nl - line numbering filter

SYNOPSIS

nl
[-b*type*]
[-f*type*]
[-h*type*]
[-i*incr*]
[-l*num*]
[-n*format*]
[-p]
[-s*sep*]
[-v*start#*]
[-w*width*]
file

DESCRIPTION

Nl reads lines from the named *file* or the standard input if no *file* is named and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

Nl views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid.

Different line numbering options are independently available for header, body, and footer (e.g. no numbering of header and footer lines while numbering blank lines only in the body).

The start of logical page sections are signaled by input lines containing nothing but the following character(s):

Line contents	*Start of*
\:\:\:	header
\:\:	body
\:	footer

Unless signaled otherwise, **nl** assumes the text being read is in a single logical page body.

Command options may appear in any order and may be intermingled with an optional file name. Only one file may be named.

OPTIONS

-b_type_ Specifies which logical page body lines are to be numbered. Recognized _types_ and their meaning are: **a**, number all lines; **t**, number lines with printable text only; **n**, no line numbering; **p**_string_, number only lines that contain the regular expression specified in _string_. Default _type_ for logical page body is **t** (text lines numbered).

-f_type_ Same as **-b**_type_ except for footer. Default for logical page footer is **n** (no lines numbered).

-h_type_ Same as **-b**_type_ except for header. Default _type_ for logical page header is **n** (no lines numbered).

-i_incr_ _Incr_ is the increment value used to number logical page lines. Default is **1**.

-l_num_ _Num_ is the number of blank lines to be considered as one. For example, **-l2** results in only the second adjacent blank being numbered (if the appropriate **-ha**, **-ba**, and/or **-fa** option is set). Default is **1**.

-n_format_ _Format_ is the line numbering format. Recognized values are: **ln**, left justified, leading zeroes supressed; **rn**, right justified, leading zeroes supressed; **rz**, right justified, leading zeroes kept. Default _format_ is **rn** (right justified).

-p Do not restart numbering at logical page delimiters.

-s_sep_ _Sep_ is the character(s) used in separating the line number and the corresponding text line. Default _sep_ is a tab.

-v_start#_ _Start#_ is the initial value used to number logical page lines. Default is **1**.

-w_width_ _Width_ is the number of characters to be used for the line number. Default _width_ is **6**.

SEE ALSO

nroff(1), pr(1).

NAME

nm - print name list

SYNOPSIS

nm [**-gnoprsu**] [file] ...

DESCRIPTION

nm prints the name list (symbol table) of each load module in the argument list. If an argument is an archive, a listing for each load module file in the archive is produced. If no file is given, the symbols in a.out are listed.

The output is sorted alphabetically by default. Each symbol name is preceded by its value in hex, or by blanks if undefined, and one of the letters:

U	undefined
A	absolute
T	text section symbol
D	data section symbol
B	bss section symbol
C	common symbol
F	file name
S	section name

An uppercase letter means global or external. A lowercase type letter means local.

OPTIONS

-g Print only global (external) symbols.

-n Sort numerically rather than alphabetically.

-o Prepend file or archive element name to each output line.

-p Print in symbol-table order rather than in sorted order.

-r Sort in reverse order.

-s Sort according to the size of the external symbol (computed from the difference between the value of the symbol and the value off the symbol with the next highest value). This difference is the value printed. This flag turns on **-g** and **-n** and turns off **-u** and **-p**.

-u Print only undefined symbols.

FILES

a.out load module

SEE ALSO

ar(1), a.out(5), ar(5).

LIMITATIONS

The -s options does not work on segmented object files.

NAME

nohup - no hang-up on interupts in a dial-up situation

SYNOPSIS

nohup command

DESCRIPTION

Nohup executes command with telephone hangups and quits ignored. If output is not re-directed by the user, it will be sent to **nohup.out**. If **nohup.out** is not writable in the current directory, output is redirected to **$HOME/nohup.out**.

Nohup is used in shell scripts. It causes the shell to ignore telephone hangups for the remainder of the script. With a command argument, **nohup** executes command with hangups and quits ignored.

LIMITATIONS

Unless the shell is running detached (in background), nohup has no effect.

SEE ALSO

nice(1C), onintr(1C), signal(2).
The C Shell in the ZEUS Utilities Manual

NAME

nq - print enqueing program

SYNOPSIS

nq [option] [file] ...

DESCRIPTION

Nq is a general purpose enqueing program. It queues files to be processed by **dqueuer**(M). **Dqueuer** is responsible for handling requests for most shared devices such as line printers and text quality typewriters. **Nq** is used to queue requests for all **dqueuer**(M) devices. **Xq**(1) can then be used to examine and delete previously queued requests. **Nq** accepts any number of filenames to be printed. If none are given, standard input is read.

With no options, **nq** takes the files given (or standard input) and queues them for the default queue for the particular system being used. The default queue is the first in the list of queues printed by **xq**. All attributes used to print these files will take on reasonable values for the default device.

Options may be used to modify certain of these attributes pertaining to each file. An option affects all files following it on the command line until it is overidden by another option.

OPTIONS

-b Return to burst page printing (default) if it had been turned off in a previous command.

-c Normally, **nq** "remembers" the name of a file to be printed. The '-c' option (copy) can be used to cause **nq** to copy the file to insulate it against changes that may occur before printing occurs.

-d dest
Certain burst page formats allow for a destination field. This option is used to set that field. If the '-m' option is also specified, and if dest is a valid login name, **mail**(1) will also be sent to user 'dest'.

-m Report by **mail**(1) when printing is complete. Additionally, if the '-d' option is supplied, **mail**(1) may be sent to the identified user (see '-d' option).

-n[bcdmpqrst]
This option is used to negate previous options. For the letter given, the value of that option is reset to the default value. If no option is given, '-nm' is assumed, to retain compatibility with the obsolete **lpr**(1) format.

-p *pri*
File is to be printed at priority *pri*. Valid priorities are 'normal', 'deferred', and 'rush'. Abbreviations to one character are excepted. All 'rush' requests will print before 'normal' requests, which will print before 'deferred' requests. In addition, a queue may be set to prevent requests below a specific priority from printing.

-q *que*[:*dev*]
If this option is omitted, the file will be sent to the default queue. When specified, the files will be sent to queue *que*, to be printed on the first available device. If a queue is serviced by more than one device and a specific device is desired, the :*dev* option can be added. *Que* and *dev* are site-dependent strings. Their exact values can be found using the **xq** command or through the system administrator.

-r Remove the file once it has been printed.

-s Silence. Prevents file name from appearing in queue lists (see **xq**(1)).

-t *number*
Print file *number* times.

EXAMPLES

nq file1
Print file1 with burst pages.

nq -nb file1 -b file2
Print file1 without burst pages. File2, because of the **-b** option, prints with burst pages.

nq -nb file1
Print file1 without burst pages.

FILES

/usr/spool/queuer/activeconfig
/usr/spool/queuer/statusdir
/usr/spool/queuer/requestdir
/tmp/queuer
/usr/spool/queuer/logfile

SEE ALSO

xq(1), backend(M), dqueuer(M), xq(M).

DIAGNOSTICS

If the dequeueing daemon is not active, **nq** will print a message saying so, but accept the request anyway.

NAME

objdu - dump for object and load modules

SYNOPSIS

objdu [**-r**] [**-h**] file

DESCRIPTION

Objdu dumps S8000 load modules in hex format, since bytes in a word are displayed in the Z8000 ordering and addresses correspond to the link addresses. Header information is also displayed in a convenient format. Values are decimal unless otherwise noted.

OPTIONS

-h Print only the header information.

-r Print relocation information if the file is not stripped.

DIAGNOSTICS

"Not a loadable file ..." for non-S8000 load modules

"Premature EOF" for files with an odd number of bytes

SEE ALSO

nm(1), a.out(5), od(1), objhdr(1).

NAME

objhdr - object module header dump

SYNOPSIS

objhdr file

DESCRIPTION

Objhdr is specifically designed for dumping S8000 load module header information in hex format.

SEE ALSO

nm(1), a.out(5), od(1), objdu(1).

NAME

objsu - object module underscore stripper

SYNOPSIS

objsu file ...

DESCRIPTION

Objsu is specifically designed for removing the leading underscore from names in the symbol table section of a.out files. This is useful in PLZ/SYS programs when linking with assembly language programs.

SEE ALSO

nm(1), a.out(5), plz(1).

DIAGNOSTICS

"file not in correct a.out format" if magic number is not valid.

NAME

od, hd - octal or hex dump

SYNOPSIS

od [**-bcdox**] [file] [[+ [x]]offset[.][b]]

hd [**-bcdox**] [file] [[+ [x]]offset[.][**b**]]

DESCRIPTION

Od dumps file in one or more formats as selected by the first argument. If the first argument is missing, **-o** is default.

The file argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

The offset argument specifies the offset in the file where the dump starts. This argument is normally interpreted as octal bytes by od and hex bytes by hd. If the offset is preceded by a **x** or **Øx**, the offset is interpreted as hex. If dot is appended, the offset is interpreted in decimal. If **b** is appended, the offset is interpreted in blocks of 512 bytes. If the file argument is omitted, the offset argument must be preceded +.

Dumping continues until end-of-file.

An asterisk (*) is generated when the data to be displayed on the next two or more lines is identical to the line just displayed. The asterisk is placed in column 1 of the line.

Hd is the same program as **od** but with the default dump option of **-x**.

OPTIONS

-b Interpret bytes in octal.

-c Interpret bytes in ASCII. Certain nongraphic characters appear as C escapes: null=\Ø, backspace=\b, form feed=\f, new line=\n, return=\r, tab=\t; others appear as three-digit octal numbers (for **od**) and hex numbers (for **hd**).

-d Interpret words in decimal.

-o Interpret words in octal.

-x Interpret words in hex. Display addresses in hex.

SEE ALSO

adb(1), objdu(1), objhdr(1).

NAME

onintr - regulate responce to interupts in a shell script

SYNOPSIS

onintr [-] [arg ...]

DESCRIPTION

Onintr

restores the default action of the shell on interrupts; that is, it terminates shell scripts or returns to the terminal command input level.

onintr -

causes all interrupts to be ignored.

onintr label

causes the shell to execute a **goto** label when an interrupt is received or a child process terminates because it was interrupted.

If the shell is running detached and interrupts are being ignored, all forms of **onintr** have no meaning and interrupts continue to be ignored by the shell and all invoked commands.

Signal Handling

The shell normally ignores quit signals. The interrupt and quit signals are ignored for an invoked command if the command is followed by &; otherwise, the signals have the values which the shell inherited from its parent. The shell's handling of interrupts can be controlled by onintr. Login shells catch the terminate signal; otherwise this signal is passed on to children from the state in the shell's parent. Interrupts are not allowed when a login shell is reading the file .logout.

SEE ALSO

continue(1C), exit(1C), logout(1C), nohup(1C).
The C Shell in the ZEUS Utilities Manual

NAME

pack, pcat, unpack - compress and expand files

SYNOPSIS

pack [-] file ...

pcat file ...

unpack file ...

DESCRIPTION

Pack attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file is replaced by a packed file**.z** with the same access modes, access and modified dates, and owner as those of file. If **pack** is successful, file will be removed. Packed files can be restored to their original form using **unpack** or **pcat**.

Pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the - argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of - in place of file will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each **.z** file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

Pack returns a value that is the number of files that it failed to compress.

No packing will occur if:

the file appears to be already packed;
the file name has more than 12 characters;
the file has links;
the file is a directory;
the file cannot be opened;
no disk storage blocks will be saved by packing;
a file called file**.z** already exists;

the **.z** file cannot be created;
an I/O error occurred during processing.

The last segment of the file name must contain no more than 12 characters to allow space for the appended **.z** extension. Directories cannot be compressed.

Pcat does for packed files what **cat**(1) does for ordinary files. The specified files are unpacked and written to the standard output. Thus to view a packed file named file**.z** use:

```
pcat file.z
```

or just:

```
pcat file
```

To make an unpacked copy, say nnn, of a packed file named file**.z** (without destroying file**.z**) use the command:

```
pcat file > nnn
```

Pcat returns the number of files it was unable to unpack. Failure may occur if:

the file name (exclusive of the **.z**) has more than 12 characters;
the file cannot be opened;
the file does not appear to be the output of **pack**.

Unpack expands files created by **pack**. For each file specified in the command, a search is made for file**.z** (or just file, if file ends in **.z**). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the **.z** suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

Unpack returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in **pcat**, as well as for the following:

a file with the ``unpacked'' name already exists;
if the unpacked file cannot be created.

NAME

passwd - change login password

SYNOPSIS

passwd [name]

DESCRIPTION

This command changes (or installs) a password associated with the user name (your own name by default).

The program prompts for the old password and then for the new one. The caller must supply both. The new password must be typed twice to forestall mistakes.

New passwords must be at least four characters long if they use a sufficiently rich alphabet and at least six characters long if monocase. Only the first eight characters of the password are significant.

Only the owner of the name or the super-user can change a password; the owner must prove he knows the old password.

The password file is not changed if the new password is the same as the old password, or if the password has not "aged" sufficiently; see **passwd**(5).

FILES

/etc/passwd, /etc/ptmp

SEE ALSO

login(1), crypt(3), passwd(5).

LIMITATIONS

Under certain conditions the password file will not be updated. In these situations the new file resides in /etc/ptmp.

NAME

paste - merge same lines of several files or subsequent lines of one file

SYNOPSIS

paste file1 file2 ...
paste -dlist file1 file2 ...
paste -s [**-d**list] file1 file2 ...

DESCRIPTION

In the first two forms, **paste** concatenates corresponding lines of the given input files file1, file2, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). If you will, it is the counterpart of **cat**(1) which concatenates vertically, i.e., one file after the other. In the last form above, **paste** subsumes the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the tab character, or with characters from an optionally specified list. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if - is used in place of a file name.

OPTIONS

-d Without this option, the new-line characters of each but the last file (or last line in case of the **-s** option) are replaced by a tab character. This option allows replacing the tab character by one or more alternate characters (see below).

list One or more characters immediately following **-d** replace the default tab as the line concatenation character. The list is used circularly, i. e. when exhausted, it is reused. In parallel merging (i.e. no **-s** option), the lines from the last file are always terminated with a new-line character, not from the list. The list may contain the special escape sequences: **\n** (new-line), **\t** (tab), **** (backslash), and **\Ø** (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (e.g. to get one backslash, use "" -d"\\\\").

-s Merge subsequent lines rather than one from each input file. Use tab for concatenation, unless a list is specified with **-d** option. Regardless of the list, the very last character of the file is forced to be a new-line.

\- May be used in place of any file name, to read a line from the standard input. (There is no prompting).

EXAMPLES

```
ls | paste -d" " -
```
list directory in one column

```
ls | paste - - - -
```
list directory in four columns

```
paste -s -d"\t\n" file
```
combine pairs of lines into lines

SEE ALSO

grep(1), cut(1),
pr(1): **pr -t -m...** works similarly, but creates extra blanks, tabs and new-lines for a nice page layout.

DIAGNOSTICS

line too long
: Output lines are restricted to 511 characters.

too many files
: Except for **-s** option, no more than 12 input files may be specified.

NAME

plz - plz/sys compiler driver

SYNOPSIS

plz [option]... file...

DESCRIPTION

Plz is the ZEUS plz/sys compiler driver. Like cc(1), it provides a simpified interface for compiling programs.

In typical uses, where the plz program is targeted for the ZEUS environment, all plz/sys sources, plz/asm sources, zcode files, etc. are turned into their respective .o files and linked together by the ZEUS linker ld(1). Plz/sys source files must end with .p. Plz/asm sources must end with .s. Zcode files must end with .z and Zobj files (output of the plz code generator) must end with .t.

The following options are interpreted by plz. Other command line options are passed on directly to ld(1).

OPTIONS

-c Suppress the linking phase of the compilation and force an object file (.o) to be produced even if only one program is compiled.

-Dname=def

-Dname Define name to the preprocessor, as if by #define. If no definition is given, name is defined as 1.

-Idir

#include files whose names do not begin with / are always sought first in directory of the file argument, then in directories named in **-I** options, then in directories on a standard list.

-L Cause the plz code generator and assembler to create assembly language listings of their inputs. Listings are left on corresponding files suffixed .l. The code generator's listing is a pseudo-assembly language listing.

-ooutput

Name the final output file output. This option does not take effect if other options are invoked which inhibit linking. If this option is used, the file a.out is left undisturbed.

-P Run only the macro preprocessor and place the result for each .p file in a corresponding .i.p file that has no # lines in it.

-t Run only the plz/sys compiler and code generator and leave the zobj output on the corresponding files suffixed .t.

-U_name_
Remove any initial definition of _name_.

-v Cause all the various stages of compilation or assembly to be verbose. (Default is silent.)

-z Run only the plz/sys compiler and leave the zcode output on the corresponding files suffixed .z.

Other arguments are taken to be linker options, or plz-compatible object programs, typically produced by an earlier plz run, or perhaps libraries of plz-compatible routines. These programs, together with the results of any compilations or assemblies specified, are linked in the order given to produce an executable program with name a.out.

FILES

file.p	plz/sys source
file.z	zcode (intermediate plz/sys)
file.t	zobj (output of code generator)
file.o	object file
/lib/cpp	preprocessor
/usr/bin/plzsys	plz/sys compiler proper
/usr/bin/plzcg	plz/sys code generator proper
/usr/bin/uimage	zobj to a.out translator
/usr/lib/libp.a	plz library (plz/io, system interface, etc.)
/lib/libc.a	standard library
/usr/include	standard directory for #include files

SEE ALSO

Tod Snook, et al., _Report on the Programming Language PLZ/SYS_
ZEUS plz/sys User Guide in the ZEUS Languages / Programming Tools Manual
plzsys(1), plzcg(1), uimage(1), ld(1).

DIAGNOSTICS

The diagnostics produced by plz are self explanatory. Explanations of compiler or code generator error numbers can be found in the _ZEUS plz/sys User Guide_.

NAME

plzcg - plz/sys S8000 code generator

SYNOPSIS

plzcg [**-lsv**] [**-o** output file] file

DESCRIPTION

Plzcg takes zcode files produced by **plzsys**(1) and creates a S8000 executable object module in zobj format named by default t.out. The input file must have a .z extension. If it is not supplied as such on the command line, **plzcg** appends .z to the name before opening the file. Command line options can be specified in upper or lowercase.

To produce an object module that runs on ZEUS, the output of **plzcg** must be run through **uimage**(1) and possibly **ld**(1). **Plzcg** is invoked automatically by the plz/sys compiler driver **plz**(1).

OPTIONS

-l Produce a pseudo-assembly language listing of the module. The listing file has the same name as the input file with .l substituted for the .z suffix. No assembly listing is produced for the data in the module and there are no symbolic labels. References to code are prefaced by the letter P; local data by L; global data by G.

-s Produce code that is sharable among multiple processes. This option is only valid for segmented code and is not applicable for programs targeted to the ZEUS environment. It is currently only useful in a cross-development sense.

-v Cause **plzcg** to be slightly more verbose It announces its presence when it starts and tells how much code and data were produced when finished.

FILES

/tmp/?XXXXX	temporary file
file.z	zcode source file
file.l	listing file
t.out	default output file name

SEE ALSO

ZEUS plz/sys User Guide
plzsys(1), plz(1), uimage(1)

DIAGNOSTICS

A more detailed description of possible errors can be found in the ZEUS plz/sys User Guide.

NAME

plzsys - plz/sys compiler

SYNOPSIS

plzsys [**-elnv**] [**-n**[c/d] [**-o**outputfile] [**-t**z8000ns,z8000s,z80] file

DESCRIPTION

Plzsys is the plz/sys compiler proper. It takes a plz/sys source program with filename extension .p and translates it into a machine independent intermediate code (zcode). If the .p extension is not supplied, **plzsys** appends it before trying to open the input file. The default output filename is the input file name with .z substituted for the .p suffix. Options are recognized in uppercase or lowercase.

To produce code suitable to be run under ZEUS, the .z (zcode) file must then be run through the plz/sys code generator **plzcg**(1), **uimage**(1), and possibly **ld**(1). **Plz**(1), the plz/sys driver program, invokes all the compiler related programs automatically.

OPTIONS

-e Create an error file in the event of errors. An error file, with the file name extension .e substituted for the source file name, is created if there were any errors during compilation. These error messages are the same as the ones shown on the standard output. Normally, errors only go to the standard output.

-l Create listing file. This file (file.l) contains a complete listing of the plz/sys source file plus any error messages that have been produced by the compiler.

-n[**c/d**]
Do not produce debugging information. These two options cause the compiler to not produce debug symbols (**-nd**) or debug constant symbols (**-nc**). Normally, the compiler produces extra symbolic information for a hypothetical debugger. These options are only significant to a debugger that may wish to disregard these particular symbols.

-ooutput file
Creates zcode output on output file instead of file.z

-t[**z8000ns**,z8000s,**z80**]
Produce zcode suitable for a particular target machine. This option tells the compiler what machine the code it is producing will be targeted for. This information is necessary, as some machine dependent features such as machine address size, need to be encoded in the zcode.

The compiler produces code suitable for the non-segmented S8000 (S8000ns) in the absence of this option.

-v Be verbose during compilation. (Default is silent.) With this option, plz/sys announces its presence and tells how many zcode bytes and data bytes were produced when finished.

FILES

/tmp/pXXXXX temporary file

file.p
source file

file.z
default output file

SEE ALSO

ZEUS plz/sys User Guide in the ZEUS Languages / Programming Tools Manual.
Tod Snook, et al., Report on the Programming Language PLZ/SYS
plzcg(1), plz(1), uimage(1), ld(1).

DIAGNOSTICS

Diagnostics are self explanatory. The ZEUS plz/sys User Guide has a complete list of plz source error explanations.

NAME

pr - format files for printer output

SYNOPSIS

pr [**adefhilmnoprstw+-]**] [files]

DESCRIPTION

Pr prints the named files on the standard output. If file is -, or if no files are specified, the standard input is assumed. By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the **-s** option is used, lines are not truncated and columns are separated by the separation character.

If the standard output is associated with a terminal, error messages are withheld until **pr** has completed printing.

Interterminal messages via **write**(1) are forbidden during a **pr**.

OPTIONS

Options may appear singly or be combined in any order.

-a Print multi-column output across the page.

-d Double-space the output.

-eck Expand input tabs to character positions k+1, 2*k+1, 3*k+1, etc. If k is Ø or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If c (any non-digit character) is given, it is treated as the input tab character (default for c is the tab character).

-f Use form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.

-h Use the next argument as the header to be printed instead of the file name.

-ick In output, replace white space wherever possible by inserting tabs to character positions k+1, 2*k+1, 3*k+1, etc. If k is Ø or is omitted, default tab settings at every eighth position are assumed. If c (any non-digit character) is given, it is treated as the

output tab character (default for c is the tab character).

+k Begin printing with page k (default is 1).

-k Produce k-column output (default is 1). -e and -i are assumed for multi-column output.

-lk Set the length of a page to k lines (default is 66).

-m Merge and print all files simultaneously, one per column (overrides the -k, and -a options).

-nck Provide k-digit line numbering (default for k is 5). The number occupies the first k+1 character positions of each column of normal output or each line of -m output. If c (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for c is a tab).

-ok Offset each line by k character positions (default is Ø). The number of character positions per line is the sum of the width and offset.

-p Pause before beginning each page if the output is directed to a terminal (pr will ring the bell at the terminal and wait for a carriage return).

-r Print no diagnostic reports on failure to open files.

-sc Separate columns by the single character c instead of by the appropriate number of spaces (default for c is a tab).

-t Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page.

-wk Set the width of a line to k character positions (default is 72 for equal-width multi-column output, no limit otherwise).

EXAMPLES

Print **file1** and **file2** as a double-spaced, three-column listing headed by "file list":

```
pr -3dh "file list" file1 file2
```

Write **file1** on **file2**, expanding tabs to columns 1Ø, 19, 28, 37, ... :

```
pr -e9 -t <file1 >file2
```

FILES

/dev/tty* to suspend messages

SEE ALSO

cat(1), dog(1), mesg(1), more(1), nq(1).

NAME

printenv - display environment variables

SYNOPSIS

printenv [name]

DESCRIPTION

Printenv prints out the values of the variables in the environment. If a name is specified, only its value is printed.

If a name is specified and it is not defined in the environment, **printenv** returns exit status 1, else it returns status Ø.

SEE ALSO

sh(1), environ(5), csh(1).
The C Shell in the ZEUS Utilities Manual

NAME

prof - display profile data

SYNOPSIS

prof [**-al**] [file]

DESCRIPTION

Prof interprets the file mon.out produced by the monitor subroutine. Under default modes, the symbol table in the named object file (a.out default) is read and correlated with the mon.out profile file. For each external symbol, the percentage of time spent executing between that symbol and the next is printed (in decreasing order), together with the number of times that routine was called and the number of milliseconds per call.

In order for the number of calls to a routine to be tallied, the **-p** option of **cc** must have been given when the file containing the routine was compiled. This option also arranges for the mon.out file to be produced automatically.

OPTIONS

-a All symbols are reported rather than just external symbols.

-l Output is listed by symbol value rather than decreasing percentage.

FILES

mon.out for profile
a.out for namelist

SEE ALSO

cc(1), profil(2), monitor(3), mon.out(5).

LIMITATIONS

Beware of errors due to the granularity of the profile data; it typically is accurate to within eight bytes as a symbol.

NAME

prom - prom programming utility

SYNOPSIS

prom [**-b**] [**-d** device] [**-Dlos** hex] file

DESCRIPTION

Prom is used to output a file to a prom programming device. The user can specify the size (length) of the prom (-l), the address space of the file which is to be placed into the prom (-s and -o), and either word or byte oriented input to the device. The output from **prom** can be directed to any device and is in Tektronix Hex format without acknowledements. This allows direct communication with various programming units such as the Data I/O Model 19 with the Translation Option.

Once the command line is entered, prom will query the user before each transfer. The prompt is similar to this:

Ready: segment Ø start Ø end fff low byte ? (y/r/s/q)

Respond by typing one of these replies:

y yes - send data to device and increment to next address space

r repeat - send data and repeat same address space

s skip - do not send data; increment to next address space

q quit - end execution, do not send data

Prom does not check for bad address offsets. An odd offset is acceptable.

The prom programming device is usually connected to a different terminal port (RS232) than the user is currently using. The prom program can be controlled from one port and the actual transfers to the device controlled through another port. Any device can be named including a file to be copied into the prom programming device at a later time.

All prom programming devices have different options. This program was specifically designed for use with a Data I/O Model 19 with Translation Mode. Other devices that accept Tektronix Hex format without acknowledgements may also function correctly but have not been specifically tested.

OPTIONS

-b Break word-oriented files into high and low bytes so

that 16 bit data can be sent to two 8 bit proms. Default is off. Processing always begins with the low byte with this option on.

-d Select ouput device. Enter a full pathname for either a device or file. Default is /dev/tty, the controlling terminal.

-D For separate I and D files, only output the data section. Default for separate I and D files is only output the code section.

-l Set the length of the prom; default is 0x800. A transfer of this length is sent to the designated device. The transfer is padded with 0xFF value bytes wherever code is not found in the input file.

-o Set the offset (beginning address) of the first data to be transferred. This should be an even number for word-oriented files such as Z8000 object code files. Default is 0x000.

-s Select the segment number from the input file to be programmed. Only one segment is valid during each execution of **prom** . The value must be between 0x00 and 0x80. Default is 0x00.

FILES

/dev/tty - to send the output to the user's terminal
/tmp/prm* - temporary file

SEE ALSO

load(1), a.out(5).

DIAGNOSTICS

Diagnostics will be issued if:

the input file is not a valid object module;

a specified segment is not found in the file;

a specified segment is empty;

or the specified address range contains no code or data.

NAME

prs - print an SCCS file

SYNOPSIS

prs [**-a**]
[**-d**[dataspec]]
[**e**]
[**-l**]
[**-r**[SID]]
files

DESCRIPTION

Prs prints, on the standard output, parts or all of an SCCS file (see **sccsfile**(5)) in a user supplied format. If a directory is named, **prs** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**), and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

Arguments to **prs**, which can appear in any order, consist of keyletter arguments, and file names.

All the described keyletter arguments apply independently to each named file:

-a Requests printing of information for both removed, i.e., delta type = R, (see **rmdel**(1)) and existing, i.e., delta type = D, deltas. If the **-a** keyletter is not specified, information for existing deltas only is provided.

-d[dataspec]
Used to specify the output data specification. The dataspec is a string consisting of SCCS file data keywords (see DATA KEYWORDS) interspersed with optional user supplied text.

-e Requests information for all deltas created earlier than and including the delta designated via the **-r** keyletter.

-l Requests information for all deltas created later than and including the delta designated via the **-r** keyletter.

-r[SID]
Used to specify the SCCS IDentification (SID) string of

a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed.

DATA KEYWORDS

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see sccsfile(5)) have an associated data keyword. There is no limit on the number of times a data keyword can appear in a dataspec.

The information printed by **prs** consists of: (1) the user supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the dataspec. The format of a data keyword value is either Simple (S), in which keyword substitution is direct, or Multi-line (M), in which keyword substitution is followed by a carriage return.

User supplied text is any text other than recognized data keywords. A tab is specified by **\t** and carriage return/new-line is specified by **\n**.

TABLE 1. SCCS Files Data Keywords

Keyword	Data Item	File Section	Value	Format
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	D~or~R^	S
:I:	SCCS ID string (SID)	"	:R:.:L:.:B:.:S:	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	:Dy:/:Dm:/:Dd:	S
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S
:T:	Time Delta created	"	:Th::Tm::Ts:	S
:Th:	Hour Delta created	"	nn	S
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq-no. of deltas incl, excl, ignor	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (seq #)	"	:DS:~:DS:...	S

:Dx:	Deltas excluded (seq #)	"	:DS:~:DS:...
:Dg:	Deltas ignored (seq #)	"	:DS:~:DS:...
:MR:	MR numbers for delta	"	text
:C:	Comments for delta	"	text
:UN:	User names	User Names	text
:FL:	Flag list	Flags	text
:Y:	Module type flag	"	text
:MF:	MR validation flag	"	yes~or~no
:MP:	MR validation pgm name	"	text
:KF:	Keyword error/warning flag	"	yes~or~no
:BF:	Branch flag	"	yes~or~no
:J:	Joint edit flag	"	yes~or~no
:LK:	Locked releases	"	:R:...
:Q:	User defined keyword	"	text
:M:	Module name	"	text
:FB:	Floor boundary	"	:R:
:CB:	Ceiling boundary	"	:R:
:Ds:	Default SID	"	:I:
:ND:	Null delta flag	"	yes~or~no
:FD:	File descriptive text	Comments	text
:BD:	Body	Body	text
:GB:	Gotten body	"	text
:W	A form of what(1) string	N/A	:Z::M::I:
:A:	A form of what(1) string	N/A	:Z::Y:~:M:~:I::Z:
:Z:	what(1) string delimiter	N/A	@(#)
:F:	SCCS file name	N/A	text
:PN:	SCCS file path name	N/A	text

* :Dt:~=~:DT:~:I:~:D:~:T:~:P:~:DS:~:DP:

EXAMPLES

```
prs -d"Users and/or user IDs for :F: are: \n:UN:"
s.file
```

may produce on the standard output:

```
Users and/or user IDs for s.file are:
xyz
131
abc
```

```
prs -d"Newest delta for pgm :M:: :I: Created :D: By
:P:" -r s.file
```

can produce on the standard output:

```
Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas
```

As a special case:

```
prs s.file
```

can produce on the standard output:

```
D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
MRs:
bl78-12345
bl79-54321
COMMENTS:
this is the comment line for s.file initial delta
```

for each delta table entry of the "D" type. The only keyletter argument allowed to be used with the special case is the -a keyletter.

FILES

/tmp/pr?????

SEE ALSO

admin(1), delta(1), get(1), help(1), sccsfile(5).
Source Code Control System User's Guide in the ZEUS Utilities Manual

DIAGNOSTICS

Use help(1) for explanations.

NAME

ps - report process status

SYNOPSIS

ps [**-edalf**]
[**-n** namelist]
[**-t** tlist]
[**-p** plist]
[**-u** ulist]
[**-g** glist]

DESCRIPTION

Ps prints certain information about active processes. Without options, information is printed about processes associated with the current terminal. Otherwise, the information that is displayed is controlled by the following options:

-e Print information about all processes.

-d Print information about all processes, except process group leaders.

-a Print information about all processes, except process group leaders and processes not associated with a terminal.

-f Generate a full listing. (Normally, a short listing containing only process ID, terminal ("tty") identifier, cumulative execution time, and the command name is printed.) See below for meaning of columns in a full listing.

-l Generate a long listing. See below.

-n namelist The argument will be taken as the name of an alternate namelist (**/zeus** is the default).

-t tlist Restrict listing to data about the processes associated with the terminals given in tlist, where tlist can be in one of two forms: a list of terminal identifiers separated from one another by a comma, or a list of terminal identifiers enclosed in double quotes and separated from one another by a comma and/or one or more spaces.

-p plist Restrict listing to data about processes whose process ID numbers are given in plist, where plist is in the same format as tlist.

-u ulist Restrict listing to data about processes whose

user ID numbers or login names are given in ulist, where ulist is in the same format as tlist. In the listing, the numerical user ID will be printed unless the -f option is used, in which case the login name will be printed.

-g glist Restrict listing to data about processes whose process groups are given in glist, where glist is a list of process group leaders and is in the same format as tlist.

The column headings and the meaning of the columns in a ps listing are given below; the letters f and l indicate the option (full or long) that causes the corresponding heading to appear; all means that the heading always appears. Note that these two options only determine what information is provided for a process; they do not determine which processes will be listed.

F (l) Flags (octal and additive) associated with the process:

- 00 swapped out;
- 01 in memory;
- 02 system process;
- 04 locked in core (e.g., for physical I/O);
- 10 being swapped;
- 20 being traced by another process;
- 40 another tracing flag.

S (l) The state of the process:

- 0 non-existent;
- S sleeping;
- W waiting;
- R running;
- I intermediate;
- Z terminated;
- T stopped.

UID (f,l) The user ID number of the process owner; the login name is printed under the **-f** option.

PID (all) The process ID of the process; it is possible to kill a process if you know this datum.

PPID (f,l) The process ID of the parent process.

C (f,l) Processor utilization for scheduling.

STIME (f) Starting time of the process.

PRI (l) The priority of the process; higher numbers mean lower priority.

NI (l) Nice value; used in priority computation.

ADDR (l) The memory address in clicks (a click equals 256 bytes) in hex representation of the process, if resident; otherwise, the disk address.

SZ (l) The size in clicks (in hex) of the memory image of the process.

WCHAN (l) The event for which the process is waiting or sleeping; if blank, the process is running.

TTY (all) The controlling terminal for the process.

TIME (all) The cumulative execution time for the process.

CMD (all) The command name; the full command name and its arguments are printed under the **-f** option.

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked **<defunct>**.

Under the **-f** option, **ps** tries to determine the command name and arguments given when the process was created by examining memory or the swap area. Failing this, the command name, as it would appear without the **-f** option, is printed in square brackets.

FILES

/zeus	system namelist
/dev/mem	memory
/dev	searched to find swap device and terminal

("tty") names.

SEE ALSO

kill(1), nice(1).

LIMITATIONS

Things can change while **ps** is running; the picture it gives is only a close approximation to reality. Some data printed for defunct processes are irrelevant.

NAME

ptx - generate the permuted index

SYNOPSIS

ptx [**-bfgiortw**] ... [input [output]]

DESCRIPTION

Ptx generates a permuted index to file input on file output (standard input and output default). It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of the page. **Ptx** produces output in the form:

.xx "tail" "before keyword" "keyword and after" "head"

where .xx may be an **nroff**(1) or **troff**(1) macro for user-defined formatting. The before keyword and keyword and after fields incorporate as much of the line as will fit around the keyword when it is printed at the middle of the page. Tail and head, at least one of which is an empty string "", are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line. When original text must be discarded, `/' marks the spot.

The cshell command script "/usr/man/manØ/tools/buildptx" generates the index for this manual using **ptx.**

OPTIONS

-b break
Use the characters in the break file to separate words. In any case, tab, newline, and space characters are always used as break characters.

-f Fold upper and lower case letters for sorting.

-g n Use the next argument, n, as the number of characters to allow for each gap among the four parts of the line as finally printed. The default gap is 3 characters.

-i ignore
Do not use as keywords any words given in the ignore file. If the **-i** and **-o** options are missing, use /usr/lib/cref/eign as the ignore file.

-o only
Use as keywords only the words given in the only file.

-r Take any leading nonblank characters of each input line to be a reference identifier (as to a page or chapter) separate from the text of the line. Attach that

identifier as a 5th field on each output line.

-t Prepare the output for the phototypesetter; the default line length is 100 characters.

-w n Use the next argument, n, as the width of the output line. The default line length is 72 characters.

FILES

/bin/sort
/usr/lib/cref/eign

SEE ALSO

apropos(1), getname(1), man(1), nroff(1), troff(1).

LIMITATIONS

Line length counts do not account for overstriking or proportional spacing.

NAME

putfile - transfer files from remote to local system

SYNOPSIS

putfile [-bBfq] filename1 [[-b] filename2 ...]

DESCRIPTION

Putfile downloads one or more files from a remote ZEUS to a local system running ZEUS or RIO. The RIO System must be running the file transfer software of the ZLAB-8000 Communication Package. This program is invoked from the remote system; therefore, **remote(1)** must be executed first. Files are transferred one record at a time along with a checksum to ensure the accuracy of the data. The transfer of one file can be terminated to go to the next by entering a control-x. The entire transfer can be aborted by entering an escape.

OPTIONS

-b The next file is considered to be a binary. A binary file is created on the local system instead of an ASCII file and new lines are not replaced by carriage returns.

-B All file names on the line are treated as if they are preceded by a **-b**. This is generally desirable for a ZEUS-to-ZEUS transfer.

-f The program suppresses all nonfatal error messages.

-q The program prints a query before replacing an existing file of the same name as the one being transferred.

SEE ALSO

getfile(1), local(1), remote(1).

DIAGNOSTICS

"checksum error ... retry" Printed if the computed checksum does not match the transmitted checksum.

"timeout ... retry" Printed if a character is not read within a reasonable time after a read is issued.

"illegal end-of-text ... retry" Generally indicates a system error; not enough characters were read.

"<filename> ... transfer aborted" Printed after a specific number of retries, if the user enters a control-x or an escape, or if the transfer failed (due to lack of space or bad media).

"putfile: <n1> successful transfers <n2> unsuccessful

transfers"
Printed at program termination.

"<filename> ... unable to open file" Printed if the file cannot be opened on either system.

The program outputs a single dot (.) after each successful transfer of a 128-byte record. If the transfer appears to halt, allow 20 seconds for a retry.

NAME

pwck, grpck - password/group file checkers

SYNOPSIS

pwck [file]
grpck [file]

DESCRIPTION

Pwck scans the password file and notes any inconsistencies. The checks include validation of the number of fields, login name, user ID, group ID, and whether the login directory and optional program name exist. The default password file is **/etc/passwd.**

Grpck verifies all entries in the group file. This verification includes a check of the number of fields, group name, group ID, and whether all login names appear in the password file. The default group file is **/etc/group.**

FILES

/etc/group
/etc/passwd

SEE ALSO

group(5), passwd(5).

DIAGNOSTICS

Group entries in **/etc/group** with no login names are flagged.

NAME

pwd - print present working directory name

SYNOPSIS

pwd

DESCRIPTION

Pwd prints the path name of the working (current) directory.

SEE ALSO

cd(1).

DIAGNOSTICS

"cannot open .." and "read error in .." indicate possible file system trouble or no permission for directory access.

NAME

ranlib - convert archives to random libraries

SYNOPSIS

ranlib archive ...

DESCRIPTION

Ranlib converts each archive to a form that can be loaded more rapidly by the loader, by adding a table of contents named __.**SYMDEF** to the beginning of the archive. It uses **ar**(1) to reconstruct the archive, so that sufficient temporary file space must be available in the file system containing the current directory.

SEE ALSO

ld(1), ar(1).

LIMITATIONS

Because generation of a library by **ar** and randomization by **ranlib** are separate, phase errors are possible. The loader **ld** warns when the modification date of a library is more recent than the creation of its dictionary; but this warning appears even if the library is only copied.

NAME

regcmp - regular expression compile

SYNOPSIS

regcmp [-] files

DESCRIPTION

Regcmp precludes the need for calling **regcmp** (see **regex**(3)) from C programs. This saves on both execution time and program size.

The command **regcmp** compiles the regular expressions in file and places the output in **file.i**. If the - option is used, the output will be placed in **file.c**. The format of entries in file is a name (C variable) followed by one or more blanks followed by a regular expression enclosed in double quotes.

The output of **regcmp** is C source code. Compiled regular expressions are represented as **extern char** vectors. File.**i** files may thus be included into C programs, or **file.c** files may be compiled and later loaded. In the C program which uses the **regcmp** output, **regex**(**abc**,**line**) will apply the regular expression named abc to line. Diagnostics are self-explanatory.

EXAMPLES

```
name     "([A-Za-z][A-Za-zØ-9_]*)$Ø"

telno    "\({Ø,1}([2-9][Ø1][1-9])$Ø\){Ø,1} *"
         "([2-9][Ø-9]{2})$1[ -]{Ø,1}"
         "([Ø-9]{4})$2"
```

In the C program that uses the regcmp output,

```
regex(telno, line, area, exch, rest)
```

will apply the regular expression named telno to line.

SEE ALSO

regex(3).

NAME

rehash - re-make the hash table of commands

SYNOPSIS

rehash

DESCRIPTION

The **rehash** command is internal to the C Shell. Its purpose is to create a hash-table of the available commands (files) named in the PATH shell variable.

The **rehash** command is invoked automatically at login and must be re-invoked each time there is a change in either the PATH variable, or the commands (files) located along the search path .

SEE ALSO

The C Shell in the ZEUS Utilities Manual

NAME

remote - transfer control to a remote ZEUS/UNIX system

SYNOPSIS

remote [system name]

DESCRIPTION

Remote is used to transfer control to a remote system running ZEUS or UNIX software. Additionally, using the utilities **getfile**(1) and **putfile**(1), files can be up and down loaded between the remote and local systems. System name indicates the name of the system to which control is to be transferred. If no system name is given, the first system in "/usr/spool/uucp/remotelines" is taken as default.

If the remote system does not respond within twenty seconds, the remoteline times out and a message "remote line to system name timed out" is printed. This can occur if the remoteline is disconnected or if the remote system is down.

Also, if **remote** terminates via a "hangup", a "logout" is sent to the remote system and control is returned to the local system.

If the remote system should go down while connected to it through **remote**, it is possible to return to the local system by using the control-\ (ASCII fs). This feature can also be useful when using **remote** over a dial-up line.

FILES

/usr/spool/uucp/LCK..*	lock files
/usr/spool/uucp/remotelines	database of systems

SEE ALSO

putfile(1), getfile(1), local(1), remotelines(5).

NAME

repeat - repeat a command

SYNOPSIS

repeat count command

DESCRIPTION

The specified command is subject to the same restrictions as the command in the one line statement above and is executed count times. I/O redirections occurs exactly once, even if count is Ø.

SEE ALSO

The C Shell in the ZEUS Utilities Manual

NAME

reserv - tape drive reserving system

SYNOPSIS

reserv [**-fqr**N**m**N**i**]

DESCRIPTION

Reserv provides a mechanism for reserving a cartridge tape drive on the system. Its actions are controlled by the key argument. The key is a string of characters containing at most one function letter and possibly one or more function modifiers. If no key is specified, **reserv** reserves, by default, the tape drive linked to /dev/resdev. If /dev/resdev does not exist, tape drive ctØ is used. The drive is automatically freed 5 minutes after the last read or write. If the tape drive is already reserved by someone else, it prints:

y is using drive x

and returns exit status of 1, otherwise, it returns exit status of Ø with no message.

OPTIONS

The function portion of the key is specified by one of the following letters:

-f Frees the tape drive. Only the user who originally reserved the tape drive can free it. The super user can free any drive. If it is already free and this option specified, an exit code of 1 is returned. Otherwise, an exit code of Ø is returned.

-q Query drive to find out if it is busy. Gives one of the following messages:

Tape drive x is free

returning an exit status = Ø, or

y is using tape drive x

returning an exit status = 1.

-r Request to be put on the queue of users waiting for the tape drive if tape drive is busy. Otherwise, the tape drive will be reserved immediately. If you have been put on the queue (indicating drive is busy presently) the message

y is using tape drive x.
You have been put on the queue for drive x.

is printed. When your turn arrives, the message

Tape drive x is now reserved for y.

The maximun size of the queue is 1Ø.

The following characters can be used as a modifier to the function letter or used alone. These can be specified even if a function letter wasn't specified, since the default function is to reserve drive Ø.

-Ø,...,7 This modifier selects the cartridge tape drive number. The default is **Ø.**

-mØ,...,m7
This modifier selects the mag tape drive number. The default is **Ø.**

-i The following argument is the increment of time, in minutes, that **reserv** waits since last access to free tape (default is 5 minutes).

EXAMPLES

reserv -ql	Queries drive 1 to see if it is reserved or not. Appropriate message will be printed.
reserv	Tries to reserve tape drive Ø. No message is printed out unless drive is busy.
reserv -f	Relinquishes cartridge tape drive Ø.
reserv -lir 1Ø	Try to reserve tape drive 1, if busy it will put user in queue and reserve tape drive in his name when his turn comes up. In any case, after tape drive is reserved, it will be freed 1Ø minutes after last access.

FILES

/usr/spool/reserv/*ct* /etc/reservrc

SEE ALSO

rc(M).

DIAGNOSTICS

Reserv aborts with an error message if more than one function character was specified.

A message is printed if the user tries to put himself on the queue and it is full.

NAME

reset - reset terminal modes to default values

SYNOPSIS

reset<linefeed>

DESCRIPTION

Reset sets the terminal mode bits to the login defaults with the erase character set to control-h and the kill character to control-x.

It is most useful for recovering from program errors which leave the terminal in an unknown and unusable state. Typing <LF> (line feed) to clear any previous inputs, and then to enter the reset command is often necessary since new-line mapping and character echoing may have been turned off.

SEE ALSO

stty(1), stty(2), gtty(2).

NAME

rm, rmdir - remove (unlink) files

SYNOPSIS

rm [**-fir**] file ...

rmdir dir ...

DESCRIPTION

Rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with y, the file is deleted, otherwise the file remains. If a designated file is a directory, an error comment is printed unless the optional argument **-r** has been used.

Rmdir removes entries for the named directories, which must be empty.

OPTIONS

-f No questions are asked when the **-f** (force) option is given.

-i Asks whether to delete each file (interactive option), and, under **-r**, whether to examine each directory.

-r Recursively deletes the entire contents of the specified directory, and the directory itself.

SEE ALSO

chmod(2), unlink(2).

DIAGNOSTICS

Diagnostics are self-explanatory. It is forbidden to remove the file .. to avoid the consequences of inadvertently doing something like rm -r .*.

NAME

rmdel - remove a delta from an SCCS file

SYNOPSIS

rmdel -rSID files

DESCRIPTION

Rmdel removes the delta specified by the SID from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the specified must not be that of a version being edited for the purpose of making a delta (i.e., if a p-file (see **get**(1)) exists for the named SCCS file, the specified must not appear in any entry of the p-file).

If a directory is named, **rmdel** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

The exact permissions necessary to remove a delta are documented in the Source Code Control System User's Guide. Simply stated, they are either (1) if you make a delta you can remove it; or (2) if you own the file and directory you can remove a delta.

FILES

x-file (see **delta**(1))
z-file (see **delta**(1))

SEE ALSO

delta(1), get(1), help(1), prs(1), sccsfile(5).
Source Code Control System User's Guide in the ZEUS Utilities Manual

DIAGNOSTICS

Use **help**(1) for explanations.

NAME

rsh - restricted shell (command interpreter)

SYNOPSIS

rsh [flags] [name[argl...]]

DESCRIPTION

Rsh is a restricted version of the standard command interpreter **sh**(1). It sets up login names and execution environments with capabilities more controlled than the standard shell. The actions of **rsh** are identical to **sh**, except the following are disallowed:

cd
setting the value of **$PATH**
command names containing /
> and >>

When invoked with the name **-rsh**, **rsh** reads the user's **.profile** (from **$HOME/.profile**). It acts as the standard **sh** while doing this, except an interrupt causes an immediate exit, instead of returning to command level. These restrictions are enforced after .profile is interpreted.

When a command to be executed is found to be a shell procedure, **rsh** invokes **sh** to execute it. It is possible to provide to a user some shell procedures that have access to the full power of the standard shell, while restricting him to a limited menu of commands; this assumes that the user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the .profile has complete control over user actions, by performing guaranteed setup actions, then leaving the user in an appropriate directory (probably not the login directory).

Rsh is a link to **sh** and any flag arguments are the same as for **sh**(1).

The system administrator sets up a directory of commands that can be safely invoked by **rsh**.

SEE ALSO

sh(1), profile(5).

NAME

sact - print current SCCS file editing activity

SYNOPSIS

sact files

DESCRIPTION

Sact informs the user of any impending deltas to a named SCCS file. This situation occurs when **get**(1) with the **-e** option has been previously executed without a subsequent execution of **delta**(1). If a directory is named on the command line, **sact** behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of - is given, the standard input is read with each line being taken as the name of an SCCS file to be processed. The output for each named file consists of five fields separated by spaces.

Field 1 specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.

Field 2 specifies the SID for the new delta to be created.

Field 3 contains the logname of the user who will make the delta (i.e. executed a **get** for editing).

Field 4 contains the date that **get -e** was executed.

Field 5 contains the time that **get** -e was executed.

SEE ALSO

delta(1), get(1), unget(1).
SCCS - Source Code Control System in the ZEUS Utilites Manual

DIAGNOSTICS

Use **help**(1) for explanations.

NAME

scc - S8000 segmented C compiler

SYNOPSIS

scc [-cDEIOpPSU] ... file ...

DESCRIPTION

Scc is the portable C compiler modified to produce Z8001 code. Depending on the options, a single **scc** call can compile; compile and assembler; or compile, assemble, and link.

The **scc** compiler provides an unsigned char data type, initialized bit fields, and the new ZEUS version 7 features of C, which are structure assignments and enumeration types.

The default is to compile, assemble, and link, using internal calls to the Z8000 assembler, **as**, and segmented Z8000 linker, **sld**. File names ending in .c are taken to be C source files to be compiled. The source files are compiled, and the assembly language code can be optimized with the **-O** option before being passed to the assembler.

The **-S** option saves the Z8001 assembly language code in .s files and suppress further processing. By default, the code is assembled and then passed to the linker. The **-c** option saves the assembled code in .o files and suppress further processing. By default, the linker then links the code to produce an executable Z8001 program.

File names ending with .s are taken to be Z8001 assembly language. By default, the .s files are assembled to produce .o files and then linked. The compilation step is skipped.

Other file names are taken to be names of C-compatible object programs (typically produced by an earlier **scc** run), or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are linked (in the order given) to produce an executable load module with name a.out.

OPTIONS

Options on the **scc** call can be for **scc** or for **sld**. The following options are interpreted by **scc**:

-c Compile and assemble the named C source files but suppress the linking step. Force an object file .o to be produced even if only one source file is compiled. If a number of C source files are specified, the .o files are saved.

-Dname
-Dname=def

Define name to the preprocessor, as if by #define. If no definition is given, name is defined as 1.

-E Run only the macro preprocessor and send the result to the standard output.

-Idir
Bring in a directory of #include files. Names that do not begin with / are always sought first in the same directory as the source file, then in directories named in **-I** options, then in directories on a standard list.

-O Invoke the C optimizer for S8000 code.

-p Arrange for the compiler to produce code that counts the number of times each routine is called; also, if loading takes place, replace the standard startup routine by one which automatically calls segmon(3) at the start and arranges to write out a mon.out(5) file at normal termination of execution of the object program. An execution profile can then be generated by use of sprof(1). This option produces an execution profile; however, it should not be used since the related programs are not supplied for the S8000.

-P Run only the macro preprocessor and place the result for each .c file in a corresponding .i file with no # lines in it.

-S[1] Compile the named C source files but suppress the assembly and link step. Leave the assembly language code on corresponding files named .s. If 1 is specified, make the original C source lines appear as assembly language comments preceding the code produced for them.

-Uname Remove any initial definition of name.

Other options can be specified on the **scc** call and are passed to the linker **sld.** No options are passed to **as** from the **scc** call, but any internal call to the assembler uses the **-o** option with a name consisting of the original name and .o. The internal call to the linker specifies the options **-X** and **-e** with entry name start, and adds the library name /lib/slibc.a to the end of the list of object module names. **Sld** and **ld** recognize the same set of options. See the description of **ld** for a description of these options.

FILES

file.c	source file
file.o	object file

a.out	load module
/tmp/ztm?	temporaries
/lib/cpp	preprocessor
/lib/s1	compiler pass1
/lib/s2	compiler pass2
/lib/s3	optional optimizer
/lib/c4	optional listing pass
/lib/slibc.a	standard library
/lib/smcrt0.o	optional startup routine for profiling

SEE ALSO

as(1), ld(1).

The C Programming Language by B. W. Kernighan and D. M. Ritchie, Prentice-Hall, 1978,
C Reference Manual by D. M. Ritchie

COMPARE

cc(1)

DIAGNOSTICS

The diagnostics produced by the compiler, assembler, or linker are self-explanatory.

IMPLEMENTATION

The **scc** compiler has the following characteristics:

- As many as six register declarations can be honored, rather than three, as on the PDP-11. The Z8001 registers r8 through r13 can be used for register variables.
- The scc compiler produces object code which conforms to the S8000 calling conventions.

NAME

sccsdiff - compare two versions of an SCCS file

SYNOPSIS

sccsdiff -**r**SID1 -**r**SID2 [**-p**] [-**s**n] file...

DESCRIPTION

Sccsdiff compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files.

OPTIONS

-p pipe output for each file through **pr**(1).

-**r**SID? **SID1** and **SID2** specify the deltas of an SCCS file that are to be compared. Versions are passed to **bdiff**(1) in the order given.

-**s**n n is the file segment size that **bdiff** will pass to **diff**(1). This is useful when **diff** fails due to a high system load.

FILES

/tmp/get????? Temporary files

SEE ALSO

bdiff(1), get(1), help(1), pr(1).
Source Code Control System User's Guide in the ZEUS Utilities Manual

DIAGNOSTICS

"file: No differences" If the two versions are the same. Use help(1) for explanations.

NAME

script - make a file copy of all terminal interactions

SYNOPSIS

script [**-a**] [**-q**] [**-S** shell] [file]

DESCRIPTION

Script makes a file of everything printed on your terminal. The typescript is saved in a file, and can be sent to the line printer later with **nq**(1). If a file name is given, the typescript is saved there. If not, the typescript is saved in the file typescript.

To exit script, type control D. This sends an end of file to all processes you have started up, and causes script to exit. For this reason, control D behaves as though you had typed an infinite number of control D's.

This program is useful when using a crt and a hard-copy record of the dialog is desired, as for a student handing in a program that was developed on a crt when hard-copy terminals are in short supply.

OPTIONS

-a flag causes script to append to the typescript file instead of creating a new file.

-q Asks for "quiet mode", where the "script started" and "script done" messages are turned off.

-S Lets you specify any shell you want. The default shell is the one specified in the user's entry of the **/etc/passwd** file. If the requested shell is not available, script uses any shell it can find.

EXAMPLES

```
Script started on Thu Oct 28 13:21:23 1982
$ cat list1
boat
boathouse
boatload
boatman
boatmen
$ rev list1
taob
esuohtaob
daoltaob
namtaob
nemtaob
$
script done on Thu Oct 28 13:21:55 1982
```

LIMITATIONS

Since ZEUS has no way to write an end-of-file down a pipe without closing the pipe, there is no way to simulate a single control D without ending script.

The new shell has its standard input coming from a pipe rather than a tty, so stty will not work, and neither will ttyname. Programs such as **ls** do not produce a multi-column format because they are writing to a pipe. In general, **script** does not work well when a program attempts to use many system features other than character-oriented I/O.

When the user interrupts a printing process, **script** attempts to flush the output backed up in the pipe for better response. Usually the next prompt also gets flushed.

NAME

sdiff - side-by-side difference program

SYNOPSIS

sdiff [**-losw ...**] file1 file2

DESCRIPTION

Sdiff uses the output of **diff**(1) to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a "<" in the gutter if the line only exists in file1, a ">" in the gutter if the line only exists in file2, and a "|" for lines that are different.

For example:

```
x    |    y
a         a
b    <
c    <
d         d
     >    c
```

OPTIONS

-l Only print the left side of any lines that are identical.

-o output
Use the next argument, output, as the name of a third file that is created as a user controlled merging of file1 and file2. Identical lines of file1 and file2 are copied to output. Sets of differences, as produced by **diff**(1), are printed where a set of differences share a common gutter character. After printing each set of differences, **sdiff** prompts the user with a "%" and waits for one of the following user-typed commands:

e call the editor **ed(1)** with a zero length file

e b call the editor **ed(1)** with both (the concatenation of) left and right

e l call the editor **ed(1)** with the left column

e r call the editor **ed(1)** with the right column

l append the left column to the output file

q exit from the program

r append the right column to the output file

s turn on silent mode; do not print identical lines

v verbose; turn off silent mode

On exit from the editor **ed(1)**, the resulting file is concatenated to the end of the output file.

-s Do not print identical lines.

-w n
Use the next argument, n, as the width of the output line. The default line length is 130 characters.

EXAMPLES

```
% cat list1

boat
boathouse
boatload
boatman
boatmen
boatyard

% cat list2

boa
boar
board
boardinghouse
boast
boat
boathouse
boatload
boatman
boatmen

% sdiff list1 list2
                                        > boa
                                        > boar
                                        > board
                                        > boardinghouse
                                        > boast
boat                                      boat
boathouse                                 boathouse
boatload                                  boatload
boatman                                   boatman
boatmen                                   boatmen
boatyard                                <

% sdiff -s list1 list2
```

```
     0a1,5
                                        > boa
                                        > boar
                                        > board
                                        > boardinghouse
                                        > boast
     6d10
     boatyard                           <

     % sdiff -l list1 list2

                                        > boa
                                        > boar
                                        > board
                                        > boardinghouse
                                        > boast
     boat
     boathouse
     boatload
     boatman
     boatmen
     boatyard                           <

     %
```

SEE ALSO

diff(1), ed(1).

NAME

sed - stream editor

SYNOPSIS

sed [**-e** script] [**-f** sfile] [**-n**] [file] ...

DESCRIPTION

Sed copies the named files (standard input default) to the standard output, edited according to a script of commands.

A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation **sed** cyclically copies a line of input into a pattern space (unless there is something after a D command), applies in sequence all commands whose addresses select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **-n**) and deletes the pattern space.

An address is either a decimal number that counts input lines cumulatively across files, a $ that addresses the last line of input, or a context address, /regular expression/, in the style of **ed**(1) modified thus:

- The escape sequence \n matches a new line embedded in the pattern space.
- A command line with no addresses selects every pattern space.
- A command line with one address selects each pattern space that matches the address.
- A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function !.

In the following list of functions, the maximum number of permissible addresses for each function is indicated in parentheses.

An argument denoted text consists of one or more lines, all but the last of which end with \ to hide the new line. Backslashes in text are treated like backslashes in the replacement string of an s command, and protects initial blanks and tabs against the stripping that is done on every script line.

An argument denoted rfile or wfile must terminate the command line and must be preceded by exactly one blank. Each wfile is created before processing begins. There can be at most 10 distinct wfile arguments.

(1)a\
text
Append. Place text on the output before reading the next input line.

(2)b label
Branch to the : command bearing the label. If label is empty, branch to the end of the script.

(2)c\
text
Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place text on the output. Start the next cycle.

(2)d Delete the pattern space. Start the next cycle.

(2)D Delete the initial segment of the pattern space through the first new line. Start the next cycle.

(2)g Replace the contents of the pattern space by the contents of the hold space.

(2)G Append the contents of the hold space to the pattern space.

(2)h Replace the contents of the hold space by the contents of the pattern space.

(2)H Append the contents of the pattern space to the hold space.

(1)i\
text Insert. Place text on the standard output.

(2)l List the pattern space on the standard output in an unambiguous form. Nonprinting characters are spelled in two-digit ASCII, and long lines are folded.

(2)n Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

(2)N Append the next line of input to the pattern space with an embedded new line. (The current line number changes.)

(2)p Print. Copy the pattern space to the standard output.

(2)P Copy the initial segment of the pattern space through the first new line to the standard output.

(1)q Quit. Branch to the end of the script. Do not start a new cycle.

(2)r rfile
Read the contents of rfile. Place them on the output before reading the next input line.

(2)s/regular expression/replacement/flags
Substitute the replacement string for instances of the regular expression in the pattern space. Any character can be used instead of /. For a fuller description, see **ed**(1). Flags is zero or more of of the following:

g Global. Substitute for all nonoverlapping instances of the regular expression rather than just the first one.

p Print the pattern space if a replacement was made.

w wfile
Write. Append the pattern space to wfile if a replacement was made.

(2)t label
Test. Branch to the : command bearing the label if any substitutions have been made since the most recent reading of an input line or execution of a t. If label is empty, branch to the end of the script.

(2)w wfile
Write. Append the pattern space to wfile.

(2)x Exchange the contents of the pattern and hold spaces.

(2)y/string1/string2/
Transform. Replace all occurrences of characters in

string1 with the corresponding character in string2. The lengths of string1 and string2 must be equal.

(2)! function
(Do not use.) Apply the function (or group, if function is bracket) only to lines not selected by the address(es).

(Ø): label
This command does nothing; it bears a label for b and t commands to branch to.

(1)= Place the current line number on the standard output as a line.

(2){ Execute the following commands through a matching } only when the pattern space is selected.

(Ø) An empty command is ignored.

OPTIONS

-f Cause the script to be taken from file sfile; these options accumulate.

-e If there is just one **-e** option and no **-f's**, the flag **-e** can be omitted.

-n Suppress the default output.

SEE ALSO

ed(1), grep(1).

NAME

SEND - Uploader to the Zilog Z8000 Development Module

SYNOPSIS

SEND file start end [entry]

DESCRIPTION

SEND is invoked from the monitor and uploads memory from a Zilog Development Module (DM) to the ZEUS system. **SEND** creates a Z8000 memory image in the named file. Since the monitor translates lowercase to uppercase characters, the file created on the ZEUS system always receives an uppercase name. start, end, and entry are hexadecimal addresses, with no leading 0x. See the description of SEND in the DM manual.

The default for entry is start, but the user should either omit entry or supply a reasonable value. If entry is omitted, the user must remember the appropriate address and jump to it when the program is downloaded again.

The text size in the header of the load module accounts for all of the uploaded program since there is no way to discern the sizes of the data and bss sections.

The uploaded memory image is assumed to be a complete Z8000 program and is created in the a.out(5) load module format. If no errors occurred during transmission, the file is marked executable. An existing file with the same name is overwritten. The file is of type N_MAGIC1 (Z8000 executable) and is suitable input to the download program **LOAD**. Therefore, the downloaded program can be run or changed, uploaded to ZEUS, and then later downloaded again.

FILES

load module (in DM memory)

SEE ALSO

ld(1), LOAD(1), a.out(5).
Z8000 Development Module Hardware Reference Manual, 03-3080-01
Z8000 Development Module Hardware Reference Manual Errata, 03-3080-01

DIAGNOSTICS

The monitor displays any error messages received from SEND. The error messages are the same as for SEND in the DM manual.

LIMITATIONS

start must be on a 256-byte boundary such as 4600, 4700, 4800, and so forth. Otherwise, the **SEND** program does not

upload.

UPLOAD PROCEDURE

For uploading a memory image from the Development Module to the ZEUS system, the steps are:

1. If communicating with ZEUS, press only the DM NMI switch. Response is NMI and the prompt <.

2. Upload the program with a command such as:

   send save 4600 5000 4dba

3. Response is the prompt <. On the ZEUS system, the current working directory now contains the file named SAVE. At this point, enter other monitor commands or go back to ZEUS by entering the quit command.

NAME

set - establish or modify a C Shell variable

SYNOPSIS

set [name=value]

DESCRIPTION

Without an argument, the set command command shows the value of all shell variables. Variables that have other than a single word as value print as a word list enclosed in parentheses.

The **set** command with an argument in the form name=value, **set** sets name to the single value

Variable expansion happens for all arguments before any setting occurs.

SEE ALSO

csh(1), env(1), printenv(1), setenv(1C).
The C Shell in the ZEUS Utilities Manual

NAME

setenv - establish or revise an environment variable

SYNOPSIS

setenv [name value]

DESCRIPTION

The command **setenv** name value Sets the value of the environment variable name to be value, a single string. Useful environment variables are TERM, the type of terminal, and SHELL, the shell being used.

SEE ALSO

csh(1), env(1), printenv(1), set(1C).
The C Shell in the ZEUS Utilities Manual.

NAME

sh - shell, the Bourne shell command programming language

SYNOPSIS

sh [**-ceiknrstuvx**] [args]

DESCRIPTION

Sh is a command programming language that executes commands read from a terminal or a file. See Invocation below for the meaning of arguments to the shell.

Commands.

A simple-command is a sequence of non-blank words separated by blanks (a blank is a tab or a space). The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument Ø (see **exec**(2)). The value of a simple-command is its exit status if it terminates normally, or (octal) 2ØØ+status if it terminates abnormally (see **signal**(2) for a list of status values).

A pipeline is a sequence of one or more commands separated by **|**. The standard output of each command but the last is connected by a **pipe**(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

A list is a sequence of one or more pipelines separated by **;**, **&**, **&&**, or **||**, and optionally terminated by **;** or **&**. Of these four symbols, **;** and **&** have equal precedence, which is lower than that of **&&** and **||**. The symbols **&&** and **||** also have equal precedence. A semicolon (**;**) causes sequential execution of the preceding pipeline; an ampersand (**&**) causes asynchronous execution of the preceding pipeline (i.e., the shell does not wait for that pipeline to finish). The symbol **&&** (**||**) causes the list following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of new-lines may appear in a list, instead of semicolons, to delimit commands.

A command is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

for name [**in** word ...] **do** list **done**

Each time a **for** command is executed, name is set to the next word taken from the **in** word list. If **in** word ... is omitted, then the **for** command executes the **do** list once for each positional parameter that is set (see

Parameter Substitution below). Execution ends when there are no more words in the list.

case word **in** [pattern [| pattern] ... **)** list **;;**] ... **esac**
A **case** command executes the list associated with the first pattern that matches word. The form of the patterns is the same as that used for file-name generation (see File Name Generation below).

if list **then** list [**elif** list **then** list] ... [**else** list] **fi**
The list following **if** is executed and, if it returns a zero exit status, the list following the first **then** is executed. Otherwise, the list following **elif** is executed and, if its value is zero, the list following the next **then** is executed. Failing that, the **else** list is executed. If no **else** list or **then** list is executed, then the **if** command returns a zero exit status.

while list **do** list **done**
A **while** command repeatedly executes the **while** list and, if the exit status of the last command in the list is zero, executes the **do** list; otherwise the loop terminates. If no commands in the **do** list are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(list**)**
Execute list in a sub-shell.

{list**;}**
list is simply executed.

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments.

In general a word beginning with # causes that word and all the following characters up to a new-line to be ignored. Comments have a special meaning as the first line of a shell script. Please refer to the following chart as to specific meanings:

First Line in Script:Meaning:

#!sh This is a shell script for /bin/sh

#!csh This is a shell script for /bin/csh

#!	Error, can't determine which shell
#!/xxx	Use the shell /xxx (where the string xxx is a pathname) to execute this script
#?	Where '?' is any character not equal to '!' this is a shell script for /bin/csh
?	Where '?' is any character not equal to '#' this is a shell script for /bin/sh

Command Substitution.
The standard output from a command enclosed in a pair of grave accents as in the command:

```
`ls`
```

may be used as part or all of a word; trailing new-lines are removed.

Parameter Substitution.
The character **$** is used to introduce substitutable parameters. Positional parameters may be assigned values by **set.** Variables may be set by writing:

name=value [name=value] ...

Pattern-matching is not performed on value.

${parameter}
A parameter is a sequence of letters, digits, or underscores (a name), a digit, or any of the characters *****, **@**, **#**, **?**, **-**, **$**, and **!** . The value, if any, of the parameter is substituted. The braces are required only when parameter is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. A name must begin with a letter or underscore. If parameter is a digit then it is a positional parameter. If parameter is ***** or **@**, then all the positional parameters, starting with **$1**, are substituted (separated by spaces). Parameter **$0** is set from argument zero when the shell is invoked.

${parameter:-word}
If parameter is set and is non-null then substitute its value; otherwise substitute word.

${parameter:=word}
If parameter is not set or is null then set it to word; the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

${parameter**:?**word**}**
If parameter is set and is non-null then substitute its value; otherwise, print word and exit from the shell. If word is omitted, then the message "parameter null or not set" is printed.

${parameter**:+**word**}**
If parameter is set and is non-null then substitute word; otherwise substitute nothing.

In the above, word is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:-pwd}
```

If the colon (:) is omitted from the above expressions, then the shell only checks whether parameter is set or not.

The following parameters are automatically set by the shell:

The number of positional parameters in decimal.

- Flags supplied to the shell on invocation or by the **set** command.

? The decimal value returned by the last synchronously executed command.

$ The process number of this shell.

! The process number of the last background command invoked.

The following parameters are used by the shell:

HOME The default argument (home directory) for the cd command.

PATH The search path for commands (see Execution below).

MAIL If this variable is set to the name of a mail file, then the shell informs the user of the arrival of mail in the specified file.

PS1 Primary prompt string, by default ``$ ''.

PS2 Secondary prompt string, by default ``> ''.

IFS Internal field separators, normally **space**, **tab**, and **new-line**.

The shell gives default values to **PATH**, **PS1**, **PS2**, and **IFS**, while **HOME** and **MAIL** are not set at all by the shell (although **HOME** is set by **login**(1)).

Blank Interpretation.

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or) are retained. Implicit null arguments (those resulting from parameters that have no values) are removed.

File Name Generation.

Following substitution, each command word is scanned for the characters *****, **?**, and **[**. If one of these characters appears then the word is regarded as a pattern. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, then the word is left unchanged. The character . at the start of a file name or immediately following a **/**, as well as the character / itself, must be matched explicitly.

***** Matches any string, including the null string.

? Matches any single character.

[...]
Matches any one of the enclosed characters. A pair of characters separated by - matches any character lexically between the pair, inclusive.

Quoting.

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | < > new-line space tab

A character may be quoted (i.e., made to stand for itself) by preceding it with a ****. The pair **\new-line** is ignored. All characters enclosed between a pair of single quote marks (), except a single quote, are quoted. Inside double quote marks (""), parameter and command substitution occurs and \ quotes the characters ****, , **"**, and **$**. **"$*"** is equivalent to **"$1 $2 ..."**, whereas **"$@"** is equivalent to **"$1" "$2" ....**

Prompting.

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a new-line is

typed and further input is needed to complete a command, then the secondary prompt (i.e., the value of **PS2**) is issued.

Input/Output.

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a command and are not passed on to the invoked command; substitution occurs before word or digit is used:

<word Use file word as standard input (file descriptor Ø).

>word Use file word as standard output (file descriptor 1). If the file does not exist then it is created; otherwise, it is truncated to zero length.

>>word Use file word as standard output. If the file exists then output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.

<<[-]word The shell input is read up to a line that is the same as word, or to an end-of-file. The resulting document becomes the standard input. If any character of word is quoted, then no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (escaped) **\new-line** is ignored, and \ must be used to quote the characters ****, **$**, , and the first character of word. If - is appended to **<<**, then all leading tabs are stripped from word and from the document.

<&digit The standard input is duplicated from file descriptor digit (see **dup**(2)). Similarly for the standard output using **>**.

<&- The standard input is closed. Similarly for the standard output using **>**.

If one of the above is preceded by a digit, then the file descriptor created is that specified by the digit (instead of the default Ø or 1). For example:

```
... 2>&1
```

creates file descriptor 2 that is a duplicate of file descriptor 1.

If a command is followed by **&** then the default standard input for the command is the empty file **/dev/null.** Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Environment.

The environment (see environ(7)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. Executed commands inherit the same environment. If the user modifies the values of these parameters or creates new ones, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any simple-command may be augmented by prefixing it with one or more assignments to parameters. Thus:

```
TERM=450 cmd argsand
(export TERM; TERM=450; cmd args)
```

are equivalent (as far as the above execution of cmd is concerned).

If the **-k** flag is set, all keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **"a=b c"** and then "c":

```
echo a=b c
set -k
echo a=b c
```

Signals.

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Execution.

Each time a command is executed, the above substitutions are carried out. Except for the Special Commands listed below, a new process is created and an attempt is made to execute the command via **exec**(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a / then the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell (i.e., a separate process) is spawned to read it. A parenthesized command is also executed in a sub-shell.

Special Commands.

The following commands are executed in the shell process and, except as specified, no input/output redirection is permitted for such commands:

: No effect; the command does nothing. A zero exit code is returned.

. file
Read and execute commands from file and return. The search path specified by **PATH** is used to find the directory containing file.

break [n]
Exit from the enclosing **for** or **while** loop, if any. If n is specified then break n levels.

continue [n]
Resume the next iteration of the enclosing **for** or **while** loop. If n is specified then resume at the n-th enclosing loop.

cd [arg]
Change the current directory to arg. The shell parameter **HOME** is the default arg.

eval [arg ...]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [arg ...]
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [n]
Causes a shell to exit with the exit status specified by n. If n is omitted then the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)

export [name ...]
The given names are marked for automatic export to the environment of subsequently-executed commands. If no arguments are given, then a list of all names that are exported in this shell is printed.

newgrp [arg ...]
Equivalent to **exec newgrp** arg ....

read [name ...]
One line is read from the standard input and the first word is assigned to the first name, the second word to the second name, etc., with leftover words assigned to the last name. The return code is 0 unless an end-of-file is encountered.

readonly [name ...]
The given names are marked readonly and the values of the these names may not be changed by subsequent assignment. If no arguments are given, then a list of all readonly names is printed.

set [**-ekntuvx** [arg ...]]

-e If the shell is non-interactive then exit immediately if a command exits with a non-zero exit status.

-k All keyword arguments are placed in the environment for a command, not just those that precede the command name.

-n Read commands but do not execute them.

-t Exit after reading and executing one command.

-u Treat unset variables as an error when substituting.

-v Print shell input lines as they are read.

-x Print commands and their arguments as they are executed.

-- Do not change any of the flags; useful in setting $1 to -.

Using **+** rather than **-** causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **$-**. The remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, ... If no arguments are given then the values of all names are printed.

shift
The positional parameters from **$2** ... are renamed **$1** ....

test
Evaluate conditional expressions. See **test**(1) for usage and description.

times
Print the accumulated user and system times for processes run from the shell.

trap [arg] [n] ...
arg is a command to be read and executed when the shell receives signal(s) n. (Note that arg is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If arg is absent then all trap(s) n are reset to their original values. If arg is the null string then this signal is ignored by the shell and by the commands it invokes. If n is 0 then the command arg is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

umask [nnn]
The user file-creation mask is set to nnn (see **umask**(2)). If nnn is omitted, the current value of the mask is printed.

wait Wait for all child processes to terminate report the termination status. If n is not given then all currently active child processes are waited for. The

return code from this command is always zero.

Invocation.

If the shell is invoked through **exec**(2) and the first character of argument zero is -, commands are initially read from **/etc/profile** and then from **$HOME/.profile**, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as **/bin/sh**. The flags below are interpreted by the shell on invocation only; Note that unless the **-c** or **-s** flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

-c string If the **-c** flag is present then commands are read from string.

-s If the **-s** flag is present or if no arguments remain then commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output is written to file descriptor 2.

-i If the **-i** flag is present or if the shell input and output are attached to a terminal, then this shell is interactive. In this case TERMINATE is ignored (so that **kill Ø** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.

-r If the **-r** flag is present the shell is a restricted shell (see **rsh**(1)).

The remaining flags and arguments are described under the **set** command above.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively then execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above).

FILES

/etc/profile
$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

cd(1), env(1), login(1), newgrp(1), rsh(1), test(1), umask(1), dup(2), exec(2), fork(2), pipe(2), signal(2), umask(2), wait(2), a.out(5), profile(5), environ(7).

LIMITATIONS

The command **readonly** (without arguments) produces the same output as the command **export.**

If **<<** is used to provide standard input to an asynchronous process invoked by **&**, the shell gets mixed up about naming the input document; a garbage file **/tmp/sh*** is created and the shell complains about not being able to find that file by another name.

NAME

size - size of an object file

SYNOPSIS

size [file ...]

DESCRIPTION

Size prints the decimal number of bytes required by the text (code) sections, initialized data sections, and bss (uninitialized data) portions, and their sum in decimal and hexadecimal, of each object-file argument. If no file is specified, **a.out** is used.

EXAMPLES

```
% size /bin/ls

7276+760+3840 = 11876b = 027144b
```

SEE ALSO

objdu(1), objhdr(1), a.out(5).

NAME

sld - segmented Z8000 loader

SYNOPSIS

sld [**-bdeilMorRstuwxXz**] ... file ...

DESCRIPTION

sld combines several Z8000 object programs into one load module, resolves external references, and searches libraries. In the simplest case, several Z8000 object files are given and **sld** combines them to produce an executable load module. An object module can also be produced and used as input to a subsequent **sld** run, in which case the **-r** option must be given to preserve the relocation bits. The output of **sld** is left on the file a.out(5), unless the **-o** option is used to select a name. If no errors occur during the link, the output file is marked executable.

The order of the argument routines in the load module depends on whether the -M or -R options are used. If they are not being used then the argument routines are concatenated in the order specified with the following mapping assignments: text, data and bss in segment 0 for combined I&D; and text in segment 0, data and bss in segment 1 for separate I&D. If -M or -R is used then the range for segment assignments is between 0 and 127 segments (inclusive), and the ordering for each section of the argument routines is left up to the user. In absence of the **-e** option, the entry point of the output is the lowest text address.

If any argument is a library, it is searched once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are linked. If a routine from a library references another routine in the library, and the library has not been processed by **ranlib**(1), the referenced routine must appear after the referencing routine in the library. Thus, the order of modules within libraries is important. If the first member of a library is named __.SYMDEF, then it is understood to be a dictionary for the library, such as one produced by **ranlib**(1). The dictionary is searched iteratively to satisfy all possible references.

The symbols _etext, _edata, _end and _stkseg in assembly language code (etext, edata, end and stkseg in C programs) are reserved and cannot be redefined by the user. The symbol _etext refers to the first location above the text section. The symbol _edata refers to the first location above initialized data. The symbol _end refers to the first location above all data, and can be used as a starting location for a dynamic allocation area managed by the user. The symbol _stkseg refers to the stack segment and the starting

segment is either: segment Ø (for combined I&D) or the segment after the last segment used for text and/or data. The above default can be overriden by the -Ms option.

OPTIONS

Except for **-l**, the options appear before the file names. For the purpose of linking a program to be downloaded to a development module, the **-o** option must be used with an uppercase file name, and the **-t** option must be used with a starting address that is Øx5ØØØ or higher. The options are:

-b addr

-bx addr
Set the bottom location for the program, or for the specified section if x is specified. X can be one of **t**, **d**, or **b** for text, data, and bss, respectively. The address can be specified in decimal, hex, or octal using the standard C language conventions: a leading zero indicates octal, and a leading Øx indicates hex. The address specified must be a multiple of 256. If no section is selected, the bottom applies to all three sections if the program is combined instruction and data, or to data and bss if separate instruction and data. Only one **-t** or **-b** option per section can be specified. Errors can result if sections overlap, or the bottom address causes a section to wrap around.

-d Force definition of common storage even if the **-r** flag is present.

-e name
Take the following argument as the name of the entry point of the loaded program. (Note that C program names specified as the entry point must have the prepended underbar.) The lowest text address is the default entry point.

-i Separate the program text and data (also called instruction and data) areas when the output file is executed.

-lx Search the named library. This option is an abbreviation for the library name /lib/slibx.a, where x is a string. "-lxyz" refers to /lib/slibxyz.a. If the library is not found in /lib then /usr/lib is searched followed by /z/bin/lib. A library is searched when its name is encountered, so the placement of a -l option is significant.

-My

-Mx*y*

-My + file ... +

-Mx*y* + file ... +

-Msy This option allows the user to specify section to segment mappings. x can be one of t, d, b for text, data and bss respectively. y is a decimal integer, between 0 and 127 (inclusive), which indicates which segment the user wants to use as a target segment. -My states that the sections for all files specified after this option (and up to the next -M) are to be mapped using segment y as the starting segment. -Mxy states that the x type sections for all files specified after this option (and up to the next -M, if any) are to be mapped onto segment y. The remaining sections are assigned to their default segments or to a user specified segment that pertains to these sections. The -My + file ... + states that only the sections whose files are between the + delimiters are to be mapped onto segment y. -Mxy + file ... + states that only x type sections whose files are between the + delimiters are to be mapped onto segment y. The other sections will be assigned to their default segments or to any user specified segment (if applicable). The -Msy option refers to the stack segment. y indicates the segment that the user wants to reserve for the stack segment. Only the -My option is valid for combined I&D.

-o name
Change the name of the **sld** output file to the name specified. This option must be used to specify an uppercase file name for a load module to be downloaded to the Zilog Z8000 Development Module. The form of the load module remains the same as described in a.out(5).

-r Generate relocation bits in the output file so that it can be used in a subsequent **sld** run. This flag overrides the **-t** and **-b** options. It also prevents final definitions from being given to common symbols, and suppresses the undefined symbol diagnostics.

-Ry Creates a zero length segment. This option generates a zero segment by entering an entry into the segment table which refers to a segment of size 0. y is a decimal integer, between 0 and 127 (inclusive), that specifies which segment is to be reserved.

-s Strip the output; that is, remove the symbol table and relocation bits to save space.

-t addr

-tx addr
Set the highest location of the program or section to the hex, octal, or decimal number specified. x can be one of **t**, **d**, or **b** for text, data, and bss respectively. This option is similar to that of -b except a top address is specified instead of a bottom. The link location of the program or section is justified so that it meets the specified top. The low address of the section or program is always a multiple of 256. In most cases, the top is within 256 bytes of the specified address, and is padded with zeros to meet the address.

-u Take the following argument as a symbol and enter it as undefined in the symbol table. This is useful for linking wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the link of the first routine.

-w Suppress the symbol redefinition warning. This warning is produced while searching archives. If an archive contains a module that defines a symbol that is already defined, a redefinition warning is produced.

-x Enter only external and global symbols and do not preserve local symbols in the output symbol table. This option saves some space in the output file.

-X Save local symbols except for those whose names begin with L. This option is used by the C compiler to discard internally generated labels while retaining symbols local to routines.

-z Gives the output file the default mappings for the S8000. Segment four is the lowest segment, segment sixty-three is for the stack and segments sixty-four and sixty-five are unusable. The output file can be either separate or combined I & D.

FILES

/lib	
/usr/lib	
/z/bin/lib	libraries for -l option
/slib*.a	
a.out	output file

SEE ALSO

LOAD(1), ar(1), ld(1), ranlib(1), a.out(5)

DIAGNOSTICS

Any undefined references cause the special symbol "_end" to be reported as undefined. As a programmer aid, the "redefinition" warning occurs if, while searching an archive, a symbol that is already defined is defined in an archive member. This warning can be suppressed with the **-w** option. It also does not occur if the symbol name begins with two underscores (one underscore in C).

LIMITATIONS

The "redefinition" warning should not occur on symbols that occur twice within the same archive.

The segment numbers for the section entries in the symbol table maybe in error. However this information is not being used by sld.

The -b and -t options work only for single segment programs.

NAME

sleep - suspend execution for an interval

SYNOPSIS

sleep time

DESCRIPTION

Sleep suspends execution for time seconds. It is used to execute a command after a 105 seconds, as in:

```
(sleep 105; command)&
```

or to execute a command every 37 seconds, as in:

```
while true
do
     command
     sleep 37
done
```

SEE ALSO

wait(1), alarm(2), sleep(3).

LIMITATIONS

Time must be less than 65536 seconds.

NAME

slink - memory binder for downloading object modules.

SYNOPSIS

slink [**-eioPsvwxX**] [[**-S** addr] file ...] ...

DESCRIPTION

Slink creates load modules for downloading to target hardware. Modules may be bound to particular memory locations by using the **-S** address option.

Slink combines several object modules into one load module file. The output of **slink** is left on the file a.out(5). If the **-o** option is used, the name so specified is used instead of a.out; the file has the same format. If no errors occur during the link, the file is marked executable.

The argument routines are concatenated in the order specified. In the absence of the **-e** option, the entry point of the output is the beginning of the first routine.

The options must be specified separately.

The options are:

-e name Take the following argument as the name of the entry point of the program. The link address of the text section is the default.

-i Separate the program text and data (also called instruction and data) areas when the output file is execute. This option is only for Z8 object modules.

-o name Change the name of the **slink** output file to the name specified. The form of the load module remains the same as described in a.out(5).

-P Preserve the temporary work files in /tmp/sl..... for debug purposes.

-s Strip the output: remove the symbol table and relocation bits to save space.

-S address file ...
Set the location of the files listed. The address may be specified in hex, octal, or decimal using standard C language conventions: a leading 0x indicates hex, and a leading zero indicates octal. Errors can result if sections overlap, or if the bottom address causes a section to wrap around.

-v Verbose option. Print the command line passed to ld.

-w Suppress the symbol redefinition warning. This warning is produced while searching archives. If an archive contains a module that defines a symbol that is already defined, a redefinition warning is produced.

-x Enter only external and global symbols and do not preserve local symbols in the output symbol table. This option saves some space in the output file.

-X Save local symbols except for section name entries in the symbol table and for those whose names begin with L. This option is used by the C compiler to discard internally generated labels while retaining symbols local to routines.

FILES

/tmp/sl..... work files which will remain if -P option is specified.
a.out output file.

SEE ALSO

ld(1), a.out(5).

DIAGNOSTICS

Load not performed.
Fork failed.
No children to wait for.
Temp files, '/tmp/sl.....', were not removed.
Bad option: <option>.
<file> not in correct a.out format.
Cannot handle segmented file: <file>.
Cannot handle overlay file: <file>.
Cannot handle stripped file: <file>.
Z8000 files cannot start on odd boundaries.
Cannot mix Z8000 modules with Z8 or Z80 modules.
Cannot mix Z8 modules with Z80 modules.

LIMITATIONS

Slink is unable to handle segmented or overlay programs.
Nothing is done with common symbols; they follow bss.

NAME

sort - sort and/or merge files

SYNOPSIS

sort
[**-bcdfimnortTu**]
[+pos1 [-pos2]]
[-o name]
[-T directory]
[file] ...

DESCRIPTION

Sort sorts lines of all the named files together and writes the result on the standard output. The name minus means the standard input. If no input files are named, the standard input is sorted.

The default sort key is an entire line. Default ordering is lexicographic by bytes in machine collating sequence.

OPTIONS

The ordering is affected globally by the following options, one or more of which can appear:

- b Ignore leading blanks (spaces and tabs) in field comparisons.

-c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

-d Dictionary order: only letters, digits and blanks are significant in comparisons.

-f Fold uppercase letters onto lowercase.

-i Ignore characters outside the ASCII range Ø4Ø-Ø176 in nonnumeric comparisons.

-m Merge only, the input files are already sorted.

-n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. Option **-n** implies option **-b.**

-o The next argument is the name of an output file to use instead of the standard output. This file can be the same as one of the inputs.

-r Reverse the sense of comparisons.

-tx Tab character separating fields is x.

-T The next argument is the name of a directory in which temporary files are made.

-u Suppress all but one in each set of equal lines. Ignored bytes and bytes outside keys do not participate in this comparison. x.

The notation +pos1 -pos2 restricts a sort key to a field beginning at pos1 and ending just before pos2. Pos1 and pos2 each have the form m.n, optionally followed by one or more of the flags **bdfinr**, where m tells a number of fields to skip from the beginning of the line and n tells a number of characters to skip further.

If any flags are present, they override all the global ordering options for this key. If the -b option is in effect n is counted from the first nonblank in the field; **-b** is attached independently to pos2. A missing .n means .Ø; a missing -pos2 means the end of the line.

Under the **-t**x option, fields are strings separated by x; otherwise fields are nonempty nonblank strings separated by blanks.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

EXAMPLES

Print in alphabetical order all the unique spellings in a list of words. Capitalized words differ from uncapitalized.

```
sort -u +Øf +Ø list
```

Print the password file (passwd(5) sorted by user ID number (the third colon-separated field).

```
sort -t: +2n /etc/passwd
```

Print the first instance of each month in an already sorted file of month-day entries. The options **-um** with one input file make the choice of a unique representative from a set of equal lines predictable.

```
sort -um +Ø -1 dates
```

FILES

/usr/tmp/stm*, /tmp/*
first and second tries for temporary files

SEE ALSO

uniq(1), comm(1), rev(1), join(1).

DIAGNOSTICS

Comments and exits with nonzero status for various trouble conditions and for disorder discovered under option **-c**.

LIMITATIONS

Very long lines are silently truncated.

NAME

source - execute commands in a shell script in the current shell

SYNOPSIS

source file

DESCRIPTION

The shell reads commands from file. **Source** commands can be nested; if they are nested too deeply, the shell can run out of file descriptors.

LIMITATIONS

An error in a **source** at any level terminates all nested **source** commands. Input during **source** commands is never placed on the history list. **Source** output cannot be redirected.

SEE ALSO

The C Shell in the ZEUS Utilities Manual

NAME

spell, spellin, spellout - find spelling errors

SYNOPSIS

spell [**-bdvx**] ... [file] ...

spellin [list]

spellout [**-d**] list

DESCRIPTION

Spell collects words from the named documents, and looks them up in a spelling list. Words that do not occur and cannot be derived (by applying certain inflections, prefixes or suffixes) from words in the spelling list are printed on the standard output. If no files are named, words are collected from the standard input.

Spell ignores most **troff**(1), **tbl**(1), and **eqn**(1) constructions.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective in respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Pertinent auxiliary files can be specified by name arguments, indicated as follows with their default settings. Copies of all output are accumulated in the history file. The stop list filters out misspellings (for example, thier=thy-y+ier) that would otherwise pass.

Two routines help maintain the hash lists used by **spell**. Both expect a list of words, one per line, from the standard input. **Spellin** adds the words on the standard input to the preexisting list and places a new list on the standard output. If no list is specified, the new list is created from scratch. **Spellout** looks up each word in the standard input and prints on the standard output those that are missing from (or present on, with option **-d**) the hash list.

OPTIONS

-b British spelling is checked. Besides preferring centre, colour, speciality, travelled, etc., this option insists upon -ise in words like standardise.

-d used with spellout; prints on the standard output those that are present on the hash list.

-v all words not literally in the spelling list are printed, and plausible derivations from spelling list

words are indicated.

-x every plausible stem is printed with = for each word.

EXAMPLES

To add words in file to the list:

```
cat file >> /usr/dict/words

sort -o /usr/dict/words -u /usr/dict/words

spellin /usr/dict/hlista < /usr/dict/words > /usr/dict/hlistx

move /usr/dict/hlistx /usr/dict/hlista
```

FILES

/usr/dict/hlist[ab]
    hashed spelling lists, American & British
/usr/dict/hstop
    hashed stop list
/usr/dict/spellhist
    history file
/usr/lib/spell

SEE ALSO

deroff(1), look(1), sort(1), tee(1), sed(1).

LIMITATIONS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions.

British spelling was done by an American.

NAME

split - split a file into pieces

SYNOPSIS

split [-n] [file [name]]

DESCRIPTION

Split reads file and writes it in n-line pieces (default 1000), or n-character pieces if the number ends with a 'c' (default 10000), as many as necessary, onto a set of output files. The name of the first output file is name with **aa** appended, and so on lexicographically. The maximum number of files created is 676. If there is more data to split from the original file and all 676 files have been used, the balance will go into the last file, namely, name with **zz** appended. If no output name is given, **x** is default.

If no input file is given, or if - is given in its stead, then the standard input file is used.

EXAMPLE

```
% cat list
  boa
  boar
  board
  boardinghouse
  boast
  boat
  boathouse
  boatload
  boatman
  boatmen

% split -5 list

% ls
  list xaa xab

% cat xaa                  % cat xab
  boa                      boat
  boar                     boathouse
  board                    boatload
  boardinghouse            boatman
  boast                    boatmen
```

NAME

sprof - display profile data

SYNOPSIS

sprof [**-al** [file]

DESCRIPTION

Sprof interprets the file mon.out produced by the segmon subroutine. Under default modes, the symbol table in the named object file (a.out default) is read and correlated with the **mon.out**(5) profile file. For each external symbol, the percentage of time spent executing between that symbol and the next is printed (in decreasing order), together with the number of times that routine was called and the number of milliseconds per call.

In order for the number of calls to a routine to be tallied, the **-p** option of **scc** must have been given when the file containing the routine was compiled. This option also arranges for the **mon.out**(5) file to be produced automatically.

OPTIONS

-a Report all symbols rather than just external symbols.

-l List output by symbol value rather than decreasing percentage.

FILES

mon.out for profile
a.out for namelist

SEE ALSO

mon.out(5), segmon(3), sprofil(2), scc(1).

LIMITATIONS

Beware of quantization errors.

NAME

strings - print strings in object or other binary file

SYNOPSIS

strings [-] [-number] [**-o**] file ...

DESCRIPTION

Strings prints all strings of printable characters in file that are more than 4 characters long. If file is in object format, strings only looks for strings in the initialized data portion.

Strings is useful for identifying random object files and many other things.

OPTIONS

- Examine the entire file, even if it is in object format.

-number
number is the minimum string length instead of 4.

-o Give each string's offset in octal.

EXAMPLES

```
$ strings /bin/ls
/etc/passwd
/etc/group
total %D
%5u
%4D
%2d
%-6.6s
%-6d
%3d,%3d
%71d
%-7.7s %-4.4s
%-12.12s
%.14s
%s unreadable
ls: out of memory
ls: too many files
%s not found
(null)
Day Mon 00 00:00:00 1900
SunMonTueWedThuFriSat
```

```
$ strings -o /bin/ls
  7450 /etc/passwd
  7462 /etc/group
  7481 total %D
  7491 %5u
  7496 %4D
  7501 %2d
  7506 %-6.6s
  7513 %-6d
  7518 %3d,%3d
  7526 %71d
  7531  %-7.7s %-4.4s
  7547  %-12.12s
  7562 %.14s
  7571 %s unreadable
  7586 ls: out of memory
  7605 ls: too many files
  7625 %s not found
  7708 (null)
  7910 Day Mon 00 00:00:00 1900
  7936 SunMonTueWedThuFriSat
```

SEE ALSO

od(1).

NAME

strip - remove symbols and relocation bits and header (optional)

SYNOPSIS

strip [-h] file ...

DESCRIPTION

Strip removes the symbol table and relocation bits ordinarily attached to the output of the assembler and loader. This is useful to save space after a program has been debugged.

This effect of **strip** is the same as use of the **-s** option of **ld**.

OPTIONS

-h Cause the header and segment table to be stripped.

SEE ALSO

ld(1), objdu(1), objhdr(1), a.out(5).

DIAGNOSTICS

"file not in correct object format" if the file is not a nonsegmented executable file.

"cannot recreate file" if the file is not owned by the invoker of the program.

"file already stripped" if the file has no symbol table or relocation information and the -h option has not been specified.

NAME

stty - set the options for a terminal

SYNOPSIS

stty [-a] [-g] [options]

DESCRIPTION

Stty sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options.

OPTIONS

-a Reports all of the option settings.

-g Reports current settings in a form that can be used as an argument to another **stty** command.

MODES

Detailed information about the modes listed in the first five groups below may be found in **tty**(4). Options in the last group are implemented using options in the previous groups.

Control Modes

clocal (-clocal)
assume a line without (with) modem control.

cread (-cread)
enable (disable) the receiver.

cstopb (-cstopb)
use two (one) stop bits per character. (The System 8000 doesn't currently support **-cstopb**).

cs5 cs6 cs7 cs8
select character size (see **tty**(4)).

hup (-hup)
hang up (do not hang up) DATA-PHONE(Reg.) connection on last close.

hupcl (-hupcl)
same as **hup (-hup)**

parenb (-parenb)
enable (disable) parity generation and detection.

parodd (-parodd)
select odd (even) parity.

0 hang up phone line immediately. **Note:** this option

is provided to hang up a line (usually associated with a dial out modem) and should not be used on a line associated with a login shell.

50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 19200 extb
Set terminal baud rate to the number given (the speeds supported by the S8000 interface).

Input Modes

brkint (-brkint)
signal (do not signal) INTR on break.

icrnl (-icrnl)
map (do not map) CR to NL on input.

ignbrk (-ignbrk)
ignore (do not ignore) break on input.

igncr (-igncr)
ignore (do not ignore) CR on input.

ignpar (-ignpar)
ignore (do not ignore) parity errors.

inlcr (-inlcr)
map (do not map) NL to CR on input.

inpck (-inpck)
enable (disable) input parity checking.

istrip (-istrip)
strip (do not strip) input characters to seven bits.

iuclc (-iuclc)
map (do not map) upper-case alphabetics to lower case on input.

ixany (-ixany)
allow any character (only DC1) to restart output.

ixoff (-ixoff)
request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

ixon (-ixon)
enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.

parmrk (-parmrk)
mark (do not mark) parity errors (see **tty**(4)).

Output Modes

bsØ bs1 select style of delay for backspaces (see **tty**(4)).

crØ cr1 cr2 cr3
select style of delay for carriage returns (see **tty**(4)).

ffØ ff1 select style of delay for form-feeds (see **tty**(4)).

nlØ nl1 select style of delay for line-feeds (see **tty**(4)).

ocrnl (-ocrnl)
map (do not map) CR to NL on output.

ofdel (-ofdel)
fill characters are DELs (NULs).

ofill (-ofill)
use fill characters (use timing) for delays.

olcuc (-olcuc)
map (do not map) lower-case alphabetics to upper case on output.

onlcr (-onlcr)
map (do not map) NL to CR-NL on output.

onlret (-onlret)
on the terminal NL performs (does not perform) the CR function.

onocr (-onocr)
do not (do) output CRs at column zero.

opost (-opost)
post-process output (do not post-process output; ignore all other output modes).

tabØ tab1 tab2 tab3
select style of delay for horizontal tabs (see **tty**(4)).

vtØ vt1 select style of delay for vertical tabs (see **tty**(4)).

Local Modes

echo (-echo)

echo back (do not echo back) every character typed.

echoe (-echoe)
echo (do not echo) ERASE character as a backspace-space-backspace string. Note: this mode will erase the ERASEed character on many CRT terminals; however, it does not keep track of column position and can be confusing on escaped characters, tabs, and backspaces.

echok (-echok)
echo (do not echo) NL after KILL character.

echonl (-echonl)
echo (do not echo) NL.

icanon (-icanon)
enable (disable) canonical input (ERASE and KILL processing).

isig (-isig)
enable (disable) the checking of characters against the special control characters INTR and QUIT.

lfkc (-lfkc)
the same as **echok (-echok)**.

noflsh (-noflsh)
disable (enable) flush after INTR or QUIT.

xcase (-xcase)
canonical (unprocessed) upper/lower-case presentation.

Control Assignments

control-character c
set control-character to c, where control-character is **erase, kill, intr, quit, eof, eol, min,** or **time** (**min** and **time** are used with **-icanon**; see **tty**(4)). If c is preceded by an (escaped from the shell) caret (^), then the value used is the corresponding CTRL character (e.g., ^d is a **CTRL-d**); ^? is interpreted as DEL and ^- is interpreted as undefined.

line i set line discipline to i (Ø < i < 127).

Combination Modes

ek reset ERASE and KILL characters back to normal **^h** and **^x**.

evenp or **parity**
enable **parenb** and **cs7**.

lcase (**-lcase**)
set (unset) **xcase**, **iuclc**, and **olcuc**.

LCASE (**-LCASE**)
same as **lcase** (**-lcase**).

nl (**-nl**) unset (set) **icrnl**, **onlcr**. In addition **-nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.

oddp enable **parenb**, **cs7**, and **parodd**.

-parity, **-evenp**, or **-oddp**
disable **parenb**, and set **cs8**.

raw (**-raw** or **cooked**)
enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, EOT, or output post processing).

sane resets all modes to some reasonable values.

tabs (**-tabs** or **tab3**)
preserve (expand to spaces) tabs when printing.

term set all modes suitable for the terminal type term, where term is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.

the command

stty < /dev/ttyx.

prints the characteristics of ttyx.

SEE ALSO
tabs(1), ioctl(2), tty(4).

NAME

su - substitute user ID temporarily

SYNOPSIS

su [-] [name [arg ...]]

DESCRIPTION

Su demands the password of the specified userid, unless the current userid is ZEUS. **Su** then changes to the specified userid and invokes the user's default shell (passwd (5)) without changing the current directory or the environment (environ (5)). The new user ID stays in force until the shell exits.

If no userid is specified, ZEUS is assumed. To remind the super-user of the responsibilities, the shell substitutes # for its usual prompt.

Any additional arguments are passed to the shell, permitting the super-user to run shell procedures with restricted privileges (an arg of the form **-c** string executes string via the shell). When additional arguments are passed, **/bin/sh** is always used. When no additional arguments are passed, **su** uses the shell specified in the password file.

OPTIONS

- Cause the environment to be changed to the one that would be expected if the user actually logged in again. Otherwise, the environment is passed along with the possible exception of **$PATH**, which is set to **/bin:/etc:/usr/bin** for ZEUS.

SEE ALSO

csh(1), sh(1) env(1), login(1), environ(5).

NAME

sum - sum and count blocks in a file

SYNOPSIS

sum [-r] file

DESCRIPTION

Sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line.

OPTIONS

-r Cause the algorithm from the previous version of to be used in computing the checksum.

SEE ALSO

wc(1)

DIAGNOSTICS

"Read error" is indistinguishable from end of file on most devices; check the block count.

NAME

switch - C Shell flow control statement for decision-making

SYNOPSIS

switch (string)

case label1:

command

breaksw

case label2:

command

breaksw

default

command

endsw

DESCRIPTION

Switch is generally useful in the context of a foreach or while statement.

Each case label is successively matched against the specified string.

The file metacharacters " *****, **?**, **[**, and **]** " can be used in the case labels. If none of the labels match before a default label is found, the execution begins after the default label.

Each case label and the default label must appear at the beginning of a line. The command **breaksw** causes execution to continue after the **endsw**. Otherwise, control falls through case labels and default labels, as in C. If no label matches and there is no default, execution continues after the **endsw**.

SEE ALSO

break(1C), breaksw(1C), continue(1C), exit(1C), foreach(1C), if(1C), logout(1C), while(1C).
The C Shell in the ZEUS Utilities Manual

NAME

tabs - set tabs on a terminal

SYNOPSIS

tabs [tabspec] [**+mn**] [**-T**type]

DESCRIPTION

Tabs sets the tab stops on the user's terminal according to the tab specification tabspec, after clearing any previous settings. The user must of course be logged in on a terminal with remotely-settable hardware tabs.

Users of GE TermiNet terminals should be aware that they behave in a different way than most other terminals for some tab settings: the first number in a list of tab settings becomes the left margin on a TermiNet terminal. Thus, any list of tab numbers whose first element is other than 1 causes a margin to be left on a TermiNet, but not on other terminals. A tab list beginning with 1 causes the same effect regardless of terminal type. It is possible to set a left margin on some other terminals, although in a different way (see below).

Four types of tab specification are accepted for tabspec: "canned," repetitive, arbitrary, and file. If no tabspec is given, the default value is **-8**, i.e., ZEUS "standard" tabs. The lowest column number is 1. Note that for **tabs**, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0, e.g., the DASI 300, DASI 300s, and DASI 450.

-code Gives the name of one of a set of "canned" tabs. The legal codes and their meanings are as follows:

-a 1,10,16,36,72
Assembler, IBM S/370, first format

-a2 1,10,16,40,72
Assembler, IBM S/370, second format

-c 1,8,12,16,20,55
COBOL, normal format

-c2 1,6,10,14,49
COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows:
<:t-c2 m6 s66 d:>

-c3 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67

COBOL compact format (columns 1-6 omitted), with more tabs than **-c2**. This is the recommended format for COBOL. The appropriate format specification is:
<:t-c3 m6 s66 d:>

-f 1,7,11,15,19,23
FORTRAN

-p 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61
PL/I

-s 1,10,55
SNOBOL

-u 1,12,20,44
UNIVAC 1100 Assembler

In addition to these "canned" formats, three other types exist:

-n A repetitive specification requests tabs at columns 1+n, 1+2*n, etc. Note that such a setting leaves a left margin of n columns on TermiNet terminals only. Of particular importance is the value **-8**: this represents the ZEUS "standard" tab setting, and is the most likely tab setting to be found at a terminal. It is required for use with the **nroff**(1) **-h** option for high-speed output. Another special case is the value **-0**, implying no tabs at all.

n1,n2,...
The arbitrary format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the tab lists 1,10,20,30 and 1,10,+10,+10 are considered identical.

--file If the name of a file is given, **tabs** reads the first line of the file, searching for a format specification. If it finds one there, it sets the tab stops according to it, otherwise it sets them as **-8**. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the **pr**(1) command:
tabs -- file; pr file

Any of the following may be used also; if a given flag occurs more than once, the last value given takes effect:

-T*type* **Tabs** usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *Type* is a name listed in **term**(7). If no **-T** flag is supplied, *tabs* searches for the **$TERM** value in the *environment* (see **environ**(7)). If no *type* can be found, **tabs** tries a sequence that will work for many terminals.

+m*n* The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n*+1 the left margin. If **+m** is given without a value of *n*, the value assumed is 10. For a TermiNet, the first value in the tab list should be 1, or the margin will move even further to the right. The normal (leftmost) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly.

Tab and margin setting is performed via the standard output.

DIAGNOSTICS

illegal tabs when arbitrary tabs are ordered incorrectly.

illegal increment when a zero or missing increment is found in an arbitrary specification.

unknown tab code when a "canned" code cannot be found.

can't open if --*file* option used, and file can't be opened.

file indirection if --*file* option used and the specification in that file points to yet another file. Indirection of this form is not permitted.

SEE ALSO

nroff(1), environ(7), term(7).

LIMITATIONS

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.
It is generally impossible to usefully change the left margin without also setting tabs.
Tabs clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 40.

NAME

tail - print the last 10 lines of a file

SYNOPSIS

tail [+[number[**lbc**] [**-f**]] [file]

DESCRIPTION

Tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance +number from the beginning, or -number from the end of the input (if number is null, the value 10 is assumed). Number is counted in units of lines, blocks, or characters, according to the appended option. When no units are specified, counting is by lines.

OPTIONS

b Number is counted in blocks.

c Number is counted in characters.

l Number is counted in lines.

-f "follow" option. If the input file is not a pipe, the program will not terminate after the line of the input file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus it may be used to monitor the growth of a file that is being written by some other process.

EXAMPLE

the command:

tail -f file

prints the last ten lines of file followed by any lines that are appended to file between the time **tail** is initiated and killed.

SEE ALSO

cat(1), dog(1), head(1), more(1), dd(1).

LIMITATIONS

Tails relative to the end of the file are treasured up in a buffer, and thus are limited in length.

Various kinds of behavior may happen with character special files.

NAME

talk - communicate with another user

SYNOPSIS

talk user [ttyname]

DESCRIPTION

Talk copies lines from your terminal to that of another user. When first called, it sends the message

Message from name ttyname...

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal or an interrupt is sent. At that point **talk** writes "EOF" on the other terminal and exits.

To write to a user who is logged in more than once, the ttyname argument may be used to indicate the appropriate terminal.

Permission to write may be denied or granted by use of the **mesg**(1) command. At the outset writing is allowed. Certain commands, in particular **nroff**(1) and **pr**(1) disallow messages in order to prevent messy output.

Talk differs from **write**(1) in that the message is written as each character is read, rather than at end of line.

Typing the "escape" key, " ESC " causes a new line, and a "less than sign", " < ", to be sent to the sender's terminal, and a new line, and a "greater than sign", ">", to be sent to the receiver's terminal. This sequence is meant to indicate that it is the other persons turn to **talk.**

The following protocol is suggested for using **talk:** one user first writes to a second user, the first user should wait for the second user to respond before starting to send. Each party should end each part of their communication with an "escape" key "ESC".

FILES

/etc/utmp to find user

SEE ALSO

mesg(1), who(1), mail(1), write(1)

LIMITATIONS

Escape to the shell via "!" is not supported, as it is in **write**(1).

NAME

tar - tape archiver

SYNOPSIS

tar [**crtux**N**bflmqvw**] [name ...]

DESCRIPTION

Tar saves and restores files in an archive. This is usually on tape, but may be in a file archive. Its actions are controlled by the key argument. The key is a string of characters containing at most one function letter and possibly one or more function modifiers.

Other arguments to the command are file or directory names specifying which files are to be dumped or restored. In all cases, appearance of a directory name refers to the files and (recursively) to subdirectories of that directory.

The default destination device is /dev/tardev, which is created by linking it to a tape device. If /dev/tardev does not exist, **tar** uses /dev/rctØ for the default.

The cartridge tape unit has the capability of accessing each of the four tracks on a tape individually. The files /dev/rctØa, ..., /dev/rctØd, /dev/rctla, ... refer to the individual tracks on a tape.

OPTIONS

The function portion of the key is specified by one of the following letters:

c Create a new tape: writing begins on the beginning of the archive instead of after the end of the archive. This destroys the existing files on the archive.

r The named files are written on the end of the archive without disturbing existing files on the tape.

t The names of the specified files are listed each time they occur on the archive. If no file argument is given, all of the names on the tape are listed.

u The named files are added to the archive if either they are not already there or have been modified since last put on the archive.

x The named files are extracted from the archive. If the named file matches a directory whose contents had been written onto the archive, this directory is recursively extracted.

The owner, modification time, and mode are restored if

possible. If no file argument is given, the entire contents of the archive is extracted. If multiple entries specifying the same file are on the tape (archive), the last one overwrites all earlier files.

The following characters can be used in addition to the letter that selects the function desired.

0,...,7
this modifier selects the drive on which the tape is mounted. The default is **0**.

b the next argument is the blocking factor. The range is 1 to 20, the default is 8. This option is only used with raw magnetic tape archives (**f** below). The block size is determined automatically when reading tapes (key letters x and t).

f file
the next argument is the name of the archive or file instead of /dev/rct0. Possible alternatives are: /dev/rct0, which is the raw i/o tape device, or /dev/nrct0 for the no rewind device. If the name of the file is -, tar writes to standard output or reads from standard input, whichever is appropriate. Thus, **tar** can be used as the head or tail of a filter chain. **Tar** can also be used to move hierarchies with the command

```
cd fromdir; tar cf - . | (cd todir; tar xf -)
```

l print diagnostics if it cannot resolve all of the links to the files dumped. If this is not specified, no error messages are printed.

m do not restore the modification times. The mod time is the time of extraction.

q do a quick extract. It retrieves only the first occurrence of the file from the archive and immediatly exits. This is a modifier to the **x** option and must always be used with the **x** option.

v print the name of each file preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.

w wait for user confirmation. If y is given, the action is performed.

If the '**x**' option is used with this option, then the

user can optionally give a filename (following the y answer) for tar to extract into. For example the following sequence of statements:

```
tar xw file.name
x file.name: y file.name1
```

causes tar to put the file named 'file.name' from the tape (archive) into the file named 'file.name1' in the present working directory.

FILES

/dev/ct? /dev/rct?
/tmp/tar*

SEE ALSO

ct(4).

DIAGNOSTICS

Bad key characters and tape read/write errors.
Not enough memory is available to hold the link tables.

LIMITATIONS

There is no way to ask for the n-th occurrence of a file.

Tape errors are handled ungracefully.

The **u** option can be slow.

The **b** option must not be used with archives that are going to be updated (r or u option). The current tape driver cannot backspace raw tape. If the archive is on a disk file the **b** option should not be used, as updating an archive stored in this manner can destroy it.

The current limit on file name length is 100 characters.

If **f** is used with **c**, it will create the file specified by **f** if it does not exist.

NAME

tbl - format tables for nroff or troff

SYNOPSIS

tbl [files] ...

DESCRIPTION

Tbl is a preprocessor for formatting tables for **nroff**(1) or **troff**(1). The input files are copied to the standard output, except for lines between .TS and .TE command lines, which are assumed to describe tables and reformatted. Details are given in the reference manual.

If no arguments are given, **tbl** reads the standard input, so it may be used as a filter. When it is used with **eqn**(1) or **neqn**(1) the **tbl** command should be first, to minimize the volume of data passed through pipes.

EXAMPLES

As an example, letting \t represent a tab (which should be typed as a genuine tab) the input

```
.TS
c s s
c c s
c c c
l n n.
Household Population
Town\tHouseholds
\tNumber\tSize
Bedminster\t789\t3.26
Bernards Twp.\t3087\t3.74
Bernardsville\t2018\t3.30
Bound Brook\t3425\t3.04
Branchburg\t1644\t3.49
.TE
```

yields:

	Household Population	
Town	Households	
	Number	Size
Bedminster	789	3.26
Bernards Twp.	3087	3.74
Bernardsville	2018	3.30
Bound Brook	3425	3.04
Branchburg	1644	3.49

SEE ALSO

eqn(1), nroff(1), troff(1).
TBL -- For Formatting Tables in the ZEUS Utilities Manual.

NAME

tee - pipe fitting

SYNOPSIS

tee [**-a**] [**-i**] [file] ...

DESCRIPTION

Tee transcribes the standard input to the standard output and makes copies in the files.

OPTIONS

-a causes the output to be appended to the files rather than overwriting them.

-i ignores interrupts.

EXAMPLE

The command:

code < /etc/passwd | **tee** temp

translates the file /etc/passwd with the **code** program, printing the results on the terminal, and putting a second copy in the file temp.

NAME

test - evaluate files, strings, and numbers

SYNOPSIS

test expr

DESCRIPTION

Test evaluates the expression expr, and if its value is true then returns zero exit status; otherwise, a nonzero exit status is returned. **Test** returns a nonzero exit if there are no arguments.

The following primitives are used to construct expr:

-r file true if the file exists and is readable.

-w file true if the file exists and is writable.

-x file true if the file exists and is executable.

-f file true if the file exists and is not a directory.

-d file true if the file exists and is a directory.

-c file true if the file exists and is a character special file.

-b file true if the file exists and is a block special file.

-u file true if the file exists and its set-user-ID bit is set.

-g file true if the file exists and its set-group-ID bit is set.

-k file true if the file exists and its sticky bit is set.

-s file true if the file exists and has a size greater than zero.

-t [fildes]
true if the open file whose file descriptor number is fildes (1 by default) is associated with a terminal device.

-z s1 true if the length of string s1 is zero.

-n s1 true if the length of the string s1 is nonzero.

s1 = s2 true if the strings s1 and s2 are equal.

s1 **!=** s2 true if the strings s1 and s2 are not equal.

s1 true if s1 is not the null string.

n1 **-eq** n2
true if the integers n1 and n2 are algebraically equal. Any of the comparisons **-ne**, **-gt**, **-ge**, **-lt**, or **-le** can be used in place of **-eq**.

These primaries can be combined with the following operators:

! unary negation operator

-a binary and operator

-o binary or operator

(expr)
parentheses for grouping.

-a has higher precedence than **-o**.

All the operators and flags are separate arguments to **test**. Parentheses are meaningful to the shell and must be escaped.

SEE ALSO

csh(1), expr(1), find(1), sh(1).
The C Shell in the ZEUS Utilities Manual

NAME

time - time a command

SYNOPSIS

time [-v] command (csh internal and /bin/time)

DESCRIPTION

The given command is executed; after it is complete, **time** prints the time spent in the system, the time spent in execution of the command, the elapsed time during the command, and percentage used of cpu time to real time. Times are reported in seconds.

OPTIONS

-v A verbose version of the command's output is printed.

EXAMPLES

An output of this command is:

```
%time cc file.c
3.3u 3.5s Ø:19 35%
```

An output using the -v option is:

```
%time -v cc file.c
3.3 user-sec., 3.5 system-sec., ØØ:19 real-time, 35% of capacit
```

The times are printed on the diagnostic output stream.

SEE ALSO

csh(1), sh(1), time(1C).

LIMITATIONS

Elapsed time is accurate to the second, while the CPU times are measured to the 6Øth second. Thus the sum of the CPU times can be up to a second larger than the elapsed time.

NAME

time - time a command

SYNOPSIS

time [**-v**] command (csh internal and /bin/time)

DESCRIPTION

The given command is executed; after it is complete, **time** prints the time spent in the system, the time spent in execution of the command, the elapsed time during the command, and percentage used of cpu time to real time. Times are reported in seconds.

OPTIONS

-v A verbose version of the command's output is printed.

EXAMPLES

An output of this command is:

```
%time cc file.c
3.3u 3.5s Ø:19 35%
```

An output using the -v option is:

```
%time -v cc file.c
3.3 user-sec., 3.5 system-sec., ØØ:19 real-time, 35% of capacity
```

The times are printed on the diagnostic output stream.

SEE ALSO

csh(1), sh(1), time(1).
The C Shell in the ZEUS Utilities Manual

LIMITATIONS

Elapsed time is accurate to the second, while the CPU times are measured to the 6Øth second. Thus the sum of the CPU times can be up to a second larger than the elapsed time.

NAME

timex - time a command and generate a system activity report

SYNOPSIS

timex command

DESCRIPTION

The given command is executed; after its execution, **timex** prints the elapsed time, the time spent executing command, and the time spent in the system, as **time**(1) does. It also reports system activity that occurred during command execution, including CPU utilization, I/O activity, system switching and swapping, and file system access. All system activity is reported, not just that due to command.

The output of **timex** is written on standard error.

SEE ALSO

time(1), time(1C), sar(M).

NAME

touch - update access and modification times of files

SYNOPSIS

touch [**-acm**] [mmddhhmm[yy]] file ...

DESCRIPTION

Touch causes the access and modification times of each file to be updated. If no time is specified (see **date** (1)) the current time is used. The return code from **touch** is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

OPTIONS

-a causes **touch** to update only the access time.

-c causes **touch** not to attempt to create a file that does not exist; The default is to create one.

-m causes **touch** to update only the modification time. The default option is **-am** (both access and modification times are updated).

EXAMPLES

```
% date
  Mon Jan 17 15:28:37 PST 1983

% ls -l
  total 2
  -rw-rw-rw- 1 carolh       49 Oct 26 09:19 list1
  -rw-rw-rw- 1 carolh       75 Oct 21 14:22 list2

% touch list2; touch -m 12311000 list1

% ls -l
  total 2
  -rw-rw-rw- 1 carolh       49 Dec 31 10:00 list1
  -rw-rw-rw- 1 carolh       75 Jan 17 15:28 list2

% touch -c list3

% ls -l
  total 2
  -rw-rw-rw- 1 carolh       49 Dec 31 10:00 list1
  -rw-rw-rw- 1 carolh       75 Jan 17 15:28 list2

% touch list3

% ls -l
  total 3
  -rw-rw-rw- 1 carolh       49 Dec 31 10:00 list1
```

```
-rw-rw-rw- 1 carolh        75 Jan 17 15:28 list2
-rw-r--r-- 1 carolh         Ø Jan 17 15:29 list3
```

DIAGNOSTICS

The exit status is the number of files unsucessfully touched. If all files were sucessfully touched, the exit status is Ø.

NAME

tr - translate characters

SYNOPSIS

tr [**-cds**] [string1 [string2]]

DESCRIPTION

Tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in string1 are mapped into the corresponding characters of string2. Any combination of **-cds** can be used:

-c Complements the set of characters in string1 with respect to characters whose ASCII codes are 001 through 377 octal.

-d Deletes all input characters in string1.

-s Squeezes all strings of repeated output characters that are in string2 to single characters.

Abbreviation conventions can be used for ranges of characters or repeated characters:

[a-z] Stands for the characters whose ASCII codes run from character **a** to character **z**.

[a*n] Stands for n repetitions of **a**. If the first digit of n is **0**, n is considered octal; otherwise, n is taken to be decimal. A zero or missing n is taken to be huge; this is used for padding string2.

The escape character \ can be used as in the shell to remove special meaning from any character. A \ followed by 1, 2, or 3 octal digits stands for the character whose ASCII code is given by those digits.

The following example creates a list of all the words in file1, one per line in file2, where a word is taken to be a maximal string of alphabetics. The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for newline.

```
tr -cs "[A-Z][a-z]" "[\012*]" <file1 >file2
```

SEE ALSO

ed(1), sh(1), ascii(7).

LIMITATIONS

Won't handle ASCII **NUL** in string1 or string2; always deletes **NUL** from input.

NAME

troff, nroff - text formatting and typesetting

SYNOPSIS

troff [**imnoqrsabfptw**] ... [file] ...

nroff [**imnoqrsehT**] ... [file] ...

DESCRIPTION

Troff formats text in the named files for printing on a Graphic Systems C/A/T phototypesetter; **nroff** for typewriter-like devices. Their capabilities are described in the Nroff/Troff User's Manual.

If no file argument is present, the standard input is read. An argument consisting of a single minus (-) is taken to be a file name corresponding to the standard input.

If the file /usr/adm/tracct is writable, **troff** keeps phototypesetter accounting records there. The integrity of that file can be secured by making **troff** a "set user-id" program.

OPTIONS

-i Read standard input after the input files are exhausted.

-mname Prepend the macro file **/usr/lib/tmac/tmac.**name to the input files.

-nN Number first generated page N.

-olist Print only pages whose page numbers appear in the comma-separated list of numbers and ranges. A range N-M means pages N through M; an initial -N means from the beginning to page N; and a final N- means from N to the end.

-q Invoke the simultaneous input/output mode of the **rd** request.

-raN Set register a (one-character) to N.

-sN Stop every N pages. **Nroff** halts prior to every N pages (default N=1) to allow paper loading or changing, and resumes upon receipt of a new line. **Troff** stops the phototypesetter every N pages, produce a trailer to allow changing cassettes, and resumes when the typesetter's start button is pressed.

Nroff only

-e Produce equally-spaced words in adjusted lines, using

full terminal resolution.

-h Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every eight nominal character widths.

-Tname Prepare output for specified terminal. Known names are **37** for the (default) Teletype Corporation Model 37 terminal, **tn300** for the GE TermiNet 300 (or any terminal without half-line capability), **300S** for the DASI-300S, **300** for the DASI-300, and **450** for the DASI-450 (Diablo Hyterm).

Troff only

-a Send a printable ASCII approximation of the results to the standard output.

-b Report whether the phototypesetter is busy or available. No text processing is done.

-f Refrain from feeding paper and stopping phototypesetter at the end of the run.

-pN Print all characters in point size N while retaining all prescribed spacings and motions, to reduce phototypesetter elasped time.

-t Direct output to the standard output instead of the phototypesetter.

-w Wait until phototypesetter is available, if currently busy.

FILES

/usr/lib/suftab	suffix hyphenation tables
/tmp/ta*	temporary file
/usr/lib/tmac/tmac.*	standard macro files
/usr/lib/term/*	terminal driving tables for **nroff**
/usr/lib/font/*	font width tables for **troff**
/dev/cat	phototypesetter
/usr/adm/tracct	accounting statistics for /dev/cat

SEE ALSO

cw(1), col(1), eqn(1), man(7), me(7), ms(7), tbl(1).

Nroff/Troff User's Manual
A TROFF Tutorial
Writing Papers with Nroff Using -ME
all in the ZEUS Utilities Manual

NAME

true, false - provide truth values

SYNOPSIS

true

false

DESCRIPTION

True returns status indicating success. False returns status indicating failure. They are typically used in input to the C Shell, **csh**(1) or the Bourne Shell, **sh**(1).

EXAMPLES

Here are some examples:

for sh(1):

```
while true
do
     command
done
```

for csh(1):

```
#
while( { true } )
     command
end
```

SEE ALSO

csh(1), sh(1)

DIAGNOSTICS

True has exit status zero, **false** nonzero.

NAME

tsort - topological sort

SYNOPSIS

tsort [file]

DESCRIPTION

Tsort produces an ordered list of items consistent with a partial ordering of items mentioned in the input file. If no file is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO

lorder(1).

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

NAME

tty - get terminal name

SYNOPSIS

tty

DESCRIPTION

Tty prints the pathname of the user's terminal.

EXAMPLES

```
% tty
 /dev/tty05
```

DIAGNOSTICS

"Not a tty" if the standard input file is not a terminal.

NAME

uimage - Zobj to a.out translator

SYNOPSIS

uimage file1 [-o file2]

DESCRIPTION

Uimage translates Zobj object files produced by **plzcg**(1) to a.out format object files. The input file is in Zobj format and the default output file is a.out. With the -v option (verbose mode), more information is given about what is being done. **Uimage** is invoked automatically by the plz compiler driver, **plz**(1). It works on nonsegmented Zobj object files only.

FILES

/tmp/t* temporary files

SEE ALSO

objdu(1), plzcg(1), plzsys(1), plz(1), a.out(5)

DIAGNOSTICS.

Files produced by the code generator plzcg(1), must be error free.

Bad input files causes errors about nonexistent or incorrect Zobj opcodes.

NAME

umask - set file-creation mode mask

SYNOPSIS

umask [nnn]

DESCRIPTION

The user file-creation mode mask is set to nnn. The three octal digits refer to read/write/execute permissions for owner, group, and others, respectively (see **chmod**(2) and **umask**(2)). The value of each specified digit is subtracted from the corresponding "digit" specified by the system for the creation of a file (see creat(2)).

If nnn is omitted, the current value of the mask is printed.

Umask is recognized and executed by the shell.

EXAMPLE

```
% umask 000

% cat /dev/null > umask.00

% ls -l umask.000
-rw-rw-rw- 1 deck          0 Mar 13 12:23 umask.000
```

The other umask values are as follows:

```
-rw-rw-rw- 1 deck          0 Mar 13 12:23 umask.111
-r--r--r-- 1 deck          0 Mar 13 12:23 umask.222
-r--r--r-- 1 deck          0 Mar 13 12:23 umask.333
--w--w--w- 1 deck          0 Mar 13 12:23 umask.444
--w--w--w- 1 deck          0 Mar 13 12:23 umask.555
---------- 1 deck          0 Mar 13 12:23 umask.666
---------- 1 deck          0 Mar 13 12:23 umask.777
```

If two digits are specified, as in:

```
% umask 22
```

The first (owner) permission bits are untouched and the second and third fields are affected as in the following file:

```
-rw-r--r-- 1 deck          0 Mar 13 12:23 umask.22
```

If only a single number is specified, it affects the last field (the "others" permission bits) as in the command:

```
% umask 2
```

The following file permission bits result:

```
-rw-rw-r-- 1 deck          Ø Mar 13 12:23 umask.2
```

In other words, **umask Ø22** removes group and others write permission (files normally created with mode **777** become mode **755**; files created with mode **666** become mode **644**).

SEE ALSO

chmod(1), sh(1), chmod(2), chown(1), creat(2), umask(2), chmog(M).
The C Shell in the ZEUS Utilities Manual

NAME

uname - print the name of current ZEUS

SYNOPSIS

uname [**-anrsv**]

DESCRIPTION

Uname prints the current system name of ZEUS on the standard output file. It is mainly useful to determine what system one is using.

OPTIONS

-a print all the available information.

-n print the nodename (name that the system is known by to a communications network).

-r print the operating system release.

-s print the system name (default).

-v print the operating system version.

SEE ALSO

uname(2).

NAME

unget - undo a previous get of an SCCS file

SYNOPSIS

unget [-r*SID*] [**-s**] [**-n**] *files*

DESCRIPTION

Unget undoes the effect of a **get -e** done prior to creating the intended new delta. If a directory is named, **unget** behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of - is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

OPTIONS

Keyletter arguments apply independently to each named file.

-n Causes the retention of the gotten file which would normally be removed from the current directory.

-r*SID* Uniquely identifies which delta is no longer intended. (This would have been specified by **get** as the "new delta"). The use of this keyletter is necessary only if two or more outstanding **gets** for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified *SID* is ambiguous, or if it is necessary and omitted on the command line.

-s Suppresses the printout, on the standard output, of the intended delta's *SID*.

SEE ALSO

delta(1), get(1), sact(1).
SCCS -- Source Code Control System in the ZEUS Utilities Manual

DIAGNOSTICS

Use **help**(1) for explanations.

NAME

uniq - report repeated lines in a file

SYNOPSIS

uniq [**-cdu** [**+**n] [**-**n]] [infile [outfile]]

DESCRIPTION

Uniq reads infile comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on outfile. Repeated lines must be adjacent in order to be found (see **sort**(1)). The normal mode output is the union of the **-u** and **-d** mode outputs.

The n arguments specify skipping an initial portion of each line in the comparison:

-n The first n fields together with any blanks before each are ignored. A field is defined as a string of nonspace, nontab characters separated by tabs and spaces from its neighbors.

+n The first n characters are ignored. Fields are skipped before characters.

OPTIONS

-c Supersedes **-u** and **-d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

-d One copy of just the repeated lines is to be written.

-u The lines that are not repeated in the original file are output.

SEE ALSO

sort(1), comm(1), diff(1), diff3(1), bdiff(1).

NAME

units - conversion program

SYNOPSIS

units

DESCRIPTION

Units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have: inch
You want: cm
     * 2.54000e+00
       3.93701e-01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```
You have: 15 pounds force/in2
You want: atm
     * 1.02069e+00
       9.79730e-01
```

Units only does multiplicative scale changes. Thus it can convert Kelvin to Rankine, but not Centigrade to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter
c	speed of light
e	charge on an electron
g	acceleration of gravity
force	same as g
mole	Avogadro's number
water	pressure head per unit height of water
au	astronomical unit

`Pound' is a unit of mass. Compound names are run together, e.g. `lightyear'. British units that differ from their US counterparts are prefixed thus: `brgallon'. Currency is denoted `belgiumfranc', `britainpound', ... For a complete list of units, `cat /usr/lib/units'.

FILES

/usr/lib/units

NAME

users - compact list of users who are on the system

SYNOPSIS

users

DESCRIPTION

Users lists the login names of the users currently on the system in a compact, one-line format.

FILES

/etc/utmp

SEE ALSO

who(1), whodo(1), whois(1), whoami(1).

NAME

uucp, uulog, uuname - ZEUS to ZEUS copy

SYNOPSIS

uucp [**cCdefmn**] ... source-file ... destination-file

uulog [**su**] ...

uuname

DESCRIPTION

Uucp copies files named by the source-file arguments to the destination-file argument. A file name can be a path name on your machine, or can have the form:

system-name!path-name

where system-name is taken from a list of system names which **uucp** knows about. Shell metacharacters **?*[]** appearing in path-name are expanded on the appropriate system.

Path names can be one of:

(1) a full path name;

(2) a path name preceded by ~user where user is a login name on the specified system and is replaced by that user's login directory;

(3) a path name preceded by ~/user where user is a login name on the specified system and is replaced by that user's directory under PUBDIR;

(4) anything else is prefixed by the current directory.

If the result is an erroneous path name for the remote system the copy will fail. If the destination-file is a directory, the last part of the source-file name is used. If a simple ~user destination is inaccessible to **uucp**, data is copied to a spool directory and the user is notified by **mail**(1).

Uucp preserves execute permissions across the transmission and gives Ø666 read and write permissions (see **chmod**(2)).

OPTIONS

-c Use the source file when copying out rather than copying the file to the spool directory (default).

-C Copy the source file to the spool directory.

-d Make all necessary directories for the file copy

(default).

-esys
Send the **uucp** command to system sys to be executed there. (Note - this will only be successful if the remote machine allows the **uucp** command to be executed by **/usr/lib/uucp/uuxqt.**)

-f Do not make intermediate directories for the file copy.

-m Send mail to the requester when the copy is complete.

-nuser
Notify user on the remote system that a file was sent.

Uulog maintains a summary log of **uucp** and **uux**(1) transactions in the file **/usr/spool/uucp/LOGFILE** by gathering information from partial log files named **/usr/spool/uucp/LOG.*.?.** (These files will only be created if the **LOGFILE** is being used by another process.) It removes the partial log files.

The options cause **uulog** to print logging information:

-ssys
Print information about work involving system sys.

-uuser
Print information about work done for the specified user.

Uuname lists the uucp names of known systems. The **-l** option returns the local system name.

FILES

/usr/spool/uucp
spool directory

/usr/spool/uucppublic
public directory for receiving and sending (PUBDIR)

/usr/lib/uucp/*
other data and program files

SEE ALSO

mail(1), uux(1).

WARNING

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by path

name; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary path names. As distributed, the remotely accessible files are those whose names begin **/usr/spool/uucppublic** (equivalent to **~nuucp** or just **~**).

LIMITATIONS

All files received by **uucp** will be owned by **uucp.**
The **-m** option only works sending files or receiving a single file. (Receiving multiple files specified by special shell characters **?*[]** will not activate the **-m** option.)

NAME

uustat - uucp status inquiry and job control

SYNOPSIS

uustat [jkmosuvy] ...

DESCRIPTION

Uustat displays the status of, or cancels, previously specified **uucp** commands, or provides general status on **uucp** connections to other systems.

OPTIONS

-c*hour* Remove the status entries which are older than *hour* hours. This can only be initiated by the user **uucp** or the super-user.

-**j***all* Report the status of all the **uucp** requests.

-**k***jobn* Kill the **uucp** request whose job number is *jobn*. The killed **uucp** request must belong to the person issuing the **uustat** command or issued by the super-user.

-**m***mch* Report the status of accessibility of machine *mch*. If *mch* is specified as **all**, then the status of all machines known to the local **uucp** are provided.

-**o***hour* Report the status of all **uucp** requests older than *hour* hours.

-s*sys* Report the status of all **uucp** requests which communicate with remote system *sys*.

-u*user* Report the status of all **uucp** requests issued by *user*.

-**v** Report the **uucp** status verbosely. If this option is not specified, a status code is printed with each **uucp** request.

-**y***hour* Report the status of all **uucp** requests younger than *hour* hours.

When no options are given, **uustat** outputs the status of all **uucp** requests issued by the current user. Only one of the options **-c**, **-j**, **-k**, **-m**, or the rest of other options can be specified.

For example, the command

```
uustat -uhdc -smhtsa -y72 -v
```

prints the verbose status of all **uucp** requests issued by user hdc to communicate with system mhtsa within the last 72 hours. The meanings of the job request status are:

```
job-number user remote-system command-time status-time
status
```

where status can be either an octal number or a verbose description. The octal code corresponds to the following description:

OCTAL	STATUS
00001	the copy failed, but the reason cannot be determined
00002	permission to access local file is denied
00004	permission to access remote file is denied
00010	bad **uucp** command is generated
00020	remote system cannot create temporary file
00040	cannot copy to remote directory
00100	cannot copy to local directory
00200	local system cannot create temporary file
00400	cannot execute **uucp**
01000	copy succeeded
02000	copy finished, job deleted
04000	job is queued

The meanings of the machine accessibility status are:

```
system-name time status
```

where time is the latest status time and status is a self-explanatory description of the machine status.

FILES

/usr/spool/uucp	spool directory
/usr/lib/uucp/L_stat	system status file
/usr/lib/uucp/R_stat	request status file

SEE ALSO

uucp(1).

NAME

uux - zeus to zeus command execution

SYNOPSIS

uux [-] command-string

DESCRIPTION

Uux will gather Ø or more files from various systems, execute a command on a specified system and send standard output to a file on a specified system.

The command-string is made up of one or more arguments that look like a shell command line, except that the command and file names may be prefixed by system-name!. A null system-name is interpreted as the local system.

File names may be one of

(1) a full pathname;

(2) a pathname preceded by ~xxx; where xxx is a userid on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

Any special shell characters such as <>;| should be quoted either by quoting the entire command-string, or quoting the special characters as individual arguments.

OPTIONS

- Cause the standard input to the **uux** command to be the standard input to the command-string.

EXAMPLES

uux "!diff usg!/usr/dan/f1 pwba!/a4/dan/f1 > !fi.diff"

will get the f1 files from the usg and pwba machines, execute a **diff** command and put the results in f1.diff in the local directory. Note that `!' is a csh metacharacter and needs to be escaped by "" when running uux from the csh.

FILES

/usr/uucp/spool - spool directory
/usr/uucp/* - other data and programs

SEE ALSO

uucp(1).

WARNING

An installation may, and for security reasons generally will, limit the list of commands executable on behalf of an

incoming request from **uux**. Typically, a restricted site will permit little other than the receipt of mail via **uux**.

LIMITATIONS

Only the first command of a shell pipeline may have a system-name!. All other commands are executed on the system of the first command.

The use of the shell metacharacter * will probably not do what you want it to do.

The shell tokens << and >> are not implemented.

There is no notification of denial of execution on the remote machine.

NAME

val - validate SCCS file

SYNOPSIS

val -
val [**-m**name]
[**-r**SID]
[**-s**]
[**-y**type]
files

DESCRIPTION

Val determines if the specified file is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to **val** can appear in any order. The arguments consist of keyletter arguments, which begin with a -, and named files.

Val has a special argument, -, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

Val generates diagnostic messages on the standard output for each command line and file processed and also returns a single 8-bit code upon exit as described below.

The 8-bit code returned by **val** is a disjunction of the possible errors, i.e., can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit Ø = missing file argument
bit 1 = unknown or duplicate keyletter argument
bit 2 = corrupted SCCS file
bit 3 = can't open file or file not SCCS
bit 4 = SID is invalid or ambiguous
bit 5 = SID does not exist
bit 6 = %Y%, **-y** mismatch
bit 7 = %M%, **-m** mismatch

Note that **val** can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned - a logical **OR** of the codes generated for each command line and file processed.

KEYS

The keyletter arguments are defined as follows. The effects of any keyletter argument apply independently to each named file on the command line.

-mname
The argument value name is compared with the SCCS %M% keyword in file.

-rSID
The argument value SID (SCCS IDentification String) is an SCCS delta number. A check is made to determine if the SID is ambiguous (e.g., **r**1 is ambiguous because it physically does not exist but implies 1.1, 1.2, etc. which may exist) or invalid (e.g., **r**1.Ø or **r**1.1.Ø are invalid because neither case can exist as a valid delta number). If the SID is valid and not ambiguous, a check is made to determine if it actually exists.

-s The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line.

-ytype
The argument value type is compared with the SCCS %Y% keyword in file.

SEE ALSO
admin(1), delta(1), get(1), prs(1).

DIAGNOSTICS
Use **help**(1) for explanations.

LIMITATONS
Val can process up to 5Ø files on a single command line. Any number above 5Ø will produce a **core** dump.

NAME

vi - screen oriented (visual) display editor based on ex

SYNOPSIS

vi [**-t** tag] [**-r**] [**-R**] [**+**[command]] [**-l**] name ...

DESCRIPTION

Vi (visual) is a display oriented text editor based on **ex**(1). **Ex** and **vi** run the same code; it is possible to get to the command mode of **ex** from within **vi** and vice-versa. In addition, a "read-only" version of vi, **view**, is available for file perusing.

Vi looks at the environment of the shell to find out on which type of terminal the editing session is to take place. If the c shell (**csh**(1)) is running, the command **printenv** displays that environment; if the shell (**sh**(1)) is running, the command **set** does the display.

Introduction to Display Editing with Vi provide full details on using **vi**.

OPTIONS

-l This sets up **vi** for LISP editing; that is, the editing options, showmatch and lisp are set.

-r This option is used to recover named files after an editor or system crash; the last saved version is retrieved.

-R This option is used to invoke the "read only" version of **vi**; this is the same as using the "view" command.

-t tag
The cursor will be positioned at the definition of tag immediately after **vi** is entered.

+[command]
The editor begins by executing the command, command; if command is omitted, then the editor begins with the cursor positioned at the last line of the file.

name Name of the file(s) to be edited.

EXAMPLES

To change the terminal type under the c shell to, type vtz-2/10, enter

```
# setenv TERM vtz
```

To change the type under the shell, type

```
# set TERM=vtz
# export TERM
```

FILES

See **ex**(1)

SEE ALSO

ex (1), edit (1), termcap(5), environ(5), termlist(7).
"Introduction to Display Editing with Vi" in the ZEUS Utilities Manual

LIMITATIONS

Software tabs using ^T work only immediately after the autoindent.

Left and right shifts on intelligent terminals do not make use of insert and delete character operations in the terminal.

The wrapmargin option examines at output columns when blanks are typed. If a long word passes through the margin and onto the next line without a break, then the line is not broken.

Insert/delete within a line is slow if tabs are present on intelligent terminals.

Saving text on deletes in the named buffers is inefficient.

The source command does not work when executed as **:source**; there is no way to use the **:append**, **:change**, and **:insert** commands, since it is not possible to give more than one line of input to a : escape. To use these on a **:global**, enter q to **ex** command mode, execute them, and then reenter the screen editor with **vi** or open.

If the temporary file exceeds 128K characters, **vi** will not attempt to edit the file and exits immediately.

NAME

vls - "visually" list files and directories

SYNOPSIS

vls [**-h**] file ...

DESCRIPTION

Vls is a version of **ls**(1) that lists files and directories in "screen" format. The display is equivalent to an "ls -F" listing where directories are indicated by appending a "/" to the filename in the listing; also, executable files are indicated by appending an "*".

After the display is output, cursor manipulation is permitted by using the following keys:

down arrow
j
CTRL-j move the cursor down the screen

up arrow
k
CTRL-k move the cursor up the screen

f
w move the cursor forward (to the right)

b move the cursor backward (to the left)

If the cursor is placed over an item, typing the following keys performs various list functions:

right arrow
l change to cursored directory and list

left arrow
h change to directory "above" in the dirctory hierarchy and list

L give long format listing

Other miscellaneous commands are:

:n list the next item given in the command line

:q quit

?<CR> display "help" file

<CR> If there is more than one page of display, type carriage return to view the next page.

OPTIONS

-h Turns off "highlighting" of the cursored item; "highlighting" is on by default.

FILES

/usr/lib/screen/vlshelp "help" file

SEE ALSO

ls(1).

LIMITATIONS

Paging back to a previous page is not implemented.

NAME

vnews - "visually" display the news items

SYNOPSIS

vnews [**-h**]

DESCRIPTION

Vnews is a version of **news**(1) that lists the news items in "screen" format.

The "news titles" are displayed in three columns and cursor manipulation is permitted by using the following keys:

down arrow
j
CTRL-j move the cursor down the screen

up arrow
k
CTRL-k move the cursor up the screen

f
w move the cursor forward (to the right)

b move the cursor backward (to the left)

Typing the following keys performs the functions described:

right arrow
l display the news item

:q quit

?<CR> display "help" file

OPTIONS

-h Turns off "highlighting" of the cursored item; "highlighting" is on by default.

FILES

/usr/lib/screen/vnewshelp "help" file

SEE ALSO

news(1).

LIMITATIONS

Paging back to a previous page is not implemented.

NAME

vtzset - set up vtz terminal function keys

SYNOPSIS

vtzset file

DESCRIPTION

Vtzset is a user interface for programming the function keys on the **VTZ** terminal. The input file should contain the "definitions" for any or all of the 34 programmable keys. The format of these definitions is:

key sequence

Key is defined as:

pf1	for the PF1 key
pf2	for the PF2 key
pf3	for the PF3 key
pf4	for the PF4 key
1	for the 1 key
2	for the 2 key
3	for the 3 key
4	for the 4 key
5	for the 5 key
6	for the 6 key
7	for the 7 key
8	for the 8 key
9	for the 9 key
Ø	for the Ø key
-	for the - key
,	for the , key
.	for the . key
enter	for the enter key
u	for the up-arrow key
d	for the down-arrow key
l	for the left-arrow key
r	for the right-arrow key
^u	for control up-arrow
^d	for control down-arrow
^l	for control left-arrow
^r	for control right-arrow
U	for shift up-arrow
D	for shift down-arrow
L	for shift left-arrow
R	for shift right-arrow
^U	for control shift up-arrow
^D	for control shift down-arrow
^L	for control shift left-arrow
^R	for control shift right-arrow

Sequence is defined as a sequence of Ø to 6 characters. For special characters, the following notation should be used:

```
\n    for newline
\r    for carriage return
\     for space (i.e. backslash followed by a space)
\e    for escape
\t    for tab
\\    for backslash
^char      for a control char
```

FILES

/usr/lib/vtz/numpad	number pad "definitions" file
/usr/lib/vtz/vipad	vi(1) commands "definitions" file

LIMITATIONS

Because **vtzset** requires the terminal to be in raw mode, it cannot be used while **remote**(1) on another system.

NAME

wait - await completion of process

SYNOPSIS

wait

DESCRIPTION

All child processes are waited for. It the shell is interactive, then an interrupt can disrupt the wait, when the shell prints names and process numbers of all children known to be outstanding.

The shell waits until all processes started with **&** have completed, and report on abnormal terminations.

Because the **wait**(2) system call must be executed in the parent process, the shell itself executes **wait**, without creating a new process.

SEE ALSO

csh(1), sh(1), wait (2).

LIMITATIONS

Not all the processes of a three- or more-stage pipeline are children of the shell, and thus cannot be waited for.

NAME

wc - word count

SYNOPSIS

wc [**-clw**] [file ...]

DESCRIPTION

Wc counts lines, words, and characters in file, or in the standard input if no name appears. A word is a maximal string of characters delimited by spaces, tabs, or new lines.

OPTIONS

-c Gives only the character count.

-l Gives only the line count.

-w Gives only the word count.

EXAMPLES

```
% wc whatis.1
     29    120     641 whatis.1

% wc -c whatis.1
    641 whatis.1

% wc -l whatis.1
     29 whatis.1

% wc -w whatis.1
    120  whatis.1
```

NAME

what - identify SCCS files

SYNOPSIS

what files

DESCRIPTION

What searches the given files for all occurrences of the pattern that **get**(1) substitutes for %Z% (**@(#)**) and prints out what follows until the first ", >, new-line, \, or null character.

What is intended to be used in conjunction with the SCCS command **get**(1), which automatically inserts identifying information, but it can also be used where the information is inserted manually.

EXAMPLES

For example, if the C program in file **f.c** contains

```
char ident[] = "@(#)identification information";
```

and **f.c** is compiled to yield **f.o** and **a.out**, then the command

```
what f.c f.o a.out
```

will print

```
f.c:
        identification information

f.o:
        identification information

a.out:
        identification information
```

SEE ALSO

get(1), help(1).

DIAGNOSTICS

Use **help**(1) for explanations.

LIMITATIONS

It's possible that an unintended occurrence of the pattern **@(#)** could be found just by chance, but this causes no harm in most cases.

NAME

whatis - describe what a command is

SYNOPSIS

whatis name ...

DESCRIPTION

Whatis looks up a given command and gives the header line from the manual section. You can then run the **man**(1) command to get more information. If the line starts "name(section) ..." you can do "man section name" to get the documentation for it.

EXAMPLE

```
% whatis wc
wc (1)                  - word count
```

FILES

/usr/lib/whatis Data base

SEE ALSO

apropos(1), makewhatis(1).

NAME

whereis - locate source, binary, and or manual for program

SYNOPSIS

whereis [**-bms**] [**-u**] [**-BMS** dir ... **-f**] file ...

DESCRIPTION

Whereis locates source/binary and manuals sections for specified files. The supplied file is first stripped of leading pathname components and any (single) trailing extension of the form ".ext", e.g. ".c". **Whereis** then attempts to locate the desired program in a list of standard places.

Finally, the **-B** **-M** and **-S** flags may be used to change or otherwise limit the places where **whereis** searches.

OPTIONS

-b Search only for binaries.

-f Terminate the last such directory list and signal the start of file names.

-m Search only for manual sections.

-s Search only for sources.

-u Used to search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type. Thus "whereis -m -u *" asks for those files in the current directory which have no documentation.

EXAMPLE

The following finds all the files in /usr/bin which are not documented in /usr/man/man1 with source in /usr/src/cmd:

```
cd /usr/bin
whereis -u -M /usr/man/man1 -S /usr/src/cmd -f *
```

FILES

/usr/src/*
/usr/{doc,man}/*
/lib, /etc, /usr/{lib,bin}
/z/bin

LIMITATIONS

Since the program uses **chdir**(2) to run faster, pathnames given with the **-B** **-M** and **-S** must be full; i.e. they must begin with a "/".

NAME

while - C Shell flow control statement for loop initiation

SYNOPSIS

while (expression**)**

command

end

DESCRIPTION

While the specified expression evaluates nonzero, the commands between the **while** and the matching **end** are executed. **Break** and **continue** can be used to terminate or continue the loop prematurely. The **while** and **end** must appear on separate lines. Prompting occurs here the first time through the loop as for the foreach statement if the input is a terminal.

EXAMPLE

```
#
while ( 1 )     # "1" is always true, therefore
                # this is an endless loop

     echo "This is an endless loop"

                # The string is printed forever
                # or until it is interupted

end         # end of the while loop
```

SEE ALSO

break(1C), breaksw(1C), continue(1C), foreach(1C), if(1C), onintr(1C), switch(1C),
The C Shell in the ZEUS Utilities Manual

NAME

who - print the login names of those currently on the system

SYNOPSIS

who [-g] [who-file] [**am I**]

DESCRIPTION

Who, without an argument, lists the login name, terminal name, and login time for each current ZEUS user.

Without an argument, **who** examines the /etc/utmp file to obtain its information. If a file is given, that file is examined. Typically the given file is /usr/adm/wtmp, which contains a record of all the logins since it was created. Then **who** lists logins, logouts, and crashes since the creation of the wtmp file. Each login is listed with user name, terminal name (with /dev/ suppressed), and date and time. When an argument is given, logouts produce a similar line without a user name. Reboots produce a line with x in the place of the device name, and a fossil time indicative of when the system went down.

With two arguments, as in "who am I" and "who are you", **who** tells who you are logged in as.

OPTIONS

-g Give the group to which you belong is printed.

***** Give the same information as who plus the logout time for terminals not logged in.

EXAMPLE

```
% who
pete     ttyb    Sep 15 17:10
paul     ttyf    Oct  8 09:04
henry    ttym    Oct  8 14:37
craig    ttyo    Oct  8 15:28
betta    ttyt    Oct  8 09:21
harold   ttyv    Aug 19 14:52
carolh   tty05   Oct  8 13:06
naushik  tty12   Oct  6 16:33
```

FILES

/etc/utmp

SEE ALSO

whois(1), whodo(1), users(1), getuid(2), utmp(5).

NAME

whoami - print effective current user id

SYNOPSIS

whoami

DESCRIPTION

Whoami prints who you are. It works even if the su command has been issued to switch to another user name, while "who am I" does not work since it uses /etc/utmp.

EXAMPLES

```
% whoami
carolh

% who am i
carolh   tty05   Oct  8 15:40
```

FILES

/etc/passwd

SEE ALSO

who(1), whodo(1), whois(1), users(1), ps(1).

DIAGNOSTICS

If the message "/etc/passwd file is corrupt" appears, one of the following may have occurred after you logged in:

1) /etc/passwd file was removed
2) your user id was removed from the /etc/passwd file

NAME

whodo - print names and process status for current users

SYNOPSIS

whodo

DESCRIPTION

Whodo produces merged, reformatted, and dated output from the **who**(1) and **ps**(1) commands.

EXAMPLES

```
% whodo
Tue Nov  2 14:09:38 PST 1982
0   karen     12:23
  0       59  0:05 -csh
  0      422  1:34 vi +1100 wdc.03
  10      49  0:00 - 2
  11      50  0:00 - 2
  12     431  0:00 - 2
  13      52  0:00 - 3
  14      53  0:00 - 2
2   deck      11:49
  2       42  0:19 -csh
  2      195  0:55 vi csh.03
  4       43  0:00 - 2
8   carol     12:28
  8       47  0:08 -csh
  8      490  0:00 script
  8      491  0:01 csh -i
  8      492  0:00 script
  8      493  0:01 /bin/sh /bin/whodo
  8      494  0:05 ps -a
9   george    12:24
  9       48  0:06 -csh
  9      231  0:01 vi o.1.9t
  co      36  0:06 /etc/update
  co      38  0:06 /etc/cron
  co     479  0:00 - 2
```

SEE ALSO

ps(1), who(1), whois(1), whoami(1), users(1).

NAME

whois - access the user information database

SYNOPSIS

whois [name...] [**-a** name] [**-m**]

DESCRIPTION

Whois accesses a database containing information such as login name, actual name of the user, office phone and other similiar data. The database information is accessed by entering the login name or the actual user's name. It is also possible to modify your entry in the database file or to add your own entry if one does not already exist.

The commands have the following meanings:

whois name ... prints only those users whose login names have been specified from the database file.

whois -a name prints the user whose actual name is 'name'.

whois -m adds new user to the database. Otherwise one can modify his/her information in the database to any field except the Login name field. See the examples section.

When the system is trying to match a name, the fields are converted to lowercase and compared.

When a name is specified the pattern matching syntax of the shell is supported. This means *, ? and [] work. These symbols must be enclosed in quotes to avoid interpretation by the shell. See **csh**(1) or **sh**(1).

EXAMPLES

To get the information about the login name lindy:

```
% whois lindy

Login name        lindy
Actual user:      John Lindquist
Office phone:     4394
Home phone:
Group:
Misc:
```

The following example displays the information about the actual user, John Lindquist. Because the name contains a blank it must be enclosed in quotes.

```
% whois -a "John Lindquist"
```

```
Login name          lindy
    Actual user     John Lindquist
    Office phone    4394
    Home phone
    Group
    Misc
```

This example is the same as the previous, but using pattern matching.

```
% whois -a "*lind*" Login name          lindy
    Actual user     John Lindquist
    Office phone    4394
    Home phone
    Group
    Misc
```

To modify the database enter:

```
% whois -m
```

No name is entered in this case. An entry can be modified only by the person who owns it.

```
Login name          lindy
    Actual user     John Lindquist
    Office phone    4421
    Home phone
    Group
    Misc

OK? no
Field? office

Office phone        4394

Login name          lindy
    Actual user     John Lindquist
    Office phone    4394
    Home phone
    Group
    Misc
OK? y
```

The above example describes how to modify a record in the database file. Note that when supplying the field name any un-ambiguous abbreviation of the field name can be used. For example, for office, an "o" could have been used.

DIAGNOSTICS

Following is a list of the error messages and their meanings.

Whois database is in use; try again later

Someone is modifying the database. In order to do this they must have exclusive use of it during the operation.

You cannot modify the login name field

An attempt was made to modify the login name field. All fields but login can be modified; login cannot

....: No such user

The user looked for does not exist in the database file.

..: unknown flag

A flag has been specified which is not known to the system.

.... contains separator (:)

The information placed into a field contains the separator character used to separate characters in the database. This can't be done.

No action taken

Nothing has happened.

Usage: whois ......

Unknown options were specified.

FILES

/etc/whois	database file
/etc/owhois	old copy of the database file

SEE ALSO

who(1), whodo(1), whoami(1), whois(5).
The C Shell in the ZEUS Utilities Manual for meta-character (pattern matching) syntax.

DIAGNOSTICS

Self-explanatory or explained above.

NAME

write - write to another user

SYNOPSIS

write user [ttyname]

DESCRIPTION

Write copies lines from a terminal to that of another user. When first called, it sends the message

Message from name ttyname...

The recipient of the message writes back at this point. Communication continues until an end of file is read from the terminal or an interrupt is sent. At that point **write** writes EOT on the other terminal and exits.

To write to a user who is logged in more than once, the ttyname argument is used to indicate the appropriate terminal name.

Permission to write can be denied or granted by the **mesg** command. At log on, writing is allowed. Certain commands, in particular **nroff** and **pr**(1), disallow messages.

If the character ! is found at the beginning of a line, **write** calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using **write**. On the first write to another user, wait for a response before starting to send. Each party should end each message with a distinctive signal such as o for over. Oo for over and out is suggested when conversation is to be terminated.

write is better suited to a single message being sent. For a lengthy, two-way conversation, use talk.

EXAMPLE

```
                Message from bradf tty6...
This is a test of the write command.
o
EOF
```

FILES

/etc/utmp	to find user
/bin/sh	to execute !

SEE ALSO

mail(1), mesg(1), talk(1), who(1).

NAME

xargs - construct argument list(s) and execute command

SYNOPSIS

xargs [options] [command [initial-arguments]]

DESCRIPTION

Xargs combines the fixed initial-arguments with arguments read from standard input to execute the specified command one or more times. The number of arguments read for each command invocation and the manner in which they are combined are determined by the options specified.

Command (can be a shell file) is searched for, using one's **$PATH.** If command is omitted, /bin/echo is used.

Arguments read from standard input are defined as contiguous strings of characters delimited by one or more blanks, tabs, or new-lines; empty lines are always discarded. Blanks and tabs can be embedded in an argument if escaped or quoted. Characters enclosed in quotes (single or double) are taken literally and the delimiting quotes are removed. Outside of quoted strings, a backslash (\) will escape the next character.

Each argument list is constructed starting with the initial-arguments, followed by some number of arguments read from standard input (Exception: see **-i** flag). Options **-i**, **-l**, and **-n** determine how arguments are selected for each command invocation. When none of these options are coded, the initial-arguments are followed by arguments read continuously from standard input until an internal buffer is full. Then, command is executed with the accumulated args. This process is repeated until there are no more args. The last flag has precedence.

OPTIONS

-eeofstr — Eofstr is taken as the logical end-of-file string. Underbar (_) is assumed for the logical **EOF** string if **-e** is not coded. **-e** with no eofstr coded turns off the logical **EOF** string capability (underbar is taken literally). **Xargs** reads standard input until either end-of-file or the logical **EOF** string is encountered.

-ireplstr — Insert mode: command is executed for each line from standard input. The entire line is taken as a single arg and inserted in initial-arguments for each occurrence of replstr. A maximum of 5 arguments in initial-arguments can each contain one or

more instances of replstr. Blanks and tabs at the beginning of each line are ignored. Constructed arguments can not be larger than 255 characters, and option **-x** is also forced. **{}** is assumed for replstr if not specified.

-lnumber — Command is executed for each non-empty number lines of arguments from standard input. The last invocation of command is with fewer lines of arguments if fewer than number remain. A line is considered to end with the first new-line unless the last character of the line is a blank or a tab; a trailing blank or tab signals continuation through the next non-empty line. If number is omitted, 1 is assumed. Option **-x** is forced.

-nnumber — Execute command using as many standard input arguments as possible, up to number arguments maximum. Fewer arguments will be used if their total size is greater than size characters, and for the last invocation if fewer than number arguments remain. If option **-x** is also coded, each number argument must fit in the size limitation, else **xargs** terminates execution.

-p — Prompt mode: The user is asked whether to execute command for each invocation. Trace mode (**-t**) is turned on to print the command instance to be executed, followed by a **?...** prompt. A reply of **y** (optionally followed by anything) will execute the command; anything else (including a carriage return) skips that particular invocation of command.

-ssize — The maximum total size of each argument list is set to size characters; size must be a positive integer less than or equal to 470. If **-s** is not coded, 470 is taken as the default. The character count for size includes one extra character for each argument and the count of characters in the command name.

-t — Trace mode: The command and each constructed argument list are echoed to file descriptor 2 just prior to their execution.

-x — Causes **xargs** to terminate if any argument

list would be greater than size characters; **-x** is forced by the options **-i** and **-l**. When neither of the options **-i**, **-l**, or **-n** are coded, the total length of all arguments must be within the size limit.

Xargs will terminate if it either receives a return code of **-1** from, or if it cannot execute, command. When command is a shell program, it should explicitly exit (see **sh**(1)) with an appropriate value to avoid returning with **-1**.

EXAMPLES

To move all files from directory $1 to directory $2, and echo each move command just before doing it:

```
ls $1 | xargs -i -t mv $1/{} $2/{}
```

To combine the output of the parenthesized commands onto one line, which is then echoed to the end of file log:

```
(logname; date; echo $0 $*) | xargs >>log
```

The user is asked which files in the current directory are to be archived. Xargs archives them into arch one at a time, or many at a time.

```
1.  ls | xargs -p -l ar r arch
2.  ls | xargs -p -l | xargs ar r arch
```

To execute **diff**(1) with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs -n2 diff
```

NAME

xq - examine or delete requests from the line printer spooler

SYNOPSIS

xq [**-d** seq] [**-q** que:dev [**-s**]]

DESCRIPTION

Xq is the part of the general queuing system that allows examination and deletion of items in the queue area. With no options, **xq** lists the devices, queues and requests along with accompanying statuses, for all entries in the queue system. The information listed for queues and devices is:

Q:DEV queue name:device name (device name printed only for device entries)

STATUS
current status. Statuses are:

READY - ready for printing

RUNNING - currently printing

OFFLINE - device is offline and cannot be used

DOWN - queue or device is down. No entries will be accepted for down queues, and no files will be printed on down devices

PRIORITY
Current priority of queue. No requests with a priority less than this will be allowed to print.

FORM current form mounted (blank if standard form) (not implemented)

ACTIVE
sequence number of the active entry

PAGES DONE
number of pages that have already been printed

% DONE
percent file done

The information listed for requests is:

SEQN Sequence number assigned to this request (will remain constant until the request is printed or deleted from the queue)

USER User who submitted the request

Q:DEV Queue (and device if specified) to be used in printing.

REQUEST
Name of file to be printed (if not suppressed)

STATUS
Status of request - Statuses are:

READY - ready to be printed when a device becomes free.

DISPCHD - in the process of being sent to a printer

RUNNING - request currently being processed

FORM Form to be printed on (blank if standard form) (not implemented)

BLKS Length of request (in blocks)

Number of copies to be printed

TIME Time request was submitted

TO Destination of request (if supplied)

OPTIONS

Other options can be used to delete entries from the print queue. These options are:

-d [seqlist]
Remove the indicated request(s) from the queue. Seqlist is the list of sequence numbers to be removed. Sequence numbers can be found using the listing form of **xq.**

-q que:dev
This option indicates which queue and device the -s option will effect. It must precede the -s option on the command line.

-s Stop printing. Causes printing on specified device to stop immediately. The -q que:dev option must be given, and the request being processed must be owned by the user making the stop request.

The **-s** and **-d** options cannot be used together.

FILES

/usr/spool/queuer/activeconfig
/usr/spool/queuer/logfile
/usr/spool/queuer/statusdir
/usr/spool/queuer/requestdir
/tmp/queuer

SEE ALSO

pr(1), nq(1), backend(M), dqueuer(M), xq(M).

DIAGNOSTICS

If any part of the spooling system appears to have inconsistancies, **xq**(1) will print an error and log it in the spooler error log file.

NAME

xstr - extract strings from C programs to implement shared strings

SYNOPSIS

xstr [-c] [-] [file]

DESCRIPTION

Xstr maintains a file strings into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, most useful if they are also read-only.

The command

xstr -c name

extracts the strings from the C source in name, replacing string references by expressions of the form (&xstr[number]) for some number. An approporiate declaration of **xstr** is prepended to the file. The resulting C text is placed in the file x.c, to then be compiled. The strings from this file are placed in the strings data base if they are not there already. Repeated strings and strings which are suffices of existing strings do not cause changes to the data base.

After all components of a large program have been compiled a file xs.c declaring the common **xstr** space can be created by a command of the form

xstr

This xs.c file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared) saving space and swap overhead.

Xstr can also be used on a single file. A command

xstr name

creates files x.c and xs.c as before, without using or affecting any strings file in the same directory.

It may be useful to run **xstr** after the C preprocessor if any macro definitions yield strings or if there is conditional code which contains strings which may not, in fact, be needed. **Xstr** reads from its standard input when the argument `-' is given. An appropriate command sequence for running **xstr** after the C preprocessor is:

```
cc -E name.c | xstr -c -
cc -c x.c
mv x.o name.o
```

Xstr does not touch the file strings unless new items are added, thus **make**(1) can avoid remaking xs.o unless truly necessary.

FILES

strings	Data base of strings
x.c	Massaged C source
xs.c	C source for definition of array `xstr'
/tmp/xs*	Temp file when `xstr name' doesn't touch strings

SEE ALSO

mkstr(1).

LIMITATIONS

If a string is a suffix of another string in the data base, but the shorter string is seen first by **xstr** both strings will be placed in the data base, when just placing the longer one there will do.

NAME

yacc - yet another compiler-compiler

SYNOPSIS

yacc [**-dv**] grammar

DESCRIPTION

Yacc converts a context-free grammar into a set of tables for a simple automaton that executes an lr(1) parsing algorithm. The grammar can be ambiguous; specified precedence rules are used to break ambiguities.

The output file, y.tab.c, must be compiled by the C compiler to produce a program yyparse. This program must be loaded with the lexical analyzer program, yylex, as well as main and yyerror, an error handling routine. These routines must be supplied by the user. **Lex**(1) is useful for creating lexical analyzers usable by **yacc**.

OPTIONS

-d the file y.tab.h is generated with the define statements that associate the **yacc**-assigned token codes with the user-declared token names. This allows source files other than y.tab.c to access the token codes.

-v the file y.output is prepared. This file contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

FILES

y.output	
y.tab.c	
y.tab.h	defines for token names
yacc.tmp, yacc.acts	temporary files
/usr/lib/yaccpar	parser prototype for C programs
/lib/liby.a	library with default main and yyerror

SEE ALSO

lex(1).
LR Parsing by A. V. Aho and S. C. Johnson, Computing Surveys, June, 1974.
YACC - Yet Another Compiler Compiler in the ZEUS Languages / Programming Tools Manual

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard output; a more detailed report is found in the y.output file. If some rules are not reached from the start symbol, this is also reported.

LIMITATIONS

Only one **yacc** process can be active in a given directory at a time.

NAME

300, 300s - handle special functions of DASI terminals

SYNOPSIS

300 [-d*c*,*l*,*t*] [-*n*] [+12]
300s [-d*c*,*l*,*t*] [-*n*] [+12]

DESCRIPTION

300 supports special functions and optimizes the use of the DASI 300 (GSI 300 or DTC 300) terminal. **300s** performs the same functions for the DASI 300s (GSI 300s or DTC 300s) terminal.

300s converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. It also draw Greek letters and other special symbols. Use of 12-pitch text is permitted. Printing time is reduced 5 to 70%.

300 can be used to print equations neatly, in the sequence:

neqn *file* **| nroff | 300**

WARNING: If your terminal has a PLOT switch, make sure it is turned ON before **300** is used.

OPTIONS

-d*c*,*l*,*t* Controls delay factors.

The default setting is **-d3,90,30.** DASI 300 terminals sometimes produce peculiar output when faced with very long lines, too many tab characters, or long strings of blankless, non-identical characters.

One null (delay) character is inserted in a line for every set of *t* tabs, and for every contiguous string of *c* non-blank, non-tab characters.

If a line is longer than *l* bytes, 1+(total length)/20 nulls are inserted at the end of that line. Items can be omitted from the end of the list, implying use of the default values. Also, a value of zero for *t* (*c*) results in two null bytes per tab (character). The former may be needed for C programs, the latter for files like **/etc/passwd.**

Because terminal behavior varies according to the specific characters printed and the load on a system, the user may have to experiment with these values to get correct output.

The **-d** option exists only as a last resort for those few cases that do not otherwise print properly. For example, the file **/etc/passwd** may be printed using **-d3,30,5**. The value **-d0,1** is a good one to use for C programs that have many levels of indentation.

Note that the delay control interacts heavily with the prevailing carriage return and line-feed delays. The **stty**(1) modes **nl0 cr2** or **nl0 cr3** are recommended for most uses.

-n Controls the size of half-line spacing.

A half-line, by default, is equal to 4 vertical plot increments. Because each increment equals 1/48 of an inch, a 10-pitch line-feed requires 8 increments, while a 12-pitch line-feed needs only 6.

The first digit of n overrides the default value, allowing for individual taste in the appearance of subscripts and superscripts.

For example, **nroff**(1) half-lines could be made to act as quarter-lines by using **-2**. The user can also obtain appropriate half-lines for 12-pitch, 8 lines/inch mode by using the option **-3** alone, having set the PITCH switch to 12-pitch.

+12 Permits use of 12-pitch, 6 lines/inch text. DASI 300 terminals normally allow only two combinations: 10-pitch, 6 lines/inch, or 12-pitch, 8 lines/inch. To obtain the 12-pitch, 6 lines per inch combination, the user should turn the PITCH switch to 12, and use the **+12** option.

300 can be used with the **nroff -s** flag or **.rd** requests, when it is necessary to insert paper manually or change fonts in the middle of a document. Instead of hitting the return key in these cases, you must use the line-feed key to get any response.

In many (but not all) cases, the following sequences are equivalent:

```
nroff -T300 files

and

nroff files | 300
```

```
nroff -T300-12 files

and

nroff files | 300 +12
```

The use of **300** can thus often be avoided unless special delays or options are required; in a few cases, however, the additional movement optimization of **300** may produce better-aligned output.

The **neqn**(**eqn**(1)) names of, and resulting output for, the Greek and special characters supported by **300** are shown in **greek**(7).

SEE ALSO

450(1), eqn(1), mesg(1), stty(1), tabs(1), tbl(1), troff(1), greek(7).

LIMITATIONS

Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from there.

If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor; although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.

NAME

450 - handle special functions of the DASI 450 terminal

SYNOPSIS

450

DESCRIPTION

450 supports special functions and optimizes the use of the DASI 450 terminal, or any terminal that is functionally identical, such as the DIABLO 1620 or XEROX 1700.

450 converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. It also draws Greek letters and other special symbols in the same manner as 300(1). **450** can be used to print equations neatly, in the sequence:

neqn file **| nroff | 450**

WARNING: make sure that the PLOT switch on your terminal is ON before **450** is used. The SPACING switch should be put in the desired position (either 10- or 12-pitch). In either case, vertical spacing is 6 lines/inch, unless dynamically changed to 8 lines per inch by an appropriate escape sequence.

450 can be used with the nroff(1) **-s** flag or **.rd** requests, when it is necessary to insert paper manually or change fonts in the middle of a document. Instead of hitting the return key in these cases, use the line-feed key to get any response.

In many (but not all) cases, the use of **450** can be eliminated in favor of one of the following:

```
nroff -T450 files ...
```

or

```
nroff -T450-12 files ...
```

The use of **450** can often be avoided, unless special delays or options are required; in a few cases, however, the additional movement optimization of **450** may produce better-aligned output.

The **neqn**(**eqn**(1)) names of, and resulting output for, the Greek and special characters supported by **450** are shown in **greek**(7).

SEE ALSO

300(1), eqn(1), mesg(1), stty(1), tabs(1), tbl(1), troff(1), greek(7).

LIMITATIONS

Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from there.

If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor; although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.

NAME

intro, errno - introduction to system calls and error numbers

SYNOPSIS

#include <errno.h>

DESCRIPTION

Section 2 of this manual lists all the entries into the system. Most of these calls have an error return. An error condition is indicated by an otherwise impossible returned value. Almost always this is -1; the individual sections specify the details. An error number is also made available in the external variable errno. Errno is not cleared on successful calls, so it should be tested only after an error has occurred.

There is a table of messages associated with each error, and a routine for printing the message; See **perror**(3).

As well, the external variable deverr may be set with error numbers which relate to I/O devices. If errno is 5 or 6 then deverr should be checked to determine, further the nature of the error. The possible error numbers are not recited with each writeup in section 2, since many errors are possible for most of the calls.
Here is a list of the error numbers, their names as defined in <errno.h>, and the messages available using **perror**.

Ø Error Ø
Unused.

1 EPERM Not owner
Typically this error indicates an attempt to modify a file in some way forbidden except to its owner or super-user. It is also returned for attempts by ordinary users to do things allowed only to the super-user.

2 ENOENT No such file or directory
This error occurs when a file name is specified and the file should exist but doesn't, or when one of the directories in a path name does not exist.

3 ESRCH No such process
The process whose number was given to **signal** and **ptrace** does not exist, or is already dead.

4 EINTR Interrupted system call
An asynchronous signal (such as interrupt or quit), which the user has elected to catch, occurred during a system call. If execution is resumed after processing the signal, it will appear as if the interrupted system

call returned this error condition.

5 EIO I/O error
Some physical I/O error occurred during a **read** or **write.** This error may in some cases occur on a call following the one to which it actually applies.

6 ENXIO No such device or address
I/O on a special file refers to a subdevice that does not exist, or beyond the limits of the device. It may also occur when a cartridge tape is prematurely pulled out of the tape drive or no mag tape is loaded on a drive.

7 E2BIG Arg list too long
An argument list longer than 5120 bytes is presented to **exec.**

8 ENOEXEC Exec format error
A request is made to execute a file which, although it has the appropriate permissions, does not start with a valid magic number, see **a.out**(5).

9 EBADF Bad file number
Either a file descriptor refers to no open file, or a read (resp. write) request is made to a file that is open only for writing (resp. reading).

10 ECHILD No children
Wait and the process has no living or unwaited-for children.

11 EAGAIN No more processes
In a **fork,** the system's process table is full or the user is not allowed to create any more processes.

12 ENOMEM Not enough core
During an **exec** or **break,** a program asks for more core than the system is able to supply. This is not a temporary condition; the maximum core size is a system parameter.

13 EACCES Permission denied
An attempt was made to access a file in a way forbidden by the protection system.

14 EFAULT Bad address
The system encountered a hardware fault in attempting to access the arguments of a system call. This is usually the result of a bad address passed to the operating system as the address of the arguments.

15 ENOTBLK Block device required
A plain file was mentioned where a block device was required, operating system **mount.**

16 EBUSY Mount device busy
An attempt to mount a device that was already mounted or an attempt was made to dismount a device on which there is an active file (open file, current directory, mounted-on file, active text segment).

17 EEXIST File exists
An existing file was mentioned in an inappropriate context, e.g. **link.**

18 EXDEV Cross-device link
A link to a file on another device was attempted.

19 ENODEV No such device
An attempt was made to apply an inappropriate system call to a device; e.g. read a write-only device.

20 ENOTDIR Not a directory
A non-directory was specified where a directory is required, for example in a path name or as an argument to **chdir.**

21 EISDIR Is a directory
An attempt to write on a directory.

22 EINVAL Invalid argument
Some invalid argument: dismounting a non-mounted device; mentioning an unknown signal in **signal;** reading or writing a file for which **seek** has generated a negative pointer; passing a bad argument to **ioctl.** Also set by math functions, see **intro**(3).

23 ENFILE File table overflow
The system's table of open files is full, and temporarily no more **opens** can be accepted.

24 EMFILE Too many open files
Customary configuration limit is 20 per process.

25 ENOTTY Not a typewriter
The file mentioned in **stty** or **gtty** is not a terminal or one of the other devices to which these calls apply.

26 ETXTBSY Text file busy
An attempt to execute a pure-procedure program that is currently open for writing (or reading!). Also an attempt to open for writing a pure-procedure program that is being executed.

27 EFBIG File too large
The size of a file exceeded the maximum (about 1.ØE9 bytes).

28 ENOSPC No space left on device
During a **write** to an ordinary file, there is no free space left on the device.

29 ESPIPE Illegal seek
An **lseek** was issued to a pipe. This error should also be issued for other non-seekable devices.

3Ø EROFS Read-only file system
An attempt to modify a file or directory was made on a device mounted read-only.

31 EMLINK Too many links
An attempt to make more than 32767 links to a file.

32 EPIPE Broken pipe
A write on a pipe for which there is no process to read the data. This condition normally generates a signal; the error is returned if the signal is ignored.

33 EDOM Math argument
The argument of a function in the math package (3M) is out of the domain of the function.

34 ERANGE Result too large
The value of a function in the math package (3M) is unrepresentable within machine precision.

35 EDEADLOCK Deadlock imminent
During a **lkdata** system call, either granting of this blocking lock will cause a deadlock condition in which case the process must release some lock or the system configurable number of locks is already allocated.

36 ENOSEG No such segment
During a **mkseg** system call, a segment greater than 127 was used. During a **break** system call, a segment was used which not allocated to the program dwas used. Or during a **mkseg** system call, a segment was requested which was already allocated.

37 ENOPROF Profiling failed
When using the **profile** system call on a segmented program, the system call failed because there were more than 1Ø code segments in the program to be profiled.

The following is a list of device error numbers, their names as defined in <errno.h>, and the messages available using

perror.

Ø DGEN General error
An error which does not fall into one of the catagories below. A general, device specific, error.

1 DNUNIT No such unit
The device requested does not exist or is not on line.

2 DIO I/O error
A physical error occured during an I/O operation.

3 DBUSY Device busy
The device has not completed the previous I/O operation before a request to the device was issued for the I/O operation which generated this error.

4 DPROT Write protected
The device was physically write protected. This can occur when the write wring is missing from a mag tape or when a cartridge tape is set in "safe mode".

5 DNMEDIA No media
The device has no storage media. This can occur if no cartridge is installed in the cartridge tape drive or if no magnetic tape is installed in the mag tape drive.

6 DEDATA End of data
This can occur on the cartridge tape drive when a read was attempted on a blank tape.

SEE ALSO
intro(3)

ASSEMBLER
The Z8ØØØ assembly language interface is given for each system call.

Return values appear in register r4 or rr4 if the return value is a long. An erroneous call is always indicated by turning on the carry bit of the program status word. The value -1 is returned in r4 as well.

The external variable errno is set after the system call to indicate the error.

NAME

access - determine accessibility of file

SYNOPSIS

int access(file, mode)
char *name; **int** mode;

DESCRIPTION

Access checks the given file for accessibility according to mode, which is 4 (read), 2 (write) or 1 (execute) or a combination thereof. Specifying mode Ø tests whether the directories leading to the file can be searched and the file exists.

An appropriate error indication is returned if file cannot be found or if any of the desired access modes is not granted. Access to the file is denied if one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

Read, write, or execute (search) permission is requested for a null path name. [ENOENT]

The named file does not exist. [ENOENT]

Search permission is denied on a component of the path prefix. [EACCES]

Write access is requested for a file on a read-only file system. [EROFS]

Write access is requested for a pure procedure (shared text) file that is being executed. [ETXTBSY]

Permission bits of the file mode do not permit the requested access. [EACCES]

File points outside the process's allocated address space. [EFAULT]

This call is useful to set-UID programs, since the user and group IDs (with respect to which permission is checked) are the real UID and GID of the process.

Only access bits are checked. A directory can be announced as writable by **access**, but an attempt to open it for writing fails (although files can be created there); a file can look executable, but **exec**(2) fails unless it is in proper format.

RETURN VALUE

If the requested access is permitted, a value of Ø is returned. Otherwise, a value of -1 is returned and **errno** is set to indicate the error.

SEE ALSO

stat(2).

ASSEMBLER

```
CONSTANT ACCESS     := 33

...         !* filename, mode in rØ, r1 respectively *!
            !* (if segmented: filename, mode in rrØ, r2) *!
clr  r4,r4
sc   #ACCESS
...         !* return value in r4 *!
            !* carry flag set if error *!
```

NAME

acct - turn accounting on or off

SYNOPSIS

int acct(file)
char *file;

DESCRIPTION

The system is prepared to write a record in an accounting file for each process as it terminates. This call, with a pointer to a null-terminated string naming an existing file as argument, turns on accounting; records for each terminating process are appended to file. An argument of Ø (ØL for segmented programs) causes accounting to be turned off. Termination can be caused by one of two things: an **exit** call or a signal; see **exit**(2) and **signal**(2). The file must exist and the effective user ID of the calling process must be super-user to use this call.

It is erroneous to turn on accounting when it is already on.

The accounting file format is given in **acct**(5).

DIAGNOSTICS

Acct will fail if one or more of the following are true:

The effective user ID of the calling process is not super-user. [EPERM]

An attempt is being made to enable accounting when it is already enabled. [EBUSY]

A component of the path prefix is not a directory. [ENOTDIR]

One or more components of the accounting file's path name do not exist. [ENOENT]

A component of the path prefix denies search permission. [EACCES]

The file named by file is not an ordinary file. [EACCES]

Mode permission is denied for the named accounting file. [EACCES]

The named file is a directory. [EISDIR]

The named file resides on a read-only file system. [EROFS]

File points to an illegal address. [EFAULT]

RETURN VALUE

Upon successful completion, a value of Ø is returned. Otherwise, a value of -1 is returned and **errno** is set to indicate the error.

LIMITATIONS

No accounting is produced for programs running when a crash occurs. Nonterminating programs are never accounted for.

SEE ALSO

acct(5), sa(1).

ASSEMBLER

```
CONSTANT ACCT  := 51

...         !* filename in rØ *!
...         !* (if segmented: filename in rrØ) *!
clr  r4
sc   #ACCT
...         !* return value in r4 *!
            !* carry flag set if error *!
```

NAME

alarm - schedule signal after specified time

SYNOPSIS

unsigned alarm (seconds)
unsigned seconds;

DESCRIPTION

Alarm causes signal SIGALRM, (**signal**(2)) to be sent to the invoking process in a number of seconds given by the argument. Unless caught or ignored, the signal terminates the process.

Alarm requests are not stacked; successive calls reset the alarm clock. If the argument is Ø, any alarm request is cancelled. Because the clock has a 1-second resolution, the signal can occur up to one second early. Due to scheduling delays, resumption of execution of when the signal is caught can be delayed an arbitrary amount. The longest specifiable delay time is 65535 seconds.

RETURN VALUE

The return value is the amount of time previously remaining in the alarm clock.

SEE ALSO

pause(2), signal(2), sleep(3).

ASSEMBLER

```
        CONSTANT ALARM := 27

        ...         !* seconds in rØ     *!
        sc    #ALARM
        ...         !* return value in r4    *!
```

NAME

brk, sbrk - change core allocation

SYNOPSIS

```
char *brk (addr)
char *addr;

char *sbrk (incr)
int incr;
```

DESCRIPTION

Brk sets the system's idea of the lowest location not used by the program (called the break) to addr (rounded up to the next multiple of 256 bytes). For example, if the original break is at location 0x12ac and the user program requests 0x400 more bytes via **sbrk**(0x400), **sbrk** returns 0x12ac and the user may now address up to location 0x16ff. At this point, a brk(0x1501) would allow the user to address up to 0x15ff. Locations greater than addr and below the stack pointer are not in the address space and thus cause a memory violation if accessed.

In the alternate function **sbrk, incr** more bytes are added to the program's data space and a pointer to the start of the new area is returned.

When a program begins execution via **exec**, the break is set at the highest location defined by the program and data storage areas. Therefore, only programs with growing data areas need to use these calls.

Brk and **sbrk** will fail without making any change in the allocated space if such a change would result in more space being allocated than is allowed by a system-imposed maximum (see **ulimit**(2)).

RETURN VALUE

Upon successful completion, **brk** returns a value of 0 and **sbrk** returns the old break value. Otherwise, a value of -1 is returned and **errno** is set to indicate the error.

SEE ALSO

sgbrk(2), ssgbrk(2), exec(2), malloc(3), end(3).

DIAGNOSTICS

-1 is returned and errno is set to ENOMEM if fulfilling the request would result in more space being allocated than is allowed.

ASSEMBLER

```
CONSTANT BREAK := 17
```

```
...         !* new break value in rØ *!
clr  r4
sc   #BREAK
...         !* return value in r4 *!
            !* carry flag set if error *!
```

BREAK performs the function of **brk.** The name of the routine differs from that in C.

Sbrk is implemented in terms of **brk.** To use **sbrk,** the program must keep track of the current break value and add the argument of **sbrk** to it before calling **brk.** This current break value is initially equal to the variable _end and after successive calls to **brk** or **sbrk** must be changed to the address (the argument to **brk**) or the current break value plus the increment, respectively.

NAME

chdir - change working directory

SYNOPSIS

int chdir (dirname)
char *dirname;

DESCRIPTION

Dirname is the address of the pathname of a directory, terminated by a null byte. **Chdir** causes this directory to become the current working directory, the starting point for path names not beginning with /.

SEE ALSO

cd(1C), chroot(2).

DIAGNOSTICS

Chdir will fail and the current working directory will be unchanged if one or more of the following are true:

A component of the path name is not a directory. [ENOTDIR]

The named directory does not exist. [ENOENT]

Search permission is denied for any component of the path name. [EACCES]

Path points outside the process's allocated address space. [EFAULT]

RETURN VALUE

Upon successful completion, a value of Ø is returned. Otherwise, a value of -1 is returned and **errno** is set to indicate the error.

ASSEMBLER

```
CONSTANT CHDIR := 12
...         !* non-segmented: dirname in rØ    *!
            !* (if segmented: dirname in rrØ) *!
clr  r4
sc   #CHDIR
...         !* return value in r4 *!
            !* carry flag set if error *!
```

NAME

chown - change owner and group of a file

SYNOPSIS

int chown (path, owner, group)
char *path;
int owner, group;

DESCRIPTION

Path points to a path name naming a file. The owner ID and group ID of the named file are set to the numeric values contained in owner and group respectively.

Only processes with effective user ID equal to the file owner or super-user may change the ownership of a file.

If **chown** is invoked by other than the super-user, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, will be cleared.

Chown will fail and the owner and group of the named file will remain unchanged if one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

The named file does not exist. [ENOENT]

Search permission is denied on a component of the path prefix. [EACCES]

The effective user ID does not match the owner of the file and the effective user ID is not super-user. [EPERM]

The named file resides on a read-only file system. [EROFS]

Path points outside the process's allocated address space. [EFAULT]

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

SEE ALSO

chmod(2).

ASSEMBLER

```
CONSTANT CHOWN := 16
```

```
...         !* non-segmented: path in rØ  *!
            !* non-segmented owner in rl  *!
            !* non-segmented group in r2  *!
            !* (if segmented: path in rrØ)     *!
            !* (if segmented: owner in r2 *!
            !* (if segmented: group in r3 *!
clr   r4
sc    #CHOWN
...         !* return value in r4 *!
            !* carry flag set if error *!
```

NAME

creat - create a new file

SYNOPSIS

int creat (file, mode)
char *file;
int mode;

DESCRIPTION

Creat creates a new file or prepares to rewrite an existing file, given as the address of a null-terminated string. If the file did not exist, the file's owner ID is set to the process's effective user ID, the file's group ID is set to the process's effective group ID, and the low-order 12 bits of the file mode are set to the value of mode modified as follows:

All bits set in the process's file mode creation mask are cleared. See **umask**(2).

The "save text image after execution bit" of the mode is cleared.

See **chmod**(2) for the construction of the mode argument.

If the file did exist, its mode and owner remain unchanged but it is truncated to Ø length.

The file is also opened for writing, and its file descriptor is returned.

The mode given is arbitrary; it need not allow writing. This feature is used by programs that deal with temporary files of fixed names. The creation is done with a mode that forbids writing. If a second instance of the program attempts a **creat**, an error is returned and the program knows that the name is unusable for the moment.

SEE ALSO

write(2), close(2), chmod(2), umask(2).

DIAGNOSTICS

Creat will fail if one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

A component of the path prefix does not exist. [ENOENT]

Search permission is denied on a component of the path prefix. [EACCES]

The path name is null. [ENOENT]

The file does not exist and the directory in which the file is to be created does not permit writing. [EACCES]

The named file resides or would reside on a read-only file system. [EROFS]

The file is a pure procedure (shared text) file that is being executed. [ETXTBSY]

The file exists and write permission is denied. [EACCES]

The named file is an existing directory. [EISDIR]

There are already too many files open. [EMFILE]

File points outside the process's allocated address space. [EFAULT]

RETURN VALUE

Upon successful completion, a non-negative integer, namely the file descriptor, is returned. Otherwise, a value of -1 is returned and I errno is set to indicate the error.

ASSEMBLER

```
CONSTANT CREAT := 8
...         !* filename, mode in rØ, rl respectively *!
            !* (segmented: filename, mode in rrØ, r2) *!
sc   #CREAT
...         !* return value in r4 *!
            !* carry flag set if error *!
```

NAME

dup, dup2 - duplicate an open file descriptor

SYNOPSIS

int dup (fildes)
int fildes;
dup2 (fildes, fildes2)
int fildes, fildes2;

DESCRIPTION

Given a file descriptor returned from an **open**(2), **pipe**(2), or **creat**(2) call, **dup** allocates another file descriptor synonymous with the original. The new file descriptor is returned. It has the following in common with the original:

Same open file (or pipe).

Same file pointer. (i.e., both file descriptors share one file pointer.)

Same access mode (read, write or read/write).

The new file descriptor is set to remain open across **exec**(2) system calls. See **fcntl**(2).

The file descriptor returned is the lowest one available.

In the second form of the call, fildes is a file descriptor referring to an open file, and fildes2 is a non-negative integer less than the maximum value allowed for file descriptors. **Dup2** causes fildes2 to refer to the same file as fildes. If fildes2 already referred to an open file, it is closed first.

SEE ALSO

creat(2), close(2), exec(2), fcntl(2), open(2), pipe(2).

DIAGNOSTICS

Dup will fail if one or more of the following are true:

Fildes is not a valid open file descriptor. [EBADF]

Twenty (2Ø) file descriptors are currently open. [EMFILE]

RETURN VALUE

Upon successful completion a non-negative integer, namely the file descriptor, is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

ASSEMBLER

```
CONSTANT DUP    := 41
```

```
...          !* fildes, fildes2 in r0, r1 respectively    *!
set   r0,#6     !* dup2 calls only                         *!
sc    #DUP
...          !* return value in r4  *!
             !* carry flag set if error *!
```

The **dup2** entry is implemented by in an OR condition 0100 with fildes.

NAME

execl, execv, execle, execve, execlp, execvp - execute a file

SYNOPSIS

```
int execl (file, arg0, argl, ..., argn, 0)
char *file, *arg0, *argl, ..., *argn;

int execv (file, argv)
char *file, *argv[ ];

int execle (file, arg0, argl, ..., argn, 0,
char *file, *arg0, *argl, ..., *argn, *envp[

int execve (file, argv, envp);
char *file, *argv[ ], *envp[ ];

int execlp (file, arg0, argl, ..., argn, 0)
char *file, *arg0, *argl, ..., *argn;

int execvp (file, argv)
char *file, *argv[ ];
```

DESCRIPTION

Exec in all its forms overlays the calling process with file, then transfers to the entry point of the core image of the file. There can be no return from a successful exec; the calling core image is lost.

File descriptors open in the calling process remain open in the new process, except for those whose close-on-exec flag is set; see **fcntl**(2). For those file descriptors that remain open, the file pointer is unchanged.

Ignored signals remain ignored across these calls, but signals that are caught (**signal**(2)) are reset to their default values.

Profiling is disabled for the new process; see **profil**(2).

Each user has a real user ID and group ID and an effective user ID and group ID. The real ID identifies the person using the system; the effective ID determines the access privileges. **Exec** changes the effective user and group ID to the owner of the executed file if the file has the set-user-ID or set-group-ID modes. The real user ID is not affected.

The new process also inherits the following attributes from the calling process:

NAME

chdir - change working directory

SYNOPSIS

int chdir (dirname)
char *dirname;

DESCRIPTION

Dirname is the address of the pathname of a directory, terminated by a null byte. **Chdir** causes this directory to become the current working directory, the starting point for path names not beginning with /.

SEE ALSO

cd(1C), chroot(2).

DIAGNOSTICS

Chdir will fail and the current working directory will be unchanged if one or more of the following are true:

A component of the path name is not a directory. [ENOTDIR]

The named directory does not exist. [ENOENT]

Search permission is denied for any component of the path name. [EACCES]

Path points outside the process's allocated address space. [EFAULT]

RETURN VALUE

Upon successful completion, a value of Ø is returned. Otherwise, a value of -1 is returned and **errno** is set to indicate the error.

ASSEMBLER

```
CONSTANT CHDIR := 12
...         !* non-segmented: dirname in rØ    *!
            !* (if segmented: dirname in rrØ) *!
clr  r4
sc   #CHDIR
...         !* return value in r4 *!
            !* carry flag set if error *!
```

NAME

chmod - change mode of file

SYNOPSIS

int chmod (name, mode)
char *name;
int mode;

DESCRIPTION

The file whose name is given as the null-terminated string pointed to by name has its mode changed to mode. Modes are constructed by combining with OR some of the following octal values:

```
04000 set user ID on execution
02000 set group ID on execution
01000 save text image after execution
00400 read by owner
00200 write by owner
00100 execute (search on directory) by owner
00070 read, write, execute (search) by group
00007 read, write, execute (search) by others
```

If an executable file is set up for sharing (**-n** option of **ld**(1)) then mode 1000 prevents the system from abandoning the swap-space image of the program-text portion of the file when its last user terminates. Thus, when the next user of the file executes it, the text need not be read from the file system but can simply be swapped in, saving time. Ability to set this bit is restricted to the super-user since swap space is consumed by the images. It is recommended only for heavily used commands.

Only the owner of a file (or the super-user) can change the mode. Only the super-user can set the 1000 mode.

SEE ALSO

chmod(1), chown(2), mknod(2).

DIAGNOSTIC

Chmod will fail and the file mode will be unchanged if one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

The named file does not exist. [ENOENT]

Search permission is denied on a component of the path prefix. [EACCES]

The effective user ID does not match the owner of the

file and the effective user ID is not super-user. [EPERM]

The named file resides on a read-only file system. [EROFS]

Name points outside the process's allocated address space. [EFAULT]

RETURN VALUE

Upon successful completion, a value of Ø is returned. Otherwise, a value of -1 is returned and **errno** is set to indicate the error.

ASSEMBLER

```
CONSTANT CHMOD := 15
...         !* filename, mode in rØ, r1 respectively *!
            !* (if segmented: filename, mode in rrØ, r2) *!
clr  r4
sc   #CHMOD
...         !* return value in r4 *!
            !* carry flag set if error *!
```

NAME

chown - change owner and group of a file

SYNOPSIS

```
int chown ( path, owner, group )
char *path;
int owner, group;
```

DESCRIPTION

Path points to a path name naming a file. The owner ID and group ID of the named file are set to the numeric values contained in owner and group respectively.

Only processes with effective user ID equal to the file owner or super-user may change the ownership of a file.

If **chown** is invoked by other than the super-user, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, will be cleared.

Chown will fail and the owner and group of the named file will remain unchanged if one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

The named file does not exist. [ENOENT]

Search permission is denied on a component of the path prefix. [EACCES]

The effective user ID does not match the owner of the file and the effective user ID is not super-user. [EPERM]

The named file resides on a read-only file system. [EROFS]

Path points outside the process's allocated address space. [EFAULT]

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

SEE ALSO

chmod(2).

ASSEMBLER

```
CONSTANT CHOWN := 16
```

```
...         !* non-segmented: path in rØ  *!
            !* non-segmented owner in r1  *!
            !* non-segmented group in r2  *!
            !* (if segmented: path in rrØ)     *!
            !* (if segmented: owner in r2 *!
            !* (if segmented: group in r3 *!
clr   r4
sc    #CHOWN
...         !* return value in r4 *!
            !* carry flag set if error *!
```

NAME

chroot - change root directory

SYNOPSIS

int chroot (dirname)
char *dirname;

DESCRIPTION

Dirname is the address of the pathname of a directory, terminated by a null byte. **Chroot** sets the root directory, the starting point for path names beginning with /. The call is restricted to the super-user.

SEE ALSO

cd(1), chdir(2).

DIAGNOSTICS

Chroot will fail and the root directory will remain unchanged if one or more of the following are true:

Any component of the path name is not a directory. [ENOTDIR]

The named directory does not exist. [ENOENT]

The effective user ID is not super-user. [EPERM]

Dirname points outside the process's allocated address space. [EFAULT]

RETURN VALUE

Upon successful completion, a value of Ø is returned. Otherwise, a value of -1 is returned and **errno** is set to indicate the error.

ASSEMBLER

```
CONSTANT CHROOT     := 61
...         !* dirname in rØ *!
            !* (if segmented: dirname in rrØ) *!
clr  r4
sc   #CHROOT
...         !* return value in r4 *!
            !* carry flag set if error *!
```

NAME

close - close a file

SYNOPSIS

int close (fildes)
int fildes;

DESCRIPTION

Given a file descriptor such as returned from an **open**(2), **creat**(2), **dup**(2), or **pipe**(2) call, **close** closes the associated file. A close of all files is automatic on **exit**, but since there is a limit on the number of open files per process, **close** is necessary for programs that deal with many files.

Files are closed upon termination of a process, and certain file descriptors can be closed by **exec**(2) (**ioctl** (2)).

SEE ALSO

creat(2), dup(2), open(2), pipe(2), exec(2), fcntl(2), ioctl(2).

DIAGNOSTICS

Close will fail if fildes is not a valid open file descriptor. [EBADF]

RETURN VALUE

Zero is returned if a file is closed; -1 is returned for an unknown file descriptor and errno is set to indicate the error.

ASSEMBLER

```
CONSTANT CLOSE := 6
...         !* fildes in rØ     *!
clr  r4
sc   #CLOSE
...         !* return value in r4 *!
            !* carry flag set if error *!
```

NAME

creat - create a new file

SYNOPSIS

int creat (file, mode)
char *file;
int mode;

DESCRIPTION

Creat creates a new file or prepares to rewrite an existing file, given as the address of a null-terminated string. If the file did not exist, the file's owner ID is set to the process's effective user ID, the file's group ID is set to the process's effective group ID, and the low-order 12 bits of the file mode are set to the value of mode modified as follows:

All bits set in the process's file mode creation mask are cleared. See **umask**(2).

The "save text image after execution bit" of the mode is cleared.

See **chmod**(2) for the construction of the mode argument.

If the file did exist, its mode and owner remain unchanged but it is truncated to 0 length.

The file is also opened for writing, and its file descriptor is returned.

The mode given is arbitrary; it need not allow writing. This feature is used by programs that deal with temporary files of fixed names. The creation is done with a mode that forbids writing. If a second instance of the program attempts a **creat**, an error is returned and the program knows that the name is unusable for the moment.

SEE ALSO

write(2), close(2), chmod(2), umask(2).

DIAGNOSTICS

Creat will fail if one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

A component of the path prefix does not exist. [ENOENT]

Search permission is denied on a component of the path prefix. [EACCES]

The path name is null. [ENOENT]

The file does not exist and the directory in which the file is to be created does not permit writing. [EACCES]

The named file resides or would reside on a read-only file system. [EROFS]

The file is a pure procedure (shared text) file that is being executed. [ETXTBSY]

The file exists and write permission is denied. [EACCES]

The named file is an existing directory. [EISDIR]

There are already too many files open. [EMFILE]

File points outside the process's allocated address space. [EFAULT]

RETURN VALUE

Upon successful completion, a non-negative integer, namely the file descriptor, is returned. Otherwise, a value of -1 is returned and I errno is set to indicate the error.

ASSEMBLER

```
CONSTANT CREAT := 8
...         !* filename, mode in rØ, rl respectively *!
            !* (segmented: filename, mode in rrØ, r2) *!
sc   #CREAT
...         !* return value in r4 *!
            !* carry flag set if error *!
```

NAME

dup, dup2 - duplicate an open file descriptor

SYNOPSIS

int dup (fildes)
int fildes;
dup2 (fildes, fildes2)
int fildes, fildes2;

DESCRIPTION

Given a file descriptor returned from an **open**(2), **pipe**(2), or **creat**(2) call, **dup** allocates another file descriptor synonymous with the original. The new file descriptor is returned. It has the following in common with the original:

Same open file (or pipe).

Same file pointer. (i.e., both file descriptors share one file pointer.)

Same access mode (read, write or read/write).

The new file descriptor is set to remain open across **exec**(2) system calls. See **fcntl**(2).

The file descriptor returned is the lowest one available.

In the second form of the call, fildes is a file descriptor referring to an open file, and fildes2 is a non-negative integer less than the maximum value allowed for file descriptors. **Dup2** causes fildes2 to refer to the same file as fildes. If fildes2 already referred to an open file, it is closed first.

SEE ALSO

creat(2), close(2), exec(2), fcntl(2), open(2), pipe(2).

DIAGNOSTICS

Dup will fail if one or more of the following are true:

Fildes is not a valid open file descriptor. [EBADF]

Twenty (2Ø) file descriptors are currently open. [EMFILE]

RETURN VALUE

Upon successful completion a non-negative integer, namely the file descriptor, is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

ASSEMBLER

```
CONSTANT DUP    := 41
```

```
...          !* fildes, fildes2 in r0, r1 respectively    *!
set   r0,#6     !* dup2 calls only                         *!
sc    #DUP
...          !* return value in r4  *!
             !* carry flag set if error *!
```

The **dup2** entry is implemented by in an OR condition 0100 with fildes.

NAME

execl, execv, execle, execve, execlp, execvp - execute a file

SYNOPSIS

```
int execl (file, arg0, arg1, ..., argn, 0)
char *file, *arg0, *arg1, ..., *argn;

int execv (file, argv)
char *file, *argv[ ];

int execle (file, arg0, arg1, ..., argn, 0,
char *file, *arg0, *arg1, ..., *argn, *envp[

int execve (file, argv, envp);
char *file, *argv[ ], *envp[ ];

int execlp (file, arg0, arg1, ..., argn, 0)
char *file, *arg0, *arg1, ..., *argn;

int execvp (file, argv)
char *file, *argv[ ];
```

DESCRIPTION

Exec in all its forms overlays the calling process with file, then transfers to the entry point of the core image of the file. There can be no return from a successful exec; the calling core image is lost.

File descriptors open in the calling process remain open in the new process, except for those whose close-on-exec flag is set; see **fcntl**(2). For those file descriptors that remain open, the file pointer is unchanged.

Ignored signals remain ignored across these calls, but signals that are caught (**signal**(2)) are reset to their default values.

Profiling is disabled for the new process; see **profil**(2).

Each user has a real user ID and group ID and an effective user ID and group ID. The real ID identifies the person using the system; the effective ID determines the access privileges. **Exec** changes the effective user and group ID to the owner of the executed file if the file has the set-user-ID or set-group-ID modes. The real user ID is not affected.

The new process also inherits the following attributes from the calling process:

nice value (see **nice**(2))
process ID
parent process ID
process group ID
tty group ID (see **exit**(2) and **signal**(2))
trace flag (see **ptrace**(2) request Ø)
time left until an alarm clock signal (see **alarm**(2))
current working directory
root directory
file mode creation mask (see **umask**(2))
file size limit (see **ulimit**(2))
utime, **stime**, **cutime**, and **cstime** (see **times**(2))

From C, two interfaces are available. **Execl** is useful when a known file with known arguments is being called; the arguments to **execl** are the character strings constituting the file and the arguments; the first argument is conventionally the same as the file name or its last component. A Ø argument must end the argument list. Remember that a ØL must end the argument list in the case of a segmented program making the call to **execl**.

The **execv** version is useful when the number of arguments is unknown in advance; the arguments to **execv** are the name of the file to be executed and a vector of strings containing the arguments. The last argument string must be followed by a Ø pointer, which must be a ØL in the case of a segmented program making the call to **execv**.

When a C program is executed, it is called as follows:

```
main(argc, argv, envp)
int argc;
char **argv, **envp;
```

where argc is the argument count and argv is an array of character pointers to the arguments themselves.

Argc is at least one and the first member of the array points to a string containing the name of the file.

Argv is directly usable in another **execv** because argv [argc] is Ø.

Execlp and **execlvp** are called with the same arguments as **execl** and **execv** but duplicate the shell's actions in searching for an executable file in a list of directories. The directory list is obtained from the environment.

ARGUMENTS

The file argument is a pointer to the file to be executed. The path prefix for this file is obtained by a search of the

directories passed as the environment line "PATH =" (see **environ**(7)). The environment is supplied by the shell (see **sh**(1)).

The pointers arg0, arg1 ... address null-terminated strings. These strings constitute the argument list available to the new process. Conventionally, arg0 is the name of the file.

Argv is an array of character pointers to null-terminated strings. These strings constitute the argument list available to the new process. By convention, argv must have at least one member, and it must point to a string that is the same as file (or its last component). Argv is terminated by a null pointer.

Envp is a pointer to an array of strings that constitute the environment of the process. Each string consists of a name, an =, and a null-terminated value. The array of pointers is terminated by a null pointer. In the case of a segmented program doing the exec, all nulls will be two words of 0. The shell (**sh**(1)) passes an environment entry for each global shell variable defined when the program is called.

See **environ**(5) for some conventionally used names. The C run-time start-off routine places a copy of **envp** in the global cell environ, which is used by **execv** and **execl** to pass the environment to any subprograms executed by the current program. The **exec** routines use lower-level routines as follows to pass an environment explicitly:

```
execle(file, arg0, arg1, . . . , argn, 0, environ);
execve(file, argv, environ);
```

FILES

/bin/sh shell, invoked if command file found by **execlp** or **execvp**

SEE ALSO

fork(2), environ(5).

DIAGNOSTICS

Exec will fail and return to the calling process if one or more of the following are true:

One or more components of the new process file's path name do not exist. [ENOENT]

A component of the new process file's path prefix is not a directory. [ENOTDIR]

Search permission is denied for a directory listed in the new process file's path prefix. [EACCES]

The new process file is not an ordinary file. [EACCES]

The new process file mode denies execution permission. [EACCES]

The new process file has the appropriate access permission, but has an invalid magic number in its header. [ENOEXEC]

The new process file is a pure procedure (shared text) file that is currently open for writing by some process. [ETXTBSY]

The new process requires more memory than is allowed by the system-imposed maximum MAXMEM. [ENOMEM]

The number of bytes in the new process's argument list is greater than the system-imposed limit of 5120 bytes. [E2BIG]

The new process file is not as long as indicated by the size values in its header. [EFAULT]

Path, argv, or **envp** point to an illegal address. [EFAULT]

RETURN VALUE

If **exec** returns to the calling process an error has occurred; the return value will be -1 and errno will be set to indicate the error.

LIMITATIONS

If **execvp** is called to execute a file that is a shell command file, and if it is impossible to execute the shell, the values of argv[0] and argv[-1] are modified before return.

ASSEMBLER

```
CONSTANT EXECE      := 59

_environ LABEL


!************************************************!
!*  execl (filename, arg0, arg1, ... , 0)       *!
!*  execute a file - linear argument transmission *!
!*  environment passed automatically            *!
!************************************************!

...                 !* filename in r0 *!
...                 !* pointer to top of stack *!
                    !* in r1 *!
...                 !* (arg0 at top *!
```

```
                        !* argl below it, ... *!
...                     !* Ø below all args *!
                        !* that is, pushed first) *!
                        !* (segmented: same as above only use *
                        !* rrØ and rr2.  Zero pushed on stack *
                        !* is a long) *!
ld  r2,_environ         !* envp pointer *!
                        !* (segmented: ldl rr4,_environ) *!
sc  #EXECE
...                     !* return value in r4 *!
                        !* if returns, error *!


!*****************************************************!
!*  execv (filename, argv)                           *!
!*  execute a file - vectored argument transmission*!
!*  environment passed automatically                 *!
!*****************************************************!

...                     !* filename, argv in rØ *!
                        !* rl respectively *!
                        !* (segmented: filename, argv in *!
                        !* rrØ, rr2) *!
ld  r2,_environ         !* envp pointer *!
                        !* (segmented: ldl rr4,_environ) *!
sc  #EXECE
...                     !* return value in r4 *!
                        !* if returns, error *!


!*****************************************************!
!* execle (filename, argØ, argl, ... , Ø, env)      *!
!*  execute a file - linear argument transmission   *!
!*****************************************************!

...                     !* filename in rØ *!
...                     !* pointer to top of stack *!
                        !* in rl *!
...                     !* (argØ at top, *!
                        !* argl below it, ... *!
...                     !* Ø below all args, *!
                        !* pushed first) *!
                        !* (segmented: filename, pointer *!
                        !* in rrØ, rr2.  Push a long zero) *!
ld  r2,rl               !* note: for segmented programs, the *!
ld  r3,#32768           !* appropriate assembly language should
clr r4                  !* be used to leave rr4 pointing to *!
cpir        r4,@r2,r3,z     !* environ instead of r2 *!
sc  #EXECE
...                     !* return value in r4 *!
```

```
                        !* if returns, error *!


        !*************************************************!
        !*  execve (filename, argv, env)                 *!
        !*  execute a file - vectored argument transmission*!
        !*************************************************!

        ...             !* filename, mode, env in *!
                        !* r0,r1,r2, respectively *!
                        !* (segmented: filename, mode, env *!
                        !* in rr0, rr2, rr4) *!
        sc  #EXECE
        ...             !* return value in r4 *!
                        !* if returns, error *!
```

NAME

exit - terminate process

SYNOPSIS

```
exit (status)
int status;

_exit (status)
int status;
```

DESCRIPTION

Exit is the normal means of terminating a process. **Exit** closes all the process's files and notifies the parent process if it is executing a **wait**. The low-order eight bits of **status** are available to the parent process; see **wait**(2).

If the parent process of the calling process is not executing a **wait**, the calling process is transformed into a zombie process. A zombie process is a process that only occupies a slot in the process table, it has no other space allocated either in user or kernel space. The process table slot that it occupies is partially overlaid with time accounting information (see <sys/proc.h>) to be used by **times**(2).

The parent process ID of all of the calling process's existing child processes and zombie processes is set to 1. This means the initialization process (see **intro**(2)) inherits each of these processes.

An accounting record is written on the accounting file if the system's accounting routine is enabled; see **acct**(2).

If the process ID, tty group ID, and process group ID of the calling process are equal, the **SIGHUP** signal is sent to each processes that has a process group ID equal to that of the calling process. This call can never return.

The C function **exit** can cause cleanup actions before the final sys exit. The function **_exit** circumvents all cleanup.

SEE ALSO

signal(2), wait(2). **WARNING** See **WARNING** in **signal**(2).

ASSEMBLER

```
CONSTANT XIT   := 1

...              !* status in rØ *!
sc   #XIT !* no return   *!
```

NAME

fcntl - file control

SYNOPSIS

#include <fcntl.h>

int fcntl (fildes, cmd, arg)
int fildes, cmd, arg;

DESCRIPTION

Fcntl provides for control over open files. Fildes is an open file descriptor obtained from a **creat**(2), **open**(2), **dup**(2), **fcntl**, or **pipe**(2) system call.

The cmds available are:

F_DUPFD
Return a new file descriptor as follows:

Lowest numbered available file descriptor greater than or equal to arg.

Same open file (or pipe) as the original file.

Same file pointer as the original file (i.e., both file descriptors share one file pointer).

Same access mode (read, write or read/write).

Same file status flags (i.e., both file descriptors share the same file status flags).

The close-on-exec flag associated with the new file descriptor is set to remain open across **exec**(2) system calls.

F_GETFD
Get the close-on-exec flag associated with the file descriptor fildes. If the low-order bit is **0** the file will remain open across **exec**, otherwise the file will be closed upon execution of **exec**.

F_SETFD
Set the close-on-exec flag associated with fildes to the low-order bit of arg (**0** or **1** as above).

F_GETFL
Get file status flags.

F_SETFL
Set file status flags to arg. Only certain flags can be set; see **fcntl**(7).

Fcntl will fail if one or more of the following are true:

Fildes is not a valid open file descriptor. [EBADF]

Cmd is F_DUPFD and 20 file descriptors are currently open. [EMFILE]

Cmd is F_DUPFD and arg is negative or greater than 20. [EINVAL]

RETURN VALUE

Upon successful completion, the value returned depends on cmd as follows:

F_DUPFD
: A new file descriptor.

F_GETFD
: Value of flag (only the low-order bit is defined).

F_SETFD
: Value other than -1.

F_GETFL
: Value of file flags.

F_SETFL
: Value other than -1.

Otherwise, a value of -1 is returned and **errno** is set to indicate the error.

SEE ALSO

close(2), exec(2), open(2), fcntl(7).

NAME

fork - spawn new process

SYNOPSIS

int fork( )

DESCRIPTION

Fork is the only way new processes are created. The new process's image is a copy of the caller of **fork**. The only differences are:

The child process has a unique process ID.

The child process has a different parent process ID (i.e., the process ID of the parent process).

The child process's **utime**(2) **stime**(2) **cutime**, and **cstime** are set to **Ø**; see **times**(2).

Fork returns a value of **Ø** to the child process.

Fork returns the process ID of the child process to the parent process. This process ID is used by **wait**(2).

Files open before the **fork** are shared, and have a common read-write pointer. This is the way that standard input and output files are passed and also how pipes are set up.

Only the super-user can take the last process-table slot.

DIAGNOSTICS

Fork will fail and no child process will be created if one or more of the following are true:

The system-imposed limit on the total number of processes under execution would be exceeded. [EAGAIN]

The system-imposed limit on the total number of processes under execution by a single user would be exceeded. [EAGAIN]

RETURN VALUE

Upon successful completion, **fork** returns a value of Ø to the child process and returns the process ID of the child process to the parent process. Otherwise, a value of -1 is returned to the parent process, no child process is created, and **errno** is set to indicate the error.

SEE ALSO

exec(2), wait(2).

ASSEMBLER

```
CONSTANT FORK  := 2

sc   #FORK
<instruc> !* new process returns here, parent UserID in
             r4... this instruction must be 1 word, and
             is usually a "jr" instruction *!
<instruc> !* old process returns here, child process ID
             in r4, carry flag set if error *!
```

NAME

getpid, - get process IDs
getpgrp - get group process IDs
getppid - get parent process IDs

SYNOPSIS

int getpid( )
int getpgrp( )
int getppid( )

DESCRIPTION

Getpid returns the process ID of the calling process.

Getpgrp returns the process group ID of the calling process.

Getppid returns the parent process ID of the calling process.

SEE ALSO

exec(2), fork(2), intro(2), setpgrp(2), signal(2), mktemp(3).

ASSEMBLER

```
CONSTANT GETPID := 2Ø

...
sc   #GETPID
...  !* pid returned in r4, carry flag set if error *!
```

NAME

getuid - get user identity
getgid - get group identity
geteuid - get effective user identity
getegid - get effective group identity

SYNOPSIS

int getuid( )

int geteuid( )

int getgid( )

int getegid( )

DESCRIPTION

Getuid returns the real user ID of the current process, **geteuid** the effective user ID. The real user ID identifies the person who is logged in, and the effective user ID determines access permission.

Getgid returns the real group ID, **getegid** the effective group ID.

SEE ALSO

setuid(2).

ASSEMBLER

```
CONSTANT GETUID := 24
...
sc   #GETUID
...  !* real user ID in r4, effective user ID in r5 *!


CONSTANT GETGID := 47
...
sc   #GETGID
...  !* real group ID in r4, effective group ID in r5 *!
```

NAME

ioctl - input / output control device

SYNOPSIS

#include <sys/ioctl.h>

ioctl (fildes, request, arg)

DESCRIPTION

Ioctl performs a variety of functions on character special files (devices). The writeups of various devices in Section 4 discuss how **ioctl** applies to them.

Ioctl will fail if one or more of the following are true:

Fildes is not a valid open file descriptor. [EBADF]

Fildes is not associated with a character special device. [ENOTTY]

Request or arg is not valid. See **tty**(4). [EINVAL]

RETURN VALUE

If an error has occurred, a value of -1 is returned and **errno** is set to indicate the error.

SEE ALSO

tty(4).

NAME

kill - send signal to a process

SYNOPSIS

kill (pid, sig);
int pid, sig;

DESCRIPTION

Kill sends the signal (sig) to the process or group of processes specified by the the process ID (pid). See **signal**(2) for a list of signals. If sig is Ø (the null signal), error checking is performed but no signal is actually sent. This can be used to check the validity of pid.

The sending and receiving processes must have the same effective user ID (unless the process is sending to itself), otherwise this call is restricted to the super-user.

If the process number is Ø, the signal is sent to all other processes in the sender's process group (**tty**(4)).

If the process number is -1, and the user is not the super-user, the signal is sent to all other processes in the sender's process group excluding process Ø and process 1.

If the process number is -1, and the user is the super-user, the signal is broadcast universally except to processes Ø and 1, the scheduler and initialization processes, (**init**(M)). If pid is negative but not -1, sig will be sent to all processes whose process group ID is equal to the absolute value of pid. Processes can send signals to themselves.

DIAGNOSTICS

Kill will fail and no signal will be sent if one or more of the following are true:

Sig is not a valid signal number. [EINVAL]

No process can be found corresponding to that specified by pid. [ESRCH]

The sending process is not sending to itself, its effective user ID is not super-user, and its effective user ID does not match the real user ID of the receiving process. [EPERM]

RETURN VALUE

Upon successful completion, a value of Ø is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

SEE ALSO

kill(1), getpid(2), setpgrp(2), signal(2).

ASSEMBLER

```
CONSTANT KILL  := 37

...       !* process ID in rØ, signal in r1 *!
clr  r4
sc   #KILL
...       !* return value in r4 *
          !* carry flag set if error *!
```

NAME

link - link to a file

SYNOPSIS

int link (file1, file2)
char *file1, *file2;

DESCRIPTION

A link to file1 is created; the link has the name file1. Either name can be an arbitrary path name. Zero is returned when a link is made.

SEE ALSO

ln(1), unlink(2).

DIAGNOSTICS

Link will fail and no link will be created if one or more of the following are true:

A component of either path prefix is not a directory. [ENOTDIR]

A component of either path prefix does not exist. [ENOENT]

A component of either path prefix denies search permission. [EACCES]

The file named by file1 does not exist. [ENOENT]

The link named by file2 exists. [EEXIST]

The file named by file1 is a directory and the effective user ID is not super-user. [EPERM]

The link named by file2 and the file named by file1 are on different logical devices (file systems). [EXDEV]

file2 points to a null path name. [ENOENT]

The requested link requires writing in a directory with a mode that denies write permission. [EACCES]

The requested link requires writing in a directory on a read-only file system. [EROFS]

Path points outside the process's allocated address space. [EFAULT]

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and **errno** is set to

indicate the error.

SEE ALSO

link(1M), unlink(2).

ASSEMBLER

```
        CONSTANT LINK  := 9

        ...         !* file1, file2 in r0, r1 respectively  *!
                    !* (segmented: file1, file2 in rr0, rr2) *!
        clr  r4
        sc   #LINK
        ...         !* return value in r4 *!
                    !* carry flag set if error *!
```

NAME

lkdata, unlk - lock and unlock data against concurrent access

SYNOPSIS

```
#include <sys/lockblk.h>

long lkdata (fildes, flag, lkblk);
int fildes, flag;
struct lockblk *lkblk;

long unlk (fildes, flag, lkblk);
int fildes, flag;
struct lockblk *lkblk;
```

DESCRIPTION

Lkdata locks regions of files against access by other processes. The locking process must have write permission on the file or have the file open for writing. A process that attempts an access forbidden by another process's lock sleeps until the other process terminates or removes the lock.

Fildes is an open file descriptor that identifies the file to be locked (**open**(2)).

Flag is the bitwise or of the following constants.

Constant	Value	Function
LKRONLY	Ø	Read-only lock
LKEXCLUSIVE	1	Exclusive lock
LKUSP	Ø	Lock region begins at read/write pointer
LKEXP	2	Lock region beginning specified by lkblk.
LKBLOCKING	Ø	Block if region not available
LKNOBLOCK	4	Return error if regions not available

A lock is either exclusive or read-only; an exclusive lock forbids other processes all access to the locked region; a read-only lock forbids all access except reading.

If flag specifies no blocking and part of the specified region is already locked by another process, **lkdata** returns with an EACCES error (**intro**(2)). If flag specifies blocking and part of the specified region is already locked by another process, **lkdata** does not return until the whole region is available. If a blocking lock would cause deadlock (two process waiting indefinitely for each other), **lkdata** returns with an EDEADLOCK error.

If flag specifies LKUSP, the locked region begins relative to the read/write pointer, using the offset in the lkblk structure. (**lseek**(2)). If instead flags specifies LKEXP, the locked region is specified solely by **lkblk.**

The third argument points to a structure that specifies the length and origin of the lock region.

- lklen specifies the length of the locked region. If lklen is set to Ø, the entire file past the starting point of the lock, including the entire region past the end of file is locked: other processes are not even permitted to append to the file.

- lkoff and lkwhnce specify the beginning of the locked region. If lkwhnce is Ø, the locked region begins at lkoff bytes from the beginning of the file. If lkwhnce is 1, the locked region begins at lkoff bytes from the read/write pointer. If lkwhnce is 2, the locked region begins at lkoff bytes from the end of the file. A negative lkoff is permitted as long as the specified region does not begin before the beginning of the file. An illegal negative offset returns EINVAL.

It is permitted to lock a region past the end of the file; if this is done, only the locking process can extend the file into that region. In particular, if the first byte past the end of the file is locked, no other process can append to the file. Thus appends to the file can be locked out by locking the entire file (setting len to Ø) or by locking the first byte past the end of the file.

If the same process locks two overlapping regions the two regions are merged into a single region. If one of the regions is read-only and the other is exclusive, the overlapping region is exclusive.

Unlk unlocks regions of a file that may have been locked by **lkdata. Unlk** has no effect on bytes not previously locked and restores normal access to bytes previously locked. If part of the specified region is locked by another process, **unlk** will successfully unlock its processes' own locks without affecting locks held by other processes. It is not an error to unlock a region that includes bytes that are not locked or to unlock just part of a single locked region.

A file is also unlocked upon the last close(2) of that file by the associated process.

The arguments to **unlk** have the same meaning as those to **lkdata,** although in flag LKRONLY and LKEXCLUSIVE are ignored.

Here are the declarations in /usr/include/sys/lockblk.h:

```
struct lockblk {
    long    lklen;    /*length of region to lock
    long    lkoff;    /*offset of 1st byte, only w/
                      /*  flag=2,3,6,7
    int     lkwhnce;/*whence flag ala lseek, only w/
                      /*  flag=2,3,6,7
} ;


#define  LKRONLY     Ø  /* Request is for a read-only lock
#define  LKEXCLUSIVE 1  /* Request is for an exclusive loc
#define  LKUSP       Ø  /* Use current seek pointer
#define  LKEXP       2  /* Use explicit offset
#define  LKBLOCKING  Ø  /* Block if request cannot be
                        /*  immediately honored
#define  LKNOBLOCK   4  /* Do not block, but return an err
                        /*  if the request
                        /*  cannot immediately be honored

static long
lkdata(fd,flag,lk)
int fd;
int flag;
struct lockblk *lk;
{
    int stat;
    long temp;

    switch( lk->lkwhnce )
    {

    case Ø:
      return(lk->lkoff >= Ø1 ? lk->lkoff : -11 );
      break;

    case 1:
      return(( temp =lk->lkoff+tell(fd) ) >= Ø1 ? temp : -
      break;

    case 2:
      temp = tell(fd);
      lseek(fd,temp,Ø1);
      return(temp);
      break;

    default:
      return(-11);
    }
}
```

```
static long
unlk(fd,flag,lk)
int fd;
int flag;
struct lockblk *lk;
{
    return(lkdata(fd,flag,lk));
}
```

SEE ALSO

unlk(2), lseek(2).

DIAGNOSTICS

If successful, the call returns the offset of the beginning of the locked region. If unsuccessful, the call returns -1.

An EDEADLOCK error (**intro**(2)) is overcome by removing the lock that is causing the deadlock. One sure way to overcome EDEADLOCK is for the process receiving the error to remove its locks one by one until the **lkdata** call no longer returns EDEADLOCK.

LIMITATIONS

The lockblk structure could have been a union, but it complicates the implementation for only marginal advantage.

The access control checks for this call are somewhat different than those for other calls. In particular, if the file is open only for reading, **lkdata** checks whether the process currently has write permission to the file. This, can lead to inconsistency in the application of the access control rules. For example, consider a setuid program. While operating under the effective uid, it can lock a file opened only for reading but writable by the effective uid. If it reverts to its real uid, the file may no longer be writable and subsequent locking calls fail. There is no inconsistency if the file is opened for reading and writing. A similar inconsistency exists if the access permissions on the file are changed after a process opens it for reading only.

ASSEMBLER

```
CONSTANT LKDATA := 49
...     !*  fdes, flag, lkblk in rØ, r1 and r2    *!
...     !*  ( if segmented: fdes, flag, in rØ, r1,      *!
    !*  lkblk in rr2 )                  *!
sc    #LKDATA
...    !*  return value in rr4, carry flag set if error *!
```

NAME

lock - lock a process in primary memory

SYNOPSIS

lock (flag)

DESCRIPTION

If the flag argument is nonzero, the process executing this call is not swapped except if it is required to grow. If the argument is zero, the process is unlocked. This call can be executed only by the super-user.

LIMITATIONS

Locked processes interfere with the compaction of primary memory and can cause deadlock. This system call is not considered a permanent part of the system.

ASSEMBLER

```
CONSTANT LOCK  := 53

...         !* flag in rØ *!
clr  r4
sc   #LOCK
...         !* return value in r4 *!
            !* carry flag set if error *!
```

NAME

lseek - move read/write pointer

SYNOPSIS

long lseek (fildes, offset, whence)
long offset;
int fildes, whence;

DESCRIPTION

The file descriptor refers to a file open for reading or writing. The read (alternatively write) pointer for the file is set as follows:

- If whence is 0, the pointer is set to offset bytes.
- If whence is 1, the pointer is set to its current location plus offset.
- If whence is 2, the pointer is set to the size of the file plus offset.

The returned value is the resulting pointer location.

Seeking far beyond the end of a file then writing creates a gap that occupies no physical space and reads as zeros.

SEE ALSO

open(2), creat(2), fseek(3), dup(2), fcntl(2).

DIAGNOSTICS

Lseek will fail and the file pointer will remain unchanged if one or more of the following are true:

Fildes is not an open file descriptor. [EBADF]

Fildes is associated with a pipe or fifo. [ESPIPE]

Whence is not 0, 1 or 2. [EINVAL and SIGSYS signal]

The resulting file pointer would be negative. [EINVAL]

Some devices are incapable of seeking. The value of the file pointer associated with such a device is undefined.

RETURN VALUE

Upon successful completion, a non-negative integer indicating the file pointer value is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

LIMITATIONS

Lseek should not be used for terminals or modems. It may

produce undefined results for other character special files.

ASSEMBLER

```
CONSTANT LSEEK    := 19

...    !* fildes in rØ, high byte of offset in rl,     *!
       !* low byte of offset in r2, whence in r3       *!
subl   rr4,rr4
sc     #LSEEK
...    !* return value in rr4, carry flag set if error *!
```

NAME

mdmctl - configure port for modem or terminal line.

SYNOPSIS

mdmctl (request, &ismodem, flag)
int request;
long ismodem;
int flag;

DESCRIPTION

The **mdmctl** system call is used to configure the system hardware and software to communicate with modems or terminals. For the purpose of this system call, modems are defined as Data Communications Equipment which require a seven wire interface to the System 8000. Terminals are defined as Data Terminal Equipment which require a three wire connection to a System8000. These connections are a subset of the RS-232-C specification for such equipment.

Ports which are configured as modems have the following characteristics. When an **open**(2) is attempted on such ports, usually by the **INIT**(M) process, the System8000 will raise its DSR (Data Set Ready) line, and it's CTS (Clear to Send) line. The **open**(2) sleeps until the DTR (Data Terminal Ready) line is asserted. When the DTR line is dropped for more than 1/60 of a second, a hangup signal SIGHUP is sent to all processes in that modem's process group. This signal will terminate these processes unless they catch this signal.

Terminals use only three lines, TxD (Transmited Data), RxD (Received Data), and GND (Signal Ground). The **open**(2) will never sleep in this configuration. Interrupts from the other modem control lines are disabled. Nor will the System8000 assert any other modem control lines. Since interrupts are disabled, system performance may be improved because of the reduced overhead of processing interrupts from noise on the other modem control lines.

Mdmctl is used in the following manner:
Ismodem is the address of a long. It's value is the bitwise or of the lines which should be modems. Thus if lines zero and two are to be modems there would be a one in bit position zero and two (starting from the right, the least significant position). Ismodem for such a case would be 0x5L.
Request is one of TIOCMCG or TIOCMCS which are defined in <sys/tty.h>.
TIOCMCG causes the current setting to be placed into location denoted by ismodem.
TIOCMCS reconfigures the lines to the setting found at the location denoted by ismodem disabling or enabling interrupts from the lines as appropriate.

Flag is unused at this time. It should be set to zero.

SEE ALSO

open(2), tty(4), signal(2), init(M).
System 8000 Hardware Reference Manual.

ASSEMBLER

```
CONSTANT MDMCTL:= 62

...         !* request in r0, segment of addr in r1,     *!
      !* offset of addr in r2, flag in r3          *!
subl  rr4,rr4
sc    #MDMCTL
...         !* return value in rr4, carry flag set if error
```

NAME

mknod - make a directory or a special file

SYNOPSIS

int mknod (file, mode, dev)
char *file;
int mode, dev;

DESCRIPTION

Mknod creates a new file. The mode of the new file (including directory and special file bits) is initialized from mode, which is constructed by summing some of the following values.

```
0xf000 file type: one of the following:
    0x1000 fifo special
    0x2000 character special
    0x4000 directory
    0x6000 block special
    0x8000 or 0x0000 ordinary file

0x800 set user ID on execution
0x400 set group ID on execution
0x200 save text image after execution
0x1ff access permissions; see umask(2)
```

Values of mode other than those above are undefined and should not be used.

The file's owner ID is set to the process's effective user ID. The file's group ID is set to the process's effective group ID.

The low-order 9 bits of mode are modified by the process's file mode creation mask: all bits set in the process's file mode creation mask are cleared. (See **umask**(2)). If mode indicates a block or character special file, dev is a configuration dependent specification of a character or block I/O device. If mode does not indicate a block special or character special device, dev is ignored.

Mknod may be invoked only by the super-user for file types other than FIFO special.

DIAGNOSTICS

Mknod will fail and the new file will not be created if one or more of the following are true:

The process's effective user ID is not super-user. [EPERM]

A component of the path prefix is not a directory.

[ENOTDIR]

A component of the path prefix does not exist. [ENOENT]

The directory in which the file is to be created is located on a read-only file system. [EROFS]

file exists. [EEXIST]

Name points outside the process's allocated address space. [EFAULT]

RETURN VALUE

Upon successful completion a value of Ø is returned. Otherwise, a value of -1 is returned and **errno** is set to indicate the error.

The first block pointer of the i-node is initialized from dev.

LIMITATIONS

For ordinary files and directories, dev is normally zero. In the case of a special file, dev specifies which special file.

Mknod can be invoked only by the super-user.

SEE ALSO

mkdir(1), mknod(1M), chmod(2), exec(2), umask(2), filsys(5), mknod(M).

ASSEMBLER

```
CONSTANT MKNOD    := 14

...       !* name, mode, dev in *!
          !* rØ, r1, r2 respectively *!
          !* (segmented: name, mode, dev in rrØ, r2, r3) *!
clr     r4
sc     #MKNOD
...       !* return value in r4 *!
          !* carry flag set if error *!
```

NAME

mkseg - make a segment

SYNOPSIS

char *mkseg (segno,size);
unsigned segno, size;

DESCRIPTION

The **mkseg** system call creates a new, unnamed, private segment and returns a pointer to it. The segment is created as size bytes, rounded up to the nearest 256 byte boundary. The initial contents are all Ø's. The process issuing the **mkseg** call must be operating in segmented mode.

The segno parameter specifies a preferred segment number. Valid segment numbers range from 4-62 and 66-127. If there is no preferred segment number, a value of Ø means that the system will assign the next free segment.

SEE ALSO

sgbrk(2), exec(2), sgstat(2), sld(1).

DIAGNOSTICS

The **mkseg** call returns a -1L on an error if:

* the segment will not fit into memory [ENOMEM],

* the preferred segment number is illegal [ENOSEG],

* too many segments would exist for the process [ENOSEG],

* or the preferred segment is already allocated. [ENOSEG]

ASSEMBLER

```
CONSTANT MKSEG := 55

...         !* segno, size in      *!
            !* rØ, rl respectively *!
ldl  rr4, #Ø
sc   #MKSEG
...         !* return value in rr4 *!
            !* carry flag set if error *!
```

NAME

mount, umount - mount or remove file system

SYNOPSIS

```
int mount (special, directory, rwflag)
char *special, *directory;
int rwflag;

int umount (special)
char *special;
```

DESCRIPTION

Mount announces to the system that a removable file system has been mounted on the block-structured special file special. From then on, references to directory refer to the root file on the newly mounted file system. Special and directory are pointers to the appropriate path names.

Directory must exist already. Directory must be a directory (unless the root of the mounted file system is not a directory). Its old contents are inaccessible while the file system is mounted.

The rwflag argument determines whether the file system can be written on; if it is Ø writing is allowed, if nonzero, no writing is done. Physically write-protected and magnetic tape file systems must be mounted read-only or errors will occur when access times are updated, whether or not any explicit write is attempted.

Umount announces to the system that the special file is no longer to contain a removable file system. The associated file reverts to its ordinary interpretation.

SEE ALSO

mount(M), umount(2).

DIAGNOSTICS

Mount will fail if one or more of the following are true:

The effective user ID is not super-user. [EPERM]

Any of the named directories do not exist. [ENOENT]

A component of a path prefix is not a directory. [ENOTDIR]

Special is not a block special device. [ENOTBLK]

The device associated with special does not exist. [ENXIO]

Directory is not a directory. [ENOTDIR]

Special or directory points outside the process's allocated address space. [EFAULT]

Directory is currently mounted on, is someone's current working directory or is otherwise busy. [EBUSY]

The device associated with special is currently mounted. [EBUSY]

RETURN VALUE

Upon successful completion a value of Ø is returned. Otherwise, a value of -1 is returned and **errno** is set to indicate the error.

ASSEMBLER

```
CONSTANT MOUNT := 21

...         !* special, mode, rwflag in *!
            !* rØ, r1, r2 respectively *!
            !* (segmented: special, mode, rwflag in *!
            !* rrØ, rr2 and r4) *!
clr  r4
sc   #MOUNT
...         !* return value in r4 *!
            !* carry flag set if error *!


CONSTANT UMOUNT     := 22

...         !* special in rØ    *!
            !* (segmented: special in rrØ) *!
clr  r4
sc   #UMOUNT
...         !* return value in r4 *!
            !* carry flag set if error *!
```

NAME

nice - set program priority

SYNOPSIS

int nice (incr)
int incr;

DESCRIPTION

The scheduling priority of the process is augmented by incr. Positive priorities get less service than normal. Priority 10 is recommended to long-running programs.

Negative increments are ignored except on behalf of the super-user. The priority is limited to the range -20 (most urgent) to 20 (least).

The priority of a process is passed to a child process by **fork**(2). For a privileged process to return to normal priority from an unknown state, **nice** should be called successively with arguments -40 (goes to priority -20 because of truncation), 20 (to get to 0), then 0 (to maintain compatibility with previous versions of this call).

DIAGNOSTICS

Nice will fail and not change the nice value if incr is negative and the effective user ID of the calling process is not super-user. [EPERM]

RETURN VALUE

Upon successful completion, nice returns the new nice value minus 20. Otherwise, a value of -1 is returned and errno is set to indicate the error.

SEE ALSO

nice(1).

ASSEMBLER

```
CONSTANT NICE  := 34

...         !* incr in r0  *!
sc   #NICE
...         !* return value in r4 *!
            !* carry flag set if error *!
```

NAME

open - open for reading or writing

SYNOPSIS

```
#include <fcntl.h>
int open (file, oflag[, mode])
char *file;
int oflag, mode;
```

DESCRIPTION

Open opens file for reading (if the optional mode is Ø), writing (if the optional mode is 1) or for both reading and writing (if the optional mode is 2). File is a path name. **Open** sets the file status flags according to the value of oflag. Oflag values are constructed by or-ing flags from the following list (only one of the first three flags below may be used):

O_RDONLY Open for reading only.

O_WRONLY Open for writing only.

O_RDWR Open for reading and writing.

O_NDELAY This flag may affect subsequent reads and writes. See **read**(2) and **write**(2).

When opening a FIFO with O_RDONLY or O_WRONLY set:

If O_NDELAY is set:

An **open** for reading-only will return without delay. An **open** for writing-only will return an error if no process currently has the file open for reading.

If O_NDELAY is clear:

An **open** for reading-only will block until a process opens the file for writing. An **open** for writing-only will block until a process opens the file for reading.

When opening a file associated with a communication line:

If O_NDELAY is set:

The open will return without waiting for carrier.

If O_NDELAY is clear:

The open will block until carrier is present.

O_APPEND If set, the file pointer will be set to the end of the file prior to each write.

O_CREAT If the file exists, this flag has no effect. Otherwise, the file's owner ID is set to the process's effective user ID, the file's group ID is set to the process's effective group ID, and the low-order 12 bits of the file mode are set to the value of mode modified as follows (see **creat**(2)):

All bits set in the process's file mode creation mask are cleared. See **umask**(2).

The "save text image after execution bit" of the mode is cleared. See **chmod**(2).

Warning: if the mode parameter is not provided in the "implicit creat" case, the mode of the new file is undefined.

O_TRUNC If the file exists, its length is truncated to Ø and the mode and owner are unchanged.

O_EXCL If O_EXCL and O_CREAT are set, **open** will fail if the file exists.

The file is positioned at the beginning (byte Ø). The returned file descriptor must be used for subsequent calls for other input/output functions on the file.

The new file descriptor is set to remain open across **exec** system calls. See **fcntl**(2).

No process may have more than 2Ø file descriptors open simultaneously.

DIAGNOSTICS

The named file is opened unless one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

O_CREAT is not set and the named file does not exist. [ENOENT]

A component of the path prefix denies search permission. [EACCES]

Oflag permission is denied for the named file.

[EACCES]

The named file is a directory and oflag is write or read/write. [EISDIR]

The named file resides on a read-only file system and oflag is write or read/write. [EROFS]

Twenty (2Ø) file descriptors are currently open. [EMFILE]

The named file is a character special or block special file, and the device associated with this special file does not exist. [ENXIO]

The file is a pure procedure (shared text) file that is being executed and oflag is write or read/write. [ETXTBSY]

File points outside the process's allocated address space. [EFAULT]

O_CREAT and O_EXCL are set, and the named file exists. [EEXIST]

O_NDELAY is set, the named file is a FIFO, O_WRONLY is set, and no process has the file open for reading. [ENXIO]

RETURN VALUE

Upon successful completion, a non-negative integer, namely a file descriptor, is returned. Otherwise, a value of -1 is returned and **errno** is set to indicate the error.

SEE ALSO

creat(2), read(2), write(2), dup(2), close(2).

ASSEMBLER

```
CONSTANT OPEN  := 5

...         !* filename, oflag, and mode in rØ, r1,   *!
            !* and r2 respectively                     *!
            !* (segmented: filename, oflag, and mode  *!
            !* in rrØ, r2, r3 respectively)           *!
sc   #OPEN
...         !* return value in r4 *!
            !* carry flag set if error *!
```

NAME

pause - stop until signal

SYNOPSIS

pause( )

DESCRIPTION

Pause never returns normally. It is used to give up control while waiting for a signal from **kill**(2) or **alarm**(2). The signal received must not cause termination or be currently set up to be ignored by the calling process. If the signal is caught by the calling process (see **signal**(2)) and control is passed back from the signal catching function, the calling process resumes execution from the point of suspension; pause returns a value of -1 and **errno** is set to EINTR.

SEE ALSO

kill(1), kill(2), alarm(2), signal(2), setret(3).

ASSEMBLER

```
CONSTANT PAUSE := 29

sc   #PAUSE
```

NAME

pipe - create an interprocess channel

SYNOPSIS

int pipe (fildes)
int fildes[2];

DESCRIPTION

The **pipe** system call creates an Input/Output mechanism called a pipe. The file descriptors returned can be used in read and write operations. When the pipe is written using the descriptor fildes[1], up to 5120 bytes of data are buffered before the writing process is suspended. A read using the descriptor fildes[0] picks up the data. Writes with a count of 5120 bytes or less are treated as a unit; no other process can intersperse data.

It is assumed that after the pipe has been set up, two or more cooperating processes, created by subsequent **fork** calls, pass data through the pipe with **read** and **write** calls.

The shell has a syntax to set up a linear array of processes connected by pipes.

Read calls on an empty pipe (no buffered data) with only one end (all write file descriptors closed) returns an end-of-file.

SEE ALSO

sh(1), csh(1), read(2), write(2), fork(2).

DIAGNOSTICS

The function value zero is returned if the pipe was created; -1 if too many files (more than 20) are already open. [EMFILE]

A signal is generated if a write on a pipe with only one end is attempted.

LIMITATIONS

Should more than 5120 bytes be necessary in any pipe among a loop of processes, deadlock occurs.

ASSEMBLER

```
CONSTANT PIPE := 42

...
sc   #PIPE
...  !* read file descriptor in r4 *!
...  !* write file descriptor in r5 *!
```

NAME

profil - execution time profile

SYNOPSIS

profil (buff, bufsiz, offset, scale)
sprofil (segno, buff, bufsiz, offset, scale)
char *buff;
int segno, bufsiz, offset, scale;

DESCRIPTION

These calls are used to create a buffer of profiling information that can tell the user where a program is spending it's time. The **sprofil**(2) call is used only by segmented programs. The segno is the segment number of the code that is to be profiled. If a segmented program has many code segments, a separate call to sprofil(2) for each segment must be made. Non-segmented programs must use the **profil**(2) call.

Buff points to an area of core whose length (in bytes) is given by bufsiz. For **sprofil** remember that buff is a long (segmented) address. After the call, the user's program counter (pc) is examined each clock tick (60th second); offset is subtracted from it, and the result multiplied by scale. If the resulting number corresponds to a word inside buff, that word is incremented.

The scale is interpreted as an unsigned, fixed-point fraction with binary point at the left: 0177777(8) gives a 1-1 mapping of pc's to words in buff; 077777(8) maps each pair of instruction words together. 02(8) maps all instructions onto the beginning of buff (producing a non-interrupting core clock).

Profiling is turned off by giving a scale of 0 or 1. It is rendered ineffective by giving a bufsiz of 0. Profiling is turned off when an **exec** is executed, but remains on in child and parent both after a **fork**. Profiling may be turned off if an update in buff would cause a memory fault.

RETURN VALUE

0: no error
A -1 is returned by **sprofile if:**

- The segment number is out of the given range, unused, or a data segment [ENDSEG].

- more than 10 code segments are being profiled [ENDPROF]

SEE ALSO

monitor(3), prof(1), mon.out(5).

ASSEMBLER

```
CONSTANT PROFIL := 44

...   !* buf in rØ, bufsiz in r1, offset in r2, scale in r3 *
sc    #PROFIL
...   ! return in r4, carry bit set on error !

CONSTANT SPROFIL := 52

...   !* segno in rØ, buff in r1, r2 *!
      !*   bufsiz in r3, offset in r4, scale in r5 *!
sc    #SPROFIL
...   ! return in r4, carry bit set on error !
```

NAME

ptrace - process trace

SYNOPSIS

#include <signal.h> (if non-segmented parent)
#include <ssignal.h> (if segmented parent)

ptrace (request, pid, addr, data)
int *addr;
int request, pid, addr, data;

sptrace (request, pid, addr, data)
int *addr;

DESCRIPTION

Ptrace allows a parent process to control the execution of a child process, and examine and change its core image. Its primary use is for the implementation of breakpoint debugging.

Sptrace is the version of ptrace that should be used if a segmented child is to be traced. (It does not matter what the parent is.) Similarly, if a non-segmented child is to be traced, use **ptrace.** The only difference between the two calls is that **sptrace** must have a 32-bit "addr" as a parameter whereas **ptrace** must have a 16-bit parameter "addr". The compiler will produce addresses corresponding to the type (segmented or non-segmented) of the parent; it is up to the user to make sure the address length is correct.

There are four arguments whose interpretation depends on a request argument. Generally, pid is the process ID of the traced process, which must be a child (no more distant descendant) of the tracing process. A process being traced behaves normally until it encounters some signal whether internally generated like "privileged instruction" or externally generated like "interrupt." (See **signal**(2) for the list.) Then the traced process enters a stopped state and its parent is notified via **wait**(2). When the child is in the stopped state, its core image can be examined and modified using **ptrace.** If desired, another **ptrace** request can then cause the child either to terminate or to continue, possibly ignoring the signal.

The value of the request argument determines the precise action of the call:

Ø This request is the only one used by the child process; it declares that the process is to be traced by its parent. All the other arguments are ignored. Either the ptrace or sptrace system call can be issued; however, the third argument must be a word if ptrace is

used and a long if sptrace is used. Peculiar results occur if the parent does not expect to trace the child.

1,2 The word in the child process's address space at addr is returned. If information and data (I and D) space are separated, request 1 indicates I space and, 2 indicates D space. Addr must be even. The child must be stopped. The input data is ignored.

3 The word of the system's per-process data area corresponding to addr is returned. Addr must be even and less than the size of the per-process data area. This space contains the registers and other information about the process; its layout corresponds to the user structure in the system.

4,5 The given data is written at the word in the process's address space corresponding to addr, which must be even. No useful value is returned. If I and D spaces are separated, request 4 indicates I space, 5 D space. Attempts to write in procedure fail if another process is executing the same file.

6 The process's system data is written, as it is read with request 3. Only the general registers and certain bits of the processor status word can be written in this way. The program counter is not allowed to be changed.

7 The data argument is taken as a signal number and the child's execution continues at location addr as if it had incurred that signal. Normally the signal number is either Ø to indicate that the signal that caused the stop should be ignored, or that value fetched out of the process's image indicating which signal caused the stop. If addr is (int *)1 then execution continues from where it stopped.

8 The traced process terminates.

9 Execution continues as in request 7; however, only one instruction is executed, then execution stops. The signal number from the stop is SIGTRAP. This is part of the mechanism for implementing breakpoints.

As indicated, these calls (except for request Ø) can be used only when the subject process has stopped. The wait call is used to determine when a process stops; in such a case the "termination" status returned by wait has the value Ø177 to indicate stoppage rather than genuine termination.

To forestall possible fraud, **ptrace** inhibits the set-user-ID facility on subsequent **exec**(2) calls. If a traced process

calls **exec**, it stops before executing the first instruction of the new image showing signal SIGTRAP.

SEE ALSO

wait(2), signal(2), adb(1), szdb(1).

DIAGNOSTICS

The value -1 is returned if request is invalid, pid is not a traceable process, addr is out of bounds, or data specifies an illegal signal number.

LIMITATIONS

The error indication, -1, is a legitimate function value; errno, (**INTRO**(2)), can be used to clarify.

It is not but should be possible to stop a process on occurrence of a system call; in this way a completely controlled environment could be provided.

ASSEMBLER

```
CONSTANT PTRACE := 26

...  !* request in rØ, pid in r1, addr in r2 *!
     !* data in r3 *!
     !* (segmented: parameters in rØ, r1, rr2, r4, respectiv
sc   #PTRACE
...  !* return value in r4 *!
     !* carry bit set on error *!
```

NAME

read - read from file

SYNOPSIS

read (fildes, buffer, nbytes)
char *buffer;
int fildes, nbytes;

DESCRIPTION

A file descriptor is a word returned from a successful **open**(2) **creat**(2) **dup**(2), or **pipe**(2) call. Buffer is the location of nbytes contiguous bytes into which the input is placed. It is not guaranteed that all nbytes bytes are read; for example, if the file refers to a typewriter, at most one line is returned. In any event, the number of characters read is returned.

If the returned value is Ø, then end-of-file has been reached.

When attempting to read from an empty pipe (or FIFO):

If O_NDELAY is set, the read will return a Ø.

If O_NDELAY is clear, the read will block until data is written to the file or the file is no longer open for writing.

When attempting to read a file associated with a tty that has no data currently available:

If O_NDELAY is set, the read will return a Ø.

If O_NDELAY is clear, the read will block until data becomes available.

DIAGNOSTICS

Read will fail if one or more of the following are true:

Fildes is not a valid file descriptor open for reading. [EBADF]

Buffer points outside the allocated address space. [EFAULT]

Many conditions generate an error; for example physical I/O errors, bad buffer address, out of range nbytes, file descriptor not that of an input file.

RETURN VALUE

Upon successful completion a non-negative integer is returned indicating the number of bytes actually read.

Otherwise, a -1 is returned and **errno** is set to indicate the error.

SEE ALSO

open(2), creat(2), dup(2), pipe(2).

ASSEMBLER

```
CONSTANT READ  := 3

...         !* fildes, buffer, nbytes in
            !* rØ, r1, r2 respectively *!
            !* (segmented: rØ, rr2, r4, respectively) *!
sc   #READ
...         !* return value in r4 *!
            !* carry flag set if error *!
```

NAME

setpgrp - set process group ID

SYNOPSIS

int setpgrp ()

DESCRIPTION

Setpgrp sets the process group ID of the calling process to the process ID of the calling process and returns the new process group ID.

The system call **getpid** returns the value of the current process group.

RETURN VALUE

Setpgrp returns the value of the new process group ID.

SEE ALSO

exec(2), fork(2), getpid(2), intro(2), kill(2), signal(2).

NAME

setuid, setgid - set user and group ID

SYNOPSIS

```
int setuid (uid)
int uid;

int setgid (gid)
int gid;
```

DESCRIPTION

The user ID (group ID) of the current process is set to the argument. Both the effective and the real ID are set. These calls are only permitted to the super-user or if the argument is the real ID.

SEE ALSO

getuid(2).

DIAGNOSTICS

Setuid will fail if the real user (group) ID of the calling process is not equal to uid (gid) and its effective user ID is not super-user. [EPERM]

RETURN VALUE

Zero is returned if the user (group) ID is set; -1 is returned otherwise.

ASSEMBLER

```
CONSTANT SETUID := 23

...  ! user ID in rØ  !
sc   #SETUID
...  ! return in r4, carry bit set if error !


CONSTANT SETGID := 46

...  ! group ID in rØ !
sc   #SETGID
...  ! return in r4, carry bit set if error !
```

NAME

sgbrk - change the size of a data segment

SYNOPSIS

```
char *sgbrk (addr) ;
char *addr

char *ssgbrk (segno, incr) ;
unsigned segno, incr
```

DESCRIPTION

These system calls are the segmented versions of the **brk**(2) and **sbrk**(2) calls and are not legal for non-segmented users. The **sgbrk** system call changes the size of a presently known, non-sharable, writable data segment. The addr argument is a segmented address, with the offset equal to the desired new size of the data segment. The new segment size is rounded up to the next 256 byte boundary past the offset given by the caller. If addr is a ØL, then **sgbrk** returns the current high data address of the program. Otherwise, **sgbrk** returns a segmented pointer to the new area on a successfull call, or a -1L if the call failed.

The **ssgbrk** system call changes the size of a presently known, non-sharable, writable data segment by the given increment. The user must provide the segment number; segno. A segmented address is returned which is a pointer to the start of the requested area. If incr is Ø, the returned address points to the current end of the data segment.

If a segment is sharable with other processes, its size may not be changed. Additionally, a segment must be writable to change its size. This restriction is to prevent alteration of shared text setuid/setgid programs.

SEE ALSO

mkseg(2), exec(2), sld(1).

DIAGNOSTICS

Both of these calls return a -1L on an error. Common errors are; the segment size exceeding 64K bytes, the program size exceeding the system's limits, or illegal segment number.

ASSEMBLER

Note that the **sgbrk** call is implemented in terms of the brk(2) call.

```
CONSTANT SGBRK := 17

...         !* new break value in rrØ *!
clr  rr4
sc   #SGBRK
...         !* return value in rr4 *!
```

```
          !* carry flag set if error *!

CONSTANT SSGBRK     := 45

...       !* segment number in rØ    *!
...       !* increment in rl         *!
clr  rr4
sc   #SSGBRK
...       !* return value in rr4 *!
          !* carry flag set if error *!
```

NAME

sgstat - get highest segmented code address

SYNOPSIS

```
sgstat(&buffer)
struct {
     char segno ;
     unsigned size ;
} buffer[10] ;
```

DESCRIPTION

The **sgstat** system call returns information in buffer that describes code segments for the calling process. The calling program must be segmented. The size is in bytes. The buffer table is filled with the first 10 code segment numbers and sizes (in bytes). If there are less than 10 code segments, the table is filled out with values of -1. This call is used in conjunction with profiling by the **monitor**(3) routine.

DIAGNOSTICS

The **sgstat** call should never return an error condition.

ASSEMBLER

```
CONSTANT SGSTAT     := 56

...         !* buffer address in rr0 *!
clr  rr4
sc   #SGSTAT
...         !* return value in rr4 *!
```

NAME

signal - catch or ignore signals

SYNOPSIS

#include <signal.h>(non-segmented)
#include <ssignal.h>(segmented)

(*signal (sig, func))()
(*func)();

DESCRIPTION

A **signal** is generated by some abnormal event, initiated either by user at a terminal (quit, interrupt), by a program error (bus error), or by request of another program (**kill**(2)). Normally all signals cause termination of the receiving process, but a **signal** call allows them either to be ignored or to cause an interrupt to a specified location. Here is the list of signals with names as in the include file.

SIGHUP	1	hangup
SIGINT	2	interrupt
SIGQUIT	3*	quit
SIGILL	4*	privileged instruction (not reset when caught)
SIGTRAP	5*	trace trap (not reset when caught)
SIGIOT	6*	software IOT instruction (hardware IOT trap not possible)
SIGEMT	7*	EMT instruction (not used)
SIGFPE	8*	floating point exception
SIGKILL	9	kill (cannot be caught or ignored)
SIGBUS	10*	bus error (impossible on S8000)
SIGSEGV	11*	segmentation violation
SIGSYS	12*	bad argument to system call
SIGPIPE	13	write on a pipe or link with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination signal
	16	unassigned

The starred signals in the list above cause a core image if not caught or ignored.

Note that there are two include files, one for the non-segmented case and one for the segmented case. The return value as well as the second argument to **signal** is an integer in the non-segmented case and a long in the segmented case.

If func is SIG_DFL, the default action for signal sig is reinstated; this default is termination, sometimes with a core image. If func is SIG_IGN the signal is ignored. Otherwise when the signal occurs func is called with the signal number as argument. A return from the function continues the process at the point it was interrupted. Except as

indicated, a signal is reset to SIG_DFL after being caught. Thus, if it is desired to catch every such signal, the catching routine must issue another **signal** call.

When a caught signal occurs during certain system calls, the call terminates prematurely. This can occur during a **read**(2) or **write**(2) on a slow device (like a teletype; but not a file); and during **pause**(2) or **wait**(2). When such a signal occurs, the saved user status is arranged in such a way that when return from the signal-catching takes place, it appears that the system call returned an error status. The user's program can then, if it wishes, reexecute the call.

The value of **signal** is the previous (or initial) value of func for the particular signal.

After a **fork**(2), the child inherits all signals. **Exec**(2) resets all caught signals to default action.

ARGUMENTS

Sig can be assigned any one of the following except **SIGKILL:**

SIGHUP	Ø1	hangup
SIGINT		Ø2interrupt
SIGQUIT	Ø3*	quit
SIGILL		Ø4*illegal instruction (not reset when caught)"
SIGTRAP	Ø5*	trace trap (not reset when caught)
SIGIOT		Ø6*IOT instruction
SIGEMT	Ø7*	EMT instruction
SIGFPE		Ø8*floating point exception
SIGKILL	Ø9	kill (cannot be caught or ignored)
SIGBUS	1Ø*	bus error
SIGSEGV	11*	segmentation violation
SIGSYS		12*bad argument to system call
SIGPIPE	13	write on a pipe with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination signal
SIGUSR1	16	user defined signal 1
SIGUSR2	17	user defined signal 2
SIGCLD	18	death of a child (see **WARNING** below)
SIGPWR	19	power fail (see **WARNING** below)

See below for the significance of the asterisk in the above list.

Func is assigned one of three values: **SIG_DFL, SIG_IGN,** or a function address. The actions prescribed by these values of are as follows:

SIG_DFL - terminate process upon receipt of a signal
Upon receipt of the signal sig, the receiving process is to be terminated with the following consequences:

All of the receiving process's open file descriptors will be closed.

If the parent process of the receiving process is executing a **wait**, it will be notified of the termination of the receiving process and the terminating signal's number will be made available to the parent process; see **wait**.

If the parent process of the receiving process is not executing a **wait**, the receiving process will be transformed into a zombie process (see **exit**(2) for definition of zombie process).

The parent process ID of each of the receiving process's existing child processes and zombie processes will be set to 1. This means the initialization process (see **intro**(2)) inherits each of these processes.

An accounting record will be written on the accounting file if the system's accounting routine is enabled; see **acct**(2).

If the receiving process's process ID, tty group ID, and process group ID are equal, the signal **SIGHUP** will be sent to all of the processes that have a process group ID equal to the process group ID of the receiving process.

A core image will be made in the current working directory of the receiving process if sig is one for which an asterisk appears in the above list and the following conditions are met:

The effective user ID and the real user ID of the receiving process are equal.

An ordinary file named **core** exists and is writable or can be created. If the file must be created, it will have the following properties:

a mode of Ø666 modified by the file creation mask (see **umask**(2))

a file owner ID that is the same as the effective user ID of the receiving process

a file group ID that is the same as the effective group ID of the receiving process

SIG_IGN - ignore signal
The signal sig is to be ignored.

Note: the signal **SIGKILL** cannot be ignored.

function address - catch signal
Upon receipt of the signal sig, the receiving process is to execute the signal-catching function pointed to by func. The signal number sig will be passed as the only argument to the signal-catching function.

Upon return from the signal-catching function, the receiving process will resume execution at the point it was interrupted and the value of func for the caught signal will be set to **SIG_DFL** unless the signal is **SIGILL, SIGTRAP, SIGCLD,** or **SIGPWR.**

When a signal that is to be caught occurs during a **read**(2), **write**(2) **open**(2) or an **ioctl**(2) system call on a slow device (like a terminal; but not a file), during a **pause** system call, or during a **wait** system call that does not return immediately due to the existence of a previously stopped or zombie process, the signal catching function will be executed and then the interrupted system call will return a -1 to the calling process with errno set to EINTR.

Note: the signal **SIGKILL** cannot be caught.

DIAGNOSTICS

Signal will fail if one or more of the following are true:

Sig is an illegal signal number, including **SIGKILL.** [EINVAL]

Func points to an illegal address. [EFAULT]

RETURN VALUE

The value (int)-1 is returned if the given signal is out of range. A -1L is returned in the segmented case.

LIMITATIONS

If a repeated signal arrives before the last one can be reset, there is no chance to catch it.

The type specification of the routine and its func argument are problematical.

ASSEMBLER

```
        CONSTANT SIGNAL := 48

...     !* signal # in rØ, label in rl *!
        !* (segmented: signal # in rØ, label in rr2) *!
sc      #SIGNAL
...     !* previous label returned in r4 *!
        !* carry bit set on error !
```

If label is Ø, default action is reinstated. If label is odd, the signal is ignored. Any other even label specifies an address in the process where a signal handling rouitne is located.

SEE ALSO

kill(1), kill(2), pause(2), ptrace(2), wait(2).

WARNING

Two other signals that behave differently than the signals described above exist in this release of the system; they are:

SIGCLD	18	death of a child (not reset when caught)"
SIGPWR	19	power fail (not reset when caught)

There is no guarantee that, in future releases of Zilog, these signals will continue to behave as described below; they are included only for compatibility with other versions of UNIX. Their use in new programs is strongly discouraged.

For these signals, func is assigned one of three values: **SIG_DFL**, **SIG_IGN**, or a function address. The actions prescribed by these values of are as follows:

SIG_DFL - ignore signal
The signal is to be ignored.

SIG_IGN - ignore signal
The signal is to be ignored. Also, if sig is **SIGCLD**, the calling process's child processes will

not create zombie processes when they terminate; see **exit**.

function address - catch signal
If the signal is **SIGPWR**, the action to be taken is the same as that described above for func equal to function address. The same is true if the signal is **SIGCLD** except, that while the process is executing the signal-catching function any received **SIGCLD** signals will be queued and the signal-catching function will be continually reentered until the queue is empty.

The **SIGCLD** affects two other system calls (**wait** and **exit** in the following ways:

wait If the func value of **SIGCLD** is set to **SIG_IGN** and a **wait** is executed, the **wait** will block until all of the calling process's child processes terminate; it will then return a value of -1 with errno set to ECHILD.

exit If in the exiting process's parent process the func value of **SIGCLD** is set to **SIG_IGN**, the exiting process will not create a zombie process.

When processing a pipeline, the shell makes the last process in the pipeline the parent of the proceeding processes. A process that may be piped into in this manner (and thus become the parent of other processes) should take care not to set **SIGCLD** to be caught.

LIMITATIONS

The signals are all inherited from the PDP-11, so some names and some functions are inappropriate to the System 8000.

NAME

stat, fstat - get file status

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>

int stat (file, buf)
char *file;
struct stat *buf;

int fstat (fildes, buf)
int fildes;
struct stat *buf;
```

DESCRIPTION

Stat obtains detailed information about file. **Fstat** obtains the same information about an open file known by the file descriptor from a successful **open**(2) **creat**(2) **dup**(2), or **pipe**(2) call.

File points to a null-terminated string naming a file; buf is the address of a buffer into which information is placed concerning the file. It is unnecessary to have any permissions with respect to the file, but all directories leading to the file must be searchable. The layout of the structure pointed to by buf as defined in <stat.h> is given below. St mode is encoded according to the #define statements.

```
struct  stat
{
        dev_t           st_dev;
        ino_t           st_ino;
        unsigned short  st_mode;
        short           st_nlink;
        short           st_uid;
        short           st_gid;
        dev_t           st_rdev;
        off_t           st_size;
        time_t          st_atime;
        time_t          st_mtime;
        time_t          st_ctime;
};

#define   S_IFMT    0170000   /* type of file */
#define   S_IFDIR   0040000   /* directory */
#define   S_IFCHR   0020000   /* character special */
#define   S_IFBLK   0060000   /* block special */
#define   S_IFREG   0100000   /* regular */
#define   S_IFMPC   0030000   /* multiplexed char special */
#define   S_IFMPB   0070000   /* multiplexed block special */
#define   S_ISUID   0004000   /* set user id on execution */
```

```
#define   S_ISGID   0002000   /* set group id on execution */
#define   S_ISVTX   0001000   /* save swapped text even after use */
#define   S_IREAD   0000400   /* read permission, owner */
#define   S_IWRIT   0000200   /* write permission, owner */
#define   S_IEXEC   0000100   /* execute/search permission, owner */
```

The mode bits 0000070 and 0000007 encode group and others permissions (**chmod**(2)). The defined types, ino t, off t, time t, name various width integer values; dev t encodes major and minor device numbers; their exact definitions are in the include file <sys/types.h> (**types**(5)).

When fildes is associated with a pipe, fstat reports an ordinary file with restricted permissions. The size is the number of bytes queued in the pipe.

St atime is the time the file was last read: it is not set when a directory is searched. St mtime is the time the file was last written or created. It is not set by changes of owner, group, link count, or mode. St ctime is set both both by writing and changing the i-node.

DIAGNOSTICS

Stat will fail if one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

The named file does not exist. [ENOENT]

Search permission is denied for a component of the path prefix. [EACCES]

Buf or path points to an invalid address. [EFAULT]

Fstat will fail if one or more of the following are true:

Fildes is not a valid open file descriptor. [EBADF]

Buf points to an invalid address. [EFAULT]

RETURN VALUE

Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

SEE ALSO

ls(1), filsys(5).

ASSEMBLER

```
CONSTANT STAT          := 18
```

```
...             !* name in rØ, buf in rl *!
                !* (segmented: name in rrØ, buf in rr2)*!
clr r4
sc  #STAT
...             !* return value in r4 *!
                !* carry flag set if error *!

CONSTANT FSTAT          := 28

...             !* fildes in rØ, buf in rl*!
                !* (segmented: fildes in rØ, buf segment*!
                !* number in rl, buf offset in r2)*!
clr r4
sc  #FSTAT
...             !* return value in r4 *!
                !* carry flag set if error *!
```

NAME

stime - set time

SYNOPSIS

```
int stime (tp)
long *tp;
```

DESCRIPTION

Stime sets the system's idea of the time and date. Time, pointed to by tp, is measured in seconds from 0000 GMT Jan 1, 1970. Only the super-user can use this call.

SEE ALSO

date(1), time(2), ctime(3).

DIAGNOSTICS

Stime will fail if the effective user ID of the calling process is not super-user. [EPERM]

RETURN VALUE

Zero is returned if the time was set; -1 if user is not the super-user.

ASSEMBLER

```
CONSTANT STIME := 25

...         !* time in rr0 *!
clr  r4
sc   #STIME
...         !* return value in r4 *!
            !* carry flag set if error *!
```

NAME

sync - update super-block

SYNOPSIS

sync( )

DESCRIPTION

Sync causes all information in core memory that should be on disk to be written out. This includes modified super blocks, modified i-nodes, and delayed block I/O.

It is used by programs which examine a file system, for example **icheck**(M) **df**(M) etc. It is mandatory before a boot.

SEE ALSO

sync(M), update(M).

LIMITATIONS

The writing, although scheduled, is not necessarily complete upon return from **sync**.

ASSEMBLER

```
CONSTANT SYNC  := 36

sc   #SYNC
```

NAME

time, ftime - get date and time

SYNOPSIS

```
long time((long *) 0)

long time( tloc)
long *tloc;

#include <sys/types.h>
#include <sys/timeb.h>
ftime( tp)
struct timeb *tp;
```

DESCRIPTION

Time returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds.

If tloc is nonnull, the return value is also stored in the place to which tloc points.

The ftime entry fills in a structure pointed to by its argument, as defined by sys/timeb.h:

```
/*
 * Structure returned by ftime system call
 */
struct timeb {
    time_t    time;
    unsigned short millitm;
    short     timezone;
    short     dstflag;
};
```

The structure contains the time since the beginning of the system, up to 1000 milliseconds of more-precise interval, the local timezone (measured in minutes of time westward from Greenwich), and a flag that, if nonzero, indicates that Daylight Saving time applies locally during the appropriate part of the year.

DIAGNOSTICS

Time will fail if tloc points to an illegal address. - [EFAULT]

RETURN VALUE

If sucessful, **time** returns the time (a long value, measured in seconds). Otherwise it returns -1 and **errno** is set to indicate the error.

SEE ALSO

date(1), stime(2), ctime(3).

ASSEMBLER

```
        CONSTANT FTIME:= 35

        ...         !* tp in rØ *!
                    !* (segmented: tp in rrØ) *!
        clr         r4
        sc          #FTIME
        ...         !* return value in r4, carry flag set if error*!

        CONSTANT TIME:= 13

        ...
        sc          #TIME
        ...         !* time since 1970 in rr4 *!
```

NAME

times - get process times

SYNOPSIS

long times(buffer)
struct tbuffer ***buffer;**

DESCRIPTION

Times returns time and accounting information for the current process and for the terminated child processes of the current process. All times are in 1/HZ seconds, where HZ=60 in North America.

After the call, the buffer appears as follows:

```
struct tbuffer {
     long proc_user_time;
     long proc_system_time;
     long child_user_time;
     long child_system_time;
};
```

The children times are the sum of the children's process times and their children's times.

Utime is the CPU time used while executing instructions in the user space of the calling process.

Stime is the CPU time used by the system on behalf of the calling process.

Cutime is the sum of the **utimes** and **cutimes** of the child processes.

Cstime is the sum of the **stimes** and **cstimes** of the child processes.

Times will fail if buffer points to an illegal address. [EFAULT]

RETURN VALUE

Upon successful completion, **times** returns the elapsed real time, in 60ths of a second, since an arbitrary point in the past (e.g., system start-up time). This point does not change from one invocation of **times** to another. If **times** fails, a -1 is returned and **errno** is set to indicate the error.

SEE ALSO

exec(2), fork(2), time(2), wait(2).

ASSEMBLER

```
CONSTANT TIMES := 43

...         !* buffer in rØ *!
            !* (segmented: buffer in rrØ) *!
clr  rr4
sc   #TIMES
...         !* return value in rr4 *!
            !* carry flag set if error *!
```

NAME

ulimit - get and set user limits

SYNOPSIS

```
long ulimit (cmd, newlimit)
int cmd;
long newlimit;
```

DESCRIPTION

This function provides for control over process limits. The cmd values available are:

1 Get the process's file size limit. The limit is in units of 512-byte blocks and is inherited by child processes. Files of any size can be read. The default limit on writable file sizes is 5000 blocks.

2 Set the process's file size limit to the value of newlimit. Any process may decrease this limit, but only a process with an effective user ID of super-user may increase the limit. **Ulimit** will fail and the limit will be unchanged if a process with an effective user ID other than super-user attempts to increase its file size limit. [EPERM]

3 Get the maximum possible break value. See **brk**(2).

RETURN VALUE

Upon successful completion, a non-negative value is returned. Otherwise, a value of -1 is returned and **errno** is set to indicate the error.

SEE ALSO

brk(2), write(2).

ASSEMBLER

```
CONSTANT ULIMIT     := 64

...         !* cmd in r0, high word of new limit    *!
            !* in r1, low word of newlimit in r2    *!
clr  rr4
sc   #ULIMIT
...         !* return value in rr4 *!
            !* carry flag set if error *!
```

NAME

umask - set file creation mode mask

SYNOPSIS

umask (complmode)
int complmode;

DESCRIPTION

Umask sets a mask used whenever a file is created by **creat**(2) or **mknod**(2). The actual mode (**chmod**(2)) of the newly created file is the logical and of the given mode and the complement of the argument. Only the low-order nine bits of the mask (the protection bits) participate. In other words, the mask shows the bits to be turned off when files are created.

The previous value of the mask is returned by the call. The value is initially 0 (no restrictions). The mask is inherited by child processes.

The value of complmode is constructed by OR'ing together some of the following hexadecimal values to produce the desired protection:

0x100	read by owner
0x80	write by owner
0x40	execute by owner
0x20	read by group
0x10	write by group
0x08	execute by group
0x04	read by others
0x02	write by others
0x01	execute by others

RETURN VALUE

The previous value of the file mode creation mask is returned.

SEE ALSO

mkdir(1), mknod(1), sh(1), chmod(2), creat(2), mknod(2), open(2).

ASSEMBLER

```
CONSTANT UMASK := 60

...         !* complmode in r0  *!
sc   #UMASK
...         !* return value in r4    *!
```

NAME

umount - unmount a file system

SYNOPSIS

```
int umount (spec)
char *spec;
```

DESCRIPTION

Umount requests that a previously mounted file system contained on the block special device identified by spec be unmounted. Spec is a pointer to a path name. After unmounting the file system, the directory upon which the file system was mounted reverts to its ordinary interpretation.

Umount may be invoked only by the super-user.

DIAGNOSTICS

Umount will fail if one or more of the following are true:

The process's effective user ID is not super-user. [EPERM]

Spec does not exist. [ENXIO]

Spec is not a block special device. [ENOTBLK]

Spec is not mounted. [EINVAL]

A file on spec is busy. [EBUSY]

Spec points outside the process's allocated address space. [EFAULT]

RETURN VALUE

Upon successful completion a value of Ø is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

SEE ALSO

mount(1M), mount(2).

ASSEMBLER

```
CONSTANT UMOUNT     := 22

...         !* spec in rØ *!
            !* (segmented: spec in rrØ) *!
clr  r4
sc   #UMOUNT
...         !* return value in r4 *!
            !* carry flag set if error *!
```

NAME

uname - get name of current Zilog system

SYNOPSIS

```
#include <sys/utsname.h>

int uname (name)
struct utsname *name;
```

DESCRIPTION

Uname stores information identifying the current Zilog system in the structure pointed to by name.

Uname uses the structure defined in **<sys/utsname.h>**:

```
struct utsname {
        char sysname[9];
        char nodename[9];
        char release[9];
        char version[9];
};
extern struct utsname utsname;
```

Uname returns a null-terminated character string naming the current Zilog system in the character array sysname. Similarly, nodename contains the name that the system is known by on a communications network. Release and version further identify the operating system.

Uname will fail if name points to an invalid address. [EFAULT]

RETURN VALUE

Upon successful completion, a non-negative value is returned. Otherwise, -1 is returned and **errno** is set to indicate the error.

SEE ALSO

uname(1).

ASSEMBLER

```
CONSTANT UNAME := 57

...         !* name in rØ *!
            !* (segmented: name in rrØ) *!
clr  rr4
sc   #UNAME
...         !* return value in rr4 *!
            !* carry flag set if error *!
```

NAME

unlink - remove directory entry

SYNOPSIS

unlink (file)
char *file;

DESCRIPTION

File points to a null-terminated string. **Unlink** removes the entry for the file pointed to by file from its directory. If this entry was the last link to the file, the contents of the file are freed and the file is destroyed. If, however, the file was open in any process, the actual destruction is delayed until it is closed, even though the directory entry has disappeared. Write permission is not required on the file itself. It is illegal to unlink a directory (unless you are the super-user).

SEE ALSO

rm(1), link(2).

DIAGNOSTICS

The named file is unlinked unless one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

The named file does not exist. [ENOENT]

Search permission is denied for a component of the path prefix. [EACCES]

Write permission is denied on the directory containing the link to be removed. [EACCES]

The named file is a directory and the effective user ID of the process is not super-user. [EPERM]

The entry to be unlinked is the mount point for a mounted file system. [EBUSY]

The entry to be unlinked is the last link to a pure procedure (shared text) file that is being executed. [ETXTBSY]

The directory entry to be unlinked is part of a read-only file system. [EROFS]

File points outside the process's allocated address space. [EFAULT]

RETURN VALUE

Zero is normally returned; -1 indicates that the file does not exist, that its directory cannot be written, or that the file contains pure procedure text that is currently in use. **Errno** is set to indicate the reason.

ASSEMBLER

```
CONSTANT UNLINK := 1Ø

...          !* filename in rØ   *!
       !* (segmented: filename in rØ)     *!
clr    r4
sc     #UNLINK
...          !* return value in r4, carry flag set if error
```

NAME

utime - set file times

SYNOPSIS

```
#include <sys/types.h>
int utime (file, times)
char *file;
struct utimbuf *times;
```

DESCRIPTION

The **utime** call uses the accessed and updated times in that order from the structure referred to by times to set the corresponding recorded times for file.

If times is **NULL**, the access and modification times of the file are set to the current time. A process must be the owner of the file or have write permission to use **utime** in this manner.

If times is not **NULL**, times is interpreted as a pointer to a **utimbuf** structure and the access and modification times are set to the values contained in the designated structure.

The times in the following structure are measured in seconds since 00:00:00 GMT, Jan. 1, 1970.

```
struct    utimbuf        {
     time_t      actime; /* access time */
     time_t      modtime;/* modification time */
};
```

The caller must be the owner of the file or the super-user. The inode-changed time of the file is set to the current time.

DIAGNOSTICS

Utime will fail if one or more of the following are true:

The named file does not exist. [ENOENT]

A component of the path prefix is not a directory. [ENOTDIR]

Search permission is denied by a component of the path prefix. [EACCES]

The effective user ID is not super-user and not the owner of the file and times is not **NULL**. [EPERM]

The effective user ID is not super-user and not the owner of the file and times is **NULL** and write access is denied. [EACCES]

The file system containing the file is mounted read-only. [EROFS]

Times is not **NULL** and points outside the process's allocated address space. [EFAULT]

File points outside the process's allocated address space. [EFAULT]

RETURN VALUE

Upon successful completion, a value of Ø is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

SEE ALSO

stat (2).

ASSEMBLER

```
CONSTANT UTIME      := 30

...             !* file, timep in r0, r1 respectively*!
                !* (segmented: file, timep in rr0, rr2) *!
clr  r4
sc   #UTIME
...             !* return value in r4 *!
                !* carry flag set if error *!
```

NAME

wait - wait for process to terminate

SYNOPSIS

int wait (status)
int *status;

int wait((int *) Ø)

DESCRIPTION

Wait causes its caller to delay until a signal is received or one of its child processes terminates. If any child has terminated since the last **wait**, return is immediate; if there are no children, return is immediate with the error bit set (a value of -1 returned). The normal return yields the process ID of the terminated child. In the case of several children, several **wait** calls are needed to learn of all the deaths.

If (int)status is nonzero, the high byte of the word pointed to receives the low byte of the argument of **exit** when the child terminated. The low byte receives the termination status of the process. See **signal**(2) for a list of termination statuses (signals); Ø status indicates normal termination. A special status (Øx7f) is returned for a stopped process which has not terminated and can be restarted (**ptrace**(2)). If the seventh bit (Øx8Ø) of the termination status is set, a core image of the process was produced by the system.

If the parent process terminates without waiting on its children, the initialization process (process ID = 1) inherits the children.

SEE ALSO

exit(2), fork(2), signal(2).

DIAGNOSTICS

Wait will fail and return immediately if one or more of the following are true:

The calling process has no existing unwaited-for child processes. [ECHILD]

Status points to an illegal address. [EFAULT]

RETURN VALUE

If **wait** returns due to the receipt of a signal, a value of -1 is returned to the calling process and errno is set to EINTR. If **wait** returns due to a stopped or terminated child process, the process ID of the child is returned to the calling process. Otherwise, a value of -1 is returned and

errno is set to indicate the error.

ASSEMBLER

```
CONSTANT WAIT  := 7

...
sc   #WAIT
...        !* return value in rr4 *!
           !* carry flag set if error *!
```

NAME

write - write on a file

SYNOPSIS

write (fildes, buffer, nbytes)
char *buffer;

DESCRIPTION

A file descriptor is a word returned from a successful **open**(2) **creat**(2) **dup**(2), or **pipe**(2) call.

Buffer is the address of nbytes contiguous bytes that are written on the output file. The number of characters actually written is returned. It is an error if this is not the same as requested.

If the O_APPEND flag of the file status flags is set, the file pointer will be set to the end of the file prior to each write.

If the file being written is a pipe (or FIFO), no partial writes will be permitted. Thus, the write will fail if a write of nbyte bytes would exceed a limit.

If the file being written is a pipe (or FIFO) and the O_NDELAY flag of the file flag word is set, then write to a full pipe (or FIFO) will return a count of Ø. Otherwise (O_NDELAY clear), writes to a full pipe (or FIFO) will block until space becomes available.

Writes which are multiples of 512 characters long and begin on a 512-byte boundary in the file are more efficient than any others.

SEE ALSO

creat(2), open(2), pipe(2).

DIAGNOSTICS

Write will fail and the file pointer will remain unchanged if one or more of the following are true:

Fildes is not a valid file descriptor open for writing. [EBADF]

An attempt is made to write to a pipe that is not open for reading by any process. [EPIPE and SIGPIPE signal]

An attempt was made to write a file that exceeds the process's file size limit or the maximum file size. See ulimit(2). [EFBIG]

Buffer points outside the process's allocated address

space. [EFAULT]

RETURN VALUE

Upon successful completion the number of bytes actually written is returned. Otherwise, -1 is returned and errno is set to indicate the error.

ASSEMBLER

```
        CONSTANT WRITE   := 4

        ...          !* fildes, buffer, nbytes in          *!
                     !* r0, r1, r2 respectively             *!
                !* (segmented: fildes in r0, segment number  *!
                !* of buffer in r1, offset of buffer in r2,  *!
                !* nbytes in r3)                        *!
        sc      #WRITE
        ...          !* return value in r4 *!
                     !* carry flag set if error *!
```

NAME

intro - introduction to library functions

SYNOPSIS

```
#include <stdio.h>
#include <math.h>
```

DESCRIPTION

This section describes functions that are found in various libraries. (Functions that directly invoke ZEUS system primitives are described in Section 2.)

NOTE

Care must be taken when using any library function to read or write files that might be locked against access by another process. Many functions automatically buffer input/output in one-block buffers; that is, if any part of a 512-byte disk block is needed, the whole block is read. If any byte in a block is locked against access buffered access on any other byte in that block causes the accessing process to sleep until the access is unlocked.

Record locking is primarily used by COBOL programs. System files that are routinely locked by system programs are indicated in Section 5.

FILES

/lib/libc.a /lib/slibc.a /lib/libm.a /lib/slibm.a

SEE ALSO

nm(1), ld(1), cc(1), intro(2), sld(1), scc(1)

For a description of ZEUS record locking, see lkdata(2). To turn off buffering for Standard I/O Package functions, see setbuf(3).

NAME

a64l, l64a - convert between long and base-64 ASCII

SYNOPSIS

```
long a64l (s)
char *s ;

char *l64a (l)
long l ;
```

DESCRIPTION

These routines are used to maintain numbers stored in base-64 ASCII. This is a notation by which long integers can be represented by up to six characters; each character represents a "digit" in a radix-64 notation.

The characters used to represent "digits" are: **.** for **Ø**, **/** for 1, **Ø** through **9** for 2-11, **A** through **Z** for 12-37, and **a** through **z** for 38-63.

A64l takes a pointer to a null-terminated base-64 representation and returns a corresponding **long** value. **L64a** takes a **long** argument and returns a pointer to the corresponding base-64 representation.

LIMITATIONS

The value returned by **l64a** is a pointer into a static buffer, the contents of which are overwritten by each call.

NAME

abort - generate IOT fault

DESCRIPTION

Abort sends an IOT signal that normally terminates the process with a memory dump which can be used for debugging. (An IOT trap cannot occur in the hardware; this is a software trap.)

SEE ALSO

adb(1), signal(2), exit(2).

DIAGNOSTICS

"IOT trap - core dumped" from the shell.

NAME

abs - integer absolute value

SYNOPSIS

```
int abs ( i )
   int i;
```

DESCRIPTION

Abs returns the absolute value of its integer operand.

SEE ALSO

floor(3) for fabs.

LIMITATIONS

The returned value is what the hardware gives on the largest negative integer. On the Z8000, 0x8000 returns 0x8000.

NAME

assert - program verification

SYNOPSIS

#include <assert.h>

assert (*expression*)

DESCRIPTION

Assert is a macro that indicates *expression* is expected to be true at this point in the program. It causes an **exit**(2) with a diagnostic comment to the standard error when *expression* is false (return of 0). Compiling with the **cc**(1) preprocessor option **-DNDEBUG** effectively deletes *assert* from the program.

DIAGNOSTICS

"Assertion failed: file *f* line *n*." *F* is the source file and *n* the source line number of the **assert** statement.

NAME

atof, atofs, atofd, _atof, atoi, atol - convert ASCII to numbers

SYNOPSIS

```
float atofs(nptr)
char *nptr;

double atof(nptr)
char *nptr;

double atofd(nptr)
char *nptr;

_atof(nptr)
char *nptr;

int atoi(nptr)
char *nptr;

long atol(nptr);
char *nptr;
```

DESCRIPTION

These functions convert a string pointed to by nptr to floating, integer, and long integer representation respectively. The first unrecognized character ends the string. A value is returned according to the following convention: **atofs** returns in rr2, **atofd** and **atof** return in rq4, and **_atof** returns in f0. **Atoi** returns an integer value and **atol** returns a long value.

In its various forms, **atof** recognizes an optional sequence of tabs and spaces, an optional sign, then a numeric string. For finite numbers, the numeric string consists of a string of digits containing an optional decimal point, then an optional exponent consisting of the letter e or E followed by an optional sign and an integer.

A numeric string beginning with the letter I denotes signed infinity. A NAN (Not-A-Number) string has the form "NAN(xxxxxx)." The x's are hexadecimal digits to be mapped into the leading 24 significant bits of the NAN. **Atoi** and **atol** recognize an optional string of tabs and spaces, then an optional sign, then a string of digits.

SEE ALSO

scanf(3).

LIMITATIONS

All the aliases do not but should collapse into a single routine (called **atof** here) that returns an extended value in

f0. Also, a variant of **atof** should be created to subsume the task of recognizing floating-point strings, something now left to other parsers.

NAME

bsearch - binary search

SYNOPSIS

```
char *bsearch (key, base, nel, width, compar)
     char *key;
     char *base;
     int nel, width;
     int (*compar)();
```

DESCRIPTION

Bsearch is a binary search routine generalized from Knuth (6.2.1) Algorithm B. It returns a pointer to a table indicating the location where a datum may be found. The table must have been sorted in increasing order. The first argument, key, is a pointer to the datum in the table. The second argument, base, is a pointer to the base of the table. The third argument, nel, is the number of elements in the table. The fourth argument, width, is the width of an element in bytes. The last argument, compar, is the name of the comparison routine. It is called with two arguments which are pointers to the elements being compared. The routine must return an integer less than, equal to, or greater than Ø, according to whether the first argument considered is less than, equal to, or greater than the second.

DIAGNOSTICS

Zero is returned if the key can not be found in the table.

SEE ALSO

lsearch(3), qsort(3).

NAME

toupper, tolower, toascii - character translation

SYNOPSIS

```
#include <ctype.h>

int toupper (c)
int c;

int tolower (c)
int c;

int _toupper (c)
int c;

int _tolower (c)
int c;

int toascii (c)
int c;
```

DESCRIPTION

Toupper and **tolower** have as domain the range of **getc:** the integers from -1 through 255. If the argument of **toupper** represents a lower-case letter, the result is the corresponding upper-case letter. If the argument of **tolower** represents an upper-case letter, the result is the corresponding lower-case letter. All other arguments in the domain are returned unchanged.

_toupper and **_tolower** are macros that accomplish the same thing as **toupper** and **tolower** but have restricted domains and are faster. **_toupper** requires a lower-case letter as its argument; its result is the corresponding upper-case letter. **_tolower** requires an upper-case letter as its argument; its result is the corresponding lower-case letter. Arguments outside the domain cause garbage results.

Toascii yields its argument with all bits turned off that are not part of a standard ASCII character; it is intended for compatibility with other systems.

SEE ALSO

ctype(3).

NAME

crypt, setkey, encrypt - DES encryption

SYNOPSIS

```
char *crypt(key, salt)
char *key, *salt;

setkey(key)
char *key;

encrypt(block, edflag)
char *block;
int edflag;
```

DESCRIPTION

Crypt is the password encryption routine. It is based on the National Bureau of Standards' Data Encryption Standard (DES), with variations intended to frustrate use of hardware implementations of the DES for key search.

The first argument to **crypt** is a user's typed password. The second is a two-character string chosen from the set [a-zA-Z0-9./]. The salt string disturbs the DES algorithm in one of 4096 different ways, after which the password is used as the key to encrypt repeatedly a constant string. The returned value points to the encrypted password, in the same alphabet as the salt. The first two characters are the salt itself.

The other entries provide rather primitive access to the actual DES algorithm. The argument of **setkey** is a character array of length 64 containing only the characters with numerical value 0 and 1. If this string is divided into groups of eight, the low-order bit in each group is ignored and leads to a 56-bit key that is set into the machine.

The argument to the **encrypt** entry is likewise a character array of length 64 containing 0's and 1's. The argument array is modified in place to a similar array representing the bits of the argument after having been subjected to the DES algorithm using the key set by **setkey**. If edflag is 0, the argument is encrypted; if nonzero, it is decrypted.

SEE ALSO

crypt(1), passwd(1), login(1), getpass(3), passwd(5).

LIMITATIONS

The return value points to static data whose content is overwritten by each call.

NAME

ctermid - generate file name for terminal

SYNOPSIS

```
#include <stdio.h>

char *ctermid(s)
     char *s;
```

DESCRIPTION

Ctermid generates a string that refers to the controlling terminal for the current process when used as a file name.

If (int)s is zero, the string is stored in an internal static area, the contents of which are overwritten at the next call to **ctermid**, and the address of which is returned. If (int)s is non-zero, then s is assumed to point to a character array of at least **L_ctermid** elements; the string is placed in this array and the value of s is returned. The manifest constant **L_ctermid** is defined in **<stdio.h>**.

NOTES

The difference between **ctermid** and **ttyname**(3) is that **ttyname** must be handed a file descriptor and returns the actual name of the terminal associated with that file descriptor, while **ctermid** returns a magic string (/dev/tty) that will refer to the terminal if used as a file name. Thus **ttyname** is useless unless the process already has at least one file open to a terminal.

SEE ALSO

ttyname(3).

NAME

ctime, localtime, gmtime, asctime, tzset - convert date and time to ASCII

SYNOPSIS

```
char *ctime (clock)
long *clock;

#include <time.h>

struct tm *localtime (clock)
long *clock;

struct tm *gmtime (clock)
long *clock;

char *asctime (tm)
struct tm *tm;

tzset ( )
```

DESCRIPTION

Ctime converts a time pointed to by clock such as returned by **time**(2) into ASCII and returns a pointer to a 26-character string in the following form. All the fields have constant width.

```
Sun Sep 16 Ø1:Ø3:52 1973\n\Ø
```

Localtime and **gmtime** return pointers to structures containing the broken-down time. **Localtime** corrects for the time zone and possible daylight savings time; **gmtime** converts directly to GMT, which is the time the ZEUS system uses. **Asctime** converts a broken-down time to ASCII and returns a pointer to a 26-character string.

The structure declaration from the include file is:

```
struct    tm { /* see ctime(3) */
     int  tm_sec;
     int  tm_min;
     int  tm_hour;
     int  tm_mday;
     int  tm_mon;
     int  tm_year;
     int  tm_wday;
     int  tm_yday;
     int  tm_isdst;
};
```

These quantities give the time on a 24-hour clock, day of month (1-31), month of year (Ø-11), year - 19ØØ, day of week

(Sunday = 0), day of year (0-365), and a flag that is non-zero if daylight saving time is in effect.

The external **long** variable timezone contains the difference, in seconds, between GMT and local standard time (in EST, timezone is 5*60*60); the external variable daylight is non-zero if and only if the standard U.S.A. Daylight Savings Time conversion should be applied. The program knows about the peculiarities of this conversion in 1974 and 1975; if necessary, a table for these years can be extended.

If an environment variable named **TZ** is present, **asctime** uses the contents of the variable to override the default time zone. The value of **TZ** must be a three-letter time zone name, followed by a number representing the difference between local time and Greenwich time in hours, followed by an optional three-letter name for a daylight time zone. For example, the setting for New Jersey would be **EST5EDT.** The effects of setting **TZ** are thus to change the values of the external variables timezone and daylight; in addition, the time zone names contained in the external variable

```
char *tzname[2] = {"EST", "EDT"};
```

are set from the environment variable. The function **tzset** sets the external variables from **TZ**; it is called by **asctime** and may also be called explicitly by the user.

SEE ALSO

time(2), getenv(3), environ(7).

LIMITATIONS

The return values point to static data whose content is overwritten by each call.

NAME

isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace , ispunct, isprint, isgraph, iscntrl, isascii - character classification

SYNOPSIS

#include <ctype.h>

int isalpha (c)
int c;

. . .

DESCRIPTION

These macros classify ASCII-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. **Isascii** is defined on all integer values; the rest are defined only where **isascii** is true and on the single non-ASCII value **EOF** (see **stdio**(3)).

isalpha	c is a letter
isupper	c is an upper case letter
islower	c is a lower case letter
isdigit	c is a digit [Ø-9]
isxdigit	c is a hexidecimal digit [Ø-9], [A-F] or [a-f]
isalnum	c is an alphanumeric
isspace	c is a space, tab, carriage return, new-line, vertical tab, or form-feed
ispunct	c is a punctuation character (neither control nor alphanumeric)
isprint	c is a printing character, code Ø4Ø (space) through Ø176 (tilde)
isgraph	c is a printing character, like **isprint** except false for space
iscntrl	c is a delete character (Ø177) or ordinary control character (less than Ø4Ø).
isascii	c is an ASCII character, code less than Ø2ØØ

SEE ALSO

ascii(7).

NAME

curses - screen functions with "optimal" cursor motion

SYNOPSIS

cc [flags] files **-lcurses -ltermlib** [libraries]

scc [flags] files **-lcurses -ltermlib** [libraries]

DESCRIPTION

These routines give the user a method of updating screens with reasonable optimization. They keep an image of the current screen and the user sets up an image of a new one. Then the refresh() tells the routines to make the current screen look like the new one. In order to initialize the routines, the routine initscr() must be called before any of the other routines that deal with windows and screens are used.

Also, be sure to include the header file, /usr/include/curses.h in your source program.

FILES

/usr/lib/libcurses.a /usr/lib/slibcurses.a

SEE ALSO

Screen Updating and Cursor Movement Optimization: A Library Package , in the ZEUS Languages / Programming Tools Manual.
termcap(5), ioctl(2), setenv (in csh(1)).

FUNCTIONS

addch(ch)	add a character to stdscr
addstr(str)	add a string to stdscr
box(win,vert,hor)	draw a box around a window
clear()	clear stdscr
clearok(scr,boolf)	set clear flag for scr
clrtobot()	clear to bottom on stdscr
clrtoeol()	clear to end of line on stdscr
delwin(win)	delete win
echo()	set echo mode
erase()	erase stdscr
getch()	get a char through stdscr
getstr(str)	get a string through stdscr
gettmode()	get tty modes
getyx(win,y,x)	get (y,x) co-ordinates
inch()	get char at current (y,x) co-ordinates
initscr()	initialize screens
leaveok(win,boolf)	set leave flag for win
longname(termbuf,name)	get long name from termbuf
move(y,x)	move to (y,x) on stdscr
mvcur(lasty,lastx,newy,newx)	actually more cursor
newwin(lines,cols,beg_x,beg_y)	create a new window
nl()	set newline mapping

noecho()	unset echo mode
nonl()	unset newline mapping
noraw()	unset raw mode
overlay(win1,win2)	overlay win1 on win2
overwrite(win1,win2)	overwrite win1 on top of win2
printw(fmt,arg1,arg2,...)	printf on stdscr
raw()	set raw mode
refresh()	make current screen look like stdscr
restty()	reset tty flags to stored value
savetty()	stored current tty flags
scanw(fmt,arg1,arg2,...)	scanf from stdscr
scroll(win)	scroll win one line
scrollok(win,boolf)	set scroll flag
setterm(name)	set term variables for name
subwin(win,lines,cols,beg_y,beg_x)	create a subwindow win
unctrl(ch)	printable version of ch
waddch(win,ch)	add char to win
waddstr(win,str)	add string to win
wclear(win)	clear win
wclrtobot(win)	clear to bottom of win
wclrtoeol(win)	clear to end of line on win
werase(win)	erase win
wgetch(win)	get a char through win
wgetstr(win,str)	get a string through win
winch(win)	get char at current (y,x) from win
wmove(win,y,x)	set current (y,x) co-ordinates on w
wprintw(win,fmt,arg1,arg2,...)	printf on win
wrefresh(win)	make screen look like win
wscanw(win,fmt,arg1,arg2,...)	scanf through win

NAME

cuserid - character login name of the user

SYNOPSIS

```
#include <stdio.h>

char *cuserid (s)
     char *s;
```

DESCRIPTION

Cuserid generates a character representation of the login name of the owner of the current process. If (int)s is zero, this representation is generated in an internal static area, the address of which is returned. If (int)s is non-zero, s is assumed to point to an array of at least **L_cuserid** characters; the representation is left in this array. The manifest constant **L_cuserid** is defined in **<stdio.h>**.

DIAGNOSTICS

If the login name cannot be found, **cuserid** returns **NULL**; if s is non-zero in this case, **\0** will be placed at ***s**.

SEE ALSO

getlogin(3), getpwuid(3).

LIMITATIONS

Cuserid uses **getpwnam**(3); thus the results of a user's call to the latter will be obliterated by a subsequent call to the former.

The name **cuserid** is rather a misnomer.

NAME

disasm, disinit - disassemble Z8000 instructions

SYNOPSIS

```
#include <disasm.h>

disinit(getfunc,symfunc,flag)
unsigned *(getfunc)();
int *(symfunc)();
int flag;

ADDR *disasm(adr,space,insptr)
 ADDR *adr;
int space;
 INSTR *insptr;
```

On **cc**(1) or **scc**(1) command include a **-lz8000** option to access proper library archive file.

DESCRIPTION

These functions are used to disassemble binary words into the equivalent Z8000 assembly instructions. The routines handle segmented and non-segmented code, and floating point instructions. Disinit is called before **disasm** to initialize the various parameters that the disassembly routines require. The different structures, variables, and constants used to communicate with **disasm** as defined in **<disasm.h>** are:

```
/* @[$]disasm.h 1.1  12/11/81 17:07:54 - Zilog Inc */
/*   disasm/disinit  .... disassembler include file */

typedef struct address { char a_segno;
                long a_offset;
              } ADDR;

typedef struct _instr { char *d_opcode;
                char *d_operand[4];
                char *d_comment;
              } INSTR;

#define NON_SEG 0
#define SEG     1

#define ISYMBOL 0
#define DSYMBOL 1

int     disinit();
ADDR    *disasm();
```

Getfunc is the address of a routine that **disasm** will use to request words to be disassembled. The calling sequence is:

```
getfunc(adr,space)
     ADDR *adr;
     int space;
```

where adr is a pointer to a structure containing the address of the word **disasm** is requesting and space is an integer value containing the value passed to **disasm** (see below).

Symfunc is the address of a routine that **disasm** calls in order to allow the calling routine to translate hexadecimal addresses to symbolic representations. Its calling sequence is:

```
symfunc(adr,symtype,s)
     ADDR *adr;
     int symtype;
     char *s;
```

where adr indicates the address to be translated. Symtype contains either ISYMBOL or DSYMBOL (defined in **<disasm.h>**) indicating whether the address should be considered to be in I-Space or D-Space. S is a string pointer into which the function should copy the null-terminated string associated with the address given. For programs that do not handle symbols, a simple function containing

```
sprintf(s,"%%%04X",adr->a_offset); /* non-seg. example */
```

is sufficient.

Flag indicates whether the information is to be interpreted as segmented or non-segmented code. The only valid values for this parameter are SEG and NON_SEG (defined in **<disasm.h>** and any others will give undefined results.

Disasm is the routine that actually does the disassembly. Adr is the address of an ADDR structure that contains the starting address of the instruction to be disassembled. Space is an integer containing information to indicate in what space the words to be disassembled reside. The disassembler does not use this parameter in any way except to pass it to **getfunc.** Insptr is the address of an INSPTR structure (defined in **<disasm.h>**) into which **disasm** will return the disassembled instruction. It contains five null-terminated character string fields: an opcode, four operands, and a comment. The opcode field contains the standard **ZILOG** mnemonic for the decoded instruction (see note on errors below). In the case of extended processing unit (epu) instructions, if it is determined that the code

does not represent a valid floating point instruction, the generalized form suggested in chapter 3 of Z8000 Assembly Language Programming, published by Osborne/Mc Graw-Hill, is used.

The four operand fields contain between zero and four null-terminated strings containing the translated operands. If any operand does not exist for a particular instruction, the first character of that field will be a null. The last field, comment, contains auxiliary information. For instance, the disassembly of the system call instruction **sc #5** contains a comment field containing the string "! open !" indicating that a system call number five is the **ZEUS open**(2) system call.

Disasm returns a pointer to an ADDR structure containing the address of the beginning of the next instruction in the case of successful completion, and a null in the case of an error. An error is indicated when **disasm** is unable to translate the given code into an instruction. In this case the opcode field will contain a string of the form %xxxx where xxxx is the hexadecimal representation of the word given to **disasm.** In an error situation, all fields except the opcode are guaranteed to start with a null.

FILES

/usr/lib/libz8000.a /usr/lib/slibz8000.a

SEE ALSO

adb(1).

DIAGNOSTICS

A null is returned instead of a pointer when instruction cannot be disassembled.

LIMITATIONS

Disasm will always disassemble valid instructions correctly. It may, however, misinterpret invalid binary sequences to be valid instructions. This is most noticeable in floating point instructions where some bits contain constant fields other than the opcode.

NAME

abs, atoi, close, creat, exit, getc, getchar, goodmagic, length, longswap, lseek, open, printf, putc, putchar, read, swab, swap, write - Z8000 development module library

SYNOPSIS

```
abs(i)
int i;

atoi(s, len, result)
char *s;
int len;
int *result;

close(fd)
int fd;

creat(name, mode)
char *name;
int mode;

getc (fd)
int fd;

getchar()

goodmagic(magic)
int magic;

length(s)
char *s;

longswap(lptr)
long *lptr;

lseek(fd, offset, whence)
int fd;
long offset;
int whence;

open(name, mode)
char *name;
int mode;

printf(frmt [ , arg ... ] )
char *frmt;

putc(ch, fd)
char ch;
int fd;

putchar(c)
```

```
char c;

read(fd, addr, cnt)
int fd, cnt;
char *addr;

swab(from, to, len)
int *from, *to;
int len;

swap(i)
int i;

int write (fd, addr, cnt)
int fd, cnt;
char *addr;
```

DESCRIPTION

The file /usr/lib/slibdm.a is a library containing functions for use with the segmented C compiler. The file /usr/lib/libdm.a is a library containing functions for use with the nonsegmented C compiler. These routines are set up to enable a program running on a development module to open, create, read, write and close files on the host S8000. While a program is running on the development module, the **SYS**(3) program is run on the S8000 (see **load**(1)). The I/O routines in this package communicate with the SYS program to perform the functions described. A brief description of each routine follows:

Abs is used to determine the absolute value of its argument.

Atoi converts its argument from ascii to integer. The result is returned at the address specified by the third argument.

Open, **creat**, and **close** are ZEUS-style file access calls to files on the host system. The SYS program interprets these calls and opens files on behalf of the program running on the development module. The arguments to these calls are the same as the equivalent ZEUS calls, **open**(2), **creat**(2) and **close**(2).

Getc, **getchar**, **putc**, **putchar** and **printf** are ZEUS-style standard i/o calls to get and put characters. **Lseek** performs seeks within files. **Read** and **write** are ZEUS-style read and write functions. These calls perform i/o by communicating with the SYS program on the host. See **getc**(3), **lseek**(3), **putc**(3), **printf**(3) **read**(2) and **write**(2) for complete explanations.

Goodmagic determines if the value passed is a valid object module magic number. See **goodmagic**(3) for complete details.

Length is passed a null-terminated string. A null is a zero byte. It returns the number of bytes preceeding the null, or 256 if no null is encountered in the first 256 bytes.

Longswap, swab and **swap** are byte-swapping routines. See **swab**(3) and **swap**(3) for complete details.

The loader option **-ldm** can be used to access the libraries, both segmented and nonsegmented. The entry point to a segmented program should be specified **-e start** for a segmented program, **-e startup** for a nonsegmented program. See **ld**(1) and **sld**(1).

FILES

/usr/lib/libdm.a	nonsegmented library
/usr/lib/slibdm.a	segmented library

SEE ALSO

ld(1), load(1), sld(1), close(2), creat(2), lseek(2), open(2), read(2), write(2), getc(3), goodmagic(3), printf(3), putc(3), swab(3), swap(3).

NAME

ecvt, _ecvt, fcvt, _fcvt, gcvt, _gcvt - output conversion

SYNOPSIS

```
char *ecvt(value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *fcvt(value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *gcvt(value, ndigit, buf)
double value;
int ndigit;
char *buf;
```

DESCRIPTION

Ecvt converts the value to a null-terminated, rounded string of ndigit ASCII digits (limited to 19) and returns a pointer to it. The position of the decimal point relative to the beginning of the string is stored indirectly through **decpt** negative means to the left of the returned digits. If the sign of the result is negative, the word pointed to by sign is nonzero, otherwise it is zero.

Fcvt is identical to **ecvt**, except that the correct digit has been rounded for Fortran F-format output with ndigits fraction digits. No more than 19 integer and fraction digits can be converted.

Gcvt converts the value to a null-terminated ASCII string in buf and returns a pointer to buf. It produces a string, ready for printing, with ndigit significant digits. Fortran F format is used, if reasonable; otherwise E format is used.

SEE ALSO

printf(3).

DIAGNOSTICS

If the total number of siginificant digits requested in fcvt exceeds 19 the digit string is set to ?????????.

LIMITATIONS

The return values of ecvt and fcvt are static buffers whose content is overwritten by each call; thus the routines are not reentrant.

NAME

end, etext, edata - last locations in program

SYNOPSIS

```
extern end;
extern etext;
extern edata;
```

DESCRIPTION

The address of **etext** is the first address above the program text, **edata** above the initialized data region, and **end** above the uninitialized data region.

When execution begins, the program break coincides with **end**, but many functions reset the program break, among them the routines of **brk**(2), **malloc**(3), standard input/output (stdio(3)), and the profile (**-p**) option of **cc**(1). The current value of the program break is reliably returned by sbrk(0) (brk (2)).

SEE ALSO

brk(2), malloc(3).

NAME

exp, log, log1Ø, pow, sqrt - exponential functions

SYNOPSIS

```
#include <math.h>

double exp(x)
double x;

double log(x)
double x;

double log1Ø(x)
double x;

double pow(x, y)
double x, y;

double sqrt(x)
double x;
```

DESCRIPTION

Exp returns the exponential function of x.

Log returns the natural logarithm of x; **log1Ø** returns the base 1Ø logarithm.

Pow returns x raised to the power of y.

Sqrt returns the square root of x.

SEE ALSO

hypot(3), sinh(3), intro(2).

DIAGNOSTICS

Exp and **pow** return a huge value when the correct value would overflow; errno is set to ERANGE. **Pow** returns Ø and sets errno to EDOM when the second argument is negative and non-integral and when both arguments are Ø.

Log returns Ø when x is zero or negative; errno is set to EDOM.

Sqrt returns Ø when x is negative; errno is set to EDOM.

NAME

fclose, fflush - close or flush a stream

SYNOPSIS

```
#include <stdio.h>

fclose(stream)
FILE *stream;

fflush(stream)
FILE *stream;
```

DESCRIPTION

Fclose causes any buffers for the named stream to be emptied, and the file to be closed. Buffers allocated by the standard input/output system are freed.

Fclose is performed automatically upon calling **exit**(2).

Fflush causes any buffered data for the named output stream to be written to that file. The stream remains open.

SEE ALSO

close(2), fopen(3), setbuf(3).

DIAGNOSTICS

These routines return **EOF** if stream is not associated with an output file, or if buffered data cannot be transferred to that file.

NAME

ferror, feof, clearerr, fileno - stream status inquiries

SYNOPSIS

```
#include <stdio.h>

feof(stream)
FILE *stream;

ferror(stream)
FILE *stream;

clearerr(stream)
FILE *stream;

fileno(stream)
FILE *stream;
```

DESCRIPTION

Feof returns nonzero if end of file has been read on stream; it returns zero otherwise.

Ferror returns nonzero if an error has occurred reading or writing the named stream; it returns zero otherwise. **Ferror** does not clear the error status: once **ferror** returns nonzero it continues to return nonzero until the stream is closed or **clearerr** is used to reset the error status.

Clearerr resets the error status on the stream. Subsequent calls to **ferror** with the same stream name argument return zero until another error occurs.

Fileno returns the integer file descriptor associated with the stream **open**(2)).

These functions are implemented as macros; they cannot be redeclared.

SEE ALSO

fopen(3), open(2).

NAME

floor, fabs, ceil, fmod - absolute value, floor, ceiling functions

SYNOPSIS

```
#include <math.h>

double fabs(x)
double(x);

double floor(x)
double x;

double ceil(x)
double x;

double fmod(x, y)
double x, y;
```

DESCRIPTION

Fabs returns the absolute value |x|.

Floor returns the largest integer not greater than x.

Ceil returns the smallest integer not less than x.

Fmod returns the number f such that x = iy + f, for some integer, i, and 0 <= f < y.

SEE ALSO

abs(3).

NAME

fopen, freopen, fdopen - open a stream

SYNOPSIS

```
#include <stdio.h>

FILE *fopen(filename, type)
char *filename, *type;

FILE *freopen(filename, type, stream)
char *filename, *type;
FILE *stream;

FILE *fdopen(fildes, type)
char *type;
```

DESCRIPTION

Fopen opens the file named by file-name and associates a stream with it. **Fopen** returns a pointer to be used to identify the stream in subsequent operations.

Type is a character string having one of the following values:

"r" open for reading

"w" create for writing

"a" append: open for writing at end of file, or create for writing

"r+" open for update (reading and writing)

"w+" create for update

"a+" append: open or create for update at end of file

Freopen substitutes the named file in place of the open stream. It returns the original value of stream. The original stream is closed, regardless of whether the open ultimately succeeds.

Freopen is typically used to attach the preopened constant names **stdin**, **stdout**, and **stderr** to specified files.

Fdopen associates a stream with a file descriptor obtained from **open**(2), **dup**(2), **creat**(2), or **pipe**(2). The type of the stream must agree with the mode of the open file.

When a file is opened for update, both input and output may be done on the resulting stream. However, output may not be directly followed by input without an intervening **fseek** or

rewind, and input may not be directly followed by output without an intervening **fseek**, **rewind**, or an input operation which encounters end of file.

SEE ALSO

open(2), fclose(3).

DIAGNOSTICS

Fopen and **freopen** return the pointer **NULL** if filename cannot be accessed.

LIMITATIONS

Fdopen is not portable to systems other than ZEUS.

NAME

fread, fwrite - buffered binary input/output

SYNOPSIS

```
#include <stdio.h>

fread((char *)ptr, sizeof(*ptr), nitems, stream)
FILE *stream;
int ptr, nitems;

fwrite((char *)ptr, sizeof(*ptr), nitems, stream)
FILE *stream;
int ptr, nitems;
```

DESCRIPTION

Fread reads, into a block beginning at ptr, nitems of data of the type of *ptr from the named input stream. It returns the number of items actually read.

Fwrite appends at most nitems of data of the type of *ptr beginning at ptr to the named output stream. It returns the number of items actually written.

SEE ALSO

read(2), write(2), fopen(3), getc(3), putc(3), gets(3), puts(3), printf(3), scanf(3).

DIAGNOSTICS

Fread and **fwrite** return Ø upon end of file or error.

NAME

frexp, ldexp, modf - split into mantissa and exponent

SYNOPSIS

```
double frexp(value, eptr)
double value;
int *eptr;

double ldexp(value, exp)
double value;
int exp;

double modf(value, iptr)
double value, *iptr;
```

DESCRIPTION

Frexp returns the mantissa of a double *value* as a double quantity, *x*, of magnitude less than 1 and stores an integer *n* such that *value* = *x**2***n* indirectly through *eptr*.

Ldexp returns the quantity *value**2***exp*.

Modf returns the positive fractional part of *value* and stores the integer part indirectly through *iptr*.

NAME

fseek, ftell, rewind - reposition a stream

SYNOPSIS

```
#include <stdio.h>

fseek(stream, offset, ptrname)
FILE *stream;
long offset;
int ptrname;

long ftell(stream)
FILE *stream;

rewind(stream)
```

DESCRIPTION

Fseek sets the position of the next input or output operation on the *stream*. The new position is at the signed distance *offset* bytes from the beginning, the current position, or the end of the file, according as *ptrname* has the value 0, 1, or 2.

Fseek undoes any effects of **ungetc**(3).

Ftell returns the current value of the offset relative to the beginning of the file associated with the named *stream*. It is measured in bytes on ZEUS; on some other V7 Unix systems it is the only foolproof way to obtain an *offset* for **fseek**.

Rewind(stream) is equivalent to **fseek**(*stream*, 0L, 0).

SEE ALSO

lseek(2), fopen(3).

DIAGNOSTICS

Fseek returns -1 for improper seeks.

NAME

gamma - log gamma function

SYNOPSIS

```
#include <math.h>
extern int signgam;

double gamma (x)
    double x;
```

DESCRIPTION

Gamma returns (GAMMA). The sign of (GAMMA) is returned in the external integer signgam. The following C program fragment might be used to calculate Gamma

```
y = gamma (x);
if (y > 88.Ø)
{
     perror ("progname");
     exit (1);
}
y = exp (y) * signgam;
```

DIAGNOSTICS

For negative integer arguments, a huge value is returned, and errno is set to **EDOM** (see INTRO(2)).

NAME

getc, getchar, fgetc, getw - get character or word from stream

SYNOPSIS

```
#include <stdio.h>

int getc ( stream )
FILE * stream;

int getchar()

int fgetc ( stream )
FILE * stream;

int getw ( stream )
FILE * stream;
```

DESCRIPTION

Getc returns the next character from the named input stream.

Getchar() is identical to getc(stdin).

Fgetc saves object text and behaves like getc, but is a genuine function, not a macro.

Getw returns the next word from the named input stream. It returns the constant **EOF** upon end of file or error, but feof and ferror(3) must be used to check the success of getw. Getw assumes no special alignment in the file.

SEE ALSO

fopen(3), putc(3), gets(3), scanf(3), fread(3), ungetc(3)

DIAGNOSTICS

These functions return the integer constant **EOF** at end of file or upon read error.

A stop with message "Reading bad file" means an attempt has been made to read from a stream that has not been opened for reading by fopen.

LIMITATIONS

The end-of-file return from getchar in ZEUS is incompatible with that in UNIX editions 1-6. (ZEUS is derived from UNIX seventh edition.)

Because it is implemented as a macro, getc treats a stream argument with side effects incorrectly. In particular, getc(*f++); does not work.

NAME

getenv - value for environment name

SYNOPSIS

char *getenv(name)
**char ***name;

DESCRIPTION

Getenv searches the environment list (**environ**(7)) for a string of the form name=value and returns value if such a string is present, otherwise Ø (NULL).

SEE ALSO

environ(7), exec(2).

NAME

getgrent, getgrgid, getgrnam, setgrent, endgrent - get group file information

```
#include <grp.h>

    struct group *getgrent();

    struct group *getgrgid(gid) int gid;

    struct group *getgrnam(name) char *name;

    int setgrent();

    int endgrent();
```

DESCRIPTION

Each call to **getgrent, getgrgid** and **getgrnam** reads one or more lines from a single input stream opened from the group file. The function places the contents of the last line read in a structure and returns a pointer to this structure. **Getgrent** reads a single line; **getgrgid** reads until its argument is matched by the numeric group ID field of the last line read; **getgrnam** reads until its argument string is matched by the name field of the last line read.

Setgrent rewinds the input stream: subsequent calls to **getgrent, getgrid,** or **getgrnam** start from the beginning of the file.

Endgrent closes the input stream. This can be used to prevent the process from running out of file descriptors.

Getgrent, getgrid, and **getgrnam** point to a structure of type group. This is the declaration of group:

```
struct    group { /* see getgrent(3) */
     char *gr_name;
     char *gr_passwd;
     int  gr_gid;
     char **gr_mem;
};
```

The members of this structure are:

gr_name
: The group's name.

gr_passwd
: The group's encrypted password.

gr_gid
: The numerical group-ID.

gr_mem
: Null-terminated vector of pointers to the individual

member names.

FILES

/etc/group - group file

SEE ALSO

getlogin(3), getpwent(3), group(5).

DIAGNOSTICS

A null pointer (Ø) is returned on end of file, search failure, or error.

LIMITATIONS

The pointer returned points to a static structure; information in this structure must be copied if it is to be saved.

NAME

getlogin - get login name

SYNOPSIS

char *getlogin();

DESCRIPTION

Getlogin returns a pointer to the login name as found in /etc/utmp. It is used in conjunction with **getpwnam** to locate the correct password file entry when the same user ID is shared by several login names.

If **getlogin** is called within a process that is not attached to a typewriter, it returns NULL. The correct procedure for determining the login name is to first call **getlogin** and if it fails, to call **getpwuid**.

FILES

/etc/utmp

SEE ALSO

getpwent(3), getgrent(3), utmp(5).

DIAGNOSTICS

Returns NULL (Ø) if name not found.

LIMITATIONS

The return values point to static data whose content is overwritten by each call.

NAME

getopt - get option letter from argv

SYNOPSIS

```
int getopt (argc, argv, optstring)
int argc;
char **argv;
char *optstring;

extern char *optarg;
extern int optind;
```

DESCRIPTION

Getopt returns the next option letter in argv that matches a letter in optstring. Optstring is a string of recognized option letters; if a letter is followed by a colon, the option is expected to have an argument that may or may not be separated from it by white space. Optarg is set to point to the start of the option argument on return from **getopt**.

Getopt places in optind the argv index of the next argument to be processed. Because optind is external, it is normally initialized to zero automatically before the first call to **getopt**.

When all options have been processed (i.e. up to the first non-option argument), **getopt** returns **EOF**. The special option -- may be used to delimit the end of the options; **EOF** will be returned, and -- will be skipped.

DIAGNOSTICS

Getopt prints an error message on stderr and returns a question mark (**?**) when it encounters an option letter not included in optstring.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the mutually exclusive options **a** and **b**, and the options **f** and **o**, both of which require arguments:

```
main (argc, argv)
int argc;
char **argv;
{
    int c;
    extern int optind;
    extern char *optarg;
    .
    .
    .
    while ((c = getopt (argc, argv, "abf:o:")) != EOF)
```

```
        switch (c) {
        case 'a':
            if (bflg)
                errflg++;
            else
                aflg++;
            break;
        case 'b':
            if (aflg)
                errflg++;
            else
                bproc();
            break;
        case 'f':
            ifile = optarg;
            break;
        case 'o':
            ofile = optarg;
            bufsiza = 512;
            break;
        case '?':
            errflg++;
        }
    if (errflg) {
        fprintf (stderr, "usage: . . . ");
        exit (2);
    }
    for( ; optind < argc; optind++) {
        if (access (argv[optind], 4)) {
    .
    .
    .
}
```

NAME

getpass - read a password

SYNOPSIS

```
char *getpass(prompt)
     char *prompt;
```

DESCRIPTION

Getpass reads a password from the file /dev/tty, or if that cannot be opened, from the standard input, after prompting with the null-terminated string prompt and disabling echoing. A pointer is returned to a null-terminated string of at most eight characters.

FILES

/dev/tty

SEE ALSO

crypt(3).

LIMITATIONS

The return value points to static data whose content is overwritten by each call.

NAME

getpw - get name from UID

SYNOPSIS

```
getpw(uid, buf)
char *buf;
int uid;
```

DESCRIPTION

Getpw searches the password file for the (numerical) *uid*, and fills in *buf* with the corresponding line; it returns nonzero if *uid* cannot be found. The line is null-terminated.

FILES

/etc/passwd

SEE ALSO

getpwent(3), passwd(5).

DIAGNOSTICS

Nonzero return on error.

NAME

getpwent, getpwuid, getpwnam, setpwent, endpwent - get password file entry

SYNOPSIS

```
#include <pwd.h>

struct passwd *getpwent();

struct passwd *getpwuid(uid)
int uid;

struct passwd *getpwnam(name)
char *name;

int setpwent();

int endpwent();
```

DESCRIPTION

Getpwent, getpwuid, and **getpwnam** each return a pointer to an object with the following structure containing the broken-out fields of a line in the password file.

```
struct  passwd { /* see getpwent(3) */
        char    *pw_name;
        char    *pw_passwd;
        int     pw_uid;
        int     pw_gid;
        char    pw_age;
        char    *pw_comment;
        char    *pw_gecos;
        char    *pw_dir;
        char    *pw_shell;
};
```

The fields pw_comment, and pw_gecos are unused; the others have meanings described in **passwd**(5).

Getpwent reads the next line after opening the file if necessary; **setpwent** rewinds the file; **endpwent** closes it.

Getpwuid and **getpwnam** search from the beginning until a matching uid or name is found, or until EOF is encountered.

FILES

/etc/passwd

SEE ALSO

getlogin(3), getgrent(3), passwd(5).

DIAGNOSTICS

Null pointer (Ø) returned on EOF or error.

LIMITATIONS

All information is contained in a static area and must be copied if it is to be saved.

NAME

gets, fgets - get a string from a stream

SYNOPSIS

```
#include <stdio.h>

char *gets(s)
char *s;

char *fgets(s, n, stream)
char *s;
FILE *stream;
int n;
```

DESCRIPTION

Gets reads a string into s from the standard input stream **stdin.** The string is terminated by a new line character, that is replaced in s by a null character. **Gets** returns its argument.

Fgets reads n-1 characters, or up to a new line character, whichever comes first, from the stream into the string s. The last character read into s is followed by a null character. Fgets returns its first argument.

SEE ALSO

puts(3), getc(3), scanf(3), fread(3), ferror(3).

DIAGNOSTICS

Gets and **fgets** return the constant pointer **NULL** upon end of file or error.

LIMITATIONS

Gets deletes a new line, and **fgets** keeps it.

NAME

goodmagic - determine magic number validity

SYNOPSIS

```
extern int swap_flg;

goodmagic(magic);
int magic;
```

DESCRIPTION

Goodmagic determines if magic is a valid **a.out**(5) magic number. If the value is not valid, Ø is returned. If magic appears to be a byte-swapped magic number, swap_flg is non-zero. If magic is valid, the value returned is in the range 1 through 6. Values 1 through 3 are nonsegmented magic numbers N_MAGIC1, N_MAGIC3, and N_MAGIC4 respectively; values 4 through 6 are returned for segmented magic numbers S_MAGIC1, S_MAGIC3, and S_MAGIC4.

SEE ALSO

a.out(5).

DIAGNOSTICS

Goodmagic returns Ø if magic is not a valid **a.out**(5) magic number.

NAME

hypot, cabs - euclidean distance

SYNOPSIS

```
#include <math.h>

double hypot(x, y)
     double x, y;

double cabs(z)
     struct { double x, y;} z;
```

DESCRIPTION

Hypot and **cabs** return

$$\mathrm{sqrt}(x*x + y*y),$$

taking precautions against unwarranted overflows.

SEE ALSO

exp(3).

NAME

isaddindex - add an index to a C-ISAM file

SYNOPSIS

```
#include <isam.h>

isaddindex(isfd, keydesc)
int isfd;
struct keydesc *keydesc;
```

DESCRIPTION

Isadddindex adds an index to the C-ISAM file indicated by the isfd parameter. The index is defined in the keydesc structure. This call executes only if the C-ISAM file has been opened for exclusive access, and for both input and output.

There is no limit to the number of indices added through the isaddindex call. However, the maximum number of parts defined for an index is eight. Also, the maximum number of bytes in an index is 118.

FILES

/usr/include/isam.h
/usr/lib/libcisam.a

SEE ALSO

isbuild(3), isdelindex(3), isindexinfo(3), isopen(3).
C-Isam Programmer's Guide in the Zeus Languages/Programming Tools Manual

NAME

isaudit - audit trail maintenance for a C-ISAM file

SYNOPSIS

```
#include <isam.h>

isaudit(isfd, filename, mode)
int  isfd;
char *filename;
int  mode;
```

DESCRIPTION

Isaudit is used to perform operation relative to the audit trail of a C-ISAM file.

Before an audit trail can be started, the name of the audit trail file must be recorded by C-ISAM. It is recorded with the mode parameter equalling AUDSETNAME and with the filename parameter pointing to the name of the audit-trail file. The name can be any valid pathname, up to 64 bytes long. This mode can be used to change the name of the audit trail file, and create an audit trail file with a new name, if the current audit trail is inactive (see below).

Once this audit trail file name has been recorded in C-ISAM, the user can activate and deactivate the audit trail at will by using the values AUDSTART and AUDSTOP for the mode parameter, respectively. (The filename parameter is ignored in both these cases.) When the audit trail is active, all modifications will be written to the audit trail file, even if the modifications are generated by a different process! It will remain active until it is deactivated, even after the originating process is terminated.

If an audit trail file is deactivated, and later activated again, any new modifications to the file will be appended to the existing audit trail data from the previous period of activation.

If the mode parameter equals AUDINFO, the isaudit call is used to query the status of the audit trail. If the first byte of the record array is (hex) Ø1, then the audit trail is active; if it is (hex) ØØ, then the audit trail is not active.

The mode parameter can be set to AUDGETNAME to have C-ISAM return the name of the audit trail file (specified by isfd), presuming it has been previously set. The name will be returned in the filename parameter.

The format of the audit trail file is as follows:

A one character code (see below)
The date and time in long format (see **time**(2))
The process id (int)
The user id (int)
The contents of the record

The code is as follows:

A - addition of a record
D - deletion of a record
R - results of a read
W - record being written

The audit trail file is needed to store all modifications to the file. Thus an isrewcurr call will generate two entries in the audit trail file: an entry with an 'R' code giving the record before it was updated, and an entry with a 'W' code, giving the new record that replaced it.

FILES

/usr/include/isam.h
/usr/lib/libcisam.a

SEE ALSO

C-Isam Programmer's Guide in the Zeus Languages/Programming Tools Manual

NAME

isbuild - define a C-ISAM file

SYNOPSIS

```
#include <isam.h>

isbuild (filename, recordlength, keydesc, mode)
char *filename;
int recordlength;
struct keydesc *keydesc;
int mode;
```

DESCRIPTION

Isbuild defines and creates a C-ISAM file. It causes two operating system files to be created and initialized. One of these two files, named by appending .dat to the filename parameter, will hold the data. The other, named by appending ".idx" to the filename parameter, will hold the dictionary and the indices.

The filename parameter should point to a null-terminated character string no more than ten characters long.

After **isbuild** has completed successfully, the C-ISAM file will remain open for further processing. The **isbuild** function returns a C-ISAM file descriptor, used for all operations on the open file.

The keydesc points to a structure containing a description of the file's primary index. Alternate indices may be added later using **isaddindex.**

Mode specifies the user's access intentions (input, output, or both) and the user's locking requirements (exclusive, manual, or automatic). The mode will be ISEXCLLOCK, ISMANULOCK, or ISAUTOLOCK arithmetically added to ISINPUT, ISOUTPUT, or ISINOUT.

LIMITATIONS

Filenames are limited to ten or less characters.

FILES

/usr/include/isam.h
/usr/lib/libcisam.a

SEE ALSO

isaddindex(3), isindexinfo(3), isopen(3).
C-Isam Programmer's Guide in the Zeus Languages / Programming Tools Manual

NAME

isclose - close a C-ISAM file

SYNOPSIS

```
#include <isam.h>

isclose(isfd)
int iisfd;
```

DESCRIPTION

Isclose is used to close a C-ISAM file. Any locks that are held for the file by the process issuing the isclose call are released.

FILES

/usr/include/isam.h
/usr/lib/libcisam.a

SEE ALSO

isbuild(3), islock(3), isopen(3).
C-Isam Programmer's Guide in the Zeus Languages / Programming Tools Manual

NAME

isdelete, isdelcurr - delete the current record from a C-ISAM file

SYNOPSIS

```
#include <isam.h>

isdelete(isfd, record)
int isfd;
char record[];

isdelcurr(isfd)
int isfd;
```

DESCRIPTION

Both **isdelete** and **isdelcurr** delete records from C-ISAM files. **Isdelcurr** will delete the current record. **Isdelete** will locate the specified record and delete it. In order to use **isdelete** the primary key must be unique. The primary key value of the desired record must be placed at their correct positions in the the I/O buffer pointed to by the record parameter.

To use **isdelcurr** the record must previously have been located with an **isread** or **isstart**

Isfd is the C-ISAM file descriptor returned when the file was opened.

The file must be open for both input and output. The appropriate values will be deleted from the index file for each index defined.

FILES

/usr/include/isam.h
/usr/lib/libcisam.a

SEE ALSO

isread(3), isstart(3).
C-Isam Programmer's Guide in the Zeus Languages/Programming Tools Manual

NAME

isdelindex - remove an index from a C-ISAM file

SYNOPSIS

```
#include <isam.h>

isdelindex(isfd, keydesc)
int isfd;
struct keydesc *keydesc;
```

DESCRIPTION

Isdelindex removes an existing index from an C-ISAM file. The index to be removed will be identified by the keydesc structure. All indices may be deleted except the primary index. This call will only execute if the C-ISAM file has been opened for exclusive access, both input and output.

FILES

/usr/include/isam.h
/usr/lib/libcisam.a

SEE ALSO

isaddindex(3).
C-Isam Programmer's Guide in the Zeus Languages/Programming Tools Manual

NAME

iserase - remove a C-ISAM file and any associated audit trail file

SYNOPSIS

```
#include <isam.h>

iserase(filename)
char *filename;
```

DESCRIPTION

Iserase removes the filename.idx and filename.dat files that comprise the C-ISAM file, as well as any audit trail file for the C-ISAM file, if it exists. The file should not be open when it is erased.

FILES

/usr/include/isam.h
/usr/lib/libcisam.a

SEE ALSO

isaudit(3), isbuild(3).
C-Isam Programmer's Guide in the Zeus Languages / Programming Tools Manual

NAME

isindexinfo - access a C-ISAM file's directory information

SYNOPSIS

```
#include <isam.h>

isindexinfo(isfd, buffer, number)
int isfd;
int number;
struct keydesc *buffer;
or
struct dictinfo *buffer;
```

DESCRIPTION

Isindesinfo gives the caller access to information about the C-ISAM file such as information about the defined indexes, their location within the record, their length and whether duplicate values are allowed.

Information about a particular index is obtained by specifying the number of the index using the number parameter. General information such as the number of indexes, index record size, and data record size is obtained by calling **isindexinfo** with the number parameter set to Ø and reading the buffer into a structure of type dictinfo.

Buffer can contain information in the format of either keydesc or dictinfo depending on whether the number parameter is positive or Ø. As indexes are added and deleted the number of a particular index may vary. However, if the number of indexes indicated in dictinfo are examined, review of all of the indexes is guaranteed.

FILES

/usr/include/isam.h
/usr/lib/libcisam.a

SEE ALSO

isaddindex(3), isbuild(3), isdelindex(3).
C-Isam Programmer's Guide in the Zeus Languages / Programming Tools Manual

NAME

lddbl, ldfloat, ldint, ldlong - C-ISAM load routines

SYNOPSIS

```
#include <isam.h>

double lddbl(p)
char  *p;

float  ldfloat(p)
char  *p;

int  ldint(p)
char  *p;

long  ldlong(p)
char  *p;
```

DESCRIPTION

These routines load a byte string into a numeric field. The byte field need not be word-alligned. The parameter p points to the byte string.

These routines require the C-ISAM library.

FILES

/usr/include/isam.h
/usr/lib/libcisam.a

SEE ALSO

isst(3).
C-Isam Programmer's Guide in the Zeus Languages / Programming Tools Manual

NAME

islock - read-lock a C-ISAM file

SYNOPSIS

```
#include <isam.h>

islock(isfd)
int isfd;
```

DESCRIPTION

Islock will lock the entire file that is specified by *isfd*. (*isfd* is the ISAM file descriptor returned by **isbuild** or **isopen** when the file was opened.) This is a "read" lock, other processes will be able to read the file but not update it, as long as they were opened for manual locking, not automatic locking.

FILES

/usr/include/isam.h
/usr/lib/libcisam.a

SEE ALSO

isclose(3), isunlock(3).
C-Isam Programmer's Guide in the Zeus Languages / Programming Tools Manual

ISOPEN

isopen - open a C-ISAM file for processing

SYNOPSIS

```
#include <isam.h>

isopen(filename, mode)
char *filename;
int mode;
```

DESCRIPTION

Isopen opens a C-ISAM file for processing. It returns the C-ISAM file descriptor to be used in subsequent accesses to the C-ISAM file.

The filename parameter must point to a null-terminated string, which is the file name of the C-ISAM file to be processed.

Mode specifies the user's access intentions (input, output, or both) and the user's locking requirements (exclusive, manual, or automatic). The mode will be ISEXCLLOCK, ISMANULOCK, or ISAUTOLOCK arithmetically added to ISINPUT, ISOUTPUT, or ISINOUT.

This call automatically positions the current record pointer to the first record in order of the primary index. If another ordering is desired, the isstart call can be used to select another index, after the file is opened.

FILES

/usr/include/isam.h
/usr/lib/libcisam.a

SEE ALSO

isbuild(3), isclose(3), isstart(3).
C-Isam Programmer's Guide in the Zeus Languages / Programming Tools Manual

NAME

isread - read records from a C-ISAM file

SYNOPSIS

```
#include <isam.h>

isread(isfd, buffer, mode)
int isfd;
char buffer[];
int mode;
```

DESCRIPTION

Isread reads records sequentially or randomly as indicated by the mode parameter.

When sequential processing is desired, mode indicates that the current (ISCURR), first (ISFIRST), last (ISLAST), next (ISNEXT), or previous (ISPREV) record is to be read. The result will be read into the buffer.

When random selection is desired, mode indicates that the record to be returned has a value that is equal to (ISEQUAL), greater than (ISGREAT), or greater than or equal to (ISGTEQ) the specified search value. The search value is placed in the buffer array in the correct byte positions for the key, as defined in the index's keydesc when the index was created. **Isread** will fill in the buffer with the results of the search.

If manual locking was specified when the file was opened, the record can be read-locked before being read, by adding the value ISLOCK to the mode. The record will remain locked until unlocked with the **isrelease** call.

Following **isopen** or **isbuild** calls or an **isstart** call to the start of a file, either **isnext** or **iscurr** will give the first record.

LIMITATIONS

Isread can not be used for partial string searches. To do this, **Isstart** must be used.

FILES

/usr/include/isam.h
/usr/lib/libcisam.a

SEE ALSO

isbuild(3), islock(3), isopen(3), isrelease(3), isstart(3). C-Isam Programmer's Guide in the Zeus Languages / Programming Tools Manual

NAME

isrelease - unlock records in a C-ISAM file

SYNOPSIS

```
#include <isam.h>

isrelease(isfd)
int isfd;
```

DESCRIPTION

Isrelease unlocks records which have been manually locked using the **isread** call. All locked records in the file indicated by isfd will be unlocked.

FILES

/usr/include/isam.h
/usr/lib/libcisam.a

SEE ALSO

isclose(3), islock(3), isread(3).
C-Isam Programmer's Guide in the Zeus Languages / Programming Tools Manual

NAME

isrename - rename a C-ISAM file

SYNOPSIS

```
#include <isam.h>

isrename(oldname, newname)
char *oldname;
char *newname;
```

DESCRIPTION

Isrename will rename the file specified by the oldname parameter to the name specified by the newname parameter. The name parameters must be null terminated strings.

LIMITATIONS

No name may be longer than 10 characters.

FILES

/usr/include/isam.h
/usr/lib/libcisam.a

SEE ALSO

C-Isam Programmer's Guide in the Zeus Languages / Programming Tools Manual

NAME

isrewrite - rewrite a record in a C-ISAM file
isrewcurr - rewrite the current record in a C-ISAM file

SYNOPSIS

```
#include <isam.h>

isrewrite(isfd, record)
int isfd;
char record[];

isrewcurr(isfd, record)
int isfd;
char record[];
```

DESCRIPTION

Isrewrite and isrewcurr change one or more values for a C-ISAM record file. Isrewcurr will rewrite the current record (obtained after an isread or isstart call.) Isrewrite will locate the record and rewrite it.

The buffer pointed to by record will contain the new value for the record. The primary key can not be changed.

In order to call isrewrite the primary key must be unique. The value of the primary key in the buffer pointed to by record is used to locate the record.

Changed secondary index values will cause modification of the appropriate index files.

These calls will not change the position of the current-record pointer.

FILES

/usr/include/isam.h
/usr/lib/libcisam.a

SEE ALSO

isdelete(3).
C-Isam Programmer's Guide in the Zeus Languages / Programming Tools Manual

NAME

stdbl, stfloat, stint, stlong, - C-ISAM store routines

SYNOPSIS

```
#include <isam.h>

stdbl(d, p)
double d;
char  *p;

stfloat(f, p)
float  f;
char  *p;

stint(i, p)
int i;
char  *p;

stlong(l, p)
long l;
char *p;
```

DESCRIPTION

These routines store a numeric value into a byte string. The byte field need not be word-alligned. The parameter p points to the byte string.

They can only be used with C-ISAM.

FILES

/usr/include/isam.h
/usr/lib/libcisam.a

SEE ALSO

isld(3).
C-Isam Programmer's Guide in the Zeus Languages / Programming Tools Manual

NAME

isstart - select the current index and record within an C-ISAM file

SYNOPSIS

```
isstart(isfd, keydesc, length, record, mode)
int isfd;
struct deydesc *keydesc;
int length;
char record[];
int mode;
```

DESCRIPTION

Isstart selects the current index and record to be used for subsequent operations in an C-ISAM file. (When an C-ISAM file is created with **isbuild** or opened with **isopen** the current index is the primary index and the current record is the first record.)

The keydesc is used to designate the desired index. It must be either the primary index, or an index that was previously added using the **isaddindex** call.

The mode will indicate which record should become the current record. If ISFIRST or ISLAST modes are selected, the length and record parameters are not needed. (**Islast** will position on the end of the file following the last record.)

If the current record is to be selected by way of key search, the record buffer must have the search values inserted in it at the correct offsets within the record. In this case, the length parameter contains Ø if the entire key is being used for comparison, or a positive value k if only the first "k" bytes of the key are being compared.

In this case the mode may be ISEQUAL (to find the first record with a key value equal to that supplied in the record buffer), ISGREAT (to find the first record with a key value greater than that supplied in the record buffer), or ISGTEQ (to find the first record with a keyvalue greater or equal to that supplied in the record buffer).

LIMITATIONS

If mode is ISFIRST or ISLAST, the parameters' length and record are unneeded and are not used by the isstart call.

FILES

/usr/include/isam.h
/usr/lib/libcisam.a

SEE ALSO

isbuild(3), isopen(3), isread(3).
C-Isam Programmer's Guide in the Zeus Languages / Programming Tools Manual

NAME

isuniqueid - obtain a unique ID for a C-ISAM file

SYNOPSIS

```
#include <isam.h>

isuniqueid(isfd, uniqueid)
int isfd;
long *uniqueid;
```

DESCRIPTION

Isuniqueid obtains a long integer that is guaranteed to be unique for a C-ISAM file. This is useful if a unique identifier for each record is desired, and the file doesn't have a natural one.

Often a programmer may want to use the calls which require a unique primary index, but may have no reasonably-sized portion of the data record which is guaranteed to be unique. In such a case, a portion of the record could contain the serially-incremented four-byte long integer obtained by isuniqueid. It is the responsibility of the programmer to place the unique long integer in the record buffer using stlong before a write or rewrite call is made.

FILES

/usr/include/isam.h
/usr/lib/libcisam.a

SEE ALSO

C-Isam Programmer's Guide in the Zeus Languages / Programming Tools Manual

NAME

isunlock - unlock a C-ISAM file

SYNOPSIS

```
#include <isam.h>

isunlock(isfd)
int isfd;
```

DESCRIPTION

Isunlock releases an existing file-level lock for the file specified by the file descriptor isfd.

FILES

/usr/include/isam.h
/usr/lib/libcisam.a

SEE ALSO

isclose(3), islock(3).
C-Isam Programmer's Guide in the Zeus Languages / Programming Tools Manual

NAME

iswrite - write a record into an C-ISAM file

SYNOPSIS

```
#include <isam.h>

iswrite(isfd, record)
int isfd;
char record[];
```

DESCRIPTION

Iswrite writes the record passed to it in the record parameter to the C-ISAM file. The appropriate values will be written to the index file for each index that is defined.

Iswrite does not change the position of the current record pointer.

FILES

/usr/include/isam.h
/usr/lib/libcisam.a

SEE ALSO

C-Isam Programmer's Guide in the Zeus Languages / Programming Tools Manual

NAME

j0, j1, jn, y0, y1, yn - bessel functions

SYNOPSIS

```
#include <math.h>

double j0(x)
     double x;

double j1(x)
     double x;

double jn(n, x)
     double x;

double y0(x)
     double x;

double y1(x)
     double x;

double yn(n, x)
     double x;
```

DESCRIPTION

These functions calculate Bessel functions of the first and second kinds for real arguments and integer orders.

DIAGNOSTICS

Negative arguments cause **y0**, **y1**, and **yn** to return a huge negative value and set errno to EDOM.

NAME

l3tol, ltol3 - convert between three-byte integers and long integers

SYNOPSIS

```
l3tol(lp, cp, n)
        long *lp;
        char *cp;
        int n;

ltol3(cp, lp, n)
        char *cp;
        long *lp;
        int n;
```

DESCRIPTION

L3tol converts a list of n three-byte integers packed into a character string pointed to by cp into a list of long integers pointed to by lp.

Ltol3 performs the reverse conversion from long integers (lp) to three-byte integers (cp).

These functions are useful for file system maintenance since disk addresses are three bytes long.

SEE ALSO

filsys(5).

NAME

logname - login name of user

SYNOPSIS

char *logname();

DESCRIPTION

Logname returns a pointer to the null-terminated login name; it extracts the **$LOGNAME** variable from the user's environment.

This routine is kept in **/usr/lib/libPW.a.**

FILES

/etc/profile

SEE ALSO

env(1), login(1), profile(5), environ(7).

NAME

lsearch - linear search and update

SYNOPSIS

```
char *lsearch (key, base, nelp, width, compar)
char *key;
char *base;
int *nelp;
int width;
int (*compar)();
```

DESCRIPTION

Lsearch is a linear search routine generalized from Knuth (6.1) Algorithm Q. It returns a pointer to a table indicating the location where a datum can be found. If the item does not occur, it is added at the end of the table. The first argument, key, is a pointer to the datum to be located in the table. The second argument, base, is a pointer to the base of the table. The third argument, nelp, is the address of an integer containing the number of items in the table. It is incremented if the item is added to the table. The fourth argument, width, is the width of an element in bytes. The last argument, compar, is the name of the comparison routine. It is called with two arguments which are pointers to the elements being compared. The routine must return zero if the items are equal and non-zero otherwise.

LIMITATIONS

Unpredictable events can occur if there is not enough room in the table to add a new item.

SEE ALSO

bsearch(3), qsort(3).

NAME

malloc, free, realloc, calloc - main memory allocator

SYNOPSIS

```
char *malloc(size)
unsigned size;

free(ptr)
char *ptr;

char *realloc(ptr, size)
char *ptr;
unsigned size;

char *calloc(nelem, elsize)
unsigned nelem, elsize;
```

DESCRIPTION

Malloc and free provide a simple general-purpose memory allocation package. **Malloc** returns a pointer to a block of at least size bytes beginning on a word boundary.

The argument to free is a pointer to a block previously allocated by **malloc**; this space is made available for further allocation, but its contents are left undisturbed.

Disorder results if the space assigned by **malloc** is overrun or if some random number is handed to **free**.

Malloc allocates the first sufficiently large contiguous reach of free space found in a circular search from the last block allocated or freed, coalescing adjacent free blocks as it searches. It calls **sbrk**(2) to get more memory from the system when there is no suitable space already free.

Realloc changes the size of the block pointed to by ptr to size bytes and returns a pointer to the block that is possibly moved. The contents is unchanged up to the lesser of the new and old sizes.

Realloc also works if ptr points to a block freed since the last call of **malloc**, **realloc**, or **calloc**; thus sequences of **free**, **malloc**, and **realloc** exploit the search strategy of **malloc** to do storage compaction.

Calloc allocates space for an array of nelem elements of size elsize. The space is initialized to zeros.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

DIAGNOSTICS

Malloc, realloc, and **calloc** return a null pointer (Ø) if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block. Therefore the user should explicitly check return values and abort his program if he receives a null pointer to avoid unexpected results.

LIMITATIONS

When **realloc** returns Ø, the block pointed to by ptr can be destroyed.

NAME

mktemp - make a unique file name

SYNOPSIS

```
char *mktemp(template)
     char *template;
```

DESCRIPTION

Mktemp replaces template by a unique file name, and returns the address of the template. The template is a file name with six trailing X's, that are replaced with the current process ID and a unique letter.

The letters are chosen to avoid file name duplication.

SEE ALSO

getpid(2).

LIMITATIONS

It is possible to run out of letters.

NAME

monitor - prepare execution profile

SYNOPSIS

```
monitor(lowpc, highpc, buffer, bufsize, nfunc)
    int (*lowpc)( ), (*highpc)( );
    short buffer[ ];
    int bufsiz,nfunc;
```

DESCRIPTION

An executable program created by `cc -p' automatically includes calls for **monitor** with default parameters; **monitor** needn't be called explicitly except to gain fine control over profiling.

Monitor is an interface to **profil**(2). Lowpc and highpc are the addresses of two functions; buffer is the address of a (user supplied) array of bufsize short integers. **Monitor** arranges to record a histogram of periodically sampled values of the program counter, and of counts of calls of certain functions, in the buffer. The lowest address sampled is that of lowpc and the highest is just below highpc. At most nfunc call counts can be kept; only calls of functions compiled with the profiling option **-p** of **cc**(1) are recorded. For the results to be significant, especially where there are small, heavily used routines, it is suggested that the buffer be no more than a few times smaller than the range of locations sampled.

To profile the entire program, it is sufficient to use

```
extern etext();
...
monitor((int)2, etext, buf, bufsize, nfunc);
```

Etext lies just above all the program text, see **end**(3).

To stop execution monitoring and write the results on the file mon.out, use

```
monitor(0);
```

then **prof**(1) can be used to examine the results.

FILES

mon.out

SEE ALSO

cc(1), prof(1), profil(2), mon.out(5).

NAME

itom, madd, msub, mult, mdiv, min, mout, pow, gcd, rpow - multiple precision integer arithmetic

SYNOPSIS

```
typedef struct { int len; short *val; } mint;

madd(a, b, c)
msub(a, b, c)
mult(a, b, c)
mdiv(a, b, q, r)
min(a)
mout(a)
pow(a, b, m, c)
gcd(a, b, c)
rpow(a, b, c)
msqrt(a, b, r)
mint *a, *b, *c, *m, *q, *r;

sdiv(a, n, q, r)
mint *a, *q;
int n;
short *r;

mint *itom(n)
int n;
```

DESCRIPTION

These routines perform arithmetic on integers of arbitrary length. The integers are stored using the defined type **mint**. Pointers to a **mint** should be initialized using the function **itom**, which sets the initial value to *n*. After that, space is managed automatically by the routines.

madd, **msub**, **mult**, assign to their third arguments the sum, difference, and product, respectively, of their first two arguments. **mdiv** assigns the quotient and remainder, respectively, to its third and fourth arguments. **sdiv** is like **mdiv** except that the divisor is an ordinary integer. **msqrt** produces the square root and remainder of its first argument. **rpow** calculates *a* raised to the power *b*, while **pow** calculates this reduced modulo *m*. **min** and **mout** do decimal input and output.

The functions are obtained with the loader option **-lmp**.

FILES

/usr/lib/libmp.a
/usr/lib/slibmp.a

DIAGNOSTICS

Illegal operations and running out of memory produce messages and core images.

NAME

nlist - get entries from name list

SYNOPSIS

```
#include <nlist.h>

nlist(filename, nl)
     char *filename;
     struct nlist nl[ ];
```

DESCRIPTION

Nlist examines the name list in the given executable output file and selectively extracts a list of values. The name list consists of an array of structures containing names, types and values. The list is terminated with a null name. Each name is looked up in the name list of the file. If the name is found, the type and value of the name are inserted in the next two fields. If the name is not found, both entries are set to Ø.

The structure is different than the actual symbol table entry for **a.out**(5). The actual structure is:

```
struct nlist {
       char     *nl_name;
       long     nl_value;
       char     nl_type;
};
```

This subroutine is useful for examining the system name list kept in the file **/zeus**. In this way, programs can obtain system addresses that are up to date.

SEE ALSO

a.out(5).

DIAGNOSTICS

All type entries are set to Ø if the file cannot be found or if it is not a valid name list.

NAME

perror, deverr, sys_errlist, sys_nerr, errno - system error messages

SYNOPSIS

```
perror(s)
char *s;

int deverr;
int sys_nerr;
char *sys_errlist[];
int errno;
```

DESCRIPTION

Perror produces a short error message on the standard error file describing the last error encountered during a call to the system from a C program. First, the argument string s is printed, then a colon, then the message and a new line. Most usefully, the argument string is the name of the program that incurred the error. The error number is taken from the external variable errno or deverr (**intro**(2)), that is set when errors occur but not cleared when nonerroneous calls are made.

To simplify variant formatting of messages, the vector of message strings **sys_errlist** is provided; errno or deverr is used as an index in this table to get the message string without the new line. **Sys_nerr** is the number of messages provided for in the table; it must be checked because new error codes can be added to the system before they are added to the table.

SEE ALSO

intro(2).

NAME

popen, pclose - initiate I/O to or from a process

SYNOPSIS

```
#include <stdio.h>

FILE *popen(command, type)
char *command, *type;

int pclose(stream)
FILE *stream;
```

DESCRIPTION

The arguments to **popen** are pointers to null-terminated strings containing respectively a shell command line and an I/O mode, either r for reading or w for writing. It creates a pipe between the calling process and the command to be executed. The value returned is a stream pointer that can write to the standard input of the command or read from its standard output.

A stream opened by **popen** should be closed by **pclose**, which waits for the associated process to terminate and returns the exit status of the command.

Because open files are shared, a type r command is used as an input filter, and a type w as an output filter.

SEE ALSO

pipe(2), fopen(3), fclose(3), system(3), wait(2).

DIAGNOSTICS

Popen returns a null pointer if files or processes cannot be created, or the shell cannot be accessed.

Pclose returns -1 if stream is not associated with a popened command.

LIMITATIONS

Buffered reading before opening an input filter sometimes leaves the standard input of that filter mispositioned. Similar problems with an output filter forestalled by careful buffer flushing, for example, with fflush (**fclose**(3)).

NAME

printf, fprintf, sprintf - output formatters

SYNOPSIS

```
#include <stdio.h>

int printf (format [ , arg ] ... )
char *format;

int fprintf (stream, format [ , arg ] ... )
FILE *stream;
char *format;

int sprintf (s, format [ , arg ] ... )
char *s, *format;
```

DESCRIPTION

The following description pertains to the printf routines normally included by the C compiler. These routines do not completely conform to System 3 specifications: they maintain Version 7 UNIX's interpretation of the capitalized conversion characters **D**, **O**, **U**, and **X** as specifying conversion of long arguments. See printf(3x) for a description of routines that do follow System 3 specifications.

Printf places output on the standard output stream stdout. Fprintf places output on the named output stream. Sprintf places ``output'', followed by the null character (**\Ø**) in consecutive bytes starting at *s; it is the user's responsibility to ensure that enough storage is available. Each function returns the number of characters transmitted (not including the **\Ø** in the case of sprintf), or a negative value if an output error was encountered.

Each of these functions converts, formats, and prints its args under control of the format. The format is a character string that contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which results in fetching of zero or more args. The results are undefined if there are insufficient args for the format. If the format is exhausted while args remain, the excess args are simply ignored.

Each conversion specification is introduced by the character **%**. After the **%**, the following appear in sequence:

Zero or more flags, which modify the meaning of the conversion specification.

An optional decimal digit string specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded on the

left (or right, if the left-adjustment flag (see below) has been given) to the field width;

A precision that gives the minimum number of digits to appear for the **d**, **o**, **u**, or **x** conversions, the number of digits to appear after the decimal point for the **e** and **f** conversions, the maximum number of significant digits for the **g** conversion, or the maximum number of characters to be printed from a string in **s** conversion. The precision takes the form of a period (.) followed by a decimal digit string: a null digit string is treated as zero.

An optional **l** specifying that a following **d**, **o**, **u**, or **x** conversion character applies to a long integer arg. Capitalizing **d**, **o**, **u**, or **x** has the same effect.

A character that indicates the type of conversion to be applied.

A field width or precision may be indicated by an asterisk (*) instead of a digit string. In this case, an integer arg supplies the field width or precision. The arg that is actually converted is not fetched until the conversion letter is seen, so the args specifying field width or precision must appear before the arg (if any) to be converted.

The flag characters and their meanings are:

- The result of the conversion will be left-justified within the field.

+ The result of a signed conversion will always begin with a sign (+ or -).

blank If the first character of a signed conversion is not a sign, a blank will be prepended to the result. This implies that if the blank and + flags both appear, the blank flag will be ignored.

This flag specifies that the value is to be converted to an ``alternate form.'' For **c**, **d**, **s**, and **u** conversions, the flag has no effect. For **o** conversion, it increases the precision to force the first digit of the result to be a zero. For **x** conversion, a non-zero result will have **0x** prepended to it. For **e**, **E**, **f**, **g**, and **G** conversions, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For **g** and G conversions, trailing zeroes will not be removed from the result (which they normally are).

The conversion characters and their meanings are:

d,o,u,x The integer arg is converted to signed decimal, unsigned octal, decimal, or hexadecimal notation respectively. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeroes. The default precision is 1. The result of converting a zero value with a precision of zero is a null string (unless the conversion is **o** or **x** and the **#** flag is present).

f The float or double arg is converted to decimal notation in the style ``[-]ddd.ddd'', where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, 6 digits are output; if the precision is explicitly Ø, no decimal point appears.

e,E The float or double arg is converted in the style ``[-]d.ddd**e**±dd'', where there is one digit before the decimal point and the number of digits after it is equal to the precision; when the precision is missing, 6 digits are produced; if the precision is zero, no decimal point appears. The **E** format code will produce a number with **E** instead of **e** introducing the exponent. The exponent always contains exactly two digits.

g,G The float or double arg is printed in style **f** or **e** (or in style **E** in the case of a **G** format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style e will be used only if the exponent resulting from the conversion is less than -4 or greater than the precision. Trailing zeroes are removed from the result; a decimal point appears only if it is followed by a digit.

c The character arg is printed.

s The arg is taken to be a string (character pointer) and characters from the string are printed until a null character (**\Ø**) is encountered or the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first null character are printed.

% Print a **%**; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by printf and fprintf are printed as if putchar had been called

(see putc(3S)).

EXAMPLES

To print a date and time in the form ``Sunday, July 3, 10:02'', where weekday and month are pointers to null-terminated strings:

```
printf("%s, %s %d,  %.2d:%.2d", weekday, month, day,
hour, min);
```

To print pi to 5 decimal places:

```
printf("pi = %.5f", 4*atan(1.0));
```

SEE ALSO

ecvt(3), putc(3), scanf(3), stdio(3), printf(3x)

NAME

printf, fprintf, sprintf - System 3 output formatters

SYNOPSIS

```
#include <stdio.h>

int printf(format [, arg ] ... )
char *format;

int fprintf(stream,format [, arg ] ... )
FILE *stream;
char *format;

int sprintf(s,format [, arg ] ... )
char *s, format;"
```

DESCRIPTION

The following describes the System 3 version of **printf** and its related routines. These routines are not normally included by the C compiler, which instead uses the routines described in **printf**(3). This System 3 version differs from the other only in its interpretation of the capitalized conversion character **X** and in not allowing the conversion characters **D**, **O**, and **U**.

Printf places output on the standard output stream stdout. **Fprintf** places output on the named output stream. **Sprintf** places ``output'', followed by the null character (**\Ø**) in consecutive bytes starting at ***s**; it is the user's responsibility to ensure that enough storage is available. Each function returns the number of characters transmitted (not including the **\Ø** in the case of **sprintf**), or a negative value if an output error was encountered.

Each of these functions converts, formats, and prints its args under control of the format. The format is a character string that contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which results in fetching of zero or more args. The results are undefined if there are insufficient args for the format. If the format is exhausted while args remain, the excess args are simply ignored.

Each conversion specification is introduced by the character **%**. After the **%**, the following appear in sequence:

Zero or more flags, which modify the meaning of the conversion specification.

An optional decimal digit string specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded on the

left (or right, if the left-adjustment flag (see below) has been given) to the field width;

A precision that gives the minimum number of digits to appear for the **d**, **o**, **u**, **x**, or **X** conversions, the number of digits to appear after the decimal point for the **e** and **f** conversions, the maximum number of significant digits for the **g** conversion, or the maximum number of characters to be printed from a string in **s** conversion. The precision takes the form of a period (.) followed by a decimal digit string: a null digit string is treated as zero.

An optional **l** specifying that a following **d**, **o**, **u**, **x**, or **X** conversion character applies to a long integer arg.

A character that indicates the type of conversion to be applied.

A field width or precision may be indicated by an asterisk (*) instead of a digit string. In this case, an integer arg supplies the field width or precision. The arg that is actually converted is not fetched until the conversion letter is seen, so the args specifying field width or precision must appear before the arg (if any) to be converted.

The flag characters and their meanings are:

- The result of the conversion will be left-justified within the field.

+ The result of a signed conversion will always begin with a sign (+ or -).

blank If the first character of a signed conversion is not a sign, a blank will be prepended to the result. This implies that if the blank and + flags both appear, the blank flag will be ignored.

This flag specifies that the value is to be converted to an ``alternate form.'' For **c**, **d**, **s**, and **u** conversions, the flag has no effect. For **o** conversion, it increases the precision to force the first digit of the result to be a zero. For **x** (**X**) conversion, a non-zero result will have **0x** (**0X**) prepended to it. For **e**, **E**, **f**, **g**, and **G** conversions, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For **g** and G conversions, trailing zeroes will not

be removed from the result (which they normally are).

The conversion characters and their meanings are:

d,o,u,x,X The integer arg is converted to signed decimal, unsigned octal, decimal, or hexadecimal notation (**x** and **X**), respectively; the letters **abcdef** are used for **x** conversion and the letters **ABCDEF** for **X** conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeroes. The default precision is 1. The result of converting a zero value with a precision of zero is a null string (unless the conversion is **o**, **x**, or **X** and the **#** flag is present).

f The float or double arg is converted to decimal notation in the style ``[-]ddd.ddd'', where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, 6 digits are output; if the precision is explicitly Ø, no decimal point appears.

e,E The float or double arg is converted in the style ``[-]d.ddd**e**±dd'', where there is one digit before the decimal point and the number of digits after it is equal to the precision; when the precision is missing, 6 digits are produced; if the precision is zero, no decimal point appears. The **E** format code will produce a number with **E** instead of **e** introducing the exponent. The exponent always contains exactly two digits.

g,G The float or double arg is printed in style f or e (or in style E in the case of a G format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style e will be used only if the exponent resulting from the conversion is less than -4 or greater than the precision. Trailing zeroes are removed from the result; a decimal point appears only if it is followed by a digit.

c The character arg is printed.

s The arg is taken to be a string (character pointer) and characters from the string are printed until a null character (**\Ø**) is encountered or the number of characters indicated by the precision specification is reached. If the precision

is missing, it is taken to be infinite, so all characters up to the first null character are printed.

% Print a **%**; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by **printf** and **fprintf** are printed as if **putchar** had been called (see **putc**(3)).

EXAMPLES

To print a date and time in the form ``Sunday, July 3, 10:02'', where weekday and month are pointers to null-terminated strings:

```
printf("%s, %s %d,  %.2d:%.2d", weekday, month, day,
hour, min);
```

To print pi to 5 decimal places:

```
printf("pi = %.5f", 4*atan(1.0));
```

FILES

/lib/doprnt.o System 3 printf library routine
/lib/cnvt.o System 3 cnvt library routine

SEE ALSO

ecvt(3), putc(3), scanf(3), stdio(3), printf(3).

NAME

putc, putchar, fputc, putw - put character or word on a stream

SYNOPSIS

```
#include <stdio.h>

int putc(c, stream)
char c;
FILE *stream;

putchar(c)

fputc(c, stream)
FILE *stream;

putw(w, stream)
FILE *stream;
```

DESCRIPTION

Putc appends the character c to the named output stream. It returns the character written.

Putchar(c) is defined as **putc**(c, stdout).

Fputc saves on object text and behaves like **putc**, but is a genuine function rather than a macro.

Putw appends word (that is, int) w to the output stream and returns the word written. **Putw** neither assumes nor causes special alignment in the file.

The standard stream stdout is normally buffered if and only if the output does not refer to a terminal; this default is changed by **setbuf**(3). The standard stream stderr is by default unbuffered unconditionally, but use of **freopen** (**fopen**(3)) causes it to become buffered. Setbuf, sets the state to whatever is desired. When an output stream is unbuffered, information appears on the destination file or terminal as soon as written. When it is buffered, many characters are saved up and written as a block. **Fflush** (**fclose**(3)) is used to force the block out early.

SEE ALSO

fopen(3), fclose(3), getc(3), puts(3), printf(3), fread(3).

DIAGNOSTICS

These functions return the constant **EOF** upon error. Since this is a good integer, use **ferror**(3) to detect **putw** errors.

LIMITATIONS

Because it is implemented as a macro, **putc** treats a stream

argument with side effects improperly. In particular putc(c, *f++); does not work.

NAME

putpwent - write password file entry

SYNOPSIS

```
#include <pwd.h>

int putpwent (p, f)
     struct passwd *p;
     FILE *f;
```

DESCRIPTION

Putpwent is the inverse of **getpwent**(3). Given a pointer to a passwd structure created by **getpwent** (or **getpwuid**(3) or **getpwnam**(3)), **putpwent** writes a line on the stream f which matches the format of **/etc/passwd.**

DIAGNOSTICS

Putpwent returns non-zero if an error was detected during its operation, otherwise zero.

NAME

puts, fputs - put a string on a stream

SYNOPSIS

```
#include <stdio.h>

puts(s)
     char *s;

fputs(s, stream)
     char *s;
     FILE *stream;
```

DESCRIPTION

Puts copies the null-terminated string s to the standard output stream stdout and appends a new line character.

Fputs copies the null-terminated string s to the named output stream.

Neither routine copies the terminal null character.

SEE ALSO

fopen(3), gets(3), putc(3), printf(3), ferror(3), fread(3).

LIMITATIONS

Puts appends a new line, but **fputs** does not.

NAME

any, anystr, balbrk, cat, clean_up, curdir, dname, fatal, fdfopen, giveup, imatch, index, lockit, move, patoi , patol, rename, repeat, satoi, seisigl, setsig, sname, strend, substr, trnslat, unlockit, userdir, userexit, username, verify, xalloc, xcreat, xfree, xfreeall, xlink, xmsg, xopen, xpipe, xunlink, xwrite, zero, zeropad - library routines from the PWB library

SYNOPSIS

```
any (c,str)
char c, *str;

anystr (str1,str2)
char *str1, *str2;

balbrk (str,open,clos,end)
char *str, *open, *clos, *end;

char *cat (dest,source1,source2,source3...sourcen,0)
char *dest,  *source1, *source2, *source3, *sourcen;

clean_up()

curdir (path)
char *path;

char *dname(pathname)
char *pathname;

fatal (msg)
char *msg;

FILE *fdfopen(fd,mode)
int fd,mode;

giveup (dump)
int dump;

imatch (prefix,str)
char *prefix, *str;

index (str1,str2)
char *str1, *str2;

lockit (lockfile,count,pid)
char *lockfile;
unsigned count,pid;

char *move(a,b,n)
char *a, *b;
unsigned n;
```

```
patoi (str)
char *str;

long patol (str)
char *str;

rename (oldname,newname)
char *oldname, *newname;

char *repeat (result,str,repfac)
char *result, *str;
unsigned repfac;

char *satoi (str,ip)
char *str;
int *ip;

setsigl()

setsig()

char *sname (str)
char *str;

char *strend (str)
char *str;

char *substr (str,result,origin,len)
char *str, *result;
int origin;
unsigned len;

char *trnslat (str,old,new,result)
char *str, *old, *new, *result;

unlockit (lockfile,pid)
char *lockfile;
unsigned pid;

char *userdir(uid)
int uid;

userexit (code)
int code;

char *username (uid)
int uid;

char *verify (str1,str2)
char *str1, *str2;
```

```
xalloc (size)
unsigned size;

xcreat (name,mode)
char *name;
int mode;

xfree (ptr)
char *ptr;

xfreeall()

xlink (f1,f2)
char *f1, *f2;

xmsg (file,func)
char *file, *func;

xopen(name, mode)
char *name;
int  mode;

xpipe (t)
int *t;

xunlink (f)
char *f;

xwrite (fildes,buffer,nbytes)
int fildes;
char *buffer;
int nbytes;

char *zero(ptr,cnt)
char *ptr;
int cnt;

char *zeropad(str)
char *str;
```

DESCRIPTION

any returns a 1 if character c is equal to any character in the string str; else returns Ø.

anystr
returns the offset (in str1) of the first character matched from str2. If no character matches -1 is returned.

balbrk
finds the offset, in string str, of the first of the characters in the string end occurring outside of a

balanced string. A balanced string contains matched occurrences of any character in the string open and the corresponding character in the string clos. Balanced strings may be nested. In addition to the characters in end, the null character is implicitly an end character. Unmatched members of open or clos result in an error return (a value of -1 is returned).

Example 1:

```
s = "a[bc=2]=3;
o = "({[";
c = ")}]";
e = "=";
```

balbrk(s,o,c,e) returns 7.

Example 2:

```
s = "a[bc=2=3";
```

with o,c, and e as in Example 1, balbrk(s,o,c,e) returns -1

cat concatenates strings. First, string sourcel is copied to string dest. Then subsequent **sourcek** strings are concatenated (by copying) onto the end of dest. The space for dest must be allocated by the caller (i.e. dest is taken to be the address of an area of memory large enough to hold the result). The address of the result (i.e., dest) is returned.

repeat
copies the string str to the string result. Then str is copied repfac -1 times onto the end of result. As with **cat()** (see above), allocation of space for result is the caller's responsibility. Result is returned.

satoi
is similar to **patoi** (see below), except that the integer value is stored through the integer pointer ip, and a pointer to the first non-numeric character encountered is returned.

sname
returns a pointer to the "simple" name of path name str; i.e., a pointer to the first character after the last "/" in str. If str does not contain a "/", a pointer to the original string is returned.

strend

returns a pointer to the end (null byte) of the string str

trnslat
copies string str to string result replacing any character found in string old with the corresponding character from string new; result is returned.

zero sets to zero the area of memory cnt bytes long, starting at address ptr; ptr is returned.

zeropad
replaces initial blanks with "Ø" characters in string str; str is returned.

dname
returns a pointer to the name of the directory that contains the file pointed to by pathname. **Dname** is the complement of **sname()** (see above). If pathname is a simple name (e.g. "file"), a pointer to "." is returned. If pathname is "/zeus", a pointer to "/" is returned. If pathname is "/bin/who", a pointer to "/bin" is returned, etc. The string pointed to by pathname is modified by **dname;** pathname is returned.

imatch
returns 1 if string prefix is a prefix of string str; else returns Ø.

index
returns the offset of the first occurrence of str2 in str1 if string str2 is a substring of string str1, otherwise -1 is returned.

patol
converts an ASCII string to a long integer. The string str is taken to be a string of decimal digits; the numeric value represented by str is returned. Converts positive numbers only. Returns -1 if a non-numeric character is encountered.

move copies the first n characters from string a to string b.

patoi
converts an ASCII string to an integer. the string str is taken to be a string of decimal digits; the numeric value represented by str is returned. Converts positive numbers only. Returns -1 if a non-numeric character is encountered.

substr

copies at most len characters from the string str starting at str[origin] to the string pointed to by result. Sufficient space must exist for that string; result is returned. There is no checking for the reasonableness of the arguments. The copying of str to result stops if either the specified number (i.e., len, which is taken as an unsigned integer) characters have been copied, or if the end of str (i.e., a null byte) is found. A large value of len (e.g., -1) will usually cause all of str to be copied.

verify
checks to see if string str1 contains any characters not in string str2. It returns the offset of the first such character in str1; else returns -1.

Error Handling Routines

The error set of subroutines consist of a general-purpose error handling routine called **fatal()**, and general-purpose signal-setting and signal-catching routines called **setsig()** and **setsigl()**, respectively. There are also two additional routines called **clean_up** and **userexit**, which may be called by **fatal** or **setsigl**. Default versions of these two additional routines are supplied in the library. Users may define their own **clean_up** and **userexit** routines.

The include file, /usr/include/fatal.h contains definitions needed to use **fatal**. These definitions are:

```
extern int   Fflags;
extern char  *Ffile;
extern int   Fvalue;
extern int   (*Ffunc)();
extern ret_buf   Fjmp[10];

# define FTLMSG   0100000
# define FTLCLN    040000
# define FTLFUNC   020000
# define FTLACT       077
# define FTLJMP        02
# define FTLEXIT       01
# define FTLRET         0

# define FSAVE(val)  SAVE(Fflags,old_Fflags): Fflags = val;
# define FRSTR()     RSTR(Fflags,old_Fflags);
```

clean_up
is a default **clean_up** routine provided to resolve external references. It simply returns. User-supplied **clean_up** routines are often used for removing temporary files, etc.

fatal
is a general-purpose error handler. Typically, low-level subroutines that detect error conditions (an open or create routine, for example) return as a value a call of **fatal** with an appropriate message string. For example:

```
return(fatal("can't do it"));
```

Higher-level routines control the execution of **fatal** via the global Fflags. The macros FSAVE() and FRSTR() in "/usr/include/fatal.h" can be used by higher-level subroutines to save and restore the Fflags value.

The argument to **fatal** is a pointer to a message string. The action of **fatal** is driven completely from the Fflags variable, which is interpreted as explained below.

The FTLMSG bit controls the writing of the message on file descriptor 2. The message is preceded by the string "ERROR:", unless the global character pointer Ffile is non-zero, in which case the message is preceded by a string equivalent to:

```
s = sprintf(space, "ERROR [%s]:", Ffile);
```

A new-line character is written after the user-supplied message.

If the FTLCLN bit is on, **clean_up** is called with an argument of Ø (see above).

If the FTLFUNC bit is on, the function pointed to by the global function pointer Ffunc is called with the user-supplied message pointer as an argument. This feature can be used to log these messages.

The FTLACT bits determine how fatal should return. If the FTLJMP bit is one, **longret(Fjmp)** (see **setret**(3)) is called. If the FTLEXIT bit is one the value of userexit(1) is passed as an argument to **exit**(2) (see below). If none of the FTLACT bits is on (the default value for Fflags is Ø), the global variable Fvalue (initialized to -1) is returned.

If all **fatal** globals have their default values, **fatal** simply returns -1.

setsig
sets signals. All signals not already ignored or caught are made to be caught by the signal catching routine **setsig1**

setsigl
catches signals set by setsigl. If a signal other than hangup interrupt or quit is caught, a "user-oriented" **help**(1) message to be printed on file descriptor 2. If hangup, interrupt, or quit is caught, subsequent occurrences of that signal will be ignored. Termination is similar to the FTLCLN and FTLEXIT options of **fatal**, in that **clean_up(sig)** (where sig is the signal number) and **exit(userexit**(1)) are called.

If the file "dump.core" exists in the current directory, the IOT signal is set to Ø and **abort**(3) is called to produce a core dump (after calling **clean_up**, but before calling **userexit**),

userexit(code)
is a default **userexit** routine provided to resolve external references. It returns the value of code. User-supplied **userexit** routines are often used for logging usage statistics.

System Call Interface Routines
These routines provide interfaces to system calls. In addition, the routines process error conditions and call **fatal**.

curdir
places the complete pathname of the current directory in string path. Returns Ø on success, non-zero on failure. On successful return, the current directory is the same as it was on entry; on failure return, the current directory is not known.

fdfopen
provides a file-descriptor interface to the Section 3 input/output routines. The first argument is a file descriptor (from **open**(2), **creat**(2), or **pipe**(2)) and the second is the read/write mode (Ø or 1, respectively). A file pointer (see **fopen**(3)) is returned on success, and NULL on failure (typically, because there are no file structures available).

giveup
changes directory to "/" if argument is Ø,
sets IOT signal to system default (Ø), and
calls **abort**(3).

Thus, if **giveup** is called with a Ø argument, and the file "/core" is not writable (or if the file "/core" doesn't exist, and the directory "/" is not writable), no core dump will be produced.

lockit

lockit
is a process semaphore implemented with files; typically, used to establish exclusive use of a resource (usually a file). The file's name is lockfile. **Lockit** tries count times to create lockfile mode 444. It sleeps 10 seconds between tries. If lockfile is created, the number pid (typically, the process ID of the current process) is written (in binary; i.e., as two bytes) into lockfile, and 0 is returned. If lockfile exists and hasn't been modified within the last 60 seconds, and if it is either empty, or if its first two bytes, interpreted as a binary number, are not the process ID of any existing process, lockfile is removed and **lockit** tries again to make lockfile. After count tries, or if the reason for the creation of lockfile failing is something other than EACCES (see **INTRO**(2)), **lockit** returns -1. See also **unlockit**, below.

rename
renames oldname to be newname; it can be thought of as:

mv oldname newname

It calls **xlink** (see below).

unlockit
is meant to be used to remove a lockfile created by **lockit**. It verifies that the pid specified is contained in the first two bytes of the named lockfile, and then removes the lockfile. If the pids match, and the file is successfully removed, **unlockit** returns 0; otherwise, -1 is returned.

userdir
returns user's login directory name. The argument must be an integer user ID. **Userdir** returns a pointer to the login directory on success, 0 on failure. It remembers its argument and the returned login directory name for subsequent calls.

username
returns user's login name. The argument must be an integer user ID. **Username** returns a pointer to the login name on success, a pointer to the string representation of the user ID on failure. **Logname**(3) is an alternative to this routine.

xalloc, xfree, xfreeall
handle the allocation of and freeing of memory. **Xalloc** and **xfree** are used in the same way as **malloc**(3) and **free**(3). The function **xfreeall()** frees all memory allocated by **xalloc** (it calls **brk**(2)). **Xalloc** returns

value of **fatal** on failure. **Xfree** and **xfreeall** don't return anything. **Xalloc** uses a "first fit" strategy. **Xfree** always coalesces continuous free blocks. **Xalloc** always allocates 2-byte words. **Xalloc** actually allocates one more word than the amount requested. The extra word (the first word of the allocated block) contains the size (in bytes) of the entire block. This size is used by **xfree** to identify continuous blocks, and is used by **xalloc** to implement the first fit strategy. Bad things will happen if that first (size) word is overwritten. Worse things happen if **xfree** is called with a garbage argument.

xcreat
is used in the same way as **creat**(2). **xcreat** requires write permission in the pertinent directory in all cases, and the created file is guaranteed to have the specified mode and be owned by the effective user (**xcreat** does this by first unlinking the file to be created); **xcreat** returns a file descriptor on success, and the value of **fatal** on failure.

xlink
is used in the same way as **link**(2). It is an interface to **link** that handles all error conditions. It returns Ø on success, and the value of **fatal** on failure.

xmsg is used by the other x -routines to generate an error message based on errno (see **INTRO**(2)). It calls **fatal** with the appropriate error message. The second argument is a pointer to the ceiling function's name (a string). There are predefined messages for the most common errors. Other errors cause a message like:

str = sprintf(space, "error = %d, function = %s", errno, funcname)

to be passed to fatal.

xopen
is used in the same way as **open**(2). It is an interface to **open**(2) that handles all error conditions. It returns a file descriptor on success, and the value of **fatal** on failure.

xpipe
is used in the same way as **pipe**(2). It is an interface to **pipe** that handles all error conditions. It returns Ø on success, and the value of **fatal** on failure.

xunlink
is used in the same way as **unlink**(2). It is an

is used in the same way as **unlink**(2). It is an interface to **unlink** that handles all error conditions. It returns Ø on success, and the value of **fatal** on failure.

xwrite
is used in the same way as **write**(2). It is an interface to **write**(2) that handles all error conditions. It returns the number of bytes written on success, and the value of **fatal** on failure.

NAME

qsort - quicker sort

SYNOPSIS

```
qsort(base, nel, width, compar)
   char *base;
   int (*compar)( );
   int nel, width;
```

DESCRIPTION

Qsort is an implementation of the quicker-sort algorithm. The first argument is a pointer to the base of the data; the second is the number of elements; the third is the width of an element in bytes; the last is the name of the comparison routine to be called with two arguments which are pointers to the elements being compared. The routine must return an integer less than, equal to, or greater than Ø according as the first argument is to be considered less than, equal to, or greater than the second.

SEE ALSO

sort(1), bsearch(3), lsearch(3), strcmp(3).

NAME

rand, srand - random number generator

SYNOPSIS

srand(seed)
int seed;

rand( )

DESCRIPTION

Rand uses a multiplicative congruent random number generator with period 28329 to return successive pseudo-random numbers in the range from Ø to 32767.

The generator is reinitialized by calling **srand** with 1 as an argument. It is set to a random starting point by calling **srand** with any argument.

NAME

readsym - read next symbol from name list

SYNOPSIS

```
#include <stdio.h>
#include <s.out.h>

readsym(sym, stream)
struct {
        struct s_nlist bol;
        char   extra[128];
}       *sym;
FILE *stream;
```

DESCRIPTION

Readsym copies the next complete symbol from stream into the supplied sym. Stream is assumed to be positioned at a symbol boundary within the name list of an **a.out**(5) format object module. Sym must be large enough to store the largest possible symbol, which is 127 characters plus control information. **Readsym** returns the number of extra s_nlist slots the symbol occupies. This number ranges between 0 for a symbol of less than eight characters to 9 for a symbol of 127 characters.

SEE ALSO

a.out(5) nlist(3).

DIAGNOSTICS

Readsym returns -1 for read past end of file or invalid data.

NAME

regex, regcmp - regular expression compile/execute

SYNOPSIS

```
char *regcmp(string1[,string2, ...],Ø);
char *string1, *string2, ...;

char *regex(re,subject[,retØ, ...]);
char *re, *subject, *retØ, ...;
```

DESCRIPTION

Regcmp compiles a regular expression and returns a pointer to the compiled form. **Malloc**(3) is used to create space for the vector. It is the user's responsibility to free unneeded space so allocated. A zero return from regcmp indicates an incorrect argument. **Regcmp**(1) has been written to generally preclude the need for this routine at execution time.

Regex executes a compiled pattern against the subject string. Additional arguments are passed to receive values back. **Regex** returns zero on failure or a pointer to the next unmatched character on success. A global character pointer locl points to where the match began. **Regcmp** and **regex** were mostly borrowed from the editor, **ed**(1) however, the syntax and semantics have been changed slightly. The following are the valid symbols and their associated meanings.

[]*.^ These symbols retain their current meaning.

$ Matches the end of the string, **\n** matches the new-line.

- Within brackets the minus means through. For example, **[a-z]** is equivalent to **[abcd...xyz]**. The - can appear as itself only if used as the last or first character. For example, the character class expression **[]-]** matches the characters **]** and -.

+ A regular expression followed by + means one or more times. For example, **[Ø-9]+** is equivalent to **[Ø-9][Ø-9]***.

{m} {m,} {m,u}

Integer values enclosed in **{}** indicate the number of times the preceding regular expression is to be applied. m is the minimum number and u is a number, less than 256, which is the maximum. If only m is present (e.g., {m}), it indicates the exact number of times the regular expression is to be applied. {m,} is analogous to {m,infinity}. The plus (+) and star (*) operations are

equivalent to {1,} and {Ø,} respectively.

(...)$n The value of the enclosed regular expression is to be returned. The value will be stored in the (n+1)th argument following the subject argument. At present, at most ten enclosed regular expressions are allowed. **Regex** makes its assignments unconditionally.

(...) Parentheses are used for grouping. An operator, e.g. *****, **+**, **{}**, can work on a single character or a regular expression enclosed in parenthesis. For example, (a*(cb+)*)$Ø.

By necessity, all the above defined symbols are special. They must, therefore, be escaped to be used as themselves.

EXAMPLES

Example 1:

```
char *cursor, *newcursor, *ptr;
     ...
newcursor = regex((ptr=regcmp("^\n",Ø)),cursor);
free(ptr);
```

This example will match a leading new-line in the subject string pointed at by cursor.

Example 2:

```
char retØ[9];
char *newcursor, *name;
     ...
name = regcmp("([A-Za-z][A-za-zØ-9_]{Ø,7})$Ø",Ø);
newcursor = regex(name,"123Testing321",retØ);
```

This example will match through the string ``Testing3'' and will return the address of the character after the last matched character (cursor+11). The string "Testing3" will be copied to the character array retØ.

Example 3:

```
#include "file.i"
char *string, *newcursor;
     ...
newcursor = regex(name,string);
```

This example applies a precompiled regular expression in **file.i** (see **regcmp**(1)) against string.

FILES

/usr/lib/libPW.a
/usr/lib/slibPW.a

SEE ALSO

ed(1), regcmp(1), free(3), malloc(3).

LIMITATIONS

The user program may run out of memory if **regcmp** is called iteratively without freeing the vectors no longer required. The following user-supplied replacement for malloc(3) reuses the same vector saving time and space:

```
/* user's program */
     ...
malloc(n) {
static int rebuf[256];
     return &rebuf;
}
```

NAME

scanf, fscanf, sscanf - formatted input conversion

SYNOPSIS

```
#include <stdio.h>

scanf (format [ , pointer ] . . . )
char *format;

fscanf (stream, format [ , pointer ] . . . )
    FILE *stream;
    char *format;

sscanf (s, format [ , pointer ] . . . )
    char *s, *format;
```

DESCRIPTION

Scanf reads from the standard input stream stdin. **Fscanf** reads from the named input stream. Sscanf reads from the character string s. Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects as arguments a control string format and a set of pointer arguments indicating where the converted input will be stored.

The control string usually contains conversion specifications that are used to direct interpretation of input sequences. The control string contains:

1. Blanks, tabs, or new lines that match optional white space in the input.

2. An ordinary character (not %) that must match the next character of the input stream.

3. Conversion specifications, consisting of the character %, an optional assignment suppressing character *, an optional numerical maximum field width, and a conversion character.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by *. An input field is defined as a string of nonspace characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. The following conversion characters are legal:

% a single % is expected in the input; no assignment is done.

d a decimal integer is expected; the corresponding argument must be an integer pointer.

o an octal integer is expected; the corresponding argument must be a integer pointer.

x a hexadecimal integer is expected; the corresponding argument must be an integer pointer.

s a character string is expected; the corresponding argument must be a character pointer pointing to an array of characters large enough to accept the string and a terminating \Ø, which will be added. The input field is terminated by a space character or a new line.

c a character is expected; the corresponding argument should be a character pointer. The normal skip over space characters is suppressed in this case; to read the next nonspace character, use %1s. If a field width is given, the corresponding argument must refer to a character array, and the indicated number of characters is read.

f a floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a float. The input format for floating point numbers is specified under atof(3).

e same as f

[indicates a string not to be delimited by space characters. The left bracket is followed by a set of characters and a right bracket; the characters between the brackets define a set of characters making up the string. If the first character is not circumflex (^), the input field is all characters until the first character not in the set between the brackets. If the first character after the left bracket is ^, the input field is all characters until the first character that is in the remaining set of characters between the brackets. The corresponding argument must point to a character array.

The conversion characters **d**, **o**, and **x** can be capitalized or preceeded by **l** to indicate which a pointer to long rather than to **int** is in the argument list. Similarly, the

conversion characters **e** or **f** can be capitalized or preceded by **l** to indicate a pointer to **double** rather than to **float.** The conversion characters **d**, **o** and **x** can be preceded by **h** to indicate a pointer to **short** rather than to **int.**

The **scanf** functions return the number of successfully matched and assigned input items. This is used to decide how many input items were found. The constant **EOF** is returned upon end of input. This is different from Ø, which means that no conversion was done; if conversion was intended, it was frustrated by an inappropriate character in the input.

For example, the call

```
int i; float x; char name[50];
scanf( "%d%f%s", &i, &x, name);
```

with the input line

```
25   54.32E-1  thompson
```

assigns to i the value 25, x the value 5.432, and name contains thompson\Ø. Or,

```
int i; float x; char name[50];
scanf("%2d%f%*d%[1234567890]", &i, &x, name);
```

with input

```
56789 0123 56a72
```

assigns 56 to i, 789.Ø to x, skip Ø123, and place the string 56\Ø in name. The next call to **getchar** returns a.

SEE ALSO

atof(3), getc(3), printf(3).

DIAGNOSTICS

The **scanf** functions return **EOF** on end of input, and a short count for missing or illegal data items.

LIMITATIONS

The success of literal matches and suppressed assignments is not directly determinable.

NAME

getkey, gonormal, goraw, wbackspace, wbackword, wcolon, wforspace, wforword, wgetword, whelp, whighlight, wleft, wmesg, wmvcursor, wpageback, wpagefor, wresscrn, wright, wsavescrn, wscrolb, wscrolf - Screen Interface Library

SYNOPSIS

```
#include <curses.h>
#include <screen.h>

getkey()

gonormal()

goraw()

wbackspace(win, top, bottom)
     WINDOW *win;
     int top, bottom;
or
backspace(top, bottom)
     int top, bottom;

wbackword(win, top, bottom)
     WINDOW *win;
     int top, bottom;
or
backword(top, bottom)
     int top, bottom;

wcolon(win)
     WINDOW *win;
or
colon()

wforspace(win, top, bottom)
     WINDOW *win;
     int top, bottom;
or
forspace(top, bottom)
     int top, bottom;

wforword(win, top, bottom)
     WINDOW *win;
     int top, bottom;
or
forword(top, bottom)
     int top, bottom;

wgetword(win, str)
     WINDOW *win;
     char *str;
```

```
or
getword(str)
     char *str;

whelp(win, file)
     WINDOW *win;
     char *file;
or
help(file)
     char *file;

whighlight(win, flag)
     WINDOW *win;
     int flag;
or
highlight(flag)
     int flag;

wleft(win, top, bottom)
     WINDOW *win;
     int top, bottom;
or
left(top, bottom)
     int top, bottom;

wmesg(win, str, data)
     WINDOW *win;
     char *str;
     char *data;
or
mesg(str, data)
     char *str;
     char *data;

wmvcursor(win, c, top, bottom)
     WINDOW *win;
     char c;
     int top, bottom;
or
mvcursor(c, top, bottom)
     char c;
     int top, bottom;

wpageback(win)
     WINDOW *win;
or
pageback()

wpagefor(win, fp, top)
     WINDOW *win;
     FILE *fp;
     int top;
```

```
or
pagefor(fp, top)
     FILE *fp;
     int top;

wresscrn(win)
     WINDOW *win;
or
resscrn()

wright(win, top, bottom)
     WINDOW *win;
     int top, bottom;
or
right(top, bottom)
     int top, bottom;

wsavescrn(win)
     WINDOW *win;
or
savescrn()

wscrolb(win)
     WINDOW *win;
or
scrolb()

wscrolf(win)
     WINDOW *win;
or
scrolf()
```

DESCRIPTION

The Screen Interface Library is designed to aid the programmer in writing "screen-oriented" or "display" programs. This set of routines uses the terminal capability data base (/etc/termcap) for terminal independence and the "Curses" library (/usr/lib/libcurses.a for nonsegmented programs and /usr/lib/slibcurses.a for segmented programs) for screen updating functions.

The major features of the library include terminal setup, cursor manipulation using the arrow keys, single character input, "highlighting" of the cursored item, paging, scrolling, saving and restoring displays, obtaining the word on which the cursor lies, handling "help" files, and general "last line" handling. Programmers familiar with the "Curses" library will recognize the "window-orientation" of these routines so that displays are not limited to the standard terminal screen.

When using the Screen Interface Library, the user program must include the header files,

```
#include <curses.h>
#include <screen.h>
```

The command lines for nonsegmented and segmented C compilation are:

```
cc file.c -lscreen -lcurses -ltermlib
scc file.c -lscreen -lcurses -ltermlib
```

The remainder of this section contains descriptions of the routines available in the Screen Interface Library; if applicable, there are two calling sequences available (one for a "window" and one for "stdscr"):

getkey()

This routine gets a single character input from the keyboard. If any of the arrow keys is typed, the standard definition found in /usr/include/screen.h (i.e. LEFT, DOWN, UP, and RIGHT) is returned. The following list contains the aliases for the arrow keys:

left - h, CTRL-h, backspace

down - j, CTRL-j

up - k, CTRL-k

right - l, CTRL-l, space

Alternatively, if none of the arrow keys is typed, the character typed is returned. In addition, if a carriage return is typed, the character "\r" is returned. This is because some terminals generate a "line feed" or "\n" character for the down arrow; therefore, a distinction must be made between the RETURN key (which is "mapped" to "line feed") and the down arrow key.

gonormal()

This routine resets standard output back to its "normal" mode and resets "_rawmode" back to FALSE (=Ø).

```
goraw()
```

This routine sets standard output for CBREAK mode; in addition, it sets the "Curses" flag, "_rawmode" to TRUE (=1).

```
wbackspace(win, top, bottom)
WINDOW      *win;
int         top, bottom;

or

backspace(top, bottom)
int         top, bottom;
```

This routine moves the cursor to the left until a space is reached on the display. If the cursor is at the leftmost position of the "top" line, the cursor is "wrapped around" to the last word of the rightmost column.

```
wbackword(win, top, bottom)
WINDOW      *win;
int         top, bottom;

or

backword(top, bottom)
int         top, bottom;
```

This routine moves the cursor to the beginning of the previous word (to the left) on the display. If the cursor is at the leftmost position of the "top" line, the cursor is "wrapped around" to the last word of the rightmost column.

```
wcolon(win)
WINDOW      *win;

or

colon()
```

This routine handles the "colon" commands (the colon is echoed on the last line of "win"). A character followed by a carriage return is the expected typein; the routine returns the character typed.

```
wforspace(win, top, bottom)
WINDOW      *win;
int         top, bottom;

or

forspace(top, bottom)
int         top, bottom;
```

This routine moves the cursor to the right until a space is reached on the display. If the cursor is at the "bottom" line of the rightmost column, the cursor is "wrapped around" to the "top" of the leftmost column.

```
wforword(win, top, bottom)
WINDOW      *win;
int         top, bottom;

or

forword(top, bottom)
int         top, bottom;
```

This routine moves the cursor to the beginning of the next word (to the right) on the display. If the cursor is at the "bottom" line of the rightmost column, the cursor is "wrapped around" to the "top" of the leftmost column.

```
wgetword(win, str)
WINDOW      *win;
char        *str;

or

getword(str)
char        *str;
```

This routine gets a word from the display and puts the string in "str"; if the word is "highlighted" in the display, the "standout" mode bit in each character is masked out and returned in "str".

```
whelp(win, file)
WINDOW      *win;
char        *file;
```

or

```
help(file)
char        *file;
```

This routine opens the given help file, "file", and displays its contents; following the display, the original screen is restored. If the help file cannot be opened or if there was a problem restoring the original screen, the routine returns ERR; otherwise, the routine returns OK.

```
whighlight(win, flag)
WINDOW      *win;
int         flag;
```

or

```
highlight(flag)
```

This routine puts the word at the current cursor position in "standout" mode (thus, "highlighting" the word) if "flag" is TRUE. If "flag" is FALSE, "standout" mode is turned off for the word at the current position.

```
wleft(win, top, bottom)
WINDOW      *win;
int         top, bottom;
```

or

```
left(top, bottom)
int         top, bottom;
```

This routine moves the cursor one position to the left. If the cursor is at the leftmost position of the "top" line, the cursor is "wrapped around" to the last word of the rightmost column.

```
wmesg(win, str, data)
WINDOW      *win;
char        *str;
char        *data;
```

or

```
mesg(str, data)
```

```
char        *str;
char        *data;
```

This routine outputs the "printf"-formatted message, "str", on the last line of "win" or "stdscr". It is noted here that a "newline" or "\n" is not required since the message is output on the last line of the window. Any additional data to be output (for example, for a "%s" in "mesg"), is stored in the variable, "data". If there is no additional data (that is, "str" is a simple informational message), "data" should contain NULL. After outputting the message, the cursor is returned to the current position.

```
wmvcursor(win, c, top, bottom)
WINDOW      *win;
char        c;
int         top, bottom;

or

mvcursor(c, top, bottom)
char        c;
int         top, bottom;
```

This routine uses the given character, "c", to move the cursor appropriately about "win" within the display limits of the "top" and "bottom" line. The valid values for "c" (and therefore, valid cursor movements) are DOWN, UP, FORWARD (or WORD) or BACKWARD as defined in /usr/include/screen.h (see APPENDIX B). If the "bottom" line limit is exceeded, the cursor will be moved to the "top" of the next column to the right or to the "top" of the leftmost column; therefore, there is cursor "wraparound". After the movement is performed, the routine returns OK. If "c" represents an invalid cursor movement, the routine returns the value ERR. If the global flag, "hilite", is set, "highlighting" of the word at the current cursor position is handled automatically.

```
wpageback(win)
WINDOW      *win;

or

pageback()
```

This routine outputs the previous "page" of the file associated with the pointer, "fp"; this has not been implemented yet.

```
wpagefor(win, fp, top)
WINDOW      *win;
FILE        *fp;
int         top;
```

or

```
pagefor(fp, top)
FILE        *fp;
int         top;
```

This routine outputs a page of the file associated with the pointer, "fp". If the number of lines in "win" is exceeded, the prompt

Type ^f for next page

is output. If "^f" is not typed, the routine returns; otherwise, the next "page" of data is output.

```
wresscrn(win)
WINDOW      *win;
```

or

```
resscrn()
```

This routine tests the global flag, "scrnflg" (set to TRUE in "wsavescrn") and checks whether the contents of the WINDOW "scrn" will fit on "win". If so, the contents of "scrn" is overwritten onto "win" and the routine returns OK; otherwise, the routine returns ERR.

```
wright(win, top, bottom)
WINDOW      *win;
int         top, bottom;
```

or

```
right(top, bottom)
int         top, bottom;
```

This routine moves the cursor one position to the right. If the cursor is at the "bottom" line of the rightmost column, the cursor is "wrapped around" to the "top" of the leftmost column.

```
wsavescrn(win)
WINDOW      *win;
```

or

```
savescrn()
```

This routine saves the contents of "win" in the global WINDOW, "scrn" (found in /usr/include/screen.h), which is allocated memory in this routine. If there is a problem with the allocation, this routine returns ERR. Otherwise, the global flag "scrnflg" is set to TRUE, the contents of "win" is saved, and the routine returns OK.

```
wscrolb(win)
WINDOW      *win;
```

or

```
scrolb()
```

This routine performs scrolling backward on "win"; this has not been implemented yet.

```
wscrolf(win)
WINDOW      *win;
```

or

```
scrolf()
```

This routine performs scrolling forward on "win"; this has not been implemented yet.

FILES

/usr/lib/libscreen.a /usr/lib/slibscreen.a

SEE ALSO

vls(1), vnews(1), termcap(3), curses(3).

"Screen Updating and Cursor Movement Optimization", "The

Screen Interface Library" in the Zeus Languages / Programming Tools Manual

LIMITATIONS

Routines wscrolf (or scrolf), wscrolb (or scrolb) and wpageback (or pageback) have not been implemented yet.

The termcap capabilities, "kl, kd, ku, kr and kh", are not recognized by the Screen Interface Library at the present time.

NAME

segmon (segmented monitor) - prepare execution profile

SYNOPSIS

```
struct segs {
     char segno;
     unsigned highpc;
     unsigned lowpc;
}

segmon(sbuff,cntsiz)
     struct segs sbuff[ ];
     int cntsiz;
```

DESCRIPTION

An executable program created by `scc -p' automatically includes calls for **segmon** with default parameters; **segmon** needn't be called explicitly except to gain fine control over profiling.

Segmon is an interface to **sprofil**(2). It arranges to record a histogram of periodically sampled values of the program counter, and of counts of calls of certain functions, in a particular segment. Sbuff is a structure which contains information on the segments to be profiled. Segno contains the segment number; highpc and lowpc are the high and low offsets that are within segment segno that make up the sample range. The lowest address sampled is that of lowpc and the highest is just below highpc. At most cntsiz call counts can be kept; only calls of functions compiled with the profiling option **-p** of **scc**(1) are recorded.

To profile a program with multiple segments, sbuff must be filled with the segment information that the user requires. The user may call **sgstat(2)** with the address of sbuff. In turn **sgstat** will fill sbuff with information concerning the text segments (lowpc is set to Ø, highpc is set to size of the text segment). Or the user may define his/her own parameters. Note, that the maximum amount of text segments that can be profiled is 1Ø. In addition to the limit, the segno after the last used segno must be set to Øxff in order to indicate there are no more segments to be profiled. Thus if there are 5 segments to be profiled, sbuff[5].segno must be set to Øxff.

EXAMPLES

An example using sgstat:

```
sgstat(sbuff);
...
segmon(sbuff,cntsiz)
```

To stop execution monitoring and write the results to the file **mon.out**(5), use

```
sbuff[Ø].segno = Øxff;
segmon(sbuff,Ø);
```

then **sprof**(1) can be used to examine the results.

FILES

mon.out

SEE ALSO

mon.out(5), sgstat(2), sprof(1), sprofil(2), scc(1).

NAME

setbuf - assign buffering to a stream

SYNOPSIS

```
#include <stdio.h>

setbuf(stream, buf)
     FILE *stream;
     char *buf;
```

DESCRIPTION

Setbuf is used after a stream has been opened but before it is read or written. It causes the character array buf to be used instead of an automatically allocated buffer. If buf is the constant pointer **NULL**, input/output is completely unbuffered.

A constant **BUFSIZ** tells how big an array is needed:

```
char buf[BUFSIZ];
```

A buffer is normally obtained from **malloc**(3) upon the first **getc** or **putc**(3) on the file. Output streams directed to terminals and the standard error stream stderr are normally not buffered.

SEE ALSO

fopen(3), getc(3), putc(3), malloc(3).

NAME

setret, longret - nonlocal goto

SYNOPSIS

```
#include <setret.h>

setret(env)
        ret_buf env;

longret(env, val)
        ret_buf env;
        int val;
```

DESCRIPTION

These routines deal with errors and interrupts encountered in a low-level subroutine of a program.

Setret saves its stack environment in env for later use by **longret.** It returns value Ø.

Longret restores the environment saved by the last call of **setret.** It then returns so that execution continues as if the call of **setret** had just returned the value val to the function that invoked **setret,** which must not return in the interim. All accessible data have values as of the time **longret** was called.

Longret and **setret** are like longjmp and setjmp routines on other V7 Unix systems. The only difference is that there must not be any declaration of register variables in the routine that calls **setret.**

SEE ALSO

signal(2).

NAME

sin, cos, tan, asin, acos, atan, atan2 - trigonometric functions

SYNOPSIS

```
#include <math.h>

double sin(x)
double x;

double cos(x)
double x;

double asin(x)
double x;

double acos(x)
double x;

double atan(x)
double x;

double atan2(x, y)
double x, y;
```

DESCRIPTION

Sin, cos and **tan** return trigonometric functions of radian arguments. The magnitude of the argument should be checked by the caller to make sure the result is meaningful.

Asin returns the arc sin in the range -pi/2 to pi/2.

Acos returns the arc cosine in the range 0 to pi.

Atan returns the arc tangent of x in the range -pi/2 to pi/2.

Atan2 returns the arc tangent of x/y in the range -pi to pi.

DIAGNOSTICS

Arguments of magnitude greater than 1 cause **asin** and **acos** to return value 0; errno is set to EDOM. The value of **tan** at its singular points is a huge number, and errno is set to ERANGE.

LIMITATIONS

The value of **tan** for arguments greater than about 2**31 is garbage.

NAME

sinh, cosh, tanh - hyperbolic functions

SYNOPSIS

```
#include <math.h>

double sinh(x)
double x;

double cosh(x)
double x;

double tanh(x)
double x;
```

DESCRIPTION

These functions compute the designated hyperbolic functions for real arguments.

DIAGNOSTICS

Sinh and **cosh** return a huge value of appropriate sign when the correct value would overflow.

NAME

sleep - suspend execution for interval

SYNOPSIS

sleep(seconds)
unsigned seconds;

DESCRIPTION

The current process is suspended from execution for the number of seconds specified by the argument. The actual suspension time can be up to 1 second less than that requested, because scheduled wakeups occur at fixed 1-second intervals, and an arbitrary amount longer because of other activity in the system.

The routine is implemented by setting an alarm clock signal and pausing until it occurs. The previous state of this signal is saved and restored. If the sleep time exceeds the time to the alarm signal, the process sleeps only until the signal would have occurred, and the signal is sent 1 second later.

SEE ALSO

alarm(2), pause(2).

NAME

ssignal, gsignal - software signals

SYNOPSIS

```
#include <signal.h>

int (*ssignal (sig, action))( )
    int sig, (*action)( );

int gsignal (sig)
    int sig;
```

DESCRIPTION

and implement a software facility similar to This facility is used by the standard C Library to enable the user to indicate the disposition of error conditions, and is also made available to the user for his own purposes.

Software signals made available to users are associated with integers in the inclusive range 1 through 15. An action for a software signal is established by a call to **ssignal**, and a software signal is raised by a call to **gsignal**. Raising a software signal causes the action established for that signal to be taken.

The first argument to **ssignal** is a number identifying the type of signal for which an action is to be established. The second argument defines the action; it is either the name of a (user defined) action function or one of the manifest constants **SIG_DFL** (default) or **SIG_IGN** (ignore). **Ssignal** returns the action previously established for that signal type; if no action has been established or the signal number is illegal, **ssignal** returns **SIG_DFL**.

Gsignal raises the signal identified by its argument, sig:

If an action function has been established for sig, then that action is reset to **SIG_DFL** and the action function is entered with argument sig. **Gsignal** returns the value returned to it by the action function.

If the action for sig is **SIG_IGN**, **gsignal** returns the value 1 and takes no other action.

If the action for sig is **SIG_DFL**, **gsignal** returns the value Ø and takes no other action.

If sig has an illegal value or no action was ever specified for sig, gsignal returns the value Ø and takes no other action.

NOTES

There are some additional signals with numbers outside the range 1 through 15 which are used by the standard C Library to indicate error conditions. Thus, some signal numbers outside the range 1 through 15 are legal, although their use may interfere with the operation of the standard C Library.

NAME

stdio - standard buffered input/output package

SYNOPSIS

#include <stdio.h>

```
FILE *stdin;
FILE *stdout;
FILE *stderr;
```

DESCRIPTION

The functions declared in the **#include** file, **<stdio.h>** constitute an efficient user-level buffering scheme. The inline macros **getc** and **putc**(3) handle characters quickly. The higher level routines **gets, fgets, scanf, fscanf, fread, puts, fputs, printf, fprintf, fwrite fgetc, fputc, getw,** and **putw** all use **getc** and **putc;** they can be freely intermixed.

A file with associated buffering is called a stream, and is declared to be a pointer to a defined type **FILE. Fopen**(3) creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. There are three normally open streams with constant pointers declared in the include file and associated with the standard open files:

stdin	standard input file
stdout	standard output file
stderr	standard error file

A constant pointer **NULL** (Ø) designates no stream at all.

An integer constant **EOF** (-1) is returned upon end of file or error by integer functions that deal with streams.

Any routine that uses the standard input/output package must include the header file <stdio.h> of pertinent macro definitions. The functions and constants are declared in the include file and need no further declaration. The constants and the following functions are implemented as macros; redeclaration of these names is perilous: **getc, getchar, putc, putchar, feof, ferror, fileno.**

SEE ALSO

open(2), close(2), read(2), write(2).

DIAGNOSTICS

The value **EOF** is returned uniformly to indicate that a **FILE** pointer has not been initialized with **fopen,** input (output) has been attempted on an output (input) stream, or a **FILE** pointer designates corrupt or otherwise unintelligible **FILE** data.

NAME

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok, index, rindex - string operations

SYNOPSIS

```
char *strcat (s1, s2)
char *s1, *s2;

char *strncat (s1, s2, n)
char *s1, *s2;
int n;

int strcmp (s1, s2)

char *s1, *s2;

int strncmp (s1, s2, n)
char *s1, *s2;
int n;

char *strcpy (s1, s2)
char *s1, *s2;

char *strncpy (s1, s2, n)
char *s1, *s2;
int n;

int strlen (s)
char *s;

char *strchr (s, c)
char *s, c;

char *strrchr (s, c)
char *s, c;

char *strpbrk (s1, s2)
char *s1, *s2;

int strspn (s1, s2)
char *s1, *s2;

int strcspn (s1, s2)
char *s1, *s2;

char *strtok (s1, s2)
char *s1, *s2;

char *index(s, c)
char *s, c;
```

char *rindex(s, c)
char *s;

DESCRIPTION

These functions operate on null-terminated strings. They do not check for overflow of any receiving string.

Strcat appends a copy of string s2 to the end of string s1. **Strncat** copies at most n characters. Both return a pointer to the null-terminated result.

Strcmp compares its arguments and returns an integer greater than, equal to, or less than 0, according as s1 is lexicographically greater than, equal to, or less than s2. **Strncmp** makes the same comparison but looks at at most n characters.

Strcpy copies string s2 to s1, stopping after the null character has been moved. **Strncpy** copies exactly n characters, truncating or null-padding s2; the target may not be null-terminated if the length of s2 is n or more. Both return s1.

Strlen returns the number of non-null characters in s.

Strchr (**strrchr**) returns a pointer to the first (last) occurrence of character c in string s, or **NULL** if c does not occur in the string. The null character terminating a string is considered to be part of the string.

Strpbrk returns a pointer to the first occurrence in string s1 of any character from string s2, or **NULL** if no character from s2 exists in s1.

Strspn (**strcspn**) returns the length of the initial segment of string s1 which consists entirely of characters from (not from) string s2.

Strtok considers the string s1 to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string s2. The first call (with pointer s1 specified) returns a pointer to the first character of the first token, and will have written a **NULL** character into s1 immediately following the returned token. Subsequent calls with zero for the first argument, will work through the string s1 in this way until no tokens remain. The separator string s2 may be different from call to call. When no token remains in s1, a **NULL** is returned.

Index (rindex) returns a pointer to the first (last) occurrence of character c in string s, or zero if c does not occur in the string.

LIMITATIONS

Strcmp uses native character comparison, which is signed on PDP-11s, unsigned on other machines.

All string movement is performed character by character starting at the left. Thus overlapping moves toward the left will work as expected, but overlapping moves to the right may yield surprises.

NAME

swab - swap bytes

SYNOPSIS

```
swab(from, to, nbytes)
     char *from, *to;
     int nbytes;
```

DESCRIPTION

Swab copies nbytes bytes pointed to by from to the position pointed to by to, exchanging adjacent even and odd bytes. It is useful for carrying binary data between S8000 and PDP-11. Nbytes should be even.

NAME

swap - swap routines - swap
longswap - swap routines - swap
swapsegt - swap routines - swap

SYNOPSIS

```
#include <a.out.h>

int val;
val = swap (val);

long val;
longswap (&val);

struct segt s_segt;
swapsegt (&s_segt);
```

DESCRIPTION

Swap is used as a function to swap both bytes of an integer val.

Longswap is given the address of a long value (val) and swaps both of its words.

Swapsegt is given the address to a segment table entry (s_segt) and swaps its word values. A segment table entry has the following structure:

```
struct segt {
     char sg_segno; /* segment number */
     char sg_coff;  /* offset/256 for code section */
     char sg_doff;  /* offset/256 for data section */
     char sg_boff;  /* offset/256 for bss section */
     unsigned sg_code;   /* size of the code portion */
     unsigned sg_data;   /* size of the data portion */
     unsigned sg_bss;    /* size of the bss portion */
     int sg_atr;    /* attributes */
     long sg_unused;     /* unused */
}
```

These routines are generally used in conjunction with swap_flg in order to determine whether bytes must be swapped for the PDP11.

SEE ALSO

a.out(5), goodmagic(3).

NAME

symscan - scan name list

SYNOPSIS

```
#include <stdio.h>
#include <nlist.h>

symscan(nl, stream)
    struct nlist *nl;
    FILE *stream;
```

DESCRIPTION

Symscan searches stream for the next occurrence of the symbol identified by nl. Nl is one element of the same structure used by **nlist**(3). One field of this structure points to a null-terminated string that defines the search object. If the symbol is found, the other fields of the structure are set appropriately. If the symbol is not found, the nl type field is set to Ø. Unlike **nlist**, **symscan** returns both external and nonexternal symbols. If the program finds a local symbol of the same name, **symscan** can be called again to find the next occurrence of a symbol with the same name. Stream is assumed to be positioned at a symbol boundary within the name list of an **a.out**(5) format object module.

SEE ALSO

a.out(5), nlist(3), readsym(3).

DIAGNOSTICS

Symscan returns nl type field zero if the symbol was not found, or if the input was invalid.

NAME

SYS - system call relay program

SYNOPSIS

execl ("/bin/SYS", "SYS", Ø);

DESCRIPTION

The **SYS** program runs on a host concurrently with a program on a satellite Development Module (DM). The program running on the DM must have been downloaded by load(1), that executes **SYS** as the last step of the loading process.

SYS runs even if the satellite system calls are not utilized. It is terminated by a normal exit from the downloaded program.

SEE ALSO

load(1), exec(2).

LIMITATIONS

Abnormal termination of a program leaves **SYS** still running on the host. In this case, RUBOUT stops the execution of **SYS** on the host.

NAME

system - issue a shell command

SYNOPSIS

system(string)
char *string;

csystem(string)
char *string;

DESCRIPTION

System causes the string to be given to **sh**(1) as input as if the string had been typed as a command at a terminal. The current process waits until the shell has completed, then returns the exit status of the shell.

Csystem cause the string to be give to **csh**(1) instead of the shell. The current process waits until the cshell has complete, then returns the exit status in the same manner as the shell.

SEE ALSO

popen(3), exec(2), wait(2).

DIAGNOSTICS

Exit status 127 indicates the shell could not be executed.

NAME

tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs - terminal independent operation routines

SYNOPSIS

```
char PC;
char *BC;
char *UP;
short ospeed;

tgetent(bp, name)
char *bp, *name;

tgetnum(id)
char *id;

tgetflag(id)
char *id;

char *
tgetstr(id, area)
char *id, **area;

char *
tgoto(cm, destcol, destline)
char *cm;
int destcol, destline;

tputs(cp, affcnt, outc)
register char *cp;
int affcnt;
int (*outc)();
```

DESCRIPTION

These functions extract and use capabilities from the terminal capability data base **termcap**(5).

Tgetent extracts the entry for terminal name into the buffer at bp. Bp is a character buffer of size 1Ø24 and must be retained through all subsequent calls to **tgetnum, tgetflag,** and **tgetstr. Tgetent** returns -1 if it cannot open the termcap file, Ø if the terminal name given does not have an entry, and 1 if all goes well. It looks in the environment for a TERMCAP variable. If found and the value does not begin with a slash, and the terminal **name** is the same as the environment string TERM, the TERMCAP string is used instead of reading the termcap file. Alternatively, if the TERMCAP variable does not begin with a slash and the terminal **name** is not the same as the TERM variable, then /etc/termcap is used to find TERM's description. If it does begin with a slash, the string is used as a path name rather than /etc/termcap. This speeds up entry into programs that call

tgetent helps debug new terminal descriptions, and helps to make one for the terminal if the file /etc/termcap cannot be written.

Tgetnum gets the numeric value of capability id, returning -1 if is not given for the terminal. **Tgetflag** returns 1 if the specified capability is present in the terminal's entry, Ø if it is not. **Tgetstr** gets the string value of capability id, placing it in the buffer at area, advancing the area pointer. It decodes the abbreviations for this field described in **termcap**(5), except for cursor addressing and padding information.

Tgoto returns a cursor addressing string decoded from cm to go to column destcol in line destline. It uses the external variables UP (from the **up** capability) and BC (if bc is given rather than bs) if necessary to avoid placing **\n**, **^D**, or **^@** in the returned string. Programs that call tgoto must turn off the XTABS bit(s), since tgoto can now output a tab. Programs using termcap must always turn off XTABS. If a **%** sequence is given that is not understood, then **tgoto** returns the string OOPS.

Tputs decodes the leading padding information of the string cp; affcnt gives the number of lines affected by the operation, or 1 if this is not applicable. Outc is a routine called with each character in turn. The external variable ospeed must contain the output speed of the terminal as encoded by **stty** (in **ioctl**(2)). The external variable **PC** must contain a pad character to be used (from the **pc** capability) if a null (**^@**) is inappropriate.

FILES

/usr/lib/libtermlib.a	nonsegmented library
/usr/lib/slibtermlib.a	segmented library
/etc/termcap	default data base

SEE ALSO

ex(1), termcap(5), tty(4), ioctl(2).

NAME

tmpfile - create a temporary file

SYNOPSIS

```
#include <stdio.h>

FILE *tmpfile ()
```

DESCRIPTION

Tmpfile creates a temporary file and returns a corresponding **FILE** pointer. Arrangements are made so that the file will automatically be deleted when the process using it terminates. The file is opened for update.

FILES

/usr/tmp directory for temporary files

SEE ALSO

creat(2), unlink(2), fopen(3), mktemp(3), tmpnam(3).

NAME

tmpnam - create a name for a temporary file

SYNOPSIS

```
#include <stdio.h>

char *tmpnam (s)
    char *s;
```

DESCRIPTION

Tmpnam generates a file name that can safely be used for a temporary file. If (int)s is zero, **tmpnam** leaves its result in an internal static area and returns a pointer to that area. The next call to **tmpnam** will destroy the contents of the area. If (int)s is nonzero, s is assumed to be the address of an array of at least **L_tmpnam** bytes; **tmpnam** places its result in that array and returns s as its value. **Tmpnam** generates a different file name each time it is called. Files created using **tmpnam** and either **fopen**(2) or **creat**(2) are only temporary in the sense that they reside in a directory intended for temporary use, and their names are unique. It is the user's responsibility to use **unlink**(2) to remove the file when its use is ended.

FILES

/usr/tmp directory for temporary files

SEE ALSO

creat(2), unlink(2), fopen(3), mktemp(3).

LIMITATIONS

If called more than 17,576 times in a single process, **tmpnam** will start recycling previously used names.
Between the time a file name is created and the file is opened, it is possible for some other process to create a file with the same name. This can never happen if that other process is using **tmpnam** or **mktemp**, and the file names are chosen so as to render duplication by other means unlikely.

NAME

ttyname, isatty, ttyslot - find name of a terminal

SYNOPSIS

char *ttyname(fildes)

isatty(fildes)

ttyslot()

DESCRIPTION

Ttyname returns a pointer to the null-terminated path name of the terminal device associated with file descriptor fildes.

Isatty returns 1 if fildes is associated with a terminal device, Ø otherwise.

Ttyslot returns the number of the slot in the **utmp**(5) file for the control terminal of the current process.

FILES

/dev/*
/etc/utmp

SEE ALSO

ioctl(2), utmp(5).

DIAGNOSTICS

Ttyname returns a null pointer (Ø) if fildes does not describe a terminal device in directory /dev.

Ttyslot returns -1 if the slot is not found.

LIMITATIONS

The return value points to static data whose content is overwritten by each call.

NAME

ungetc - push character back into input stream

SYNOPSIS

```
#include <stdio.h>

ungetc(c, stream)
FILE *stream;
char c;
```

DESCRIPTION

Ungetc pushes the character c back on an input stream. That character is returned by the next **getc** call on that stream.

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered. Attempts to push EOF are rejected.

Fseek(3) erases all memory of pushed-back characters.

SEE ALSO

getc(3), setbuf(3), fseek(3).

DIAGNOSTICS

Ungetc returns **EOF** if it cannot push a character back.

NAME

acu - automatic dialing out unit

DESCRIPTION

An **Acu** is a modem that contains an automatic dialer that will allow a System 8000 to be connected to the telephone network.
The **Acu** is used by uucp(1) and cu(1). Currently uucp(1) and cu(1) only have the capabilities to dial-out using a Ventel 212-plus. This particular modem has a interactive feature that allows the user to do much more than just dialing one number.

The Ventel 212-plus must be connected to one of the serial I/O ports with a NULL modem cable. Such a cable can be purchased from many suppliers. The serial I/O port must be disabled in the **/etc/inittab** file.
A brief list of the commands that the Ventel 212-plus understands is:

H Display the help messages.

K Dial from the keyboard.

S1-S5
Store/Change numbers.

1-5 Dial a stored number.

R Re-dial last number.

F Re-dial forever.

D Display stored numbers.

C Clear numbers.

B busy out.

Q Quit.

Because of the nature of both uucp(1) and cu(1), not all of the Ventel features are used. The sequence that cu(1) follows when talking to the Ventel 212-plus is:

1. Send two carriage returns. This lets the Ventel 212-plus recognize your transmission speed.

2. Send the K command. This means dial from the keyboard.

3. Send the telephone number. This step is difficult

because the speed at which the Ventel 212-plus accepts each digit of the telephone number is much slower than the rate at which the System 8000 transmits them.

For a more complete definition of the caplablities of the Ventel 212-plus, please refer to the Ventel 212-plus, Integral Dialer, Users Manual Supplement.

SEE ALSO

cu(1), uucp(1)

NAME

ct - cartridge tape interface

DESCRIPTION

The files **ctØ, ctl, ...** refer to the cartridge tape units. When opened for reading or writing, the tape is not rewound. When closed, it is rewound unless the Ø2ØØ bit is on. If the tape was open for writing, an end-of-file is written, and the next four feet of tape are erased (indicating end-of-tape). If the tape is not to be rewound, the tape is backspaced to after the filemark.

Exiting a program which has opened a tape device automatically closes the device, even if there is no explicit close of the device. If the tape was opened for writing, an end-of-file is written at the current location, and the next four feet of tape are erased. Therefore, one should not del out of programs which have opened a tape device for writing unless truncation is desired. It is generally desirable to be at the end of the tape upon close of the tape device if it was opened for writing.

The **ct** files allow reads and writes of arbitrary length. Seeks are also supported. These files are intended to allow tape files to be accessed similarly to other files to a certain extent.

The files **rctØ, rctl, ...** refer to the raw interface. These are useful to read and write long records and on the ZEUS system. **Tar**(1), for example, uses the raw interface. The rest of this discussion continues to refer to the raw device names. Except that seeks are not supported on the raw device, the nonraw device can be accessed similarly by dropping the r from the device name.

The cartridge tape unit has the capability of accessing each of the four tracks on a tape individually. The files **rctØa, ..., rctØd, rctla, ...** refer to the individual tracks on a tape.

The lowest four bits of the minor device number select the tape track to be used. The next two bits specify the drive. Bit 6 specifies which of up to two controllers is to be used. If bit 7 is on, the tape is not rewound on close. The minor device number has no necessary connection with the file name.

The name of the raw tape device is rct*n*, where *n* is the drive number. A leading *n* (for example, nrctØ) specifies the no-rewind device, and a trailing letter in the range a-d indicates single-track operation.

Each **read** or **write** call to a raw tape device reads or writes the next record on the tape. In the write case, the record has the same length as the buffer given. During a read, the record size is passed back as the number of bytes read, provided it is not greater than the buffer size. If the record is long, the extra data is skipped over without notification. The buffer must begin on a word boundary. Writes can be of any length. The buffer length for reads must be even; if the record is shorter, the actual length (even or odd) will be returned. Seeks are ignored on the raw device. A zero count is returned when a tape mark is read. When the no-rewind device is closed and reopened, the tape head is positioned at the beginning of the next tape file.

The following **ioctl** calls are available for performing forward and backward file and record spacing, respectively, on the raw tape device:

```
#include <ct.h>
ioctl(fildes, CTIOFF, n)
ioctl(fildes, CTIOFR, n)
ioctl(fildes, CTIOBF, n)
ioctl(fildes, CTIOBR, n)
```

where fildes is a file descriptor returned from an open of the raw tape device, and n is an integer specifying the number of files or records to space forward or backward. These calls do not affect the seek pointer. An attempt to move past the logical end of tape using CTIOFF will return an end of data error in errno; an attempt to move before the logical beginning of tape using CTIOFR or CTIOBR will return an end of media error. CTIOBF and CTIOBR will not cross over a filemark. **ioctl** returns the number of blocks or files skipped, or -1 if an end of data or end of media error has occurred.

FILES

/dev/rct?,
/dev/nrct?,
/dev/ct?,
/dev/nct?
/dev/rct?[a-d],
/dev/nrct?[a-d],
/dev/ct?[a-d],
/dev/nct?[a-d]

SEE ALSO

dd(1), tar(1), ioctl(2).

LIMITATIONS

In raw I/O, there is not but should be a way to write an EOF mark explicitly.

NAME

root, rroot, usr, rusr, tmp, rtmp, z, rz, tardev, dumpdev and resdev

DESCRIPTION

The files root, rroot, usr, rusr, tmp, rtmp, z and rz, found in /dev, are links to the default devices for the filesystems /, /usr, /tmp and /z. The following table shows these files and the default devices to which they are linked for different types of disks:

	Devices		
Files	5-1/4 inch Disk	8 inch Disk	SMD Disk
/dev/root	/dev/md2	/dev/zd2	/dev/smd2
/dev/rroot	/dev/rmd2	/dev/rzd2	/dev/rsmd2
/dev/usr	/dev/mdØ	/dev/zdØ	/dev/smdØ
/dev/rusr	/dev/rmdØ	/dev/rzdØ	/dev/rsmdØ
/dev/tmp	/dev/md3	/dev/zd3	/dev/smd3
/dev/rtmp	/dev/rmd3	/dev/rzd3	/dev/rsmd3
/dev/z	/dev/md4	/dev/zd4	/dev/smd4
/dev/rz	/dev/rmd4	/dev/rzd4	/dev/rsmd4

The files tardev and resdev are used by **tar**(1) and **reserv**(1) their default devices. The file dumpdev is used by **dump**(M) and **restor**(M) as their default device. These files should be linked to the appropriate raw tape device. If not linked, /dev/rctØ is used by default. For example, if the 9-track tape device is to be the default dump device, then the following link should be made:

```
ln /dev/rmtØ /dev/dumpdev
```

FILES

/dev/root
/dev/rroot
/dev/usr
/dev/rusr
/dev/tmp
/dev/rtmp
/dev/z
/dev/rz
/dev/tardev
/dev/dumpdev
/dev/resdev

SEE ALSO

reserv(1), tar(1), dump(M), makenewfs(M), mfs(M), rc(M), restor(M).

NAME

icp - general Intelligent Communication Processor interface

DESCRIPTION

An **ICP** device driver implements a protocol independent access method to **ICP** controller boards and general control functions of an **ICP**. It is also the link between a ZEUS Protocol Driver and an associated **ICP** Protocol Driver. All communication between the protocol driver pair passes through the **ICP** device driver.

The **ICP** is an exclusive open device, i.e. only one process may use the control functions (**open**(2), **close**(2), **read**(2), **write**(2), **ioctl**(2)) at a time.

The **ICP** is viewed as any other special file under ZEUS. For example, a user can read memory using **hd**(1) to get a dump of an **ICP**'s memory contents.

The files /dev/icp* are the special files which allow access to the **ICP**s (up to 8 maximum) installed on a system.

Each **ICP** has 9 ports and logical port numbers, which represent to the ports of all **ICP**s ranging from 0 to 71.

Several **ioctl**(2) calls apply to **ICP**s. Most use the following structure (defined in <icpio.h>):

```
#define   ICP_PMAX  9

struct icpquery {
       char     icpq_istate;          /* state of the ICP    */

       struct  {
               char     icpq_pstate;  /* function to perform */
                                      /* on port             */
               char     icpq_pp;      /* protocol on port    */
       } iport[ICP_PMAX];
};
```

A few **ioctl** calls have the form:

```
#include <icpio.h>

ioctl(fildes, code, arg)
int fildes;
int code;
struct icpquery *arg;
```

The applicable codes are:

STARTPP

This command looks through the icpquery structure passed to it for protocols other than the NULL protocol. Upon finding one, it then associates the given protocol with the port (iport[x]). If successful, it will start the protocol running on this specific port on the **ICP**. This command may return an error under the following conditions:

the icpquery structure pointer passed is null;
the protocol may not be associated with this port;
the **ICP** has a hardware error;
or there has been a system memory fault.

If the indicated protocal is started sucessfully, the appropriate icpq-pstate will be equal to the value P_ASSOC, otherwise the protocal count not be started and icpq-pstate will be equal to P_NOASSOC.

STOPPP

This command looks through the icpquery structure passed to it for protocols other than the NULL protocol. Upon finding one, it then attempts to dis-associate the given protocol with the port (iport[x]). If successful, it will stop the protocol running on this specific port on the **ICP**. This command may return an error on the following conditions:

the icpquery structure pointer passed is null;
the protocol may not be dis-associated with this por
the **ICP** has a hardware error;
or there has been a system memory fault.

The values that the various members of the query structure may contain are:

icpq-istat ICP_RUNNING
ICP_ISOPEN
ICP_ERROR
ICP_SOFTERR
ICP_ACK

icpq-pp the protocol that is active on this port otherwise NULLPROTO. Defined protocal values are in "icpio.h"

QUERY

This command fills the icpquery structure passed to it (rather, the pointer is passed to it), with the state of an **ICP**, and the protocols that are running on the **ICP**. This command will fail under these conditions:

the icpquery structure pointer is null;
or there was a system memory fault.

Additional **ioctl** calls have the form:

```
#include <icpio.h>

ioctl(fildes, code, Ø)
int fildes;
int code;
```

STARTI
Removes reset from an **ICP** controller board, and attempts to set it running. Also resets any hardware error indication, if there was a previous hardware error. This command will return an error if the **ICP** cannot be started.

STOPI
This unconditionally stops an **ICP** controller board. There is no error indication returned if the **ICP** is already stopped.

FILES
/dev/icp*

SEE ALSO
ioctl(2), icpload(M), icpcntrl(M).

NAME

lp - line printer driver

DESCRIPTION

The files lp, lp2 refer to line printer ports.

ZEUS can accommodate up to three line printers, depending on the hardware configuration. The printers can have either a Centronics or Data Products interface. During SYSGEN the kernel can be configured to have no line printers.

The default configuration is for Centronics line printer. A Data Products interface will need a different hardware configuration. Refer to the System 8000 hardware reference manual for details.

The device files /dev/lp and /dev/lp2 refer to the line printer drivers. Each has a major device number of 9 and a minor device number that depends on the kind of printer and port used.

The line printers are accessed by the line printer spooler **lp**(1) or by the **ioctl**(2) system call.

The **ioctl** system call is used to modify some parameters of the line printer driver. This call uses the following structure and commands which are defined in

/usr/include/sgtty.h:

```
#define  SHOWLPR    (('l' << 8) | 0)
#define  SETLP      (('l' << 8) | 1)

/*
* This is the structure of the
* arguments to the lpr ioctl program
*/

struct lparms

{

    int     lines;          /* number of lines per page     */
    int     cols;           /*number of columns per page    */
    int     indent;         /*default indentation for       */
                            /*line printer                  */

};
```

The way the the system call is implemented is similar to the tty(2) ioctl call.

```
#include <sgtty.h>

struct lparms args ;

    if ((fd = open("/dev/lp", 2)) < Ø )
         printf("can't open /dev/lp\n");
    rtn = (ioctl(fd, SHOWLPR, &args));
    rtn = (ioctl(fd, SETLP, &args));
```

If the command SHOWLPR is issued to the line printer, the line printer driver will return to the user's lparms structure the parameters: lines per page, columns per page, and standard indent.

Lines per page is defined as the number of lines printed before an FF character (ØxC) is output.

Columns per page means the number of characters printed before a "carriage return" is output. For Data Products printers this is a '\n' character (ØxA) which causes a line feed and carriage return to be emitted. For Centronics printers the line printer driver must emit a '\r' (ØxD) and '\n' (ØxA) to emit a carriage return and line feed. If the Centronics printer has an "automatic line feed" option, it emits a line feed for '\r' (ØxD). This should be disabled to allow underlining.

Indent is the default indentation for all output to /dev/lp. Note that the the length of the line printed will be cols - indent thus lines longer than this are truncated rather than shifted to the right.

The command SETLP allows the system administrator to set these values in the line printer driver to accommodate printers with different characteristics. The default values set at compile time for lines, cols and indent are 66, 13Ø, and 1 respectively.

The following is a description of how the line printer driver handles some special characters.

'\n' (ØxA) - Newline: This is output 'as is' for Data Products printers. For Centronics printers, a '\r' is output also.

'\r' (ØxD) - Carriage Return: This is output 'as is'.

'\f' (ØxC) - Form Feed: This output 'as is' for Data Products printers and appended with a '\r' for Centronics printers.

'\b' (Øx8) - Backspace: A '\r' is emitted and spaces are printed up to the previous character printed.

'\t' (Øx9) - Horizontal Tab: Blanks are output to the nearest multiple of four.

FILES

/dev/lp
/dev/lp2
/usr/include/sgtty.h

SEE ALSO

ioctl(2), tty(4).

Zeus System Administrators Manual,
System 8000 Hardware Reference Manual Manual

LIMITATIONS

There is no way at this time of choosing CR/LF default for automatic LF printers.

NAME

md - 5.25" Winchester disk

DESCRIPTION

The **md** device provides the interface to 5.25-inch Winchester disk drives.

The files md0 ... md9 refer to sections of disk drive 0. The files md10 ... md19 refer to drive 1 etc. This allows a large disk to be broken up into more manageable pieces.

The origin and size of the pseudo-disks on each drive as delivered from the factory are as follows:

disk	start	length
0	0	5000
1	5000	2000
2	7000	6000
3	13000	4000
4	17000	11000
5-9	unassigned	

Note that these values can be changed by using **sysgen**(M).

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

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

FILES

/dev/md*, /dev/rmd*

LIMITATIONS

In raw I/O **read** and **write**(2) truncate file offsets to 512-byte block boundaries, and **write** scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, **read, write** and **lseek**(2) should always deal in 512-byte multiples.

NAME

mem, kmem - memory

DESCRIPTION

Mem is a special file that is an image of the memory of the computer. For example, it examines and even patches the system. **Kmem** is the same as **mem** except that kernel virtual memory rather than physical memory is accessed.

Byte addresses are interpreted as memory addresses. References to nonexistent locations return errors.

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

On the System 8000, the kernel instruction space begins at physical location 0. Kernel data space starts at the next 256-byte boundary following the kernel instruction space.

The per-process data for the current process begins at kernel virtual memory address 0xF800.

FILES

/dev/mem, /dev/kmem.

NAME

mt - Zilog streaming magnetic tape interface

DESCRIPTION

The files **mtØ, ..., mt7** refer to the streaming magnetic tape control and transports at 16ØØbpi. The files **mtØ, ..., mt7** are designated normal-rewind on close, and the files **nmt4, ..., nmt7** are no-rewind on close. When opened for reading or writing, the tape is assumed to be positioned as desired. When a file is closed, a double end-of-file (double tape mark) is written if the file was opened for writing. If the file was normal-rewind, the tape is rewound. If it is no-rewind and the file was open for writing, the tape is positioned before the second EOF just written. If the file was no-rewind and opened read-only, the tape is positioned after the EOF following the data just read. Once opened, reading is restricted to between the position when opened and the next EOF or the last write. The EOF is returned as a zero-length read. By judiciously choosing **mt** files, it is possible to read and write multi-file tapes.

A standard tape consists of several 512 byte records terminated by an EOF. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time.

The **mt** files discussed above are useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, and especially when long records are to be read or written, the "raw" interface is appropriate. The associated files are named **rmtØ, ..., rmt7** Each **read** or **write** call reads or writes the next record on the tape. In the write case the record has the same length as the buffer given. During a read, the record size is passed back as the number of bytes read, up to the buffer size specified. In raw tape I/O, seeks are ignored. An EOF is returned as a zero-length read, with the tape positioned after the EOF, so that the next read will return the next record.

In addition, the files **smtØ, ..., smt7** represent short inter record gap (.6in) rather than the default (1.2in) gap. This applies only to writing. Either length gap can be read with either option. The files **fmtØ, ..., fmt7** represent the high speed mode of the drives which is 1ØØips rather than the default 12.5ips.

Any combination of these options are available through one of the /dev/*mt? files.

The minor device number of the device special files determines which combination of the above options applies to them. The minor device number is formed by 'anding' the bits of the desired options. Each bit position in the minor device number represents an option.

The rightmost two bits, bits Ø and 1, represent one of four controllers.
The next three bits, bits 2, 3, and 4, represent one of eight tape drives for each controller.
The next position, bit 5, if on, represents high speed mode.
The next position, bit 6, if on, means the tape is to be rewound on close.
The next position, bit 7, denotes long gap if on, short gap if off.

These bits are only relevent when using the **mknod**(M) command to create new device files for nine track mag. tape drives. The device files for all combinitions of options for four tape drives are automaticaly created by using the makemt(M) command.

Streaming automatically occurs when there are several reads or writes and the data does not need to be read in from the disk. If the tape drives do not stream, then the drive must reposition the tape after each block read or written.

There are several **ioctl**(2) system calls which apply to the nine track tape drive. The calls apply to an open /dev/*mt* special file. The syntax of the calls is:

ioctl(fd, REQUEST, count);

Fd is a valid file descriptor of a /dev/*mt* file.
REQUEST is one of the options below.
Count applies to certain of the requests below.

MTSKBF - Skips forward the number of tape blocks specified by the count field. This command will not skip over file marks or past the end of tape mark. The tape blocks may be of any size.

MTSKBR - Skips backward the specified number of tape blocks. This command will not skip back over file marks or before the beginning of tape mark. The tape blocks may be of any size.

MTSKFF - Skips forward the number of files (as determined by file marks) specified by the count field. The tape is positioned at the beginning of the next file. The tape will not skip past the end of tape mark.

MTSKFR - Skips backward the number of files (as determined by file marks) specified by the count field. The tape is positioned at the beginning of the appropriate file. The tape will not skip past the beginning of tape mark.

MTWFM - Writes a file mark at the current position. If the short gap device is used the file mark will have a short gap before it rather than a long gap. This command does not take a count.

MTSE - Security erase. The tape is erased at high speed from the current position to one meter past the end of tape mark. This command does not take a count.

MTRWD - Rewind the tape. Leave the tape positioned at the load point. This command does not take a count.

MTRWDUL - Rewind and unload the tape, leave the drive off line. The tape is completely rewound from the current position. This command does not take a count.

MTOL - Put the drive on line. This command does not take a count.

FILES

/dev/mt
/dev/rmt
/dev/smt
/dev/fmt
/dev/srmt
/dev/frmt
/dev/fsmt

NAME

null - data sink

DESCRIPTION

Data written on a **null** special file is discarded.

Reads from a **null** special file always return Ø bytes.

FILES

/dev/null

NAME

smd - Storage module disk

DESCRIPTION

The **smd** device provides the interface to disks conforming tp the Storage Module Device industry standard.

The files smdØ ... smd9 refer to sections of disk drive Ø. The files smdlØ ... smdl9 refer to drive 1 etc. This allows a large disk to be broken up into more manageable pieces.

The origin and size of the pseudo-disks on each drive as delivered from the factory are as follows:

disk	start	length
Ø	Ø	12ØØØ
1	12ØØØ	32ØØ
2	15200	6ØØØ
3	21200	6ØØØ
4	27200	104739
5-9	unassigned	

Note that these values can be changed by using **sysgen**(M).

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

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

FILES

/dev/smd*
/dev/rsmd*

LIMITATIONS

In raw I/O **read** and **write**(2) truncate file offsets to 512-byte block boundaries, and **write** scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, **read, write** and **lseek**(2) should always deal in 512-byte multiples.

NAME

tty - general terminal interface

DESCRIPTION

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

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

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

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, users' programs seldom open these files; they are opened by **getty**(8) and become a user's standard input, output, and error files. The very first terminal file opened by the process group leader of a terminal file not already associated with a process group becomes the control terminal for that process group. The control terminal plays a special role in handling quit and interrupt signals, as discussed below. The control terminal is inherited by a child process during a **fork**(2). A process can break this association by changing its process group using **setpgrp**(2).

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

Normally, terminal input is processed in units of lines. A line is delimited by a new-line (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. This means that a program attempting to read will be suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not, however, necessary to read a whole line at once; any number of

characters may be requested in a read, even one, without losing information.

During input, erase and kill processing is normally done. By default, the character **<BS>** erases the last character typed, except that it will not erase beyond the beginning of the line. By default, the character **^X** kills (deletes) the entire input line, and optionally outputs a new-line character. Both these characters operate on a key-stroke basis, independently of any backspacing or tabbing that may have been done. Both the erase and kill characters may be entered literally by preceding them with the escape character (\). In this case the escape character is not read. The erase and kill characters may be changed.

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR (Rubout or ASCII DEL) generates an interrupt signal which is sent to all processes with the associated control terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location; see **signal**(2).

QUIT (Control-| or ASCII FS) generates a **quit** signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called core) will be created in the current working directory.

ERASE (ASCII BS) erases the preceding character. It will not erase beyond the start of a line, as delimited by a NL, EOF, or EOL character.

KILL (Control-x) deletes the entire line, as delimited by a NL, EOF, or EOL character.

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

NL (ASCII LF) is the normal line delimiter. It can not be changed or escaped.

EOL (ASCII NUL) is an additional line delimiter, like NL. It is not normally used.

STOP (Control-s or ASCII DC3) can be used to temporarily suspend output. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.

START (Control-q or ASCII DC1) is used to resume output which has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The start/stop characters can not be changed or escaped.

The character values for INTR, QUIT, ERASE, KILL, EOF, and EOL may be changed to suit individual tastes. The ERASE, KILL, and EOF characters may be escaped by a preceding \ character, in which case no special function is done.

When the carrier signal from the data-set drops, a hangup signal is sent to all processes that have this terminal as the control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hangup signal is ignored, any subsequent read returns with an end-of-file indication. Thus programs that read a terminal and test for end-of-file can terminate appropriately when hung up on.

When one or more characters are written, they are transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed by putting them in the output queue as they arrive. If a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold, the program is resumed.

Several **ioctl**(2) system calls apply to terminal files. The primary calls use the following structure, defined in **<termio.h>:**

```
#define     NCC  8
struct      termio {
    unsigned    short    c_iflag;     /* input modes */
    unsigned    short    c_oflag;     /* output modes */
    unsigned    short    c_cflag;     /* control modes */
    unsigned    short    c_lflag;     /* local modes */
    char        c_line;          /* line discipline */
    unsigned    char     c_cc[NCC];    /* control chars */
};
```

The special control characters are defined by the array c_cc. The relative positions and initial values for each function are as follows:

0	INTR	DEL
1	QUIT	FS
2	ERASE	**BS**
3	KILL	**^X**
4	EOF	EOT
5	EOL	NUL
6	reserved	
7	reserved	

The c_iflag field describes the basic terminal input control:

IGNBRK	0000001	Ignore break condition.
BRKINT	0000002	Signal interrupt on break.
IGNPAR	0000004	Ignore characters with parity errors.
PARMRK	0000010	Mark parity errors.
INPCK	0000020	Enable input parity check.
ISTRIP	0000040	Strip character.
INLCR	0000100	Map NL to CR on input.
IGNCR	0000200	Ignore CR.
ICRNL	0000400	Map CR to NL on input.
IUCLC	0001000	Map upper-case to lower-case on input.
IXON	0002000	Enable start/stop output control.
IXANY	0004000	Enable any character to restart output.
IXOFF	0010000	Enable start/stop input control.

If IGNBRK is set, the break condition (a character framing error with data all zeros) is ignored, that is, not put on the input queue and therefore not read by any process. Otherwise if BRKINT is set, the break condition will generate an interrupt signal and flush both the input and output queues. If IGNPAR is set, characters with other framing and parity errors are ignored.

If PARMRK is set, a character with a framing or parity error which is not ignored is read as the three character sequence: 0377, 0, X, where X is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of 0377 is read as 0377, 0377. If PARMRK is not set, a framing or parity error which is not ignored is read as the character NUL (0).

If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is disabled. This allows output parity generation without input parity errors.

If ISTRIP is set, valid input characters are first stripped to 7-bits, otherwise all 8-bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). Otherwise if ICRNL is set, a received CR character is translated into a NL character.

If IUCLC is set, a received upper-case alphabetic character is translated into the corresponding lower-case character.

If IXON is set, start/stop output control is enabled. A received STOP character will suspend output and a received START character will restart output. All start/stop characters are ignored and not read. If IXANY is set, any input character, will restart output which has been suspended.

If IXOFF is set, the system will transmit START/STOP characters when the input queue is nearly empty/full.

The initial input control value is ICRNL, ISTRIP, IGNPAR, IXON, BRKINT.

The c oflag field specifies the system treatment of output:

OPOST	0000001	Postprocess output.
OLCUC	0000002	Map lower case to upper on output.
ONLCR	0000004	Map NL to CR-NL on output.
OCRNL	0000010	Map CR to NL on output.
ONOCR	0000020	No CR output at column 0.
ONLRET	0000040	NL performs CR function.
OFILL	0000100	Use fill characters for delay.
OFDEL	0000200	Fill is DEL, else NUL.
NLDLY	0000400	Select new-line delays:
NL0	0	
NL1	0000400	
CRDLY	0003000	Select carriage-return delays:
CR0	0	
CR1	0001000	
CR2	0002000	
CR3	0003000	
TABDLY	0014000	Select horizontal-tab delays:
TAB0	0	
TAB1	0004000	
TAB2	0010000	
TAB3	0014000	Expand tabs to spaces.
BSDLY	0020000	Select backspace delays:
BS0	0	
BS1	0020000	
VTDLY	0040000	Select vertical-tab delays:
VT0	0	
VT1	0040000	
FFDLY	0100000	Select form-feed delays:
FF0	0	
FF1	0100000	

If OPOST is set, output characters are post-processed as indicated by the remaining flags, otherwise characters are transmitted without change.

If OLCUC is set, a lower-case alphabetic character is transmitted as the corresponding upper-case character. This function is often used in conjunction with IUCLC.

If ONLCR is set, the NL character is transmitted as the CR-NL character pair. If OCRNL is set, the CR character is transmitted as the NL character. If ONOCR is set, no CR character is transmitted when at column Ø (first position). If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer will be set to Ø and the delays specified for CR will be used. Otherwise the NL character is assumed to do just the line-feed function; the column pointer will remain unchanged. The column pointer is also set to Ø if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of Ø indicates no delay. If OFILL is set, fill characters will be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character is DEL, otherwise NUL.

If a form-feed or vertical-tab delay is specified, it lasts for about 2 seconds.

New-line delay lasts about Ø.1Ø seconds. If ONLRET is set, the carriage-return delays are used instead of the new-line delays. If OFILL is set, two fill characters will be transmitted.

Carriage-return delay type 1 is dependent on the current column position, type 2 is about Ø.1Ø seconds, and type 3 is about Ø.15 seconds. If OFILL is set, delay type 1 transmits two fill characters, and type 2 four fill characters.

Horizontal-tab delay type 1 is dependent on the current column position. Type 2 is about Ø.1Ø seconds. Type 3 specifies that tabs are to be expanded into spaces. If OFILL is set, two fill characters will be transmitted for any delay.

Backspace delay lasts about Ø.Ø5 seconds. If OFILL is set, one fill character will be transmitted.

The actual delays depend on line speed and system load.

The initial output control value is TAB3, OPOST, ONLCR.

The c_cflag field describes the hardware control of the terminal:

CBAUD	0000017	Baud rate:
B0	0	Hang up
B75	0000002	75 baud
B110	0000003	110 baud
B134	0000004	134.5 baud
B150	0000005	150 baud
B200	0000006	200 baud
B300	0000007	300 baud
B600	0000010	600 baud
B1200	0000011	1200 baud
B1800	0000012	1800 baud
B2400	0000013	2400 baud
B4800	0000014	4800 baud
B9600	0000015	9600 baud
B19200	0000016	19200 baud
EXT	0000017	External
CSIZE	0000060	Character size:
CS5	0	5 bits
CS6	0000020	6 bits
CS7	0000040	7 bits
CS8	0000060	8 bits
CSTOPB	0000100	Send two stop bits, else one.
CREAD	0000200	Enable receiver.
PARENB	0000400	Parity enable.
PARODD	0001000	Odd parity, else even.
HUPCL	0002000	Hang up on last close.
CLOCAL	0004000	Local line, else dial-up.

The CBAUD bits specify the baud rate. The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal will not be asserted. Normally, this will disconnect the line. For any particular hardware, impossible speed changes are ignored.

The CSIZE bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If CSTOPB is set, two stop bits are used, otherwise one stop bit. For example, at 110 baud, two stops bits are required.

If PARENB is set, parity generation and detection is enabled and a parity bit is added to each character. If parity is enabled, the PARODD flag specifies odd parity if set, otherwise even parity is used.

If CREAD is set, the receiver is enabled. Otherwise no characters will be received.

If HUPCL is set, the line will be disconnected when the last process with the line open closes it or terminates. That is, the data-terminal-ready signal will not be asserted.

If CLOCAL is set, the line is assumed to be a local, direct connection with no modem control. Otherwise modem control is assumed.

The initial hardware control value after open is B9600, CS7, CREAD, HUPCL, CSTOPB.

The c_lflag field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline (0) provides the following:

ISIG	0000001	Enable signals.
ICANON	0000002	Canonical input (erase and kill processing)
XCASE	0000004	Canonical upper/lower presentation.
ECHO	0000010	Enable echo.
ECHOE	0000020	Echo erase character as BS-SP-BS.
ECHOK	0000040	Echo NL after kill character.
ECHONL	0000100	Echo NL.
NOFLSH	0000200	Disable flush after interrupt or quit.

If ISIG is set, each input character is checked against the special control characters INTR and QUIT. If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, no checking is done. Thus these special input functions are possible only if ISIG is set. These functions may be disabled individually by changing the value of the control character to an unlikely or impossible value (e.g. 0377).

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL. If ICANON is not set, read requests are satisfied directly from the input queue. A read will not be satisfied until at least MIN characters have been received or the timeout value TIME has expired. This allows fast bursts of input to be read efficiently while still allowing single character input. The MIN and TIME values are stored in the position for the EOL and first reserved characters respectively. The time value represents tenths of seconds.

If XCASE is set, and if ICANON is set, an upper-case letter is accepted on input by preceding it with a \ character, and is output preceded by a \ character. In this mode, the

following escape sequences are generated on output and accepted on input:

for:	use:
\	\\
|	\!
~	\^
{	\(
}	\)
\	\\

For example, **A** is input as **\a**, **\n** as **\\n**, and **\N** as **\\\n**.

If ECHO is set, characters are echoed as received.

When ICANON is set, the following echo functions are possible. If ECHO and ECHOE are set, the erase character is echoed as ASCII BS SP BS, which will clear the last character from a CRT screen. If ECHOE is set and ECHO is not set, the erase character is echoed as ASCII SP BS. If ECHOK is set, the NL character will be echoed after the kill character to emphasize that the line will be deleted. Note that an escape character preceding the erase or kill character removes any special function. If ECHONL is set, the NL character will be echoed even if ECHO is not set. This is useful for terminals set to local echo (so-called half duplex). Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

If NOFLSH is set, the normal flush of the input and output queues associated with the quit and interrupt characters will not be done.

The initial line-discipline control value is ISIG, ICANON, ECHO, ECHOE, ECHOK, ECHONL.

The primary **ioctl**(2) system calls have the form:

```
ioctl (fildes, command, arg)
struct termio *arg;
```

The commands using this form are:

TCGETA Get the parameters associated with the terminal and store in the termio structure referenced by **arg**.

TCSETA Set the parameters associated with the terminal from the structure referenced by **arg**. The change is immediate.

TCSETAW Wait for the output to drain before setting the new parameters. This form should be used when changing parameters that will affect output.

TCSETAF Wait for the output to drain, then flush the input queue and set the new parameters.

Additional **ioctl**(2) calls have the form:

```
ioctl (fildes, command, arg)
int arg;
```

The commands using this form are:

TCSBRK Wait for the output to drain. If arg is Ø, then send a break (zero bits for Ø.25 seconds).

TCXONC Start/stop control. If arg is Ø, suspend output; if 1, restart suspended output.

TCFLSH If arg is Ø, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

FILES

/dev/tty
/dev/tty*
/dev/console

SEE ALSO

stty(1), ioctl(2).

NAME

zd - Winchester disk

DESCRIPTION

The **zd** device provides the interface to disk drives. Two sizes of drives are currently supported: 24 MB (mega bytes) and 32 MB.

The files zdØ ... zd9 refer to sections of disk drive Ø. The files zd1Ø ... zd19 refer to drive 1 etc. This allows a large disk to be broken up into more manageable pieces.

The origin and size of the pseudo-disks on each drive as delivered from the factory are as follows:

24 MB Drive

disk	start	length
Ø	Ø	1ØØØØ
1	1ØØØØ	32ØØ
2	132ØØ	5ØØØ
3	182ØØ	5ØØØ
4	232ØØ	2ØØØØ
5-9	unassigned	

32 MB Drive

disk	start	length
Ø	Ø	12ØØØ
1	12ØØØ	32ØØ
2	152ØØ	6ØØØ
3	212ØØ	6ØØØ
4	272ØØ	3Ø4ØØ

Note that these values can be changed by using **sysgen**(M).

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

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

FILES

/dev/zd*
/dev/rzd*

LIMITATIONS

In raw I/O **read** and **write**(2) truncate file offsets to 512-byte block boundaries, and **write** scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, **read, write** and **lseek**(2) should always deal in 512-byte multiples.

NAME

a.out - System 8000 object code format

SYNOPSIS

#include <s.out.h>

DESCRIPTION

A.out is the output file of the assembler **as**(1) and the link editors **ld**(1) and **sld**(1). Both programs make **a.out** executable if there are neither errors nor unresolved external references. This format is a departure from the format used for PDP-11 programs. It has been upgraded to handle Z8, Z80, and segmented and nonsegmented Z8000 machine code. It also is referred to as "s.out" in several places, to distinguish it from the PDP-11 format. Layout information as given in the include file is:

```
/* The object module header structure is as defined below */
/* for S.OUT object file formats. */

    struct s_exec {
      int      s_magic;   /* Magic number */
      long     s_imsize;  /* Size of memory image section */
      long     s_bss;     /* Size of bss sections */
      unsigned s_segt;    /* Size of segment table section */
      unsigned s_syms;    /* Size of symbol table section */
      long     s_entry;   /* Entry point address */
      unsigned s_flag;    /* Flags word */
      unsigned s_codesz;  /* 8-bit padded code size */
      unsigned s_unused;  /* Unused */
    };

/*  The valid magic numbers for the s.out are defined . */
/*  below. Valid a.out magic numbers ( defined in       */
/*  a.out.h ) are : e807, e810, and e811                */

#define   S_MAGIC1  0xE607  /* Segmented executable */
#define   S_MAGIC3  0xE611  /* Segmented separate I & D */
#define   S_MAGIC4  0xE605  /* Segmented overlay */
#define   N_MAGIC1  0xE707  /* Nonsegmented executable */
#define   N_MAGIC3  0xE711  /* Nonsegmented separate I & D */
#define   N_MAGIC4  0xE705  /* Nonsegmented overlay file */
#define   X_MAGIC1  0xE507  /* 8-bit executable */
#define   X_MAGIC3  0xE511  /* 8-bit separate I & D */
#define   X_MAGIC4  0xE505  /* 8-bit overlay */

/* Valid flags in the s_flag field are defined as follows:  */

#define   SF_STRIP  0x0001  /* Reloc info has been stripped */
#define   SF_OPREP  0x0002  /* Changed by OPREP program */
#define   SF_Z8     0x0004  /* Z8 program */
#define   SF_Z80    0x0080  /* Z80 program */
```

```
#define   SF_SEND   0x0010  /* module created by SEND */

/* The segment table is an array of the following structure.  *
/* Each entry relates to one segment in the program.  For file*
/* types with  magic numbers S_MAGIC1 and S_MAGIC3, there are *
/* 128  entries  maximum.  For file types S_MAGIC4, there  are*
/* 256  entries  maximum.    For  the remaining  file  types, *
/* there is always one entry in the segment table.            *

    struct segt {
      char     sg_segno;  /* Segment number */
      char     sg_coff;   /* Offset/256 for code section */
      char     sg_doff;   /* Offset/256 for data section */
      char     sg_boff;   /* Offset/256 for bss section */
      unsigned sg_code;   /* Size of the code portion */
      unsigned sg_data;   /* Size of the data portion */
      unsigned sg_bss;    /* Size of the bss portion */
      int      sg_atr;    /* Attributes, defined below */
      long     sg_unused; /* Unused */
    };

/* These definitions apply to the sg_atr field in the */
/* segment table*/

#define   SG_CODE   0x0001  /* Segment contains code */
#define   SG_DATA   0x0002  /* Contains initialized data */
#define   SG_BSS    0x0004  /* Contains uninitialized data */
#define   SG_STAK   0x0008  /* The segment grows downward */
#define   SG_OCODE  0x0010  /* Code section is offset */
#define   SG_ODATA  0x0020  /* Data section is offset */
#define   SG_OBSS   0x0040  /* Bss section is offset */
#define   SG_BOUND  0x0080  /* Segment number bound to */
                                  /* this section */

/* The symbol  table is an  array containing entries  with  */
/* information on symbols.  All entries are fixed size with */
/* the symbol  having <= 8  characters.   If a  longer name */
/* is used, the first character of the name has the high bit*/
/* set and is interpreted as the length of the name rather  */
/* than a character.  The "overflow" characters are put into*/
/* the following storage areas and padded out with zeros to */
/* fill a  symbol table entry  exactly.  Names shorter than */
/* eight characters are also padded with zeros.             */

#define   SHORTNAME 8       /* Length of normal sized name */
#define   LONGNAME 127      /* Max characters in a long name */

    struct s_nlist {
      long     sn_value;       /* Value */
      char     sn_type;        /* Type field */
      char     sn_segt;        /* Segment table entry # */
      char     sn_name[SHORTNAME]; /* Symbol name padded with 0
```

```
    };

/* These definitions apply to the sn_type field in the */
/* symbol table */

#define   N_UNDF    Ø        /* Undefined */
#define   N_ABS     Øx01     /* Absolute */
#define   N_DATA    Øx03     /* Data symbol */
#define   N_BSS     Øx04     /* Bss symbol */
#define   N_TYPE    Øx1F     /* Mask for type */
#define   N_REG     Øx14     /* Register name */
#define   N_FN      Øx1F     /* File name */
#define   N_EXT     Øx2Ø      /* External bit, or'ed in */

#define   N_CODE    Øx02     /* Code symbol */
#define   N_SN      Øx1E     /* Section name */
#define   N_SEG     Øx4Ø      /* Segmented bit, or'ed in */
#define   LONGMASK Øx007F    /* Strip off long bit */

/* Complete s.out header structure */

struct s_head {
    struct s_exec   s_exc;       /* s.out header */
    struct segt     segtable[2]; /* nonsegmented seg table */
};
```

The file has five sections: a header, the segment table, the program and data text, relocation information, and a symbol table (in that order). The last two may be empty if the program was loaded with the "-s" option of **ld**(1) or if the symbols and relocation have been removed by **strip**(1).

In the header the sizes of each section are given in bytes. The sizes of the code and data in the file are always even. If a size in the header is odd, the real size is the rounded up size. For example, a size of hexadecimal FFFF in the sg_code field of a segment table entry implies a real size of hexadecimal 10000. The size of the header is not included in any of the other sizes.

For 8-bit processors (X_MAGIC1, X_MAGIC3, and X_MAGIC4), the memory image in the object file can be padded for word alignment. In this case the s_imsize field differs from the size of the resulting object code and cannot be used to determine memory requirements. For 8-bit separate I & D files, the padded code size can be obtained from the s_codesz field. The memory requirement information must be obtained from the segment table.

This format can be used for files intended to run on the ZEUS system, and for programs that are to be downloaded to target hardware. When an **a.out** file is loaded into memory

by ZEUS for execution, three logical segments are set up: the text segment, the data segment (with uninitialized data, which starts off as all Ø, following initialization), and a stack. The text segment begins at Ø in the image; the header is not loaded. If the magic number in the header is N_MAGIC1, it indicates that the text segment is not to be write-protected and shared, so the data segment is immediately contiguous with the text segment. If the magic number is N_MAGIC3, the text segment is pure, write-protected, and shared, and instruction and data space are separated; the text and data segment both begin at location Ø. If the magic number is N_MAGIC4, the text segment is overlaid on an existing (N_MAGIC3 or N_MAGIC4) text segment and the existing data segment is preserved. For segmented files, the above holds true except that there can be more than one segment in each category (text, data, and bss).

The stack will occupy the highest possible locations in the image: from FFFE(hex) and growing downwards. The stack is automatically extended as required. Data segments are only extended as requested by **brk**(2) or **sgbrk**(2).

The start of the segment table in the file is 18(hex); the start of the text segment is 18+Ss (the size of the segment table); the start of the data segment is 18+Ss+St (the size of the text); the start of the relocation information is 18+Ss+St+Sd (the size of the memory image); the start of the symbol table is 18+Ss+2*(St+Sd) if the relocation information is present, 18+Ss+St+Sd if not.

The layout of a symbol table entry and the principal flag values that distinguish symbol types are given in the include file. Other flag values can occur if an assembly language program defines machine instructions.

If the first character of a symbol name has a value greater than 127 (its high order bit 1), the symbol name is longer than eight characters. The low order seven bits of this byte are taken as the length of the name, including the length byte. A long name always occupies an integral number of s_nlist entries. This means that a long name can be padded out with zeros until the next entry is filled. The pad bytes are not included in the length.

If a symbol's type is undefined external, and the value field is non-zero, the symbol is interpreted by the loader **ld**(1) as the name of a common region whose size is indicated by the value of the symbol.

The value of a word in the text or data portions which is not a reference to an undefined external symbol is exactly that value which will appear in memory when the file is

executed. If a word in the text or data portion involves a reference to an undefined external symbol, as indicated by the relocation information for that word, then the value of the word as stored in the file is an offset from the associated external symbol. When the file is processed by the loader and the external symbol becomes defined, the value of the symbol will be added into the word in the file.

If relocation information is present, it amounts to one word per word of program text or initialized data. There is no relocation information if the "relocation info stripped" flag in the header is on.

Each relocation word is interpreted as follows:

Relocation bits for Z8000 object files

```
Bit 3 = 1: External reference
  Bits 2-0:
    000    offset part (or other 16 bit value)
    001    segment part
    010    short segmented address
    011    12 bit relative address (CALR)
    100    16 bit relative address (LDR, LDAR)
  Bits 15-4:
    Symbol number starting from 0

Bit 3 = 0: Resolved reference
  Bits 2-1:
    00     absolute
    01     code reference
    10     data reference
    11     bss reference
  Bit 0 = 1: segmented reference
    Bits 15-8:
                segment table entry #
    Bits 6-4:
      000    offset part (or other 16 bit value)
      001    segment part
      010    short segmented address
      011    12 bit relative address (CALR)
      100    16 bit relative address (LDR, LDAR)
  Bit 0 = 0: non-segmented reference
    Bits 15-4:
      unused
```

Relocation bits for 8-bit object files

```
Bit 3 = 1: External reference
  Bits 2-0:
    000    word value
    001    byte value
```

```
        Ø1Ø     high byte of word value
        Ø11     low byte of word value
      Bits 15-4:
        Symbol number starting from Ø

    Bit 3 = Ø: Resolved reference
      Bits 2-Ø:
        ØØØ      absolute
        Ø1Ø      code reference
        1ØØ      data reference
        11Ø      bss reference
      Bits 6-4:
        ØØØ     word value
        ØØ1     byte value
        Ø1Ø     high byte of word value
        Ø11     low byte of word value
      Bits 15-7:
          unused
```

SEE ALSO

as(1), ld(1), nm(1), objdu(1), objhdr(1).

NAME

acct - per-process accounting file format

SYNOPSIS

#include <sys/acct.h>

DESCRIPTION

Files produced as a result of calling acct(2) have records in the form defined by **<sys/acct.h>**, whose contents are:

```
/* @[$]acct.h    2.4  03/07/83 17:56:39 - Zilog Inc */
/*
 * Accounting structures
 */

typedef    ushort comp_t;        /* "floating point" */
        /* 13-bit fraction, 3-bit exponent  */

struct    acct
{
    char      ac_flag;      /* Accounting flag */
    char      ac_stat;      /* Exit status */
    ushort    ac_uid;       /* Accounting user ID */
    ushort    ac_gid;       /* Accounting group ID */
    dev_t     ac_tty;       /* control typewriter */
    time_t    ac_btime;     /* Beginning time */
    comp_t    ac_utime;     /* acctng user time in clock ticks
    comp_t    ac_stime;     /* acctng system time in clock tick
    comp_t    ac_etime;     /* acctng elapsed time in clock tic
    comp_t    ac_mem;       /* memory usage */
    comp_t    ac_io;        /* chars transferred */
    comp_t    ac_rw;        /* blocks read or written */
    char      ac_comm[8];   /* command name */
};

extern  struct  acct  acctbuf;
extern  struct  inode *acctp;  /* inode of accounting file */

#define  AFORK  01        /* has executed fork, but no exec */
#define  ASU    02        /* used super-user privileges */
#define    ACCTF    0300        /* record type: 00 = acct */
```

In ac flag, the AFORK flag is turned on by each **fork**(2) and turned off by an **exec**(2). The ac comm field is inherited from the parent process and reset by any exec. Each time the system charges the process with a clock tick, it also adds the current process size to ac mem, computed as follows:

(data size) + (text size) / (number of in-core

processes using text)

The value of ac mem/ac stime can be viewed as an approximation to the mean process size, as modified by text-sharing.

The following structure represents the total accounting format used by the various accounting commands:

```
/*
 *    total accounting (for acct period), also for day
 */

struct    tacct    {
    uid_t       ta_uid;        /* userid */
    char        ta_name[8];    /* login name */
    float       ta_cpu[2];     /* cum. cpu time, p/np (mins) */
    float       ta_kcore[2];   /* cum kcore-minutes, p/np */
    float       ta_con[2];     /* cum. connect time, p/np, mins *,
    float       ta_du;         /* cum. disk usage */
    long        ta_pc;         /* count of processes */
    unsigned short    ta_sc;    /* count of login sessions */
    unsigned short    ta_dc;    /* count of disk samples */
    unsigned short    ta_fee;   /* fee for special services */
};
```

SEE ALSO

acctcom(1), acct(2), acct(M).

LIMITATIONS

The ac mem value for a short-lived command gives little information about the actual size of the command, because ac mem can be incremented while a different command (e.g., the shell) is being executed by the process.

NAME

ar - archive (library) file format

SYNOPSIS

#include <ar.h>

DESCRIPTION

The archive command **ar** is used to combine several files into one. Archives are used mainly as libraries to be searched by the link-editor **ld.**

A file produced by **ar** has a magic number at the start, followed by the constituent files, each preceded by a file header. The magic number and header layout as described in the include file are:

```
/* @[$]ar.h    1.1  12/11/81 17:07:41 - Zilog Inc */
#define   ARMAG     0177545
struct    ar_hdr {
     char ar_name[14];
     long ar_date;
     char ar_uid;
     char ar_gid;
     int  ar_mode;
     long ar_size;
};
```

The name is a null-terminated string; the date is in the form of **time**(2); the user ID and group ID are numbers; the mode is a bit pattern per **chmod**(2); the size is counted in bytes.

Each file begins on a word boundary; a null byte is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

Notice there is no provision for empty areas in an archive file.

SEE ALSO

ar(1), ld(1), nm(1).

LIMITATIONS

Coding user and group IDs as characters is a botch.

NAME

core - format of core image file

DESCRIPTION

ZEUS writes a memory image of a terminated process when any error occurs. (See **signal**(2) for the list of reasons.) The most common errors are memory violations, illegal instructions, and user-generated quit signals. The memory image, called core, is written in the process's working directory, provided normal access controls apply.

The first 2Ø48 bytes of the memory image are a copy of the system's per-user data for the process, including the registers as they were at the time of the fault. The remainder represents the actual contents of the user's memory when the image was written. If a user program is separate I & D, the code segment(s) are not dumped, just the data segment(s). In the combined I & D case, all user segments are dumped. In both cases the user's stack appears in the core file after all the user segments.

The debugger **adb**(1) is sufficient to deal with memory images.

SEE ALSO

adb(1), signal(2).

LIMITATIONS

The name core is a throwback to an earlier era.

NAME

cpio - format of cpio archive

DESCRIPTION

The header structure, when the **c** option is not used, is:

```
struct {
        short     h_magic,
                  h_dev,
                  h_ino,
                  h_mode,
                  h_uid,
                  h_gid,
                  h_nlink,
                  h_rdev,
                  h_mtime[2],
                  h_namesize,
                  h_filesize[2];
        char h_name[h_namesize rounded to word];
} Hdr;
```

When the **c** option is used, the header information is described by the statement below:

```
sscanf(Chdr,"%6o%6o%6o%6o%6o%6o%6o%6o%11o%6o%6o%s",
        &Hdr.h_magic,&Hdr.h_dev,&Hdr.h_ino,&Hdr.h_mode,
        &Hdr.h_uid,&Hdr.h_gid,&Hdr.h_nlink,&Hdr.h_rdev,
        &Longtime,&Hdr.h_namesize,&Longfile,Hdr.h_name);
```

Longtime and Longfile are equivalent to Hdr.h_mtime and Hdr.h_filesize, respectively. The contents of each file is recorded in an element of the array of varying length structures, archive, together with other items describing the file. Every instance of h_magic contains the constant 070707 (octal). The items h_dev through h_mtime have meanings explained in **stat**(2). The length of the null-terminated path name h_name, including the null byte, is given by h_namesize.

The last record of the archive always contains the name TRAILER!!!. Special files, directories, and the trailer are recorded with h_filesize equal to zero.

SEE ALSO

cpio(1), find(1), stat(2).

NAME

cshrc, cshprofile, login - setting up an environment at login time

DESCRIPTION

Users of the C-shell, **csh,** can set up a consistent environment before a session begins. If the home directory contains a file named .login, that file will be executed (via the C-shell's source command) only at login time. If a second file, .cshrc, exists in the home directory, this file also will be executed. The .cshrc file will be executed before the .login file. The difference is that the .cshrc file is executed by every shell, not just the login shell.

If the file /etc/cshprofile exists, it will be executed for every C-shell user at login before either the .login or the .cshrc are executed.

FILES

$home/.cshrc
$home/.login
/etc/cshprofile

SEE ALSO

csh(1), login(1), printenv(1), sh(1), su(1), profile(5), environ(7), term(7).

NAME

dir - format of directories

SYNOPSIS

#include <sys/dir.h>

DESCRIPTION

A directory behaves exactly like an ordinary file, save that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its i-node entry see, **filsys**(5). The structure of a directory entry as given in the include file is:

```
/* @[$]dir.h    2.1  05/13/82 13:50:31 - Zilog Inc */
#ifndef DIRSIZ
#define DIRSIZ  14
#endif
struct  direct
{
        ino_t   d_ino;
        char    d_name[DIRSIZ];
};
```

By convention, the first two entries in each directory are for `.' and `..'. The first is an entry for the directory itself. The second is for the parent directory. The meaning of `..' is modified for the root directory of the master file system and for the root directories of removable file systems. In the first case, there is no parent, and in the second, the system does not permit off-device references. Therefore in both cases `..' has the same meaning as `.'.

SEE ALSO

filsys(5).

NAME

dump, ddate - incremental dump format

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ino.h>
#include <dumprestor.h>
```

DESCRIPTION

Tapes used by **dump** and **restor**(M) contain:

- a header record
- two groups of bit map records
- a group of records describing directories
- a group of records describing files

The format of the header record and of the first record of each description as given in the include file <dumprestor.h> is:

```
#define NTREC     8
#define MLEN      16
#define MSIZ      4096

#define TS_TAPE   1
#define TS_INODE  2
#define TS_BITS   3
#define TS_ADDR   4
#define TS_END    5
#define TS_CLRI   6
#define MAGIC     (int)60011
#define CHECKSUM  (int)84446
struct            spcl
{
     int          c_type;
     time_t       c_date;
     time_t       c_ddate;
     int          c_volume;
     daddr_t      c_tapea;
     ino_t        c_inumber;
     int          c_magic;
     int          c_checksum;
     struct       dinode  c_dinode;
     int          c_count;
     union
     {
                  struct
                  {
                  char c_dev[30];
```

```
                char c_string[BSIZE-30];
                }c_dinfo;
                char    c_addr[BSIZE];
        }c_block;
} spcl;

struct          idates
{
        char    id_name[16];
        char    id_incno;
        time_t  id_ddate;
};
```

NTREC is the number of 512 byte records in a physical tape block. MLEN is the number of bits in a bit map word. MSIZ is the number of bit map words.

The TS_ entries are used in the c_type field to indicate header type. The types and their meanings are as follows:

TS_TAPE Tape volume label

TS_INODE
: A file or directory follows. The c_dinode field is a copy of the disk inode and contains bits telling file type.

TS_BITS A bit map follows. This bit map has a one bit for each inode that was dumped.

TS_ADDR A subrecord of a file description. See the following c_addr entry.

TS_END End of tape record.

TS_CLRI A bit map follows. This bit map contains a zero bit for all inodes that were empty on the file system when dumped.

MAGIC All header records have this number in c_magic.

CHECKSUM
: Header records checksum to this value.

The fields of the header structure are as follows:

c_type The type of the header.

c_date The date the dump was taken.

c_ddate The date the file system was dumped from.

c_volume The current volume number of the dump.

c_tapea The current number of this 512-byte record.

c_inumber
: The number of the inode being dumped if this is of type TS_INODE.

c_magic This contains the value MAGIC above, truncated as needed.

c_checksum
: This contains whatever value is needed to make the record sum to CHECKSUM.

c_dinode This is a copy of the inode as it appears on the file system (filsys (5)).
c_count The count of characters in c_addr.
c_addr An array of characters describing the blocks of the dumped file. A character is zero if the block associated with that character was not present on the file system; otherwise, the character is non-zero. If the block was not present on the file system, no block was dumped; the block is restored as a hole in the file. If there is not sufficient space in this record to describe all of the blocks in a file, TS_ADDR records are scattered through the file, each one picking up where the last left off.

Each volume except the last ends with a tapemark (read as an end of file). The last volume ends with a TS_END record and then the tapemark.

The structure idates describes an entry of the file /etc/ddate where dump history is kept. The fields of the structure are:

id_name The dumped filesystem is /dev/id_nam.
id_incno The level number of the dump tape; (dump (M)).
id_ddate The date of the incremental dump in system format (types (5)).

FILES

/etc/ddate

SEE ALSO

dump(M), dumpdir(M), restor(M), filsys(5), types(5).

NAME

environ - user environment

SYNOPSIS

extern char **environ;

DESCRIPTION

An array of strings called the environment is made available by **exec**(2) when a process begins. These strings have the form name=value. **csh**(1) and **vi**(1) internalize environment variables and display them via the set command. The following names are used by various commands:

PATH The sequence of directory prefixes that **sh**(1), **time**(1), **nice**(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by :. **login**(1) sets PATH=:/bin:/usr/bin. **csh**(1) uses the variable path.

HOME A user's login directory, set by **login** from the password file **passwd**(5). **csh** uses the variable home.

TERM The kind of terminal for which output is to be prepared. This information is used by commands, such as **nroff**(1) that exploit special terminal capabilities. **login** sets TERM based on entries found in /etc/ttytype. See **term**(7) for a list of terminal types.

Further names are placed in the environment by the export command and name=value arguments in **sh**(1), or by **exec**(2). Do not conflict with certain shell variables that are frequently exported by .profile files: MAIL, PS1, PS2, IFS.

SEE ALSO

csh(1), login(1), sh(1), exec(2), ttytype(5), term(7), termlist(7).

NAME

filsys, flblk, ino - format of file system volume

SYNOPSIS

```
#include <sys/param.h>
#include <sys/types.h>
#include <sys/fblk.h>
#include <sys/filsys.h>
#include <sys/ino.h>
```

DESCRIPTION

Every file system storage volume has a common format for certain vital information. Block Ø is 512-bytes and contains the bootstrap program.

Each virtual file system on disk is divided into a certain number of 512-byte blocks. Block 1 is the super block. The layout of the super block as defined by the include file <sys/filsys.h> is:

```
/*
 * Structure of the super-block
 */
struct   filsys {
   unsigned short s_isize;    /* size in blocks of i-list */
   daddr_t    s_fsize;        /* size in blocks of entire volume
   short      s_nfree;        /* number of addresses in s_free *,
   daddr_t    s_free[NICFREE];/* free block list */
   short      s_ninode;       /* number of i-nodes in s_inode */
   ino_t      s_inode[NICINOD];/* free i-node list */
   char       s_flock;        /* lock during free list manipulat:
   char       s_ilock;        /* lock during i-list manipulation
   char       s_fmod;         /* super block modified flag */
   char       s_ronly;        /* mounted read-only flag */
   time_t     s_time;         /* last super block update */
   daddr_t    s_tfree;        /* total free blocks*/
   ino_t      s_tinode;       /* total free inodes */
   short      s_m;            /* interleave factor */
   short      s_n;            /* " " */
   char       s_fname[6];     /* file system name */
   char       s_fpack[6];     /* file system pack name */
   char    s_mach;            /* byte order flag (Ø = native, */
                              /*    1 = foreign) */
};
```

S_isize is the address of the first block after the i-list, that starts after the super block, in block 2. Thus the i-list is s_isize-2 blocks long. S_fsize is the address of the first block not potentially available for allocation to a file. These numbers are used by the system to check for bad block addresses; if an impossible block address is allocated from the free list or is freed, a diagnostic is

written on the on-line console. The free array is cleared, to prevent further allocation from a corrupted free list.

The free list for each volume is maintained as follows:

The s_free array contains, in s_free[1], ..., s_free[s_nfree-1], up to NICFREE free block numbers. NICFREE is a configuration constant defined to be 50. S_free[0] is the block address of the head of a chain of blocks constituting the free list. The layout of each block of the free chain as defined in the include file <sys/fblk.h> is:

```
struct fblk
{
     int       df_nfree;
     daddr_t   df_free[NICFREE];
};
```

The fields df_nfree and df_free in a free block are used exactly like s_nfree and s_free in the super block. To allocate a block, decrement s_nfree, and the new block number is s_free[s_nfree]. If the new block address is 0, there are no blocks left, so give an error. If s_nfree became 0, read the new block into s_nfree and s_free. To free a block, check if s_nfree is NICFREE; if so, copy s_nfree and the s_free array into it, write it out, and set s_nfree to 0. In any event, set s_free[s_nfree] to the freed block's address and increment s_nfree.

S_ninode is the number of free i-numbers in the s_inode array. To allocate an i-node, if s_ninode is greater than 0, decrement it and return s_inode[s_ninode]. If it was 0, read the i-list on disk and place the numbers of all free inodes (up to NICINOD) into the s_inode array, then try again. (NICINOD is a configuration constant equal to 100.) To free an i-node, provided s_ninode is less than NICINODE, place its number into s_inode[s_ninode] and increment s_ninode. If s_ninode is already NICINODE, do not enter the freed i-node into any table. This list of i-nodes is only to speed up the allocation process; the information as to whether the inode is really free or not is maintained in the inode itself.

S_flock and s_ilock are flags maintained in the core copy of the file system while it is mounted and their values on disk are immaterial. The value of s_fmod on disk is likewise immaterial; it is used as a flag to indicate that the super block has changed and should be copied to the disk during the next periodic update of file system information. S_ronly is a write-protection indicator; its disk value is also immaterial.

S time is the last time the super block of the file system was changed. During a reboot, s time of the super block for the root file system sets the system's time.

The fields s tfree, s tinode, s fname and s fpack are not currently maintained.

I-numbers begin at 1, and the storage for i-nodes begins in block 2. I-nodes are 64 bytes long, so eight of them fit into a block. I-node 2 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each i-node represents one file. The format of an i-node as given in the include file <sys/ino.h> is:

```
/*
 * Inode structure as it appears on
 * a disk block.
 */
struct dinode
{
     unsigned short di_mode;       /* mode and type of file */
     short     di_nlink;       /* number of links to file */
     short     di_uid;         /* owner's user id */
     short     di_gid;         /* owner's group id */
     off_t     di_size;        /* number of bytes in file */
     char      di_addr[40];    /* disk block addresses */
     time_t    di_atime;       /* time last accessed */
     time_t    di_mtime;       /* time last modified */
     time_t    di_ctime;       /* time created */
};
#define   INOPB     8    /* 8 inodes per block */
/*
 * the 40 address bytes:
 *   39 used; 13 addresses
 *   of 3 bytes each.
 */
```

Di mode tells the kind of file; it is encoded identically to the st mode field of **stat**(2). Di nlink is the number of directory entries (links) that refer to this i-node. Di uid and di gid are the owner's user and group IDs. Size is the number of bytes in the file. Di atime and di mtime are the times of last access and modification of the file contents (read, write or create) (**times** (2)); Di ctime records the time of last modification to the inode or to the file, and determines whether it should be dumped.

Special files are recognized by their modes and not by their i-number. A block-type special file is one that can potentially be mounted as a file system; a character-type special file cannot, though it is not necessarily character-oriented. For special files, the di addr field is occupied

by the device code (**types** (5)). The device codes of block and character special files overlap.

Disk addresses of plain files and directories are kept in the array di addr packed into three bytes each. The first 10 addresses specify device blocks directly. The last three addresses are singly, doubly, and triply indirect and point to blocks of 128 block pointers. Pointers in indirect blocks have the type daddr t (**types** (5)).

For block b in a file to exist, it is not necessary that all blocks less than b exist. A zero block number either in the address words of the i-node or in an indirect block indicates that the corresponding block has never been allocated. Such a missing block reads as if it contained all zero words.

SEE ALSO

stat(2), dir(5), types(5), dcheck(M), icheck(M), mount(M).

NAME

group - group file

DESCRIPTION

Group contains for each group the following information:

- group name
- encrypted password
- numerical group ID
- a comma separated list of all users allowed in the group

This is an ASCII file. The fields are separated by colons; Each group is separated from the next by a new-line. If the password field is null, no password is demanded.

This file resides in directory /etc. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group ID's to names.

FILES

/etc/group

SEE ALSO

newgrp(1), passwd(1), crypt(3), passwd(5).

NAME

inittab - control information for init

DESCRIPTION

When a state is entered, **init** reads the file /etc/inittab. Lines in this file have the format:

state:id:flags:command

All lines in which the state field match **init**'s current state are recognized. If a process is active under the same two character id as a recognized line, it may be terminated (signal 15), killed (signal 9), or both by including the flags **t** and **k** in the order desired. The signal is sent to all processes in the process group associated with the id. The command field is saved for later execution. The flag **c** requires the command to be continuously reinvoked whenever the process with that id dies. Otherwise the command is invoked a maximum of one time in the current state.

FILES

/etc/inittab

NAME

mnttab - mounted file system table

SYNOPSIS

```
struct mnttab {
        char mt_dev[10];
        char mt_filsys[10];
        short     mt_ro_flg;
        time_t    mt_time;
};
```

DESCRIPTION

Mnttab resides in directory /etc and contains a table of devices mounted by the **mount**(M) command.

Each entry is 26 bytes in length. The first 10 bytes are the null-padded name of where the special file is mounted; the next 10 bytes represent the null-padded root name of the mounted special file; the remaining 6 bytes contain the mounted special file's read/write permissions and date it was mounted.

The maximum number of entries in mnttab is based on the system parameter **Nmount** located in the kernel, which defines the number of allowable mounted special files.

SEE ALSO

mount(M).

NAME

mon.out - profile information

DESCRIPTION

Mon.out is a file containing profiling information. This information is produced by either **monitor**(3) or **segmon**(3), depending on whether the user program was compiled nonsegmented or segmented. Mon.out has two formats, one for nonsegmented code and one for segmented code.

The format produced by **monitor**(3) is a file divided into 3 sections - a header, the routine counts, and the timing information. The header is a fixed structure, whereas the other sections vary in size depending on the number of routines profiled and the size of the sample range.

```
    typedef short UNIT;
/* structure of the header */
    struct hdr {        /* header information  */
        UNIT *lowpc;    /* low end of range  */
        UNIT *highpc;   /* high end of range */
        int  ncount;    /* number of routines counted */
            };

/* strucure used for the routine counts */
    struct cnt {            /* routine information */
        unsigned cvalue;    /* number of times routine*/
                            /* is called */
        long cncall;        /* address to the routine */
                            /* being counted */
                } cbuf[350];
```

Although cbuf has a maximum size of 350 structures, the number of structures is determined by the number of routines counted (ncount). The rest of the file is timing information, read one UNIT at a time until the end of file is reached.

The format produced by **segmon**(3) is a bit different from the above format. This is because the previous format is a profile of only 1 segment and 1 sample range whereas this format must handle multiple segments and multiple sample ranges. The file consists of routine counts, segment headers, and timing information. The routine count section consists of 2 parts a header and the actual count.

```
/* structure for the routine header*/
    struct rhdr {       /* header information    */
        long seghdr;    /* offset where seg headers start */
        long size;      /* size of rountine section and */
                        /*  seg headers section in bytes */
        long ncount;    /* number of routines    */
```

```
        long numseg;    /* number of segments     */
        }
/* structure for the routine counts */
    struct cnt  {       /* area for the routine counts  */
        unsigned long cvalue; /* no. of times routine */
                        /* is called */
        long cncall;    /* address of routine */
        } cbuf[350];
```

As in the above format the amount of information in cbuf is determined by ncount. The structure used for the segment header is:

```
    struct shdr {     /* segment header      */
        long lowpc;   /* low end for range   */
        long highpc;  /* high end for range  */
        long scale;   /* scale of timing     */
        long segnum;  /* segment number      */
        }
```

The number of segment headers depends on the number of segments being profiled. The timing section is read a UNIT at a time until the end of file is reached.

EXAMPLE

If there were 3 segments and 10 routines profiled, the layout would appear as:

```
routine header (rhdr)
routine information (cbuf[0])
        .
        .
routine information (cbuf[9])
segment    header 1 (shdr)
segment header 2
segment header 3
timing information for seg header 1
(length determined by highpc - lowpc)
    (of segment header 1)
timing information block for seg 2
timing information block for seg 3
```

SEE ALSO

prof(1), sprof(1), monitor(1), segmon(3), sprofil(2), profil(2), cc(1), scc(1).

NAME

motd - message of the day file

DESCRIPTION

Motd contains text that is printed on the terminal as a person logs in (.BR login (1)). The file should be readable and writable by everyone. A specific format is suggested here. The file contains brief announcements of important, impending events, such as system availability or large-scale changes. Next is a line introducing news items (**news**(1)). Following that is one line for each news item. Each line contains the date, title, and short description for each news item. The last line indicates how to access a news item. An example:

```
System down between 12 and 1 today for maintenance

The following are news items of current interest:

3/14    news    How to use the news program
3/12    meeting Agenda for Friday group meeting

For information on any news item type: news item
```

This file resides in directory /etc.

FILES

/etc/motd

SEE ALSO

login(1), news(1).

NAME

passwd - password file

DESCRIPTION

Passwd contains for each user the following information:

name (login name, contains no uppercase)

encrypted password

numerical user ID

numerical group ID

optional

initial working directory

program to use as shell

This is an ASCII file. Each field within each user's entry is separated from the next by a colon. The optional field can contain any desired information. Each user is separated from the next by a new line. If the password field is null, no password is demanded; if the shell field is null, the shell itself (**sh**(1)) is used.

This file resides in directory /etc. Because of the encrypted passwords, it has general read permission and can be used, for example, to map numerical user ID's to names.

The encrypted password consists of 13 characters chosen from a 64 character alphabet (**.**, **/**, **Ø-9**, **A-Z**, **a-z**), except when the password is null in which case the encrypted password is also null. Password aging is effected for a particular user if his encrypted password in the password file is followed by a comma and a non-null string of characters from the above alphabet. (Such a string must be introduced in the first instance by the super-user.) The first character of the age, M say, denotes the maximum number of weeks for which a password is valid. A user who attempts to login after his password has expired will be forced to supply a new one. The next character, m say, denotes the minimum period in weeks which must expire before the password may be changed. The remaining characters define the week (counted from the beginning of 197Ø) when the password was last changed. (A null string is equivalent to zero.) M and m have numerical values in the range Ø-63. If m = M = Ø (derived from the string . or ..) the user will be forced to change his password the next time he logs in (and the ``age'' will disappear from his entry in the password file). If m > M

(signified, e.g., by the string ./) only the super-user will be able to change the password.

FILES

/etc/passwd

SEE ALSO

login(1), passwd(1), getpwent(3), crypt(3), group(5).

NAME

remotelines, LCK - remote line control and lock files

DESCRIPTION

Remote is used to transfer control to a remote system running ZEUS or UNIX software. The system administrator is responsible for creating and maintaining the file "/usr/spool/uucp/remotelines" which contains the information detailing what systems are available and which devices are to be used to connect to them. This file has one line for every remote system known to **remote**(1).

Each line contains two or more entries. The first entry is the system name string that **remote**(1) recognizes when it is invoked. The second entry is the name of the device (usually a tty line) that connects to the remote system. The third entry is an optional baud rate (the default is 9600). The entries are separated by one tab or space. The first line in the file will become the default if no system name is given. An example would be:

```
pdp11 /dev/tty3
rzeus /dev/tty4 4800
```

which defines two remote systems, a PDP 11, called `pdp11', accessed through /dev/tty3 and a remote zeus system, known as `rzeus', accessed through /dev/tty4 at 4800 baud. If no system name was given remote(1) would default to `pdp11'. Note: it is necessary that the remotelines file be readable by everyone, so a mode of 0664 (see chmod(1)) is required.

When remote(1) is invoked, it scans the remotelines file and determines the entry to be used. It then checks to see if a lock file is present for each device for the desired system, stopping when it finds a free one. The name of the lock file is composed of the string "/usr/spool/uucp/LCK.." appended with the name of the device to be used (all leading path components removed). If this file is not present it is assumed that the remote line is not in use. **Remote**(1) then creates the file to lock out other requests for the line. It then proceeds to access the proper device and begin the remote operation. The system administrator is responsible for making sure all devices defined as remote lines are connected to other machines. It is also necessary to make the line accessible to everyone by changing the mode to 0666 (see **chmod**(1)) and disabling it in /etc/inittab (see **inittab**(5)).

FILES

/usr/spool/uucp/LCK..*	lock files
/usr/spool/uucp/remotelines	database of systems

SEE ALSO

remote(1), putfile(1), getfile(1), local(1).

NAME

sccsfile - format of SCCS file

DESCRIPTION

An SCCS file is an ASCII file. It consists of six logical parts: the checksum, the delta table (information about each delta), user names (login names and/or numerical group IDs of users who add deltas), flags (definitions of internal keywords), comments (arbitrary descriptive information about the file), and the body (the actual text lines intermixed with control lines).

Throughout an SCCS file there are lines beginning with the **ASCII SOH** (start of heading) character (octal 001). This character is hereafter referred to as the control character and is represented graphically as **@**. Any line described below which is not depicted as beginning with the control character is prevented from beginning with the control character.

Entries of the form DDDDD represent a five digit string (a number between 00000 and 99999).

Each logical part of an SCCS file is described in detail below.

Checksum

The first line of an SCCS file. The form of the line is:

@hDDDDD

The value of the checksum is the sum of all characters, except those of the first line. The @h provides a magic number of (octal) 064001.

Delta table

Consists of a variable number of entries of the form:

@s DDDDD/DDDDD/DDDDD
@d <type> <**SCCS ID**> yr/mo/da hr:mi:se <pgmr> DDDDD DDDDD
@i DDDDD ...
@x DDDDD ...
@g DDDDD ...
@m <MR number>
.
.
.
@c <comments> ...
.
.
.
@e

The first line (**@s**) contains the number of lines inserted/deleted/unchanged respectively. The second line (**@d**) contains the type of the delta (currently, normal: **D**, and removed: **R**), the SCCS ID of the delta, the date and time of creation of the delta, the login name corresponding to the real user ID at the time the delta was created, and the serial numbers of the delta and its predecessor, respectively.

The **@i**, **@x**, and @g lines contain the serial numbers of deltas included, excluded, and ignored, respectively. These lines are optional.

The **@m** lines (optional) each contain one MR number associated with the delta; the @c lines contain comments associated with the delta.

The @e line ends the delta table entry.

User names

The list of login names and/or numerical group IDs of users who may add deltas to the file, separated by new-lines. The lines containing these login names and/or numerical group IDs are surrounded by the bracketing lines **@u** and **@U**. An empty list allows anyone to make a delta.

Flags

Keywords used internally (see **admin**(1) for more information on their use). Each flag line takes the form:

```
@f <flag> <optional text>
```

The following flags are defined:

```
@f t <type of program>
@f v <program name>
@f i
@f b
@f m <module name>
@f f <floor>
@f c <ceiling>
@f d <default-sid>
@f n
@f j
@f l <lock-releases>
@f q <user defined>
```

b The **-b** keyletter can be used on the get command to cause a branch in the delta tree.

c Defines the "ceiling" release; the release above which no deltas may be added.

d Defines the default SID to be used when none is specified on a **get** command.

f Defines the "floor" release; the release below which no deltas may be added.

i Controls the warning/error aspect of the "No id keywords" message. When the **i** flag is not present, this message is only a warning; when the **i** flag is present, this message will cause a "fatal" error (the file will not be gotten, or the delta will not be made).

j Causes **get** to allow concurrent edits of the same base SID.

l Defines a list of releases that are locked against editing (**get**(1) with the -e keyletter).

m Defines the first choice for the replacement text of the **%M%** identification keyword.

n Causes **delta** to insert a "null" delta (a delta that applies no changes) in those releases that are skipped when a delta is made in a new release (e.g., when delta 5.1 is made after delta 2.7, releases 3 and 4 are skipped). The absence of the **n** flag causes skipped releases to be completely empty.

q Defines the replacement for the **%Q%** identification keyword.

t Defines the replacement for the **%Y%** identification keyword.

v Controls prompting for MR numbers in addition to comments; if the optional text is present it defines an MR number validity checking program.

Comments

Arbitrary text surrounded by the bracketing lines **@t** and **@T**. The comments section typically will contain a description of the file's purpose.

Body

The body consists of text lines and control lines. Text lines don't begin with the control character, control lines do. There are three kinds of control lines: insert, Idelete, and end, represented by:

```
@I DDDDD
@D DDDDD
@E DDDDD
```

respectively. The digit string is the serial number corresponding to the delta for the control line.

SEE ALSO

admin(1), delta(1), get(1), prs(1).

Source Code Control System User's Guide in the ZEUS Utilities Manual

NAME

spool - information for writing backends for the Zeus printer spooler

SYNOPSIS

#include <spool.h>

DESCRIPTION

This section gives information on how to write backends for the Zeus printer spooler.

Backends for the Zeus printer spooler are expected to copy data from RFFD to SFFD while doing any filtering required. When it is time to print something, dqueuer(1) invokes the backend, passing it status and printing information. Other than copying input to output, there is only one other condition that backends must handle. The backends do not read RFFD if a **-B** flag was passed to it by dqueuer. The dqueuer generates the **-B** flag when the file to be printed cannot be found. It is used by the dqueuer to cause system supplied backends to indicate 'file not found' on a banner page by itself. The backend handles flags and status passed to it by the dqueuer.

Backends are also used to generate banner pages and burst pages for easy separation and identification of print requests.

Backends expect the following flags to be passed as arguments when invoked. They are used for banner page information by the system backends.

-B	File not found.
-c x	Print file x number of times.
-d dest	Destination string
-f file	File name being printed.
-F from	'from' is the user making the request.
-s time	String with time that file was spooled
-t title	Title requested

Any extra options included in the configuration file are passed directly to the backend.

The spooling system also allows the backend to return certain statuses so that xq(1) can monitor its progress. This is done through the file on STATUSFD. The contents of this file has the structure of Dstat (in the next section). The only fields that the backend may access are Dpgsdone (number of pages printed) and Dpercentd (percentage of file printed) fields. Modification of other fields can cause unpredictable results. Use the OFFSET macro included in this file.

```
/* Macro definition to find the offset (in bytes)
of a certain element in a structure
```

```
*/
#define OFFSET(struct,item)
(off_t)((char *)&(struct.item) - (char *)&struct)
```

The fields Dpgsdone and Dpercentd are printed by xq(1). They are originally set to -1, indicating that the backend does not support these counters. No problems are caused by not including this support. Do not try to read or write any other fields of the Dstat structure. The spooler uses record locking extensively, and even reading a field can cause a deadlock situation.

Upon completion of processing, the backend exits. If the exit returns a non-zero value, the spooler assumes an error has occurred and tries to recover. If error logging is enabled, a message is written and the code is returned into the error file. It will then try to repeat the request. If the error occurs more than twice, the dqueuer sets the device status to DOWN. The Device may be reenabled by using the command

```
xq -q q:dev -Ud
```

where q:dev is replaced with the queue name and device number. See xq(M) for more details.

REQUEST and STATUS INFORMATION

The directory /usr/spool/queuer/requestdir holds files that contain request records. Each time **nq** is run, one file is created. Each file contains one or more request records, one for each file requested to be printed. The first field in the file is an off_t pointer to the first request record in the file. The remainder of the records are arranged as a link list. The structure definition is:

```
/* structure defining normal (print) requests */
struct normal {
   ippid   bepid;          /* pid of a backend if active */
   char    user[USERL];    /* owner of request */
   char    from[USERL];    /* loginid of user */
   int     uid;            /*  "    "     " */
   int     gid;            /*  "    "     " */
   char    qnam[QUEL];     /* queue for request */
   char    dnam[DEVL];     /* device for request */
   struct  opts noptions;
   int     seqno;
   int     pri;
   char    request[FILENL];/* file name of request */
   char    temp[FILENL];   /* file name if using cp */
   char    wdir[2*FILENL]; /* working directory */
   int     status;
   time_t  dsptime;        /* time dispatched */
   char    form[FORML];
```

```
long    length;
int     copies;
time_t  time;
char    dest[FILENL];   /* destination of output */
char    title[FILENL];  /* printed on banner page */
off_t   nextstruct;
 };
```

Qnam contains the queue desired. (If the default queue was requested, **nq** fills in the default name.) **Dnam** is the device desired. If the first available device is desired, this field is null.

Opts is a bit array that describes certain boolean options that the user requested. The definition of the bit array is:

```
/* define option struct */
struct opts {
        unsigned burst      : 1;
        unsigned copy       : 1;
        unsigned encrypt    : 1;
        unsigned mail       : 1;
        unsigned rm         : 1;
        unsigned silent     : 1;
       };
```

Seqn contains the sequence number of this request.

Status contains READY, HELD, or DISPCHD depending upon the status of this request. Form contains the string describing the special form needed for this request.

The directory /usr/spool/queuer/statusdir contains the status files for all queues and devices. The status file name for queues is just the name of the queue. For example, the name of the status file for a queue called lp would be /usr/spool/queuer/statusdir/lp. The format of the status file is defined by the following structure:

```
/* entries in queue status directory */
struct Qstat {
        int     Qstatus;
        int     Qpri;
        int     Qselect;
        };
```

The values for Qstatus, Qpri, and Qselect are defined by:

```
/* priority defines */
#define PRIR    0   /*  rush        */
#define PRIN    1   /*  normal      */
#define PRID    2   /*  deferred    */

/* selection criteria defines */
#define SELF    0   /*  fifo    */
#define SELS    1   /*  size    */

/* valid statuses for queues, devices, and requests
 don't use zero ... used for certain checks
*/
#define HELD    1
#define OFFLINE 2
#define DOWN    3
#define RUNNING 4
#define DISPCHD 5
#define READY   6
```

A description of the priority and selection criteria is given in the system administrator manual.

In addition to the queue status files, one file is created for each device known to the system. The name of the file is the queue name appended with a colon and then the device name. For example, the device 'mrm' on queue 'lp' would have status file called /usr/spool/queuer/statusdir/lp:mrm. The format of the status file is given by the following:

```
/* entries in device status directory */
struct Dstat {
        int     Dstatus;
        int     Duid;               /* UID of current request
        char    Dform[FORML];       /* current form on device
        time_t  last_dispatched;    /* last time we were
                                       dispatched */
        int     retrys;             /* number of dispatch
                                       retrys */
        ippid   Dpid;               /* pid of backend active
                                       on this device */
        int     Dpgsdone;
        int     Dseqn;              /* current sequence number
        int     Dpercentd;
        };
```

The Dstatus field contains one of the values described above under queues. Last dispatched is set by the dqueuer just before it dispatches a backend to this device and is used to detect time outs on the device. Dpgsdone and Dprecentd are values set by intelligent backends to indicate how much processing has been completed.

One status file is created for each device and queue when the configuration file is parsed and it is determined that the required file does not already exist. In this case, the initial values are taken from the configuration file. Once the status file exists, reparsing the configuration file will not affect it. Changes can be made using **xq**.

SEE ALSO

dqueuer(1), xq(1), xq(M), nq(1), backend(M).

NAME

strfile - software trouble report data base

DESCRIPTION

strfile contains the following information on software trouble reports:
{status}
{suspense date}
{internally assigned sequence number}
{title}
{submission date}
{program affected}
{priority}
{assignee}
{action required at suspense date}

This is an ASCII file with the free-form fields separated by colon. To make sorts and selections easier suspense dates are of the form MM/DD/YY.

FILES

/etc/strfile

SEE ALSO

str(M).

NAME

tar - tar tape format

DESCRIPTION

A tape written with the program **tar**(1) has a certain format: each file on the tape is preceded with a header of information . This header has the following structure.

```
struct header
{
        char name[NAMSIZ];
        char mode[8];
        char uid[8];
        char gid[8];
        char size[12];
        char mtime[12];
        char chksum[8];
        char linkflag;
        char linkname[NAMSIZ];
};
```

Here, NAMSIZ currently is equal to 100. Each header block is 512 bytes long. This structure takes less that 512 bytes so the remainder are filled in with nulls.
Meaning of the fields:

name	The name of the file as typed on the command line.
mode	The mode the file had when it was written. Refer to the description of st_mode in stat(2) for more information.,
uid	Owner of the file
gid	Group the owner is in
size	Size in bytes of the file
mtime	Last modified time
chksum	Checksum of the header
linkflag	Set if this file has a link to another
linkname	The name of the file linked to this one

SEE ALSO

tar(1).

NAME

termcap - terminal capability data base

SYNOPSIS

/etc/termcap

DESCRIPTION

Termcap is a data base describing terminals used, for example, by **vi**(1). Terminals are described in termcap by giving a set of capabilities which they have, and by describing how operations are performed. Padding requirements and initialization sequences are included in termcap.

Entries in termcap consist of a number of colon separated fields. The first entry for each terminal gives the names known for the terminal, separated by | characters. The first name is always two characters long and describes each terminal type in /etc/ttytype. The second name given is the most common abbreviation for the terminal, and the last name given is a long name fully identifying the terminal. The second name cannot contain any blanks; the last name can contain blanks for readability.

CAPABILITIES

(P) indicates padding can be specified
(P*) indicates that padding is based on number of lines affected

Name	**Type**	**Pad?**	**Description**
ae	str	(P)	End alternate character set
al	str	(P*)	Add new blank line
am	bool		Terminal has automatic margins
as	str	(P)	Start alternate character set
bc	str		Backspace if not ^H
bs	bool		Terminal can backspace with ^H
bt	str	(P)	Back tab
bw	bool		Backspace wraps from column Ø to last column
CC	str		Command character in prototype if it can be set on terminal
cd	str	(P*)	Clear to end of display
ce	str	(P)	Clear to end of line
ch	str	(P)	Like cm but horizontal motion only, line stays same
cl	str	(P*)	Clear screen
cm	str	(P)	Cursor motion
co	num		Number of columns in a line
cr	str	(P*)	Carriage return, (default ^M)
cs	str	(P)	Change scrolling region (vtlØØ), like cm
cv	str	(P)	Like ch but vertical only.
da	bool		Display is retained above
dB	num		Number of millisec of bs delay needed
db	bool		Display is retained below

dC	num		Number of millisec of cr delay needed
dc	str	(P*)	Delete character
dF	num		Number of millisec of ff delay needed
dl	str	(P*)	Delete line
dm	str		Delete mode (enter)
dN	num		Number of millisec of nl delay needed
do	str		Down one line
dT	num		Number of millisec of tab delay needed
ed	str		End delete mode
ei	str		End insert mode; give :ei=: if **ic**
eo	str		Can erase overstrikes with a blank
ff	str	(P*)	Hardcopy terminal page eject (default **^L**)
hc	bool		Hardcopy terminal
hd	str		Half-line down (forward 1/2 linefeed)
ho	str		Home cursor (if no **cm**)
hu	str		Half-line up (reverse 1/2 linefeed)
hz	str		Hazeltine; cannot print ~'s
ic	str	(P)	Insert character
if	str		Name of file containing **is**
im	bool		Insert mode (enter); give :im=: if **ic**
in	bool		Insert mode distinguishes nulls on display
ip	str	(P*)	Insert pad after character inserted
is	str		Terminal initialization string
kØ-k9	str		Sent by other function keys Ø-9
kb	str		Sent by backspace key
kd	str		Sent by terminal down arrow key
ke	str		Out of keypad transmit mode
kh	str		Sent by home key
kl	str		Sent by terminal left arrow key
kn	num		Number of other keys
ko	str		Termcap entries for other nonfunction keys
kr	str		Sent by terminal right arrow key
ks	str		Put terminal in keypad transmit mode
ku	str		Sent by terminal up arrow key
lØ-l9	str		Labels on other function keys
li	num		Number of lines on screen or page
ll	str		Last line, first column (if no **cm**)
ma	str		Arrow key map, used by vi version 2 only
mi	bool		Safe to move while in insert mode
ml	str		Memory lock on above cursor
mu	str		Memory unlock (turn off memory lock).
nc	bool		No correctly working carriage return (DM2500,H2000)
nd	str		Nondestructive space (cursor right)
nl	str	(P*)	Newline character (default **\n**)
ns	bool		Terminal is a CRT but does not scroll.
os	bool		Terminal overstrikes
pc	str		Pad character (rather than null)
pt	bool		Has hardware tabs (may need to be set with **is**)
se	str		End stand out mode
sf	str	(P)	Scroll forwards

sg	num		Number of blank chars left by so or se
so	str		Begin stand out mode
sr	str	(P)	Scroll reverse (backwards)
ta	str	(P)	Tab (other than ^I or with padding)
tc	str		Entry of similar terminal - must be last
te	str		String to end programs that use **cm**
ti	str		String to begin programs that use **cm**
uc	str		Underscore one char and move past it
ue	str		End underscore mode
ug	num		Number of blank chars left by us or ue
ul	bool		Terminal underlines even though it does not overstrike
up	str		Upline (cursor up)
us	str		Start underscore mode
vb	str		Visible bell (may not move cursor)
ve	str		Sequence to end open/visual mode
vs	str		Sequence to start open/visual mode
xb	bool		Beehive (f1=escape, f2=ctrl C)
xn	bool		A newline is ignored after a wrap (Concept)
xr	bool		Return acts like **ce** \r \n (Delta Data)
xs	bool		Standout not erased by writing over it (HP 264?)
xt	bool		Tabs are destructive, magic so char (Teleray 1061)
AB	str		alternate intensity, normal video, blinking (for RMCOBOL only)
AL	str		alternate intensity, normal video, no blinking (for RMCOBOL only)
AR	str		alternate intensity, reverse video, no blinking (for RMCOBOL only)
AS	str		alternate intensity, reverse video, blinking (for RMCOBOL only)
CF	str		cursor off (for RMCOBOL only)
CN	str		cursor on (for RMCOBOL only)
NB	str		normal intensity, normal video, blinking (for RMCOBOL only)
NM	str		normal intensity, normal video, no blinking (for RMCOBOL only)
NR	str		normal intensity, reverse video, no blinking (for RMCOBOL only)
NS	str		normal intensity, reverse video, blinking (for RMCOBOL only)
OV	str		overhead; the maximum number of screen positions occupied by the above defined capabilities (for RMCOBOL only)

A Sample Entry

The following entry describes the Concept-100 and is among the more complex entries in the termcap file as of this writing. (This particular entry for the Concept is outdated, and is used as an example only.)

```
c1|c100|concept100:is=\EU\Ef\E7\E5\E8\E1\ENH\EK\E\200\Eo&\200:\
        :al=3*\E^R:am:bs:cd=16*\E^C:ce=16\E^S:cl=2*^L:\
        :cm=\Ea%+ %+ :co#80:dc=16\E^A:dl=3*\E^B:\
        :ei=\E\200:eo:im=\E^P:in:ip=16*:li#24:mi:nd=\E=:\
        :se=\Ed\Ee:so=\ED\EE:ta=8\t:ul:up=\E;:vb=\Ek\EK:xn:
```

Entries can continue onto multiple lines by giving a \ as the last character of a line. Capabilities in termcap are of three types: Boolean capabilities that indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities that give a sequence to perform particular terminal operations.

Types of Capabilities

All capabilities have two-letter codes. For instance, the fact that the Concept has automatic margins (that is, an automatic return and line feed when the end of a line is reached) is indicated by the capability am. Hence the description of the Concept includes am. Numeric capabilities are followed by the character # and then the value. Thus co that indicates the number of columns the terminal has gives the value 80 for the Concept.

Finally, string valued capabilities, such as ce (clear to end of line sequence) are given by the two-character code, an =, and a string ending at the next following :. A delay in milliseconds can appear after the = in such a capability, and padding characters are supplied by the editor after the remainder of the string is sent to provide this delay. The delay can be either an integer, for example, 20, or an integer followed by an *, for example, 3*. A * indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected unit padding required. When a * is specified, give a delay of the form 3.5 to specify a delay per unit to tenths of milliseconds.

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. A **\E** maps to an ESCAPE character, **^x** maps to a control-x for any appropriate x, and the sequences **\n, \r, \t, \b, \f** give a new line, return, tab, backspace, and form feed respectivley. Finally, characters can be given as three octal digits after a \, and the characters ^ and \ are given as **\^** and ****. If it is necessary to place a : in a capability, it must be escaped in octal as **\072**. If it is necessary to place a null character in a string capability, it must be encoded as **\200**. The routines that deal with termcap use C strings, and strip the high bits of the output very late so

that a **\200** comes out as a **\000** would.

Preparing Descriptions

The most effective way to prepare a terminal description is by imitating the description of a similar terminal in termcap and to build up a description gradually, using partial descriptions with **vi**(1) to check that they are correct. A very unusual terminal can expose deficiencies in the ability of the termcap file to describe it or expose bugs in **vi**. To easily test a new terminal description, set the environment variable TERMCAP to a pathname of a file containing the description being worked on and the editor looks there rather than in /etc/termcap. TERMCAP can also be set to the termcap entry itself to avoid reading the file when starting up the editor. (This only works on version 7 systems.)

Basic Capabilities

The number of columns on each line for the terminal is given by the co numeric capability. If the terminal is a CRT, then the number of lines on the screen is given by the li capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it must have the am capability. If the terminal can clear its screen, then this is given by the cl string capability. If the terminal can backspace, then it must have the bs capability. An exception is when a backspace is accomplished by a character other than **^H** in which case this character must have the bc string capability. If it overstrikes rather than clearing a position when a character is struck over, then it must have the os capability.

The local cursor motions encoded in termcap are undefined at the left and top edges of a CRT terminal. The editor never attempts to backspace around the left edge, nor does it attempt to go up locally off the top. The editor assumes that feeding from the bottom of the screen causes the screen to scroll up, and the am capability tells whether the cursor sticks at the right edge of the screen. If the terminal has switch selectable automatic margins, the termcap file usually assumes that this is on, that is, am.

These capabilities suffice to describe hardcopy and glasstty terminals. Thus the model 33 teletype is described as

```
t3|33|tty33:co#72:os
```

while the Lear Siegler ADM-3 is described as

```
cl|adm3|3|lsi adm3:am:bs:cl=^Z:li#24:co#80
```

Cursor Addressing

Cursor addressing in the terminal is described by a cm string capability, with **printf**(3) like escapes **%x** in it. These substitute to encodings of the current line or column position, while other characters are passed through unchanged. If the cm string is thought of as being a function, then its arguments are the line and the column to which motion is desired, and the **%** encodings have the following meanings:

```
%d   as in printf, 0 origin
%2   like %2d
%3   like %3d
%.   like %c
%+x  adds x to value, then %
%>xy if value > x adds y, no output
%r   reverses order of line and column, no output
%i   increments line/column (for 1 origin)
%%   gives a single %
%n   exclusive or row and column with 0140 (DM2500)
%B   BCD (16*(x/10)) + (x%10), no output
%D   Reverse coding (x-2*(x%16)), no output
     (Delta Data)
```

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. The order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its cm capability is cm=6\E&%r%2c%2Y. The Microterm ACT-IV needs the current row and column sent preceded by a **^T**, with the row and column simply encoded in binary, cm=^T%.%.. Terminals which use %. need to be able to backspace the cursor (bs or bc), and to move the cursor up one line on the screen (**up** introduced in following text). This is necessary because it is not always safe to transmit **\t**, **\n** **^D**, and **\r**, as the system may change or discard them.

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus cm=\E=%+ %+ .

Cursor Motions

If the terminal can move the cursor one position to the right, leaving the character at the current position unchanged, then this sequence should be given as nd (nondestructive space). If it can move the cursor up a line on the screen in the same column, this should be given as up.

If the terminal has no cursor addressing capability, but can home the cursor (to very upper left corner of screen), then this is given as ho; similarly a fast way of getting to the lower left hand corner can be given as ll; this can involve going up with up from the home position, but the editor never does this itself (unless ll does) because it makes no assumption about the effect of moving up from the home position.

Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as ce. If the terminal can clear from the current position to the end of the display, then this should be given as cd. The editor only uses cd from the first column of a line.

Insert/Delete Line

If the terminal can open a new blank line before the line where the cursor is, this should be given as al; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, then this should be given as dl; this is done only from the first position on the line to be deleted. If the terminal can scroll the screen backwards, then this can be given as sb, but al suffices. If the terminal can retain display memory above, then the da capability should be given; if display memory can be retained below, then db should be given. These tell the editor that deleting a line on the screen can bring nonblank lines up from below or that scrolling back with sb can bring down nonblank lines.

Insert/Delete Character

There are two kinds of intelligent terminals with respect to insert/delete character that can be described using termcap. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen that is either eliminated, or expanded to two untyped blanks. To find out the kind of terminal, clear the screen then type text separated by cursor motions. Type abc def using local cursor motions (not spaces) between the abc and

the def. Then position the cursor before the abc and put the terminal in insert mode. If typing characters causes the rest of the line to shift and characters to fall off the end, then the terminal does not distinguish between blanks and untyped positions. If the abc shifts over to the def that moves together around the end of the current line and onto the next, this is have the second type of terminal, and the capability in must be given (insert null). If the terminal does something different, then the editor must be modified to get it to use the insert mode that the terminal defines.

The editor handles both terminals that have an insert mode and terminals that send a simple sequence to open a blank position on the current line. Give as im the sequence to get into insert mode, or give it an empty value if the terminal uses a sequence to insert a blank position. Give as ei the sequence to leave insert mode (give this, with an empty value also if you gave im one). Now give as ic any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode do not give ic. Terminals that send a sequence to open a screen position should give it here. (Insert mode is preferable to the sequence to open a position on the screen if the terminal has both.) If post-insert padding is needed, give this as a number of milliseconds in ip (a string option). Any other sequence needed to be sent after an insert of a single character can also given in ip.

It is occasionally necessary to move around while in insert mode to delete characters on the same line, for example, if there is a tab after the insertion position. If the terminal allows motion while in insert mode, give the capability mi to speed up inserting in this case. Omitting mi affects only speed. Some terminals (notably Datamedia's) must not have mi because of the way their insert mode works.

Finally, specify delete mode by giving dm and ed to enter and exit delete mode, and dc to delete a single character while in delete mode.

Highlighting, Underlining, and Visible Bells

If the terminal has sequences to enter and exit standout mode, these are given as so and se respectively. If there are several kinds of standout mode: inverse video, blinking, or underlining (half bright is not usually an acceptable standout mode unless the terminal is in inverse video mode constantly). The preferred mode is inverse video by itself. If the code to change into or out of standout mode leaves one or two blank spaces on the screen, as the TVI 912 and

Teleray 1061 do, this is acceptable.

Codes to begin underlining and end underlining are given as us and ue respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, this can be given as uc. (If the underline code does not move the cursor to the right, give the code followed by a nondestructive space.)

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement) then this is given as vb; it must not move the cursor. If the terminal is placed in a different mode during open and visual modes of ex, this is given as vs and ve, sent at the start and end of these modes respectively. These are used to change modes, for example, from a underline to a block cursor and back.

If the terminal needs to be in a special mode when running a program that addresses the cursor, the codes to enter and exit this mode are given as ti and te. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly.

If the terminal correctly generates underlined characters with no special codes needed even though it does not overstrike, then give the capability ul. If overstrikes are erasable with a blank, then this must be indicated by giving eo.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information is given. It is not possible to handle terminals where the keypad only works in local; for example, the unshifted HP 2621 keys. If the keypad can be set to transmit or not transmit, give these codes as ks and ke. Otherwise the keypad is assumed to always transmit. The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys are given as kl, kr, ku, kd, and kh respectively. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as k0, k1, ..., k9. If these keys have labels other than the default f0 through f9, the labels are given as **10, 11, ...,** 19. If there are other keys that transmit the same code as the terminal expects for the corresponding function, such as clear screen, the termcap 2-letter codes are given in the ko capability; for example, :ko=cl,ll,sf,sb:, says that the

terminal has clear, home down, scroll down, and scroll up keys that transmit the same thing as the cl, ll, sf, and sb entries.

The ma entry is also used to indicate arrow keys on terminals that have single character arrow keys. It is obsolete but still in use in version 2 of vi, run on some minicomputers. This field is redundant with kl, kr, ku, kd, and kh. It consists of groups of two characters. In each group, the first character is what an arrow key sends, the second character is the corresponding vi command. These commands are h for kl, j for kd, k for ku, l for kr, and H for kh. For example, the mime would be **:ma=^Kj^Zk^Xl:** indicating arrow keys left (^H), down (^K), up (^Z), and right (^X). (There is no home key on the mime.)

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this is given as pc.

If tabs on the terminal require padding, or if the terminal uses a character other than **^I** to tab, then this is given as ta.

Hazeltine terminals that do not allow ~ characters to be printed must indicate hz. Datamedia terminals that echo carriage-return line feed for carriage return and then ignore a following line feed must indicate nc. Early Concept terminals that ignore a line feed immediately after an am wrap must indicate xn. If an erase-eol is required to get rid of standout instead of merely writing on top of it, xs must be given. Teleray terminals, where tabs turn all characters moved over to blanks, must indicate **xt**. Other specific terminal problems can be corrected by adding more capabilities of the form xx.

Other capabilities include is, an initialization string for the terminal, and if, the name of a file containing long initialization strings. These strings are expected to properly clear and then set the tabs on the terminal. If both are given, is is printed before if. This is useful where if is /usr/lib/tabset/std but is clears the tabs first.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability tc can be given with the name of the similar terminal. This capability must be last and the combined length of the two entries must not exceed 1024. Since

termlib routines search the entry from left to right, and since the tc capability is replaced by the corresponding entry, the capabilities given at the left override the ones in the similar terminal. A capability can be cancelled with **xx@** where xx is the capability. For example, the entry

```
hn|2621nl:ks@:ke@:tc=2621:
```

defines a 2621nl that does not have the ks or ke capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

FILES

/etc/termcap	file containing terminal descriptions
/etc/termcap.others	file containing more terminal descriptions (for supporting nonstandard terminals)

The system administrator may want to arrange the /etc/termcap file such that only the terminals attached to the system or most likely to be attached to the system reside in that file. Since the file is searched sequentially, it is also a good idea to arrange the file with the most common terminals at the front of the file. This makes the intialization process for files searching /etc/termcap more efficient. It is recommended that any terminal specifiers not used be placed in /etc/termcap.others for backup purposes.

SEE ALSO

ex(1), vi(1), termcap(3), ttytype(5).

LIMITATIONS

Vi allows only 256 characters for string capabilities, and the routines in **termcap**(3) do not check for overflow of this buffer. The total length of a single entry (excluding only escaped new lines) cannot exceed 1024.

The ma, vs, and ve entries are specific to the vi program.

Not all programs support all entries. There are entries that are not supported by any program.

NAME

ttytype - terminal/types table

DESCRIPTION

Ttytype resides in the directory /etc and associates terminal types with ports. There is one line per terminal port. Each line contains two fields; a two-character terminal capabilities code and the tty name for that port. The fields are separated by a blank character. The two-character code is found in /etc/termcap and is the first two characters describing each entry in that file. A typical eight-user configuration using vtz-2/10 and Adm31 terminals looks like this:

```
vz console
vz tty1
31 tty2
vz tty3
vz tty4
31 tty5
31 tty6
vz tty7
```

This file is read by **login**(1) to set up the environment variable TERM. 'TERM' is used by **vi**(1),**and ex**(1).

FILES

/etc/ttytype
/etc/termcap

SEE ALSO

login(1), termcap(5).

NAME

types - primitive system data types

SYNOPSIS

#include <sys/types.h>

DESCRIPTION

The data types defined in the include file are used in ZEUS system code; some data of these types are accessible to user code:

```
/* @[$]types.h 1.1  12/11/81 17:21:31 - Zilog Inc */
typedef   long           daddr_t;       /* disk address */
typedef   char *         caddr_t;       /* core address */
typedef   unsigned int   ino_t;         /* i-node number */
typedef   long           time_t;        /* a time */
typedef   int            label_t[9];    /* program status */
typedef   int            dev_t;         /* device code */
typedef   long           off_t;         /* offset in file */
     /* selectors and constructor for device code */
#define   major(x)       (int)(((unsigned)x>>8))
#define   minor(x)       (int)(x&0377)
#define   makedev(x,y)   (dev_t)((x)<<8|(y))
```

The form daddr_t is used for disk addresses except in an i-node on disk, see **filsys**(5). Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The label_t variables are used to save the processor state while another process is running.

SEE ALSO

adb(1), lseek(2), time(2), filsys(5).

NAME

utmp, wtmp - login records

SYNOPSIS

#include <utmp.h>

DESCRIPTION

The utmp file contains information about who is currently using ZEUS. The file is a sequence of entries with the following structure declared in the include file:

```
/* @[$]utmp.h  1.1  12/11/81 17:Ø9:29 - Zilog Inc */
struct utmp {
     char ut_line[8];          /* tty name */
     char ut_name[8];          /* user id */
     long ut_time;        /* time on */
};
```

This structure gives the name of the special file associated with the user's terminal, the user's login name, and the time of the login in the form of **time**(2).

The wtmp file records all logins and logouts. Its format is exactly like utmp except that a null user name indicates a logout on the associated terminal. The terminal name ~ indicates that the system was rebooted at the indicated time; the adjacent pair of entries with terminal names | and } indicate the system-maintained time just before and just after a date command has changed the system's time.

Wtmp is maintained by **login**(1) and **init**(M). Neither of these programs creates the file, so if it is removed, record-keeping is turned off. It is summarized by **acct**(M).

FILES

/etc/utmp
/usr/adm/wtmp

SEE ALSO

acct(M), login(1), who(1), init(M).

NAME

whois - whois database file

DESCRIPTION

Whois contains an entry for each user. Each entry contains the following information:

Login name
Actual user's name (full name)
Office phone
Home phone
Group associated with
Miscellaneous comments

This is an ASCII file. Each field within each user's entry is separated from the next by a colon. Each user is separated from the next by a newline.

This file resides in the directory /etc.

FILES

/etc/whois

SEE ALSO

whois(1).

NAME

zsc - Zilog Source Control File conventions

DESCRIPTION

A Zilog Source Control File contains all the information necessary to re-create any version of an associated source file. Text lines in more than one version of the source are in the control file only once. This ensures that the working version of the source is not destroyed in the development process. This method avoids the usual practice of saving several versions of a file. It is less cumbersome and wastes less disk storage.

The control file is a conventional ZEUS text file. Each successive release of the source file is represented by a release entry in the source file. The release entry consists of one or more level entries, representing levels within a release of the source file. A version is extracted from a control file is assigned a version number of the form

rel.lev

where rel indicates that this is the rel'th release and lev indicates that this is the lev'th level in release rel. (It is possible to include the version number in the source file.)

The first line of each release entry is a release line, consisting of the the characters

&R

The release line is immediately followed by the first level entry. Each level begins with a history line which resembles:

&H isaac at Tue Sep 15 10:17:34 1981

The history line shows the user who entered the version into the control file and shows when this was done.

Each history line is followed by zero or more comment lines which resemble:

&C terminal-independent functions added

Comment lines hold documenting information entered by the person who added the version to the control file.

Following the history and comment lines are the lines which show the differences between this version and the previous version. (For version 1.1, these lines show the differences

between the version and an empty file.) The difference line are created by the diff program run with the -f option; see diff(1).

The end of the difference lines and of the level entry is indicated with a done line:

```
&D
```

The control file for a source code file called file is called file+.

The following operations may be done on a Zilog Source Control file:

(1) Check in (add) a new version of the source file to the control file. This is done with **chkin**(1).

(2) Check out (extract) a version of the source from the control file. This is done with **chkout**(1); chkin also checks out the same version it is told to check in, unless the -r option is used.

(3) Check differences between the source file and a version in the control file or between two versions in the control file. This is done with **chkdiff**(1).

A source file is checked out either editable or read-only. An editable source file is meant to be modified and checked back in, thus creating a new version. When an editable source file is checked out, a lock file is created; without a lock file a modified source file cannot be checked back in to an existing control file. A read-only source file is meant to be listed, compiled, or otherwise input to a program. It must not be modified or checked back in. Chkout by default creates a read-only source file; used with the -e option, chkout creates an editable source file. Chkin by default replaces the editable source file with a read-only source file; used with the -r option, the source file is simply removed.

A read-only source file should contain information identifying its version. This can be provided by including one or more keywords in the file before it is first checked in. A keyword is one of the following three-character sequences. In a read-only source file, the keywords are expanded as follows.

W This is expanded to a what string, which identifies which version of which module (file) this is. As in this example,

```
@[$]master.c   1.2
```

a what string has four elements: the four characters @[$], to identify this as a what string; the name of the source code file (module); a tab character; and the version number (release). Enter W in the editable source in such a way that the what string appears in both the read-only source and the object file. For example, if

```
extern char Version[] = "$W$";
```

appears in an editable C source

```
extern char Version[] = "@[$]master.c   1.2";
```

appears in the read-only source and the what string appears in the object file.

Z Expands to the characters @[$] to designate user-designed what strings.

R Expands to the version number (release).

D Expands to date and time (MM/DD/YY HH:MM:SS) source file was checked out.

The following procedure shows a possible use of the source code control programs and illustrates how they are used together.

(1) Create the original source file, including W and any other keywords that are appropriate.

(2) Check in the original source file. Chkin replaces the original source file with a read-only source file, substituting keywords.

(3) Compile and test. If no further changes are indicated, stop.

(4) Check out an editable source file of the latest version, using the -e option of chkout.

(5) Edit indicated changes. Go to step 3.

SEE ALSO

chkin(1), chkout(1), chkdiff(1), diff(1).

LIMITATIONS

A read-only source file is protected only by its lack of write permissions. Thus its status may go unnoticed until

the text editor refuses to overwrite it.

NAME

adventure

SYNOPSIS

adventure

DESCRIPTION

Adventure is a treasure hunt game. General directions are given at the beginning of the game. The computer prompts for answers and directions in response to the descriptions given regarding location. Commands can be one or two words long, but only the first five letters of each word are read. Typing **help** produces a list of general hints. Creating a map while playing the game is very useful.

LIMITATIONS

The list of commands is limited. The list of hints is very limited and general. One must be quite imaginative to succeed.

FILES

/usr/games/adventure the program

NAME

arithmetic - provide drill in number facts

SYNOPSIS

arithmetic [**+ - x /**] [range]

DESCRIPTION

Arithmetic types out simple arithmetic problems, and waits for an answer to be typed in. If the answer is correct, it types back "Right!", and a new problem. If the answer is wrong, it replies "What?", and waits for another answer. Every twenty problems, it publishes statistics on correctness and the time required to answer.

To quit the program, type an interrupt (delete).

The first optional argument determines the kind of problem to be generated; **+ - x /** respectively cause addition, subtraction, multiplication, and division problems to be generated. One or more characters can be given; if more than one is given, the different types of problems will be mixed in random order; default is **+ -** .

Range is a decimal number; all addends, subtrahends, differences, multiplicands, divisors, and quotients will be less than or equal to the value of range. Default range is 10.

At the start, all numbers less than or equal to range are equally likely to appear. If the respondent makes a mistake, the numbers in the problem which was missed become more likely to reappear.

As a matter of educational philosophy, the program will not give correct answers, since the learner should, in principle, be able to calculate them. Thus the program is intended to provide drill for someone just past the first learning stage, not to teach number facts de novo. For almost all users, the relevant statistic should be time per problem, not percent correct.

FILES

/usr/games/arithmetic the program

NAME

backgammon - the game

SYNOPSIS

backgammon

DESCRIPTION

This program does what you expect. It will ask whether you need instructions.

FILES

/usr/games/backgammon the program

NAME

craps - the game of craps

SYNOPSIS

craps

DESCRIPTION

Craps is a form of the game of craps that is played in Las Vegas. The program simulates the roller, while the user (the player) places bets. The player may choose, at any time, to bet with the roller or with the House. A bet of a negative amount is taken as a bet with the House, any other bet is a bet with the roller.

The player starts off with a ``bankroll'' of $2,000.

The program prompts with:

bet?

The bet can be all or part of the player's bankroll. Any bet over the total bankroll is rejected and the program prompts with ``bet?'' until a proper bet is made.

Once the bet is accepted, the roller throws the dice. The following rules apply (the player wins or loses depending on whether the bet is placed with the roller or with the House; the odds are even). The first roll is the roll immediately following a bet.

1. On the first roll:

7~or~11	wins for the roller;
2,~3,~or~12	wins for the House;
any~other~number	is the point, roll again (Rule 2 applies).

2. On subsequent rolls:

point	roller wins;
7	House wins;
any~other~number	roll again.

If a player loses the entire bankroll, the House will offer to lend the player an additional $2,000. The program will prompt:

marker?

A ``yes'' (or ``y'') consummates the loan. Any other reply terminates the game.

If a player owes the House money, the House reminds the player, before a bet is placed, how many markers are outstanding.

If, at any time, the bankroll of a player who has outstanding markers exceeds $2,000, the House asks:

Repay marker?

A reply of ``yes'' (or ``y'') indicates the player's willingness to repay the loan. If only 1 marker is outstanding, it is immediately repaid. However, if more than 1 marker are outstanding, the House asks:

How many?

markers the player would like to repay. If an invalid number is entered (or just a carriage return), an appropriate message is printed and the program will prompt with ``How many?'' until a valid number is entered.

If a player accumulates 10 markers (a total of $20,000 borrowed from the House), the program informs the player of the situation and exits.

Should the bankroll of a player who has outstanding markers exceed $50,000, the total amount of money borrowed will be automatically repaid to the House.

Any player who accumulates $100,000 or more breaks the bank. The program then prompts:

New game?

Any reply other than ``yes'' is considered ``no'' (except in the case of ``bet?'' or ``How many?''). To exit, send an interrupt (break), DEL, or control-D. The program will indicate whether the player won, lost, or broke even.

MISCELLANEOUS

The random number generator for the die numbers uses the seconds from the time of day. Depending on system usage, these numbers, at times, may seem strange but occurrences of this type in a real dice situation are not uncommon.

FILES

/usr/games/craps the program

NAME

fortune - print fortune cookie

SYNOPSIS

fortune

DESCRIPTION

Fortune emulates the well-known Chinese munchie-of-wisdom. The program selects a line at random from a text file. Add or delete fortunes by adding or removing lines from this file.

FILES

/usr/games/lib/fortunes - wisdom file

DIAGNOSTICS

The System 8000 does not trap memory errors.

LIMITATIONS

Humor is subjective.

NAME

hangman - word games

SYNOPSIS

hangman [dict]

vhm

DESCRIPTION

Hangman chooses a word at least seven letters long from a word list. The user is to guess letters one at a time.

The optional argument names an alternate word list. The special name `-a' gets a particular very large word list.

FILES

/usr/games/hangman	the program
/usr/games/vhm	the visual program
/usr/dict/words	the regular word list

DIAGNOSTICS

After each round, **hangman** reports the average number of guesses per round and the number of rounds.

LIMITATIONS

Hyphenated compounds are run together.

NAME

quiz - test your knowledge

SYNOPSIS

quiz [**-i** file] [**-t**] [category1 category2]

DESCRIPTION

Quiz gives associative knowledge tests on various subjects. It asks items chosen from category1 and expects answers from category2. If no categories are specified, **quiz** gives instructions and lists the available categories.

Quiz tells a correct answer whenever you type a bare newline. At the end of input, upon interrupt, or when questions run out, **quiz** reports a score and terminates.

The **-t** flag specifies `tutorial' mode, where missed questions are repeated later, and material is gradually introduced as you learn.

The **-i** flag causes the named file to be substituted for the default index file. The lines of these files have the syntax:

```
line     = category newline | category `:' line
category = alternate | category `|' alternate
alternate = empty | alternate primary
primary  = character | `[' category `]' | option
option   = `{' category `}'
```

The first category on each line of an index file names an information file. The remaining categories specify the order and contents of the data in each line of the information file. Information files have the same syntax. Backslash `\' is used as with **sh**(1) to quote syntactically significant characters or to insert transparent newlines into a line. When either a question or its answer is empty, **quiz** will refrain from asking it.

FILES

/usr/games/quiz.k/*

LIMITATIONS

The construct `a|ab' doesn't work in an information file. Use `a{b}'.

NAME

wump - the game of hunt-the-wumpus

SYNOPSIS

wump

DESCRIPTION

Wump plays the game of `Hunt the Wumpus.' A Wumpus is a creature that lives in a cave with several rooms connected by tunnels. You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile avoiding being eaten by the Wumpus and falling into Bottomless Pits. There are also Super Bats which are likely to pick you up and drop you in some random room.

The program asks various questions which you answer one per line; it will give a more detailed description if you want.

This program is based on one described in People's Computer Company, 2, 2 (November 1973).

FILES

/usr/games/wump the program

LIMITATIONS

It will never replace Space War.

NAME

ascii - map of ASCII character set

SYNOPSIS

cat /usr/pub/ascii

DESCRIPTION

Ascii is a map of the ASCII character set, to be printed as needed. It contains:

00	nul	01	soh	02	stx	03	etx	04	eot	05	enq	06	ack	07	bel
08	bs	09	ht	0A	nl	0B	vt	0C	np	0D	cr	0E	so	0F	si
10	dle	11	dc1	12	dc2	13	dc3	14	dc4	15	nak	16	syn	17	etb
18	can	19	em	1A	sub	1B	esc	1C	fs	1D	gs	1E	rs	1F	us
20	sp	21	!	22	"	23	#	24	$	25	%	26	&	27	'
28	(	29	)	2A	*	2B	+	2C	,	2D	-	2E	.	2F	/
30	0	31	1	32	2	33	3	34	4	35	5	36	6	37	7
38	8	39	9	3A	:	3B	;	3C	<	3D	=	3E	>	3F	?
40	@	41	A	42	B	43	C	44	D	45	E	46	F	47	G
48	H	49	I	4A	J	4B	K	4C	L	4D	M	4E	N	4F	O
50	P	51	Q	52	R	53	S	54	T	55	U	56	V	57	W
58	X	59	Y	5A	Z	5B	[	5C	\	5D	]	5E	^	5F	_
60	`	61	a	62	b	63	c	64	d	65	e	66	f	67	g
68	h	69	i	6A	j	6B	k	6C	l	6D	m	6E	n	6F	o
70	p	71	q	72	r	73	s	74	t	75	u	76	v	77	w
78	x	79	y	7A	z	7B	{	7C	|	7D	}	7E	~	7F	del

FILES

/usr/pub/ascii

NAME

dmalias - Z8000 Development Module protocol

SYNOPSIS

source /usr/pub/dmalias

DESCRIPTION

This cshell script is required for proper handshaking between a Z8000 Development Module and the shell. Prior to a LOAD or SEND command, this script should be sourced so that the cshell will correctly accept the 'load' command from the development module.

SEE ALSO

load(1), send(1).

NAME

environ - user environment

DESCRIPTION

An array of strings called the "environment" is made available by **exec**(2) when a process begins. By convention, these strings have the form name=value. The following names are used by various commands:

HOME Name of the user's login directory, set by **login**(1) from the password file **passwd**(5).

PATH The sequence of directory prefixes that **sh**(1), **time**(1), **nice**(1), **nohup**(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by colons " : ".

Login(1) sets PATH=:/bin:/usr/bin.

TERM The kind of terminal for which output is to be prepared. This is used by commands, such as **vi**(1) which can exploit special capabilities of that terminal.

TZ Time zone information. The format is xxxnzzz where xxx is standard local time zone abbreviation, n is the difference in hours from **GMT**, and zzz is the abbreviation for the daylight-saving local time zone, if any; for example, **EST5EDT**.

Further names can be placed in the environment by the **export** command and name=value arguments in **sh**(1), or by **exec**(2).

SEE ALSO

env(1), login(1), sh(1), exec(2), getenv(3), profile(5), term(7).

LIMITATIONS

Avoid conflict with certain shell variables that are frequently exported by .profile files: MAIL, PS1, PS2, IFS.

NAME

fcntl - file control options

SYNOPSIS

#include <fcntl.h>

DESCRIPTION

The **fcntl**(2) function provides for control over open files. This include file describes requests and arguments to fcntl and **open**(2).

```
/* Flag values accessible to open(2) and fcntl(2) */
/*  (The first three can only be set by open) */
#define    O_RDONLY    0
#define    O_WRONLY    1
#define    O_RDWR      2
#define    O_NDELAY    04        /* Non-blocking I/O */
#define    O_APPEND    010       /* append (writes guaranteed */
                                 /*    at the end) */

/* Flag values accessible only to open(2) */
#define    O_CREAT     00400     /* open with file create */
                                 /*   (uses third open arg)*/
#define    O_TRUNC     01000     /* open with truncation */
#define    O_EXCL      02000     /* exclusive open */

/* fcntl(2) requests */
#define    F_DUPFD     0         /* Duplicate fildes */
#define    F_GETFD     1         /* Get fildes flags */
#define    F_SETFD     2         /* Set fildes flags */
#define    F_GETFL     3         /* Get file flags */
#define    F_SETFL     4         /* Set file flags */
```

SEE ALSO

fcntl(2), open(2).

NAME

greek - graphics for the extended TTY-37 type-box

SYNOPSIS

cat /usr/pub/greek [| **greek -T**terminal]

DESCRIPTION

Greek gives the mapping from ASCII to the "shift-out" graphics in effect between **SO** and **SI** on TELETYPE(Reg.) Model 37 terminals equipped with a 128-character type-box. These are the default greek characters produced by **nroff**(1). The filters of **greek**(1) attempt to print them on various other terminals. The file contains:

alpha	A	A	beta	B	B	gamma	\	\
GAMMA	G	G	delta	D	D	DELTA	W	W
epsilon	S	S	zeta	Q	Q	eta	N	N
THETA	T	T	theta	O	O	lambda	L	L
LAMBDA	E	E	mu	M	M	nu	@	@
xi	X	X	pi	J	J	PI	P	P
rho	K	K	sigma	Y	Y	SIGMA	R	R
tau	I	I	phi	U	U	PHI	F	F
psi	V	V	PSI	H	H	omega	C	C
OMEGA	Z	Z	nabla	[	[	not	_	_
partial	]	]	integral	^	^			

FILES

/usr/pub/greek

SEE ALSO

300(1), 4014(1), 450(1), greek(1), hp(1), tc(1), troff(1).

NAME

hier - file system hierarchy

DESCRIPTION

The following outline gives a quick tour through a representative directory hierarchy.

```
/    root
/dev/
     devices (4)
     ct*  cartridge tape, ct(4)
     rct* raw cartridge tape, ct(4)
     console
          main console, tty(4)
     tty* terminals, tty(4)
     zd*  disk, zd(4)
     rzd* raw disk, zd(4)
     md*  disk, md(4)
     rmd* raw disk, md(4)
     smd* disk, smd(4)
     rsmd*
          raw disk, smd(4)
     ...
/bin/
     utility programs, cf /usr/bin/ (1)
     as assembler first pass, cf /lib/as2
     cc C compiler executive, cf /lib/c[Ø12]
     ...
/lib/
     object libraries and other stuff, cf /usr/lib/
     libc.a
          system calls, standard I/O, etc. (2,3,3S)
     cpp  C preprocessor
     c[1234]
          passes of cc(1)
     ...
/etc/
     essential data and dangerous maintenance utilities
     passwd
          password file, passwd(5)
     group
          group file, group(5)
     motd message of the day, login(1)
     mtab mounted file table, mtab(5)
     ddate
          dump history, dump(1)
     ttys properties of terminals, ttys(5)
     getty
          part of login, getty(M)
     init the parent of all processes, init(M)
     rc   shell program to bring the system up
     cron the clock demon, cron(M)
```

```
     mount
          mount(M)
     wall wall(M)
     ...
/tmp/
     temporary files
     e*   used by ed(1)
     ctm* used by cc(1)
     ...
/usr/
     general-purpose directory, usually a mounted file
     system
     adm/ administrative information
          wtmp login history, utmp(5)
          messages
               hardware error messages
/usr/ bin/
     utility programs, to keep /bin/ small
     dict/
          word lists, etc.
          words
               principal word list, used by look(1)
          spellhist
               history file for spell(1)
     games/
     include/
          standard #include files
          s.out.h
               object file layout, a.out(5)
          stdio.h
               standard I/O, stdio(3)
          ...
          sys/ system-defined layouts, cf /usr/sys/h
               acct.h
                    process accounts, acct(5)
               buf.h
                    internal system buffers
               ...
     lib/ object libraries and stuff, to keep /lib/ small
          atrun
               scheduler for at(1)
          tmac/
               macros for troff(1)
               tmac.an
                    macros for man(7)
               tmac.s
                    macros for ms(7)
               ...
          font/
               fonts for troff(1)
               R    Times Roman
               B    Times Bold
```

```
                ...
        /usr/   man/
                volume 1 of this manual, man(1)
                man0/
                        general
                        introduction to volume 1, ms(7) format
                man1/
                        chapter 1
                        as.1
                        ...
                ...
                spool/
                        delayed execution files
                        at/  used by at(1)
                        queuer/
                                general purpose enqueing program
                                used by nq(1)
                        uucp/
                                work files and staging area for uucp(1)
                                LOGFILE
                                        summary log
                                LOG.*
                                        ... log file for one transaction
                        mail/
                                mailboxes for mail(1)
                                uid  mail file for user uid
                                uid.lock
                                        lock file while uid is receiving
                                        mail
                        secretmail/
                                mailboxes and keys for xsend(1)
                                uid.0
                                        secretmail file for user
                                uid.key
                                        key for decryption
                                        taper/
                                                reserve tape drive
```

SEE ALSO

find(1), grep(1), ls(1), ncheck(M).

LIMITATIONS

The position of files is subject to change without notice.

NAME

man - macros to nroff or troff manual entry

SYNOPSIS

nroff -man file

troff -man file

DESCRIPTION

These macros are used to lay out pages of this manual. A skeleton page is found in the file /usr/man/manØ/xx.

Any text argument t is zero to six words. Quotes are used to include blanks in a word. If text is empty, the special treatment is applied to the next input line with text to be printed. In this way, .I is used to italicize a whole line, or .SM followed by .B to make small bold letters.

A prevailing indent distance is retained between successive indented paragraphs, and is reset to default value upon reaching a nonindented paragraph. Default units for indents i are ens.

Type font and size are reset to default values before each paragraph, and after processing font and size setting macros.

These strings are predefined by **-man:**

*R `(Reg)', trademark symbol in troff.

*S Change to default type size.

FILES

/usr/lib/tmac/tmac.an
/usr/man/manØ/xx

SEE ALSO

troff(1), man(1).

LIMITATIONS

Relative indents do not nest.

REQUESTS

Request	Cause Break	If no Argument	Explanation
.B t	no	t=n.t.l.	*Text t is bold.
.BI t	no	t=n.t.l.	Join words of t alternating bold and italic.
.BR t	no	t=n.t.l.	Join words of t alternating bold and Roman.
.DT	no	.5i li...	Restore default tabs.

.HP i	yes	i=p.i.*	Set prevailing indent to i. Begin paragraph with hanging indent.
.I t	no	t=n.t.l.	Text t is italic.
.IB t	no	t=n.t.l.	Join words of t alternating italic and bold.
.IP x i	yes	x=""	Same as .TP with tag x.
.IR t	no	t=n.t.l.	Join words of t alternating italic and Roman.
.LP	yes	-	Same as .PP.
.PD d	no	d=.4v	Interparagraph distance is d.
.PP	yes	-	Begin paragraph. Set prevailing indent to .5i.
.RE	yes	-	End of relative indent. Set prevailing indent to amount of starting .RS.
.RB t	no	t=n.t.l.	Join words of t alternating Roman and bold.
.RI t	no	t=n.t.l.	Join words of t alternating Roman and italic.
.RS i	yes	i=p.i.	Start relative indent, move left margin in distance i. Set prevailing indent to .5i for nested indents.
.SH t	yes	t=n.t.l.	Subhead.
.SM t	no	t=n.t.l.	Text t is small.
.TH n c x	yes	-	Begin page named n of chapter c; x is extra commentary, for example, local, for page foot. Set prevailing indent and tabs to .5i.
.TP i	yes	i=p.i.	Set prevailing indent to i. Begin indented paragraph with hanging tag given by next text line. If tag does not fit, place it on separate line.

* n.t.l. = next text line; p.i. = prevailing indent

NAME

me - macros for formatting manuscripts using nroff or troff

SYNOPSIS

nroff -me [options] file
troff -me [options] file

DESCRIPTION

This package of **nroff** and **troff** macro definitions provides a canned formatting facility for technical papers in various formats.

Output of the **eqn**(1) **neqn**(1) and **tbl**(1) preprocessors for equations and tables is acceptable as input.

REQUESTS

.bc Start a new column

.bp Start a new page

.ce Center the next N lines

.in Indent N spaces

.ip Begin a paragraph with indented body

.lp Begin a left-justified block-style paragraph

.np Begin an indented, numbered paragraph

.pp Begin an indented paragraph

.sh Give a section header

.sp Leave N lines of blank space

.ti Begin temporary indent of NFR spaces

.tp Begin a title page

.ul Under-line the next N lines (italics in **troff**)

.lc Revert to single column output

.2c Automatic two column out-put

FILES

/usr/lib

SEE ALSO

eqn(1), troff(1), refer(1), tbl(1).

Writing Papers With Nroff Using -ME in the ZEUS Utilities Manual

NAME

mm - the MM macro package for formatting documents

SYNOPSIS

mm [*options*] [*files*]

nroff -mm [*options*] [*files*]

nroff -cm [*options*] [*files*]

mmt [*options*] [*files*]

troff -mm [*options*] [*files*]

troff -cm [*options*] [*files*]

DESCRIPTION

This package provides a formatting capability for a wide variety of documents. It is the standard package used by the BTL typing pools and documentation centers. The manner in which a document is entered and edited is essentially independent of whether the document is to be eventually formatted at a terminal or phototypeset. See the references below for further details.

OPTIONS

-cm Causes *nroff*(1) and *troff*(1) to use the compacted version of the macro package, thus speeding up the process of loading.

-mm Results in the use of the non-compacted version of the macro package.

FILES

/usr/lib/tmac/tmac.m	pointer to the non-compacted version of the package
/usr/lib/macros/mm[nt]	non-compacted version of the package
/usr/lib/macros/cmp.[nt].[dt].m	compacted version of the package
/usr/lib/macros/ucmp.[nt].m	initializers for the compacted version of the package

SEE ALSO

troff(1).

NAME

ms - macros for formatting manuscripts using nroff or troff

SYNOPSIS

nroff -ms [options] file
troff -ms [options] file

DESCRIPTION

This package of **nroff**(1) and **troff**(1) macro definitions provides a canned formatting facility for technical papers in various formats. When producing two-column output on a terminal, filter the output through **col**(1).

The macro requests are defined below. Many **nroff** and **troff** requests are unsafe in conjunction with this package. However, these requests can be used after the first .PP:

```
.bp   begin new page
.br   break output line here
.sp n insert n spacing lines
.ls n (line spacing) n=1 single, n=2 double space
.na   no alignment of right margin
```

Output of the **eqn, neqn,** and **tbl**(1) preprocessors for equations and tables is acceptable as input.

FILES

/usr/lib/tmac/tmac.s

SEE ALSO

eqn(1), troff(1), tbl(1).

Typing Documents on the ZEUS System with Troff and Nroff in the ZEUS Utilities Manual.

REQUESTS

Request	Initial Value	Cause Break	Explanation
.1C	yes	yes	One-column format on a new page.
.2C	no	yes	Two-column format.
.AB	no	yes	Begin abstract.
.AE	-	yes	End abstract.
.AI	no	yes	Author's institution follows. Suppressed in TM.
.AT	no	yes	Print "Attached" and turn off line filling.
.AU x y	no	yes	Author's name follows. x is location and y is extension, ignored except in TM.
.B x	no	no	Print x in boldface; if no argument switch to boldface.
.B1	no	yes	Begin text to be enclosed in a box.

.B2	no	yes	End text to be boxed, print it.
.BT	date	no	Bottom title, automatically invoked at foot of page. May be redefined.
.BX x	no	no	Print x in a box.
.CS x...	-	yes	Cover sheet info if TM format, suppressed otherwise. Arguments are number of text pages, other pages, total pages, figures, tables, references.
.CT	no	yes	Print "Copies to" and enter no-fill mode.
.DA x	nroff	no	Date line at bottom of page is x. Default is today.
.DE	-	yes	End displayed text. Implies .KE.
.DS x	no	yes	Start of displayed text, to appear verbatim line-by-line. x=I for indented display (default), x=L for left-justified on the page, x=C for centered, x=B for make left-justified block, then center whole block. Implies .KS.
.EN	-	yes	Space after equation produced by eqn or neqn.
.EQ x y	-	yes	Precede equation; break out and add space. Equation number is y. The optional argument x is I to indent equation (default), L to left-adjust the equation, or C to center the equation.
.FE	-	yes	End footnote.
.FS	no	no	Start footnote. The note will be moved to the bottom of the page.
.I x	no	no	Italicize x; if x missing, italic text follows.
.IP x y	no	yes	Start indented paragraph, with hanging tag x. Indentation is y ens (default 5).
.KE	-	yes	End keep. Put kept text on next page if not enough room.
.KF	no	yes	Start floating keep. If the kept text must be moved to the next page, float later text back to this page.
.KS	no	yes	Start keeping following text.
.LG	no	no	Make letters larger.
.LP	yes	yes	Start left-blocked paragraph.
.NH n	-	yes	Same as .SH, with section number supplied automatically. Numbers are multilevel, for example 1.2.3, where n tells what level is wanted (default is 1).
.NL	yes	no	Make letters normal size.
.OK	-	yes	Other keywords for TM cover sheet follow.
.PP	no	yes	Begin paragraph. First line indented.
.PT	pg #	-	Page title, automatically invoked at top of page. May be redefined.
.QE	-	yes	End quoted (indented and shorter) material.
.QP	-	yes	Begin single paragraph which is indented

			and shorter.
.QS	-	yes	Begin quoted (indented and shorter) material.
.R	yes	no	Roman text follows.
.RE	-	yes	End relative indent level.
.RP	no	-	Cover sheet and first page for released paper. Must precede other requests.
.RS	-	yes	Start level of relative indentation. Following .IP's are measured from current indentation.
.SG x	no	yes	Insert signature(s) of author(s), ignored except in TM. x is the reference line (initials of author and typist).
.SH	-	yes	Section head follows, font automatically bold.
.SM	no	no	Make letters smaller.
.TA x...	5...	no	Set tabs in ens. Default is 5 1Ø 15 ...
.TE	-	yes	End table.
.TH	-	yes	End heading section of table.
.TL	no	yes	Title follows.
.TS x	-	yes	Begin table; if x is H table has repeated heading.
.UL x	-	no	Underline argument (even in troff).
.UX	-	no	`UNIX'; first time used, add footnote

NAME

mv - a macro package for making view graphs

SYNOPSIS

mvt [options] [files]
troff -mv [options] [files]

DESCRIPTION

This package provides an easy-to-use facility for making view graphs and projection slides in a variety of formats. A dozen or so macros are provided that accomplish most of the formatting tasks needed in making transparencies. All of the facilities of **troff**(1), **eqn**(1), and **tbl**(1) are available for more difficult tasks. The output can be previewed on most terminals, and, in particular, on the Tektronix 4014 and on the Versatec printer. See the reference below for further details.

FILES

/usr/lib/tmac/tmac.v

SEE ALSO

eqn(1), mvt(1), tbl(1), troff(1).

NAME

regexp - regular expression compile and match routines

SYNOPSIS

```
#define INIT declarations
#define GETC() getc code
#define PEEKC() peekc code
#define UNGETC(c) ungetc code
#define RETURN (pointer) return code
#define ERROR (val) error code

#include <regexp.h>

char *compile(instring, expbuf, endbuf, eof)
char *instring, *expbuf, *endbuf;

int step(string, expbuf)
char *string, *expbuf;
```

DESCRIPTION

This page describes general purpose regular expression matching routines in the form of **ed**(1), defined in /usr/include/regexp.h. Programs such as **ed**(1), **sed**(1), **grep**(1), **expr**(1), etc., which perform regular expression matching use this source file. In this way, only this file need be changed to maintain regular expression compatibility.

The interface to this file is complex. Programs including this file must have the following five macros declared before the "#include <regexp.h>" statement. These macros are used by the compile routine.

GETC() — Return the value of the next character in the regular expression pattern. Successive calls to GETC() should return successive characters of the regular expression.

PEEKC() — Return the next character in the regular expression. Successive calls to PEEKC() should return the same character (which should also be the next character returned by GETC()).

UNGETC(c) — Cause the argument c to be returned by the next call to GETC() (and PEEKC()). No more that one character of pushback is ever needed and this character is guaranteed to be the last character read by GETC(). The value of the macro UNGETC(c) is always ignored.

RETURN(pointer) This macro is used on normal exit of the compile routine. The value of the argument pointer is a pointer to the character after the last character of the compiled regular expression. This is useful to programs which have memory allocation to manage.

ERROR(val) This is the abnormal return from the compile routine. The argument val is an error number (see table below for meanings). This call should never return.

ERROR	MEANING
11	Range endpoint too large.
16	Bad number.
25	"\digit" out of range.
36	Illegal or missing delimiter.
41	No remembered search string.
42	**\(\)** imbalance.
43	Too many **\(**.
44	More than 2 numbers given in **\{ \}**.
45	**}** expected after ****.
46	First number exceeds second in **\{ \}**.
49	**[ ]** imbalance.
5Ø	Regular expression overflow.

The syntax of the compile routine is as follows:

compile(instring, expbuf, endbuf, eof)

The first parameter instring is never used explicitly by the compile routine but is useful for programs that pass down different pointers to input characters. It is sometimes used in the INIT declaration (see below). Programs which call functions to input characters or have characters in an external array can pass down a value of ((char *) Ø) for this parameter.

The next parameter expbuf is a character pointer. It points to the place where the compiled regular expression will be placed.

The parameter endbuf is one more than the highest address that the compiled regular expression may be stored. If the compiled expression doesn't fit in (endbuf-expbuf) bytes, a call to ERROR(5Ø) is made.

The parameter eof is the character which marks the end of the regular expression. For example, in **ed**(1), this character is usually a /.

Each program that includes this file must have a **#define** statement for INIT. This definition will be placed right after the declaration for the function compile and the opening curly brace ({). It is used for dependent declarations and initializations. Most often it is used to set a register variable to point to the beginning of the regular expression so that this register variable can be used in the declarations for GETC(), PEEKC() and UNGETC(). Otherwise it can be used to declare external variables that might be used by GETC(), PEEKC() and UNGETC(). See the example below of the declarations taken from **grep**(1).

There are other functions in this file which perform actual regular expression matching, one of which is the function step. The call to step is as follows:

```
step(string, expbuf)
```

The first parameter to step is a pointer to a string of characters to be checked for a match. This string should be null terminated.

The second parameter expbuf is the compiled regular expression which was obtained by a call of the function compile.

The function step returns one, if the given string matches the regular expression, and zero if the expressions do not match. If there is a match, two external character pointers are set as a side effect to the call to step. The variable set in step is locl. This is a pointer to the first character that matched the regular expression. The variable loc2, which is set by the function advance, points to the character after the last character that matches the regular expression. Thus if the regular expression matches the entire line, locl will point to the first character of string and loc2 will point to the null at the end of string.

Step uses the external variable circf which is set by compile if the regular expression begins with ^. If this is set then step will only try to match the regular expression to the beginning of the string. If more than one regular expression is to be compiled before the the first is executed the value of circf should be saved for each compiled expression and circf should be set to that saved value before each call to step.

The function advance is called from step with the same arguments as step. The purpose of step is to step through the string argument and call advance until advance returns a one indicating a match or until the end of string is reached. If one wants to constrain string to the beginning of the line in all cases, step need not be called, simply call

advance.

When advance encounters a * or \{ \} sequence in the regular expression it advances its pointer to the string to be matched as far as possible and will recursively call itself trying to match the rest of the string to the rest of the regular expression. As long as there is no match, advance backs up along the string until it finds a match or reaches the point in the string that initially matched the * or \{ \}. It is sometimes desirable to stop this backing up before the initial point in the string is reached. If the external character pointer locs is equal to the point in the string at sometime during the backing up process, advance will break out of the loop that backs up and will return zero. This is used by **ed**(1) and **sed**(1) for substitutions done globally (not just the first occurrence, but the whole line). For example, expressions like **s/y*//g** do not loop forever.

The routines ecmp and getrange are trivial and are called by the routines previously mentioned.

EXAMPLES

The following is an example of how the regular expression macros and calls look from **grep**(1):

```
#define INIT   register char *sp = instring;
#define GETC() (*sp++)
#define PEEKC()     (*sp)
#define UNGETC(c)   (--sp)
#define RETURN(c)   return;
#define ERROR(c)    regerr()

#include <regexp.h>
...
        compile(*argv, expbuf, &expbuf[ESIZE], '\Ø');
...
        if(step(linebuf, expbuf))
                succeed();
```

FILES

/usr/include/regexp.h

SEE ALSO

ed(1), grep(1), sed(1).

LIMITATIONS

The handling of circf is kludgy.
The routine ecmp is equivalent to the Standard I/O routine strncmp and should be replaced by that routine.

NAME

stat - data returned by stat system call

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
```

DESCRIPTION

The system calls **stat** and **fstat**(2) return data whose structure is defined by this include file. The encoding of the field st mode is defined in this file also.

```
struct  stat
{
        dev_t   st_dev;
        ino_t   st_ino;
        unsigned short st_mode;
        short   st_nlink;
        short   st_uid;
        short   st_gid;
        dev_t   st_rdev;
        off_t   st_size;
        time_t  st_atime;
        time_t  st_mtime;
        time_t  st_ctime;
};

#define S_IFMT  0170000         /* type of file */
#define         S_IFDIR 0040000 /* directory */
#define         S_IFCHR 0020000 /* character special */
#define         S_IFBLK 0060000 /* block special */
#define         S_IFREG 0100000 /* regular */
#define         S_IFMPC 0030000 /* multiplexed char special */
#define         S_IFMPB 0070000 /* multiplexed block special */
#define S_ISUID 0004000         /* set user id on execution */
#define S_ISGID 0002000         /* set group id on execution */
#define S_ISVTX 0001000         /* save swapped text even after
#define S_IREAD 0000400         /* read permission, owner */
#define S_IWRITE        0000200 /* write permission, owner */
#define S_IEXEC 0000100         /* execute/search permission, o
```

FILES

/usr/include/sys/types.h
/usr/include/sys/stat.h

SEE ALSO

stat(2).

NAME

terminals- conventional names

DESCRIPTION

These names are used by certain commands and are maintained as part of the shell environment (**sh**(1), **environ**(5)).

1620	DIABLO 1620 (and others using HyType II)
1620-12	same, in 12-pitch mode
300	DASI/DTC/GSI 300 (and others using HyType I)
300-12	same, in 12-pitch mode
300s	DASI/DTC 300/S
300s-12	same, in 12-pitch mode
33	TELETYPE(Reg.) Model 33
37	TELETYPE Model 37
40-2	TELETYPE Model 40/2
43	TELETYPE Model 43
450	DASI 450 (same as Diablo 1620)
450-12	same, in 12-pitch mode
450-12-8	same, in 12-pitch, 8 lines/inch mode
735	Texas Instruments TI735 (and TI725)
745	Texas Instruments TI745
dumb	terminals with no special features
hp	Hewlett-Packard HP264? series terminals
4014	Tektronix 4014
tn1200	General Electric TermiNet 1200
tn300	General Electric TermiNet 300
vt05	Digital Equipment Corp. VT05

Commands whose behavior depend on the terminal accept arguments of the form **-Tterm,** where term is one of the names given above. If no such argument is present, a command consults the shell environment for the terminal type.

SEE ALSO

stty(1), tabs(1), plot(1), sh(1), environ(5) troff(1).

NAME

terminal list - terminal names and codes

DESCRIPTION

This is a list of terminals supported in /etc/termcap and /etc/termcap.others.

Under the heading "Terminal" is the manufacturer and model of the terminal and under "Code" are the possible codes that can be specified when setting up the terminal environment. To set up the terminal environment, use the following command:

setenv TERM code

For example, for a Vtz-2/10 terminal, the possible codes are "vz", "vtz", "mcz-2/60" and "vtz-2/10"; therefore, any one of these codes can be used in the "setenv" command.

In the following list, an asterisk (*) following the terminal name indicates that the terminal description is in /etc/termcap.others. If this is the case, the termcap entry must be moved to /etc/termcap, or, the following command must be specified:

setenv TERMCAP /etc/termcap.others

Otherwise, if no asterisk is indicated following the terminal name, the terminal description is in /etc/termcap.

NOTE

Some of the entries shown on the following pages have been reformatted to fit the page. Refer to the file /etc/termcap for the actual format of the file.

Terminal	Code
Act5 (skinny)*	mS | act5s
Addrinfo*	ia | addrinfo
Addrinfo	si
Adds Regent 25	cR | regent 25 | adds25
Adds Regent 25*	a3 | regent25
Adds Regent 60*	a6 | regent60 | regent200
Adds consul 980*	ac | a980
Adds regent 100*	al | regent100
Adds regent 20*	a2 | regent20
Adds regent 40*	a4 | regent40
Adds regent series*	a0 | regent
Adds Viewpoint*	a7 | viewpoint
Ampex Dialogue 80*	MA | ampex | d80 | dialogue | dialogue80
Anderson Jacobson*	Mc | aj830 | aj832 | aj
Ann Arbor*	Ma | aa | annarbor
Ann Arbor Ambassador 48 (destructive backspace)*	MB | aaadb
Ann Arbor Ambassador/48 lines*	Mb | aaa | ambas | ambassador
Arpanet network*	sa | arpanet | network
Beehive IIIm	bh | bh3m | beehiveIIIm
Beehive super bee*	bs | sb1 | superbee | superb
Beehive super bee (fixed)*	b2 | sb2 | sb3
Beehive super Bee w/insert char*	bi | superbeeic
Bill Croft homebrew*	qB | bc
Bussiplexer*	sb | bussiplexer
C. Itoh 101 w/initialization	cs | cit | cit101 | cl01 | cit-101
C. Itoh 101 for 132-column	cw | citw | cit101w | cl01w | cit-101w
C. Itoh 101 w/o initialization	cn | citn | cit101n | cl01n
Carlock*	qc | carlock | klc
Carlock	cc | klc | carlock
Cdc456*	Ca | cdc456 | cdc
Cdc456tst	Cc | cdc456tst
Cdi1203*	Mi | cdi | cdi1203
CompucolorII*	MC | compucolor | compucolorII
Concept 100	c1 | c100 | concept100
Concept 100*	co | c100 | concept | concept100
Concept 100 slow*	cs | c100s | slowconcept

	slowconcept100
Concept 100 rev slow*	cd | c100rvs
Concept 100 rev video*	cr | c100rv
Concept 100 rev video	cr | c100rv | concept100rv
Concept 100 rev w/o arrows*	cn | c100rv4pna
Concept 100 w/4 pages*	cR | c100rv4p
Concept 100 with 4 pages*	c4 | c1004p
Ct82	c8 | swtp | ct82
Data General 6053*	Mg | dg | dg6053
Datagraphix 132a*	MD | d132 | datagraphix
Datamedia 1520*	D0 | dm1520 | 1520
Datamedia 1520	cm | dm1520
Datamedia 1521*	D1 | dm1521 | 1521
Datamedia 2500*	D2 | dm2500 | datamedia2500 | 2500
Datamedia 2500	c2 | dm2500
Datamedia 3025a*	D3 | dm3025
Datamedia 3025a	c3 | dm3025
Datamedia 3045a*	D4 | 3045 | dm3045
Datamedia dt80/1*	D5 | dt80 | dmdt80 | dm80
Datamedia dt80/1 with 132 char*	D6 | dt80132 | dmdt80132
Datapoint 3360*	Md | datapoint | dp3 | dp3360
Dec gt40*	d4 | gt40
Dec gt40	g0 | gt40
Dec gt42*	d2 | gt42
Dec gt42	g2 | gt42
Dec vt100	d1 | vt100 | vt-100 | pt100 | p
Dec vt100 132 cols*	dt | vt100w | vt-100w | pt100w | pt-100w
Dec vt100/132 cols 14 lines*	ds | vt100s | vt-100s | pt100s | pt-100s
Dec vt100 w/o initialization	d0 | vt100n
Dec vt132*	d3 | vt132 | vt-132
Dec vt50*	d5 | vt50
Dec vt50	v0 | vt50
Dec vt50h*	dh | vt50h
Dec vt50h	vh | vt50h
Dec vt52*	dv | vt52
Dec vt52	v2 | vt52
Decwriter I*	dI | dw1
Decwriter I	dw | dw1
Decwriter II*	dw | dw2 | dw3 | dw4
Decwriter II	td | dw2
Delta Data 5000*	ED | delta | dd5000
Diablo 1620*	A6 | 1620 | 450
Diablo 1640*	A7 | 1640
Diablo 300s	pd | dtc300s | dtc
Dialup*	sd | du | dialup
Digilog 333*	L3 | digilog | 333
Dtc 300s*	Ad | dtc300s | 300 | 300s | gsi | dtc
Ex3000*	qb | ex3000

Execuport 4000*	e2 | ep40 | ep4000
Execuport 4080*	el | ep48 | ep4080
Exidy smart*	qs | sexidy
Exidy sorcerer as dm2500*	qe | exidy | exidy2500
Fox 1100	cf | fox
General Terminal 100A*	il | i100 | gt100 | gt100a
Gsi*	Gs | gsi
Gsi or pg	pg | gsi
Hazeltine 1500*	H5 | h1500
Hazeltine 1500	h5 | h1500
Hazeltine 1510*	H6 | h1510
Hazeltine 1510	h6 | h1510
Hazeltine 1520*	H8 | h1520
Hazeltine 2000*	H7 | h2000
Hazeltine 2000	h7 | h2000
Hazeltine Esprit I*	H9 | esprit | hazeltine esprit
Heathkit h19	kb | h19 | heath | h19b | heathkit
Heathkit w/keypad shifted	kB | h19bs
Heathkit w/keypad shifted and underscore cursor	kU | h19us
Heathkit with underscore cursor	ku | h19u
Hewlett-Packard 2621*	h2 | 2621 | hp2621 | hp2621a | hp2621p | 2621a | 2621p
Hewlett-Packard 2621 with 45 keyboard*	h3 | 2621k45 | hp2621k45 | k45
Hewlett-Packard 2621 w/labels*	hw | 2621wl | hp2621wl | 2621
Hewlett-Packard 2621 w/o labels*	hn | 2621nl | hp2621nl | 2621
Hewlett-Packard 2626*	h6 | hp2626 | hp2626a | hp2626p | 2626 | 2626p | 2626a
Hewlett-Packard 2640a*	ha | 2640 | hp2640a | 2640a
Hewlett-Packard 2645	cH | 2645 | hp2645
Hewlett-Packard 2648a graphics terminal*	h8 | hp2648 | hp2648a | 2648a | 264
Hewlett-Packard 264x series*	h4 | hp | hp2645 | 2645 | hp2640b | 2640b
Hewlett-Packard 48 line 2621*	hb | big2621
IBM 3101-10*	I9 | ibm | ibm3101 | 3101 | i3101
ISC 8001*	I8 | 8001 | ISC8001
ISC modified owl 1200*	It | intext
Infoton 100	cl | i100
Infoton 200	i2 | i200
Infoton 400	i4 | i400 | 400
InfotonKAS	ci | InfotonKAS
InfotonKAS*	ik | infotonKAS
Lear Siegler adm3	cl | adm3 | 3
Lear Siegler adm3*	l3 | adm3 | 3
Lear Siegler adm31	Cl | adm31 | 31
Lear Siegler adm31*	ll | adm31 | 31
Lear Siegler adm3a	ca | adm3a | 3a
Lear Siegler adm3a*	la | adm3a | 3a

Lear Siegler adm3b	cb	adm3b | 3b
Lear Siegler adm42*	14	adm42 | 42
Mcz vdb	mz	mcz
Mcz vdb*	mz	mcz
Micro bee series*	bm	microb | microbee
Microterm	c4	act4 | microterm
Microterm Mime2a (emulating enhanced soroc iq120)*	ms	mime2as
Microterm Mime2a (emulating enhanced vt52)*	mv	mime2a | mime2av
Microterm act iv*	m4	microterm | act4
Microterm act v*	m5	microterm5 | act5
Microterm mime1*	mm	mime | mime1 | mime2 | mimei | mimeii
Microterm5	c5	act5 | microterm5
Mime1	c6	mime | mime1 | mimei | mimeii
Mime1 emulating 3a*	m3	mime3a
Mime1 emulating enhanced 3a*	mx	mime3ax
Mime1 (half bright)*	mh	mimehb
Model 33 teletype*	T3	33 | tty33 | tty
Model 37 teletype*	T7	37 | tty37
Model 43 teletype*	T4	43 | tty43
NUC homebrew*	qN	nucterm
Netronics*	qn	netx | netronics
Omron 8025AG*	Mo	omron
Owl 1200	co	owl
Perkin Elmer 1100*	pf	fox
Perkin Elmer 1200*	po	owl
Plasma panel*	Mp	plasma
Plugboard*	sp	plugboard
Qume Sprint 5*	Aq	qume5 | qume
Sol*	Ml	
Soroc 120*	MS	soroc
Southwest Technical Products ct82*	Ms	swtp | ct82
Special*	qq	ccncrt | special
TI silent 700*	t3	ti | ti700 | ti733 | 735 | ti735
TI silent 700	ti	ti700 | ti733
TI silent 745*	t4	ti745 | 745 | 743
TI silent 745	tI	ti745
Tektronix 4012*	X1	tek | tek4012 | 4012 | tek4013 | 4013
Tektronix 4014*	X2	tek4014 | tek4015 | 4014 | 4015
Tektronix 4014	gk	tk4014 | 4014
Tektronix 4014 in small font*	X3	tek4014sm | tek4015sm | 4014sm | 4015sm
Tektronix 4023*	X4	tek4023 | 4023

Tektronix 4024/4025/4027*	X5	tek4027 | 4025 | 4027 | 4024 | tek4025 | tek4024 | 4025cu | 4027cu
Tektronix 4025 17 line window in workspace*	X8	4025-17ws | 4027-17ws
Tektronix 4025 with "!"*	Xe	4025ex | 4027ex
Tektronix 4025 w/17 line window*	X7	4025-17 | 4027-17
Teleray 1061	ct	t1061
Teleray 1061*	y6	t1061 | t10 | teleray
Teleray 1061 with fast PROMs*	yf	t1061f
Teleray 3700 (dumb)*	yl	t3700 | teleray
Teleray 3800 series*	y3	t3800
Teletec Datascreen*	Mk	teletec | tec
Televideo	cT	tvi920 | tvi912 | tvisp | tvi
Televideo (new)*	v2	912b | 912c | 920b | 920c | tvi
Televideo (old)*	vl	tvi912 | 912 | 920 | tvi920
Televideo 925*	v3	925 | tvi925
Televideo 950*	v5	tvi950 |950
Terak emulating Datamedia 1520*	Mt	terak
Terminet 1200	T2	1200
Terminet 1200*	g2	1200 | tn1200
Terminet 300	T3	300
Terminet 300*	g3	300 | tn300
Tty33	t3	33 | tty33
Tty37	t7	37 | tty37
Tty43	t4	43 | tty43
TtyWilliams	dp	
TtyWilliams*	qw	ttyWilliams
Ubellchar*	qu	ubell
Ubellchar	tu	ubell | ubellchar
Unknown*	su	dumb | un | unknown
Unknown	un	pb | dumb | unknown
Visual 200*	V2	vi200 | visual
Visual 200 reverse video*	Vr	vi200rv
Visual 200 reverse video using insert char*	VR	vi200rvic
Visual 200 using insert char*	Vt	vi200ic
Vtz-2/10 or mcz-2/60	vz	vtz | mcz-2/60 | vtz-2/10
Xerox 1720*	xl	xl720
Xitex sct-100	cx	xitex
Xitex sct-100*	qx	xitex
Zentec 30*	Mz	zen30 | z30

FILES

/usr/pub/termlist

SEE ALSO

ex(1), vi(1), environ(5).

NAME

types - primitive system data types

SYNOPSIS

#include <sys/types.h>

DESCRIPTION

The data types defined in the include file are used in UNIX system code; some data of these types are accessible to user code:

```
typedef struct { int r[1]; } *  physadr;
typedef long                    daddr_t;
typedef char *                  caddr_t;
typedef unsigned short          ushort;
typedef ushort                  ino_t;
typedef char                    cnt_t;
typedef long                    saddr_t;
typedef long                    paddr_t;
typedef long                    time_t;
typedef int                     label_t[9];
typedef int                     dev_t;
typedef long                    off_t;
```

The form daddr_t is used for disk addresses except in an i-node on disk, see **filesys**(5). Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The label_t variables are used to save the processor state while another process is running.

SEE ALSO

filsys(5).

NAME

acct - overview of accounting and miscellaneous accounting commands

SYNOPSIS

acctdisk

acctdusg [**-p** file] [**-u** file] > dtmp-file

accton [file]

acctwtmp [name[line]] >> /usr/adm/wtmp

DESCRIPTION

Accounting software is structured as a set of tools (consisting of both C programs and shell procedures) that can be used to build accounting systems. **Acctsh**(M) describes the set of shell procedures developed from C programs.

Connect time accounting is handled by various programs that write records into /usr/adm/wtmp, as described in **utmp**(5). The programs described in **acctcon**(M) convert this file into session and charging records, which are then summarized by **acctmerg**(M).

Process accounting is performed by the ZEUS kernel. Upon termination of a process, one record per process is written to a file (normally /usr/adm/pacct). The programs in **acctprc**(M) summarize this data for charging purposes; **acctcms**(M) is used to summarize command usage. Current process data may be examined using **acctcom**(1).

Process accounting and connect time accounting (or any accounting records described in **acct**(5)) can be merged and summarized into total accounting records by **acctmerg** (see tacct format in **acct**(5)). **Prtacct** (see **acctsh**(M)) is used to format any or all accounting records.

Acctdisk reads lines that contain user ID, login name, and number of disk blocks, and converts them to total accounting records that can be merged with other accounting records.

Acctdusg reads its standard input (usually from **find / -print**) and computes disk resource consumption (including indirect blocks) by login.

Accton alone turns process accounting off. If file is given, it must be the name of an existing file, to which the kernel appends process accounting records (see **acct**(2) and **acct**(5)).

Acctwtmp writes a **wtmp**(5) record to its standard output. The record contains the current time, name, and line. If line is omitted, a value is emitted that is interpreted by other programs as a reboot. For more precise accounting, the following are recommended for use in reboot and shutdown procedures, respectively:

```
acctwtmp uname >>/usr/adm/wtmp
acctwtmp reason >>/usr/adm/wtmp
```

OPTIONS

-p file is the name of the password file. This option is not needed if the password file is /etc/passwd.

-u records consisting of those file names for which acctdusg charges no one are placed in file (a potential source for finding users trying to avoid disk charges).

FILES

/etc/passwd	used for login name to user ID conversions
/usr/lib/acct	holds all accounting commands listed in section M of this manual
/usr/adm/pacct	current process accounting file
/usr/adm/wtmp	login/logoff history file

SEE ALSO

acctcom(M), acct(2), acct(5), utmp(5), acctcms(M), acctcon(M), acctmerg(M), acctprc(M), acctsh(M), fwtmp(M), runacct(M).
The PWB/ZEUS Accounting System in the ZEUS Utilities Manual

NAME

acctcms - command summary from per-process accounting records

SYNOPSIS

acctcms [options] files

DESCRIPTION

Acctcms reads one or more files, normally in the form described in **acct**(5). It adds all records for processes that executed identically-named commands, sorts them, and writes them to the standard output, normally using an internal summary format.

OPTIONS

-a Print output in ASCII rather than in the internal summary format. The output includes command name, number of times executed, total kcore-minutes, total CPU minutes, total real minutes, mean size (in K), mean CPU minutes per invocation, and "hog factor", as in **acctcom**(M). Output is normally sorted by total kcore-minutes.

-c Sort by total CPU time, rather than total kcore-minutes.

-j Combine all commands invoked only once under "***other".

-n Sort by number of command invocations.

-s Any file names encountered hereafter are already in internal summary format.

EXAMPLES

A typical sequence for performing daily command accounting and for maintaining a running total is:

```
acctcms file ... >today
cp total previoustotal
acctcms -s today previoustotal >total
acctcms -a -s today
```

SEE ALSO

acctcom(M), acct(2), acct(5), utmp(5), acct(M), acctcon(M), acctmerg(M), acctprc(M), acctsh(M), fwtmp(M), runacct(M).

NAME

acctcom - search and print process accounting file(s)

SYNOPSIS

acctcom [[options] [file]]

DESCRIPTION

Acctcom reads file, the standard input, or /usr/adm/pacct, in the form described by **acct**(5) and writes selected records to the standard output. Each record represents the execution of one process. The output shows:

COMMAND NAME
USER
TTYNAME
START TIME
END TIME
REAL (SEC)
CPU (SEC)
MEAN SIZE(K)

and optionally, **F** (the **fork/exec** flag: **1** for **fork** without **exec**) and **STAT(the system exit status).**

The command name is prepended with a **#** if it was executed with super-user privileges. If a process is not associated with a known terminal, a **?** is printed in the **TTYNAME** field.

If no files are specified and if the standard input is associated with a terminal or /dev/null (as is the case when using & in the shell), /usr/adm/pacct is read, otherwise the standard input is read.

If any file arguments are given, they are read in their respective order. Each file is normally read forward, i.e., in chronological order by process completion time. The file /usr/adm/pacct is usually the current file to be examined; a busy system may need several files, in which case all but the current will be found in **/usr/adm/pacct?.**

OPTIONS

-b Read backwards, showing latest commands first.

-C time Show only those processes that exceed time that indicates the total CPU time.

-d mm/dd Any time arguments following this flag are assumed to occur on the given month and day, rather than during the last 24 hours. This is needed for looking at old files.

-e time Show only those processes that existed on or

before *time*. Using the same *time* for both **-s** and **-e** shows the processes that existed at *time*.

-f Print the **fork/exec** flag and system exit status columns in the output.

-g *group* Show only processes belonging to *group*. The *group* may be designated by either the group ID or group name.

-h Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This "hog factor" is computed as:

(total CPU time)/(elapsed time).

-H *factor* Show only processes that exceed *factor*, where factor is the "hog factor" as explained in option **-h** above.

-i Print columns containing the I/O counts in the output.

-k Instead of memory size, show total kcore-minutes.

-l *line* Show only processes belonging to terminal /*dev*/*line*.

-m Show mean core size (the default).

-n *pattern* Show only commands matching *pattern* that may be a regular expression as in **ed**(1) except that + means one or more occurrences.

-O *time* Show only those processes with operating system CPU time exceeding *time*.

-r Show CPU factor (user time/(system-time + user-time).

-s *time* Show only those processes that existed on or after *time*, given in the form *hr*:*min*:*sec*. The :*sec* or :*min*:*sec* may be omitted.

-t Show separate system and user CPU times.

-u *user* Show only processes belonging to *user* that may be specified by: a user ID, a login name that is then converted to a user ID, a **#** which designates only those processes executed with *super-user* privileges, or **?** which designates only

those processes associated with unknown user IDs.

-v Exclude column headings from the output.

Listing options together has the effect of a logical and.

FILES

/etc/passwd
/usr/adm/pacct
/etc/group

SEE ALSO

ps(1), su(1), acct(2), acct(5), utmp(5), acct(M), acctcms(M), acctcon(M), acctmerg(M), acctprc(M), acctsh(M), fwtmp(M), runacct(M).

LIMITATIONS

Acctcom only reports on processes that have terminated; use **ps**(1) for active processes.

NAME

acctcon - connect-time accounting

SYNOPSIS

acctcon1 [options]

acctcon2

DESCRIPTION

Acctcon1 converts a sequence of login/logoff records read from its standard input to a sequence of records, one per login session. Its input should normally be redirected from /usr/adm/wtmp. Its output is ASCII, giving device, user ID, login name, prime connect time (seconds), non-prime connect time (seconds), session starting time (numeric), and starting date and time.

Acctcon2 expects as input a sequence of login session records and converts them into total accounting records (see **tacct** format in **acct**(5)).

OPTIONS

-l file
File is created to contain a summary of line usage showing line name, number of minutes used, percentage of total elapsed time used, number of sessions charged, number of logins, and number of logoffs. This file helps track line usage, identify bad lines, and find software and hardware oddities. Both hang-up and termination of the login shell generate a logoff record, so that the number of logoffs is often twice the number of sessions.

-o file
File is filled with an overall record for the accounting period, giving starting time, ending time, number of reboots, and number of date changes.

-p Print input only, showing line name, login name, and time (in both numeric and date/time formats).

-t **Acctcon1** maintains a list of lines on which users are logged in. When it reaches the end of its input, it emits a session record for each line that still appears to be active. It normally assumes that its input is a current file, so that it uses the current time as the ending time for each session still in progress. The **-t** flag causes it to use the last time found in its input, assuring reasonable and repeatable numbers for non-current files.

EXAMPLES

These commands are typically used as shown below. The file **ctmp** is created only for the use of **acctprc**(M) commands:

```
acctcon1 -t -l lineuse -o reboots <wtmp | sort +1n +2 >ctmp
acctcon2 <ctmp | acctmerg >ctacct
```

FILES

/usr/adm/wtmp

SEE ALSO

acctcom(M), acct(2), acct(5), utmp(5), acct(M), acctcms(M), acctmerg(M), acctprc(M), acctsh(M), fwtmp(M), runacct(M).

LIMITATIONS

The line usage report is confused by date changes. Use **wtmpfix** (see **fwtmp**(M)) to correct this situation.

NAME

acctmerg - merge or add total accounting files

SYNOPSIS

acctmerg [-aiptuv] [file]

DESCRIPTION

Acctmerg reads its standard input and up to nine additional files, all in the **tacct** format (see **acct**(5)), or an ASCII version. It merges these inputs by adding records whose keys (normally user ID and name) are identical, and expects the inputs to be sorted on those keys.

OPTIONS

-a Produce output in ASCII version of **tacct.**

-i Input files are in ASCII version of **tacct.**

-p Print input with no processing.

-t Produce a single record that totals all input.

-u Summarize by user ID, rather than user ID and name.

-v Produce output in verbose ASCII format, with more precise notation for floating point numbers.

EXAMPLES

The following sequence is useful for making "repairs" to any file kept in this format:

```
acctmerg -v <file1 >file2
     edit file2 as desired ...
acctmerg -i <file2 >file1
```

SEE ALSO

acctcom(M), acct(2), acct(5), utmp(5), acct(M), acctcms(M), acctcon(M), acctprc(M), acctsh(M), fwtmp(M), runacct(M).

NAME

acctprc - process accounting

SYNOPSIS

acctprc1 [**ctmp**]

acctprc2

DESCRIPTION

Acctprc1 reads input in the form described by **acct**(5), adds login names corresponding to user IDs, then writes for each process an ASCII line giving user ID, login name, prime CPU time (tics), non-prime CPU time (tics), and mean memory size (in 64-byte units). If **ctmp** is given, it is expected to contain a list of login sessions, in the form described in **acctcon**(M), sorted by user ID and login name. If this file is not supplied, it obtains login names from the password file. The information in **ctmp** helps it distinguish among different login names that share the same user ID.

Acctprc2 reads records in the form written by **acctprc1**, summarizes them by user ID and name, then writes the sorted summaries to the standard output as total accounting records.

EXAMPLES

These commands are typically used as shown below:

```
acctprc1 ctmp </usr/adm/pacct | acctprc2 >ptacct
```

FILES

/etc/passwd

SEE ALSO

acctcom(M), acct(2), acct(5), utmp(5), acct(M), acctcms(M), acctcon(M), acctmerg(M), acctsh(M), fwtmp(M), runacct(M).

LIMITATIONS

Although it is possible to distinguish among login names that share user IDs for commands run normally, it is difficult to do this for those commands run from **cron**(M), for example. More precise conversion can be done by faking login sessions on the console via the **acctwtmp** program in **acct**(M).

NAME

acctsh - shell procedures for accounting

SYNOPSIS

chargefee login-name number

ckpacct [blocks]

dodisk

lastlogin

monacct number

nulladm file

prctmp

prdaily

prtacct file [heading]

runacct [mmdd] [mmdd state]

shutacct [reason]

startup

turnacct [**on** | **off** | **switch**]

DESCRIPTION

Chargefee is invoked to charge number dollars to login-name. A record is written to /usr/adm/fee, to be merged with other accounting records during the night.

Ckpacct is initiated via **cron**. It periodically checks the size of /usr/adm/pacct. If the size exceeds blocks, 1000 by default, **turnacct** will be invoked with argument switch.

Dodisk is invoked by **cron** to perform the disk accounting functions.

Lastlogin is invoked by **runacct** to update /usr/adm/acct/sum/loginlog, which shows the last date on which each person logged in.

Monacct should be invoked once each month or each accounting period. Number indicates which month or period it is. It creates summary files in /usr/adm/acct/fiscal and restarts summary file in /usr/adm/acct/sum. **Nulladm** creates file with mode 644 and insures owner is **adm**. It is called by **lastlogin**, **runacct**, and **turnacct**.

Prctmp can be used to print the session record file (normally /usr/adm/acct/nite/ctmp created by **acctcon1** (see **acctcon**(M)).

Prdaily is invoked by **runacct** to print a report of the previous day's accounting. The report resides in /usr/adm/acct/sum/rprt xxxx where xxxx is the month and day of the report. The daily accounting reports may be printed with the command:

cat /usr/adm/acct/sum/rprt*

as often as desired and they must be explicitly deleted when no longer needed.

Prtacct can be used to format and print any total accounting file.

Runacct performs the accumulation of connect, process, fee, and disk accounting on a daily basis. It also creates summaries of command usage. For more information, see runacct(M).

Shutacct should be invoked during a system shutdown to turn process accounting off and append a "reason" record to /usr/adm/wtmp. **Startup** should be called by **rc**(8) to turn the accounting on whenever the system is brought up.

Turnacct is an interface to **accton** (see **acct**(M)) to turn process accounting **on** or **off**. The **switch** argument moves the current /usr/adm/pacct to the next free name in /usr/adm/pacct[1-9], turns accounting off, then turns it back on again. This procedure is called by **ckpacct** via the **cron** to keep the pacct file size smaller.

FILES

/usr/adm/fee	accumulator for fees
/usr/adm/pacct	current file for per-process accounting
/usr/adm/pacct[1-9]	used if pacct gets large and during execution of daily accounting procedure
/usr/adm/wtmp	login/logoff summary
/usr/adm/wtmp[1-9]	used during daily accounting procedure
/usr/adm/acct/nite	working directory
/usr/lib/acct	holds all accounting commands listed in sub-class M of this manual
/usr/adm/acct/sum	summary directory, should be saved

SEE ALSO

acct(M), acctcms(M), acctcom(M), acctcon(M), acctmerg(M), acctprc(M), fwtmp(M), runacct(M), acct(2), acct(5), utmp(5).

NAME

adduser - add a new user to the system

SYNOPSIS

adduser

DESCRIPTION

This shell procedure aids the system administrator in adding a user to the system. It updates /etc/passwd and /etc/group and creates various directories and files for the new user. **Adduser** is interactive and prompts for all needed information. The new user is assigned a uid that is the least used uid already in /etc/passwd.

Adduser prompts for the new user's name, home directory name, group at login, and other group names the user is also belong to. The home directory name must be the full, explicit pathname. The list of additional groups is used by **newgrp**(1). An example session of **adduser** is:

```
New user's name: doug
Scanning for a uid number.
Home directory: /z/doug
Login shell (/bin/csh is the default): <cr>
Default group at login ('other' is the default): <cr>

Enter groups OTHER THAN the default group that `doug'
should belong to. Enter one group name per line, with
a control-d to end. If the user should belong only to
her/his default group, just type control-d to the
question. Group: END

User:           doug
Uid:            76
Gid:            1
Dir:            /z/doug
Login shell:    /bin/csh

Okay to add (yes or no)? yes
doug: Added
New password:
Retype new password:
```

In response to the prompt "Okay to add (yes or no)? ", if anything other than a yes or a y is typed it will be interpreted an a no response.

FILES

/etc/group	group file
/etc/passwd	password file

/usr/spool/mail directory in which mailbox is created

SEE ALSO

mkdir(1), passwd(1), rmuser(M).

DIAGNOSTICS

Invalid inputs are rejected and re-requested.

LIMITATIONS

There is a tiny time window in which two simultaneously running copies of **adduser** might interact to their mutual harm. No check is made on the success of giving the new user an initial password. If the password initialization fails, (e.g. the password is not entered identically in response to the two requests from **passwd**(1) for the new password) the new user is left without a password, a possible security flaw.

Adduser is a C-shell script and is slow.

NAME

lp, text - service line printer spooler print requests

SYNOPSIS

/usr/lib/lp [options]
/usr/lib/text [options]

DESCRIPTION

These programs are used by the spooling system to do the actual copy of data from a print request to a printer device. Lp is used to print requests on line printers. Text is used to print requests on text quality printers.

Each backend is started by dqueuer(M). The way backends are started is controlled by the configuration file (see the System Adminstrators Manual, Section 7). A backend program inherits three open file descriptors from the dqueuer: RFFD (read file), SFFD (device) and STATUSFD (status log). Actual definitions of these variables can be found in /usr/include/spool.h. The lp and text backends copy data from file RFFD to file SFFD with minimal processing of input. Statuses are written to file STATUSFD.

In addition to printing, the backends perform functions required by the specific device being used. For instance, lp filters backspaces and text can perform stty(1) calls before printing. They also provide banner pages with appropriate information.

When invoked, backends are passed a parameter list generated by dqueuer(M). This list contains options generated by dqueuer(M) and those options that may have been specified in the configuration file. The options that dqueuer generates and that both backends recognize are:

-B Indicates that the file the dqueuer was to print could not be found. In this case, RFFD is not open and should not be used. This option is used to generate banner pages that say the file could not be found, so it won't appear that the file disappeared.

-c n
Indicates that the file must be printed multiple times. The backend should create n copies of the output.

-d dest
Used in a destination portion of the banner page. Dest is a string of no other importance to the backend.

-f filename
Filename, printed on the banner page.

-F from
Indicates a 'from' string, for use on the banner page.

-s spool time
A string in ctime(3) format, indicating when the file was spooled to be printed. This value is printed on banner page.

-t title
A string used as a title on banner pages.

The backend also looks in a file called '/etc/sitefile' to find a string to be printed near the page breaks on the banner page. It uses the string of characters on the first line of this file, up to the first white space. If this file does not exist, an appropriate default is used.

The text backend accepts one additional option, that can only specified through the configuration file. Its description follows:

-T [stty parameters]
Set terminal parameters. When a tty device is used as a printer, it is disabled in the /etc/inittab file. When this is done, the device is closed except when being used to print upon. This causes ZEUS to reset all terminal characteristics. The -T option is used to set these characteristics so that printing can occur. The quoted string is passed, unchanged except for removal of the quotes, to stty(1). Thus, in the configuration file the line

```
D1,R,/dev/tty0,/usr/lib/text -T "1200"
```

would cause device 1 on the most recent queue to use stty to set /dev/tty0 to 1200 BAUD before being used.

SEE ALSO
dqueuer(M)

NAME

boot - secondary bootstrapper

SYNOPSIS

boot

DESCRIPTION

Boot is invoked by the primary bootstrapper. For the Model 2Ø, the primary bootstrap resides on block Ø (the boot block) of the disk. For other models the primary bootstrapper resides in PROM on the CPU board.

Its primary function is to load the kernel from disk into memory during normal bootstrap operations, but it is also capable of loading and executing a variety of stand-alone programs from either disk or cartridge tape.

Boot must reside in the root of the first file system on the disk. As currently configured, this file system gets mounted on /usr, so the fully-qualified pathname of the primary bootstrapper is /usr/boot.

A new secondary bootstrapper can be made using **sysgen**(M).

FILES

/usr/boot program

SEE ALSO

init(M).

NAME

chmog, chog - change mode, owner and group of a file

SYNOPSIS

chmog octal-mode user group file

chog user group file

DESCRIPTION

Chmog will change the mode, owner and group of the listed files to octal-mode, user, and group respectively. **Chog** will change the owner and group of the listed files. The user and group may be specified by either the user id, group id or the user name or group name.

FILES

/etc/passwd
/etc/group
/etc/chmog
/etc/chog

SEE ALSO

chmod(1), chmod(2), chgrp(1), chown(M).

NAME

chown, chgrp - change owner or group

SYNOPSIS

chown owner file

chgrp group file

DESCRIPTION

Chown changes the owner of the files to owner. The owner may be either a decimal UID or a login name found in the password file.

Chgrp changes the group-ID of the files to group. The group may be either a decimal GID or a group name found in the group-ID file.

FILES

/etc/passwd	user names and user ID numbers
/etc/group	group names and ID numbers
/etc/chown	the program
/etc/chgrp	the program

SEE ALSO

chown(2), passwd(5), group(5), chmog(M).

NAME

chroot - change root directory for a command

SYNOPSIS

chroot newroot command

DESCRIPTION

The command is executed relative to the new root. The meaning of any initial slashes (/) in path names is changed for command and any of its children to newroot. The initial working directory is newroot. The new root path name is always relative to the current root: even if a **chroot** is currently in effect, the newroot argument is relative to the current root of the running process.

This command is restricted to the super-user.

EXAMPLES

The command syntax:

chroot newroot command > newfile

creates the file newfile relative to the original root, not the new one.

FILES

/etc/chroot the program

SEE ALSO

chdir(2).

LIMITATIONS

Exercise extreme caution when referencing special files in the new root file system.

NAME

clri - clear i-node

SYNOPSIS

clri filesystem i-number

DESCRIPTION

Clri writes zeros on the i-nodes with the decimal i-numbers on the filesystem. After **clri**, any blocks in the affected file will show up as `missing' in an **icheck**(M) of the filesystem.

Read and write permission is required on the specified file system device. The i-node becomes allocatable.

The primary purpose of this routine is to remove a file which for some reason appears in no directory. If it is used to zap an i-node which does appear in a directory, care should be taken to track down the directory entry and remove it first. Otherwise, when the i-node is reallocated to some new file, the old entry will still point to that file. At that point removing the old entry will destroy the new file. The new entry will again point to an unallocated i-node, so the whole cycle is likely to be repeated again and again.

FILES

/etc/clri program

SEE ALSO

icheck(M).

LIMITATIONS

If the file is open, **clri** is likely to be ineffective.

NAME

cron - clock daemon

SYNOPSIS

cron

DESCRIPTION

Cron executes commands at specified dates and times. It follows the instructions in /usr/lib/crontab. Because **cron** never exits, it should be executed only once. This is best done by running **cron** from the initialization process through the file /etc/rc (see **init**(M)).

The file crontab consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns specifying the minute (Ø-59), hour (Ø-23), day of the month (1-31), month of the year (1-12), and day of the week (Ø-6, with Ø=Sunday). Each of these patterns can contain:

a number in the (respective) range indicated above;
two numbers separated by a minus (indicating an inclusive range);
a list of numbers separated by commas (meaning all of these numbers); or
an asterisk (meaning all legal values).

The sixth field is a string that is executed by the shell at the specified time(s). A % in this field is translated into a new-line character. Only the first line (up to a % or the end of line) of the command field is executed by the shell. The other lines are made available to the command as standard input.

Cron examines crontab once a minute to see if it has changed; if it has, **cron** reads it. It takes only a minute for entries to become effective.

FILES

/usr/lib/crontab
/usr/lib/cronlog

SEE ALSO

sh(1), init(M).

DIAGNOSTICS

A history of all actions by **cron** are recorded in /usr/lib/cronlog.

LIMITATIONS

Cron reads crontab only when it has changed, but it reads the in-core version of that table once a minute. A more

efficient algorithm could be used. The overhead in running **cron** is about one percent of the CPU, exclusive of any commands executed by **cron.**

NAME

date - print and set the date

SYNOPSIS

date [**-u**] [mmddhhmm[yy]] [+format]

DESCRIPTION

If the **-u** option is given, GMT time is printed.

If no argument is given, or if the argument begins with **+**, the current date and time are printed. Otherwise, the current date is set. The first mm is the month number; dd is the day number in the month; hh is the hour number (24 hour system); the second mm is the minute number; yy is the last 2 digits of the year number and is optional. For example:

```
date 10080045
```

sets the date to Oct 8, 12:45 AM. The current year is the default if no year is mentioned. The system operates in GMT. **Date** takes care of the conversion to and from local standard and daylight time.

If the argument begins with **+**, the output of **date** is under the control of the user. The format for the output is similar to that of the first argument to **printf**(3). All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by **%** and will be replaced in the output by its corresponding value. A single **%** is encoded by **%%**. All other characters are copied to the output without change. The string is always terminated with a new-line character.

Field Descriptors:

n	insert a new-line character
t	insert a tab character
m	month of year - 01 to 12
d	day of month - 01 to 31
y	last 2 digits of year - 00 to 99
D	date as mm/dd/yy
H	hour - 00 to 23
M	minute - 00 to 59
S	second - 00 to 59
T	time as HH:MM:SS
j	Julian date - 001 to 366
w	day of week - Sunday = 0
a	abbreviated weekday - Sun to Sat
h	abbreviated month - Jan to Dec
r	time in AM/PM notation

EXAMPLE

```
date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'
```

would generate as output:

```
DATE: 08/01/76
TIME: 14:45:05
```

FILES

/bin/date the program

/usr/adm/wtmp to record time-setting

/dev/kmem

SEE ALSO

date(1), time(2), ctime(3), utmp(5)

DIAGNOSTICS

No permission if you aren't the super-user and you try to change the date.

bad conversion if the date set is syntactically incorrect.

bad format character if the field descriptor is not recognizable.

NAME

datem - friendly date and time setting

SYNOPSIS

datem

DESCRIPTION

Datem prompts for and sets the date and time.

Enter the date in either of the popular forms: dd-mo-yy, where dd is the day of the month, mo is the conventional abbreviation for the month, and yy is the last two digits of the year; or mm/dd/yy, where mm is the mm'th month of the year, dd is the day of the month, and yy is the last two digits of the year. Thus January 4, 1982, can be entered either as 4-Jan-82 or 1/4/82.

Enter the time using the 24-hour clock in the form hh:mm. (Time is local time.) Thus 3:31 P.M. is 15:31, midnight is Ø:ØØ, 8 a.m. is Ø8:ØØ.

All impossible and some unlikely entries are rejected. An empty date entry sets the time without changing the date.

If **datem** receives no reply in 5 minutes, it times out and exits.

Datem is normally invoked by including it in /etc/rc csh, the ZEUS startup file.

FILES

/etc/datem the program

SEE ALSO

date(1), rc(M).

DIAGNOSTICS

Not superuser if anyone but superuser attempts to change date and time.

LIMITATIONS

Doesn't review date or time.

NAME

dcheck - file system directory consistency check

SYNOPSIS

dcheck [**-i** numbers] [filesystem]

DESCRIPTION

Dcheck reads the directories in a file system and compares the link-count in each i-node with the number of directory entries by which it is referenced. If the file system is not specified, a set of default file systems is checked.

The **-i** flag is followed by a list of i-numbers; when one of those i-numbers turns up in a directory, the number, the i-number of the directory, and the name of the entry are reported.

The program is fastest if the raw version of the special file is used, since the i-list is read in large chunks.

FILES

/etc/dcheck program

SEE ALSO

filsys(5), clri(M), icheck(M), ncheck(M).

DIAGNOSTICS

When a file turns up for which the link-count and the number of directory entries disagree, the relevant facts are reported. Allocated files which have Ø link-count and no entries are also listed. The only dangerous situation occurs when there are more entries than links; if entries are removed, so the link-count drops to Ø, the remaining entries point to thin air. They should be removed. When there are more links than entries, or there is an allocated file with neither links nor entries, some disk space may be lost but the situation will not degenerate.

LIMITATIONS

Since **dcheck** is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

NAME

devnm - device name

SYNOPSIS

devnm [names]

DESCRIPTION

Devnm identifies the special file associated with the mounted file system where the argument name resides.

This command is most commonly used by /etc/rc csh to construct a mount table entry for the root device.

EXAMPLE

The command:

```
/etc/devnm /
```

produces

```
zd2 /
```

if / is mounted on /dev/zd2.

FILES

/dev/zd*
/dev/md*
/dev/smd*
/etc/mnttab
/etc/devnm program

SEE ALSO

setmnt(M).

NAME

df - report number of free disk blocks

SYNOPSIS

df [**-t**] [**-f**] [file-systems]

DESCRIPTION

Df prints out the number of free blocks and free i-nodes available for on-line file systems by examining the counts kept in the super-blocks; file-systems may be specified either by device name (e.g., /dev/zd1) or by mounted directory name (e.g., /usr). If the file-systems argument is unspecified, the free space on all of the mounted file systems is printed.

The **-t** flag causes the total allocated block figures to be reported as well.

If the **-f** flag is given, only an actual count of the blocks in the free list is made (free i-nodes are not reported). With this option, **df** will report on raw devices.

FILES

/dev/zd*
/dev/md*
/dev/smd*
/etc/mnttab

SEE ALSO

fsck(1), filesys(5), mnttab(5).

NAME

down - take the system down

SYNOPSIS

down [dtime btime]

DESCRIPTION

Down is a Shell script that can be used to take the system down in an orderly manner.

The arguments, dtime and btime, can be used to specify the time, in minutes, until system shutdown and the time between warning messages, respectively. If these arguments are not specified the user is prompted for them.

At the appropriate times a message is sent to all logged in users telling them how many minutes until system shutdown. When the total time for system shutdown has elapsed all system buffers are flushed and a **kill -1 1** is issued. This is done by invoking **halt**(M).

A line is added to the message-of-the-day file, saying that the system will be taken down shortly. The previous contents of the motd file are saved to be restored after the system has been halted.

RESTRICTIONS

This command will work only if issued by Superuser. **Down** will not let you take the system down any sooner than 5 minutes from the current time. If you specify a time until shutdown that is less than 5 minutes **down** will assume 5 minutes, and tell you so. If you want to take the system down now, but in an orderly manner, use **halt**(M).

FILES

/etc/motd message of the day file

/etc/motd.save message of the day file (saved)

SEE ALSO

kill(1), halt(M).

NAME

dqueuer - process and remove print queue command requests from **nq**

SYNOPSIS

dqueuer [-rnk]

DESCRIPTION

Dqueuer takes requests from **nq**(1) and **xq**(M) and prints files on a set of shared devices. It communicates with **nq** and **xq** through signals and sets of control files. **Dqueuer** is meant to run at all times and is usually initiated at system boot time through /etc/rc.

All the devices under the control of **dqueuer** are defined by the system administrator in /usr/spool/queuer/config. (See Section 7 of the System Administrator Manual.)

Dqueuer checks to see if the active configuration file, /usr/spool/queuer/activeconfig, exists or if the **-r** option (reread) is set. In either case, it parses the raw configuration file, /usr/spool/queuer/config, and creates the active config file.

The active configuration file contains the same information as the configuration file in a binary, machine convenient format and also includes the pid of **dqueuer** if it is running. It is the active configuration file that **nq** and **xq** use to process their requests.

If the dqueuer is already running when the -r option is given, the new process will signal the previous dqueuer and exit immediately. The signal will cause the current dqueuer process to reparse the configuration file and create the active configuration file.

The **-n** option (no execute) can be used to print a summary of the configuration file, to check for errors, without actually changing the active configuration file and without bringing up the dqueuer.

The **-k** option kills the dqueuer, although outstanding backends will finish their current file.

Dqueuer receives requests for service from **nq** and **xq**. These programs create files in the directory /usr/spool/queuer/requestdir with the appropriate information, and then send a signal to **dqueuer**. These files contain the information neccessary for processing print requests.

Upon receipt of a signal from either **nq** or **xq**, **dqueuer** wakes up and processes any outstanding requests it finds in the

request directory.

To actually print files, **dqueuer** must be supplied with a set of 'backend' programs that are specifically written for each device to be used. The names of these backends are supplied to **dqueuer** through the configuration file. Two are supplied with the queueing package, /usr/lib/lp for line printers and /usr/lib/text for text quality typewriter devices.

FILES

/usr/spool/queuer/config
/usr/spool/queuer/activeconfig
/usr/spool/queuer/statusdir
/usr/spool/queuer/requestdir
/tmp/queuer
/usr/spool/queuer/logfile

SEE ALSO

nq(1), xq(1), lp(M), spool(7), xq(M).

NAME

dump, sdump - incremental file system dump

SYNOPSIS

dump

[key [argument] [

DESCRIPTION

Dump copies to magnetic tape all files changed after a certain date in the filesystem. **Sdump** is the segmented version. It allows the 9 track tape drive to go into a higher rate of reading and writing.

The default destination device is /dev/dumpdev. If /dev/dumpdev does not exist, /dev/rctØ is used for the default.

Always perform dumps on quiescent filesystems. (Dumping active filesystems may lead to unfortunate results upon restor.)

KEYS

The key specifies the date and other options about the dump.

a If the dump will span more than one tape, then **dump** will abort before any data has been dumped and print a message to this effect onto the standard output device.

b Allow specification of blocksize to make the tape.

f Place the dump on the next argument file instead of /dev/dumpdev

h The next argument is taken as a user comment and is written onto the dump-tape header. 394 characters are allowed for this comment.

n The next argument is taken as the filesystem name being dumped and is is written onto the dump-tape header. 3Ø characters are allocated for this name.

s The next argument is the length of the tape in feet, default is 45Ø feet for cartridge tape and 24ØØ feet for mag tape.

u If the dump completes successfully, write the date of the beginning of the dump on file `/etc/ddate'. This file records a separate date for each filesystem and each dump level.

Ø-9 This number is the `dump level'. All files modified

since the last date stored in the file `/etc/ddate' for the same filesystem at lesser levels will be dumped. If no date is determined by the level, the beginning of time is assumed; thus the option **Ø** causes the entire filesystem to be dumped.

If no arguments are given, the key is assumed to be **9u** and the filesystem name and comment fields will contain nulls. The tape length will default to 45Ø feet.

EXAMPLES

Start with a full level Ø dump

```
dump Øu filesys
```

Next, periodic level 9 dumps should be made on an exponential progression of tapes. (Sometimes called Tower of Hanoi - 1 2 1 3 1 2 1 4 ... tape 1 used every other time, tape 2 used every fourth, tape 3 used every eighth, etc.)

```
dump 9u /dev/filesys
```

When the level 9 incremental approaches a full tape a level 1 dump should be made.

```
dump 1u /dev/filesys
```

After this, the exponential series should progress as uninterrupted. These level 9 dumps are based on the level 1 dump which is based on the level Ø full dump. This progression of levels of dump can be carried as far as desired.

FILES

/etc/ddate: record dump dates of filesystem/level.

SEE ALSO

dumpdir(M), restor(M).

LIMITATIONS

Read errors on the filesystem are ignored. Write errors on the magtape are usually fatal. When doing a dump of a filesystem, always use the same name for it. When **dump** reads the /etc/ddate file to determine the last date a file system was dumped, it considers '/dev/rzdØ' and '/dev/zdØ' two different filesystems, for example.

NAME

fsck - file system consistency check and interactive repair

SYNOPSIS

fsck [[**-fnsSty**] filesystem]

DESCRIPTION

Fsck audits and interactively repairs inconsistent conditions for the named filesystems. If a file system is consistent then the number of files, number of blocks used, and number of blocks free are reported. If the file system is inconsistent the operator is prompted for concurrence before each correction is attempted. Most corrections lose data; all losses are reported. The default action for each correction is to wait for the operator to respond `yes' or `no'. Without write permission **fsck** defaults to **-n** action.

If no filesystems are given to **fsck** then a default list of file systems is read from the file /etc/checklist.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one inode or the free list.

2. Blocks claimed by an inode or the free list outside the range of the file system.

3. Incorrect link counts.

4. Size checks:
   Incorrect number of blocks in file.
   Directory size not a multiple of 16 bytes.

5. Bad inode format.

6. Blocks not accounted for anywhere.

7. Directory checks:
   Discrepency between number of directory entries and inode link counts
   Inode number out of range.

8. Super Block checks:
   More than 65536 inodes.
   More blocks for inodes than there are in the file system.

9. Bad free block list format.

10. Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the "lost+found" directory. The name assigned is the inode number. The only restriction is that the directory "lost+found" must preexist in the root of the filesystem being checked and must have empty slots in which entries can be made. This is accomplished by making "lost+found", copying a large number of files to the directory, and then removing them (before **fsck** is executed). (See the System Administrator Manual for more information about making "lost+found" directories.)

Checking the raw device is almost always faster.

OPTIONS

-f Echoes the output from the command into /tmp/fsck.err. This option does not work if the filesystem being fscked is /tmp or if /tmp isn't mounted.

-n Assume a no response to all questions.

-s Ignore the actual free list and (unconditionally) construct a new one by rewriting the super-block of the file system. The file system should be unmounted while this is done, or extreme care should be taken that the system is quiescent and that it is rebooted immediately afterwards. This precaution is necessary so that the old, bad, in-core copy of the superblock will not continue to be used, or written on the file system.

The free list can be created with optimal interleaving according to the specification :

-s_c_:_s_ space free blocks _s_ blocks apart in cylinders of _c_ blocks each.

If _c_:_s_ is not given, the values used when the filesystem was created are used. If these values were not specified, then _c_=72, _s_=9 is assumed.

-S Conditionally reconstruct the free list. This option is like **-s** except that the free list is rebuilt only if there were no discrepancies discovered in the file system. It is useful for forcing free list reorganization on uncontaminated file systems. **-S** forces **-n**.

-t If **fsck** cannot obtain enough memory to keep its tables, it uses scratch files. If the **-t** option is specified, the file named in the next argument is used as the scratch file. Without the **-t** option, **fsck** prompts if it needs a scratch file. The file should not be on the

file system being checked, and if it is not a special file or did not already exist, it is removed when **fsck** completes.

-y Assume a yes response to all questions.

FILES

/etc/checklist
contains default list of file systems to check.

/etc/fsck the program

SEE ALSO

dcheck(M), icheck(M), filesys(5).

LIMITATIONS

Inode numbers for the "dot" directory, " . " and the "parent" directory, " .. " in each directory should be checked for validity.
The **-b** option of **icheck**(M) should be available.

NAME

fwtmp, wtmpfix - manipulate wtmp records

SYNOPSIS

fwtmp [-ic]
wtmpfix [files]

DESCRIPTION

Fwtmp reads from the standard input and writes to the standard output, converting binary records of the type found in wtmp to formated ASCII records. The ASCII version is useful to enable editing, via **ed**(1), bad records or general purpose maintenance of the file. **Wtmpfix** examines the standard input or named files in **wtmp** format, corrects the time/date stamps to make the entries consistent, and writes to the standard output. A - can be used in place of files to indicate the standard input. If time/date corrections are not made, **acctcon1** will fault when it encounters certain date change records.

Each time the date is set while operating in multi-user mode, a pair of date change records are written to /usr/adm/wtmp. The first record is the old date denoted by in the name field. The second record specifies the new date and is denoted by a **{** in the name field. **Wtmpfix** uses these records to synchronize all time stamps in the file.

OPTIONS

-ic Used to denote that input is in ASCII form, and output is to be written in binary form.

FILES

/usr/adm/wtmp
/usr/include/utmp.h

SEE ALSO

acctcom(M), acct(2), acct(5), utmp(5), acct(M), acctcms(M), acctcon(M), acctmerg(M), acctprc(M), acctsh(M), runacct(M).

NAME

GETTY - set the modes of a terminal

SYNOPSIS

GETTY name type delay

DESCRIPTION

GETTY is normally invoked by **INIT**(M) as the first step in allowing users to login to the system. Lines in /etc/inittab tell **INIT** to invoke **GETTY** with the proper arguments.

Name should be the name of a terminal in /dev (e.g., **ttyØ3**); type should be a single character chosen from **-, Ø, 1, 2, 3, 4, 5, 6,** or **7** (may vary locally) which selects a speed table in **GETTY,** or **!**, which tells **GETTY** to update /etc/utmp and exit; delay is relevant for dial-up ports only. It specifies the time in seconds that should elapse before the port is disconnected if the user does not respond to the **Zeus login:** request.

First, **GETTY** types the **Zeus login:** message. The **Zeus login:** message depends on the speed table being used, and may include the characters that put the GE TermiNet 3ØØ terminal into full-duplex, take the DASI terminals out of the plot mode, or put a TELETYPE(Reg.) Model 37 into full-duplex. If the terminal type as given in /etc/ttytype can have its screen cleared, the **Zeus login:** message will also contain the characters that will clear the screen. The Zeus portion of the login prompt, is obtained from the nodename field of **uname**(2).

Then the user's login name is read, a character at a time.

While reading, **GETTY** tries to adapt to the terminal, speed, and mode that is being used. If a null character is received, it is assumed to be the result of a "break" ("interrupt"). The speed is then changed based on the speed table that **GETTY** is using, and **Zeus login:** is typed again. Subsequent breaks cause a cycling through the speeds in the speed table being used.

The user's login name is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately. If the login name contains only upper-case alphabetic characters, the system is told to map any future upper-case characters into the corresponding lower-case characters.

Finally, **login**(1) is called with the user's login name as argument.

Speed sequences for the speed tables:

- B110; for 110 baud console TTY.
- 0 B300-B1200-B150-B110; normal dial-up sequence starting at B300. Useful as a default for dialup lines accessed by a variety of terminals.
- 1 B150; no sequence. Optimized for a 150-baud Teletype model 37.
- 2 B9600; no sequence. Intended for an on-line 9600-baud terminal.
- 3 B1200-B300; normal dial-up sequence starting at B1200. Useful with Bell 212 datasets where most terminals run at 1200-baud.
- 4 B300; for 300-baud terminals similar to the LA36 DECwriter.
- 5 B300-B1200; normal dial-up sequence starting at B300.
- 6 B2400; no sequence.
- 7 EXTA (19200); no sequence.

FILES

/etc/GETTY The program

SEE ALSO

login(1), tty(4), inittab(5), utmp(5), INIT(M), uname(2), ttytype(5).

BUGS

Ideally, the speed tables would be read from a file, not compiled into **GETTY**.

NAME

halt - take the system down

SYNOPSIS

halt

DESCRIPTION

Halt is a Shell file to halt the system promptly. It gives a 30-second warning, then system buffers are flushed and a **kill -1 1** is issued. After this has been done, **halt** replaces /etc/motd with /etc/motd.save if that file exists.

This command will work only if it is issued by Superuser.

FILES

/etc/halt the program

/etc/motd message of the day file

/etc/motd.save message of the day (saved) file

SEE ALSO

down(M), wall(M), kill(1).

NAME

icheck - file system storage consistency check

SYNOPSIS

icheck [**-s**] [**-b** numbers] filesystem

DESCRIPTION

Icheck examines a file system, builds a bit map of used blocks, and compares this bit map against the free list maintained on the file system. The normal output of **icheck** includes a report of

The total number of files and the numbers of regular, directory, block special and character special files.

The total number of blocks in use and the numbers of single-, double-, and triple-indirect blocks and directory blocks.

The number of free blocks.

The number of blocks missing; i.e. not in any file nor in the free list.

The **-s** option causes **icheck** to ignore the actual free list and reconstruct a new one by rewriting the super-block of the file system. The file system should be dismounted while this is done; if this is not possible (for example if the root file system has to be salvaged) care should be taken that the system is quiescent and that it is rebooted immediately afterwards so that the old, bad in-core copy of the super-block will not continue to be used. Notice also that the words in the super-block which indicate the size of the free list and of the i-list are believed. If the super-block has been curdled these words will have to be patched. The **-s** option causes the normal output reports to be suppressed.

Following the **-b** option is a list of block numbers; whenever any of the named blocks turns up in a file, a diagnostic is produced.

Icheck is faster if the raw version of the special file is used, since it reads the i-list many blocks at a time.

FILES

/etc/icheck the program

Default file systems vary with installation.

SEE ALSO

filsys(5), clri(M), dcheck(M), ncheck(M).

DIAGNOSTICS

For duplicate blocks and bad blocks (which lie outside the file system) **icheck** announces the difficulty, the i-number, and the kind of block involved. If a read error is encountered, the block number of the bad block is printed and **icheck** considers it to contain Ø.

`Bad freeblock' means that a block number outside the available space was encountered in the free list.

`n dups in free' means that n blocks were found in the free list which duplicate blocks either in some file or in the earlier part of the free list.

LIMITATIONS

Since **icheck** is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

It believes even preposterous super-blocks and consequently can get core images.

NAME

icpcntrl - start and stop ICP's and their protocols

SYNOPSIS

icpcntrl icp# [[-start [protocol:ports]] [-stop [protocol:ports]]

DESCRIPTION

Icpcntrl is used to control the running of an ICP 8/Ø2 and its protocols. If **icpcntrl** is invoked with only the icp number, information is given about the protocols that are currently active on the various ports.

When giving an protocol name, the name must be the `official' name by which the protocol is known (the name it was loaded by, using **icpload**(M)). Currently the offical protocol names are:

itty Intelligent terminal protocol

OPTIONS

-start [protocol:ports]

Start an ICP 8/Ø2 or an indicated protocol. If the start option is given with no arguments, then only the ICP is started.
When the information for the protocol and ports are given, they have the following meanings:

protocol

This is the `official' name by which the protocol is known.

ports

This field gives the ports the indicated protocol should be started on. The ports are given as a range of ports or a comma separated list. An **ICP** has 9 ports which range from Ø to 8.

-stop [protocol:ports]

Stop an ICP 8/Ø2 or an indicated protocol. If the **stop** option is given with no arguments, then the ICP is stopped (a side-effect is that all protocols are stopped ungracefully also).
The protocol and ports information has the same meanings as in the **start** options.

FILES

/dev/icp*

SEE ALSO

icpload(M), icp(4).

NAME

icpload - load and configure an ICP 8/Ø2

SYNOPSIS

icpload icp# [[**-k** kernel] [**-c** ports:config] protocols]

DESCRIPTION

Icpload configures ports and load protocols or kernels on the ICP 8/Ø2. If **icpload** is invoked with only the icp number, information is given about the current port configuration and loaded protocols.

When loading protocols, the entry point of each protocol is taken to be where it is loaded in the ICP 8/Ø2 memory. The name of the protocol object file, must be the `official' name by which the protocol is known. Only the `tail' of the path name is checked against the offical list of protocol names. So, if /usr/protocols/itty is used, only the itty portion is verified. Currently the offical protocol names are:

itty	Intelligent terminal protocol

OPTIONS

-c ports:config

Since the ICP 8/Ø2 has no way to determine what the configuration of its various ports are, this option is used to set that information into the ICP 8/Ø2. Ports are given as a range of ports or a comma separated list. An **ICP** has 9 ports which range from Ø to 8. Config is the hardware configuration of the port, and is one of the following:

A	Asynchronous	(all ports, execpt 8).
S	Synchronous	(ports 6 and 7 only).
O	Olympic	(ports 4 and 5 only).
Z	Znet	(ports 6 and 7 only).
P	Parallel	(port 8 only).

When configuring a port its adjacent port must be given the same configuration. Adjacent ports are defined as:

Ø,1
2,3
4,5
6,7

-k kernel
Load the object module kernel as the kernel on the ICP 8/Ø2. Because of various constraints the kernel object module must be segmented. The kernel entry point is where execution begins when the ICP 8/Ø2 comes out of reset. Loading a kernel also has the effect of zeroing out the port configuration and loaded protocol information.

FILES
/dev/icp*

SEE ALSO
icpcntrl(M), icp(4).

DIAGNOSTICS
All diagnostics are intended to be self-explanatory.

NAME

INIT - process control initialization

SYNOPSIS

INIT [state]

DESCRIPTION

INIT is invoked inside ZEUS as the last step in the boot procedure. It is process number one, and is the ancestor of every other process in the system. As such, it can be used to control the process structure of the system. This is done by the super-user invoking **INIT** with an argument.

INIT has 9 states, 1 through 9; it is invoked by the system in state 1 during manual bootstrapping, and in state 2 during auto-booting. It performs the same functions on entering each state. When a state is entered, **INIT** reads the file /etc/inittab which defines the transition into each state. Lines in this file have the format:

state:id:flags:command

where

state is an integer from 1 to 9,

id is a two character identifier (i.e. co for console, ØØ for tty ØØ, etc) standing for some process that might be active in that state,

flags are the characters t, k, c, or o,

command is the name of an executable file with its optional arguments.

All lines in which the state field matches **INIT**'s current state are recognized. A process whose two character id matches one on a recognized line may be terminated (signal 15), killed (signal 9), or both by including the flags t and **k** in the order desired. The signal is sent to all processes in the process group associated with the id. The command field is saved for later execution (see below).

After reading /etc/inittab and signaling running processes as required, but before invoking any processes under the new state, **INIT** invokes /etc/rc with three arguments. /etc/rc performs housekeeping such as removing temporary files, mounting file systems, and starting daemons. The three arguments that it receives are the current state, the number of times this state has been entered previously, and

the prior state.

When /etc/rc has finished executing, **INIT** invokes all commands waiting to be executed. (A command is considered to be waiting to be executed if there is no process currently running that has the same id as the command.) The flag c (continuous) requires the command to be continuously reinvoked whenever the process with that id dies. The flag o (off) causes the command to be ignored. This is useful for turning lines off without extensive editing. Otherwise, the command is invoked a maximum of one time in the current state.

INIT invokes the command field read from /etc/inittab by opening / for reading and writing on file descriptors Ø, 1, and 2, resetting all signals to system default, setting up a new process group (**setpgrp**(2)), and **exec**ing:

/bin/sh -c exec command

EXAMPLES

The inittab line

1:co:c:/bin/csh </dev/console > /dev/console 2>&1

creates a single user environment when **INIT 1** is invoked by the super-user. It states that in state 1 (by convention single-user mode), a process with the arbitrary unique identifier co should be created. The program invoked for this process should be the c-shell, taking its input from the console and directing both its default output and error output back to the console. When this process exits (i.e. logout) it should be reinvoked.

Invoking **INIT 2** when inittab contains the entry

2:co:c:/etc/getty console 2

attaches a **login** process to the console in multi-user state (state 2) and sets its baud rate to 9600. the entry

2:Ø5:c:/etc/getty tty Ø5 2

sets up line /dev/tty Ø5 for use by 9600 baud terminals, while

1:Ø5:k:/etc/getty !

will ensure that it is not active on return (or entry) to single-user mode.

DIAGNOSTICS

When **INIT** can do nothing else because of a missing /etc/inittab or when it has no children left, it will try to execute a shell on /dev/console. When the problem has been fixed, it is necessary to change states, and terminate the shell.

LIMITATIONS

INIT does not complain if the state-id pairs in /etc/inittab are not unique. For any given pair, the last one in the file is valid.

FILES

/etc/INIT The program
/etc/inittab
/etc/rc
/bin/sh
/dev/console

SEE ALSO

login(1), sh(1), exec(2), setpgrp(2), inittab(5), getty(M).

NAME

install - install commands

SYNOPSIS

install
[**-c** dira]
[**-f** dirb]
[**-i**]
[**-n** dirc]
[**-o**]
[**-s**]
file
[dirx ...]

DESCRIPTION

Install is most commonly used in "makefiles" (see **make**(1)) to install a file (updated target file) in a specific place within a file system. Each file is installed by copying it into the appropriate directory, retaining the mode and owner of the original command. The program prints messages telling exactly what files it is replacing or creating and where they are going.

If no options or directories (dirx ...) are given, **install** will search (using **find**(1)) a set of default directories (/bin, /usr/bin, /etc, /lib, and /usr/lib , in that order) for a file with the same name as file. When the first occurrence is found, **install** issues a message saying it is overwriting that file with file, and proceeds to do so. If the file is not found, the program states this and exits without further action.

If one or more directories (dirx ...) are specified after file, those directories will be searched before the directories specified in the default list.

OPTIONS

-c dira — Installs a new command in the directory specified in dira. Looks for file in dira and installs it there if not found. If found, **install** issues a message saying the file already exists, and exits without overwriting it. Can be used alone or with the **-s** option.

-f dirb — Forces file to be installed in given directory, whether or not one already exists. If it does not already exist, the mode and owner of the new file will be set to **751** and **bin**, respectively. If the file already exists, the mode and owner will be that of the already existing file. Can be used alone or

with the **-o** or **-s** options.

-i Ignores default directory list, searching only through the given directories (*dirx ...*). Can be used alone or with other options, except **-c** and **-f**.

-n *dirc* If *file* is not found in any of the searched directories, it is put in the directory specified in *dirc*. The mode and owner of the new file will be set to **755** and **bin**, respectively. Can be used alone or with other options, except **-c** and **-f**.

-o If *file* is found, this saves the "found" file by copying it to **OLD***file* in the directory in which it was found. Can be used alone or with other options, except **-c**.

-s Suppresses printing of messages other than error messages. Can be used alone or with any other options.

FILES

/etc/install the program

WARNING

Install is a Bourne shell script. If your login shell is the C shell, **install** must be invoked as a shell script from *makefile* as follows:

```
/bin/sh /etc/install file
```

This can be done easily as a **make** macro definition. For example,

```
INS=/bin/sh /etc/install
```

and to execute **install**

```
$(INS) file
```

This is not necessary if your login shell is the *Bourne Shell*, or if you use **install** from the command level of either shell, or if it is invoked from command scripts.

NAME

labelit - label file systems

SYNOPSIS

labelit special [fsname volume [**-n**]]

DESCRIPTION

Labelit can be used to provide initial labels for unmounted disk or tape file systems. With the optional arguments omitted, **labelit** prints current label values. The **-n** option provides for initial labeling of new tapes only (this destroys previous contents). The fsname argument represents the mounted name (e.g. **root**, **usr**, etc.) of the filsystem being copied.

The special argument should be the physical disk section or tape (e.g. /dev/rzd1, /dev/rmtØ, etc.)

Fsname and volname are recorded in the last 12 characters of the superblock (char fsname[6], volname[6]).

FILES

/etc/log/filesave a record of file systems/volumes copied

SEE ALSO

filesys(5).

LIMITATIONS

Only device names beginning /dev/rmt are treated as tapes.

NAME

link, unlink - exercise link and unlink system calls

SYNOPSIS

link file1 file2
unlink file

DESCRIPTION

Link and **unlink** perform their respective system calls on their arguments, abandoning all error checking. These commands can only be executed by the super-user.

FILES

/etc/link the program

/etc/unlink the program

SEE ALSO

rm(1), link(2), unlink(2).

LIMITATIONS

The programs cannot cross filesystem boundaries.

NAME

lp, text - service line printer spooler print requests

SYNOPSIS

/usr/lib/lp [options]
/usr/lib/text [options]

DESCRIPTION

These programs are used by the spooling system to do the actual copy of data from a print request to a printer device. Lp is used to print requests on line printers. Text is used to print requests on text quality printers.

Each backend is started by dqueuer(M). The way backends are started is controlled by the configuration file (see the System Adminstrators Manual, Section 7). A backend program inherits three open file descriptors from the dqueuer: RFFD (read file), SFFD (device) and STATUSFD (status log). Actual definitions of these variables can be found in /usr/include/spool.h. The lp and text backends copy data from file RFFD to file SFFD with minimal processing of input. Statuses are written to file STATUSFD.

In addition to printing, the backends perform functions required by the specific device being used. For instance, lp filters backspaces and text can perform stty(1) calls before printing. They also provide banner pages with appropriate information.

When invoked, backends are passed a parameter list generated by dqueuer(M). This list contains options generated by dqueuer(M) and those options that may have been specified in the configuration file. The options that dqueuer generates and that both backends recognize are:

-B Indicates that the file the dqueuer was to print could not be found. In this case, RFFD is not open and should not be used. This option is used to generate banner pages that say the file could not be found, so it won't appear that the file disappeared.

-c n
Indicates that the file must be printed multiple times. The backend should create n copies of the output.

-d dest
Used in a destination portion of the banner page. Dest is a string of no other importance to the backend.

-f filename
Filename, printed on the banner page.

-F from
Indicates a 'from' string, for use on the banner page.

-s spool time
A string in ctime(3) format, indicating when the file was spooled to be printed. This value is printed on banner page.

-t title
A string used as a title on banner pages.

The backend also looks in a file called '/etc/sitefile' to find a string to be printed near the page breaks on the banner page. It uses the string of characters on the first line of this file, up to the first white space. If this file does not exist, an appropriate default is used.

The text backend accepts one additional option, that can only specified through the configuration file. Its description follows:

-T [stty parameters]
Set terminal parameters. When a tty device is used as a printer, it is disabled in the /etc/inittab file. When this is done, the device is closed except when being used to print upon. This causes ZEUS to reset all terminal characteristics. The -T option is used to set these characteristics so that printing can occur. The quoted string is passed, unchanged except for removal of the quotes, to stty(1). Thus, in the configuration file the line

```
D1,R,/dev/tty0,/usr/lib/text -T "1200"
```

would cause device 1 on the most recent queue to use stty to set /dev/tty0 to 1200 BAUD before being used.

SEE ALSO
dqueuer(M)

NAME

makekey - generate encryption key

SYNOPSIS

/usr/lib/makekey

DESCRIPTION

Makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (i.e. to require a substantial fraction of a second).

The first eight input bytes (the input key) can be arbitrary ASCII characters. The last two (the salt) are best chosen from the set of digits, upper- and lower-case letters, and `.' and `/'. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the output key.

The transformation performed is essentially the following: the salt is used to select one of 4096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but modified in 4096 different ways. Using the input key as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 useful key bits in the result.

Makekey is intended for programs that perform encryption (e.g. **ed** and **crypt**(1)). Usually its input and output will be pipes.

SEE ALSO

crypt(1), ed(1).

LIMITATIONS

Although it can be invoked from the prompt, **makekey** is used mainly within the body of a program.

NAME

makenewfs - construct and restore file system

SYNOPSIS

makenewfs

DESCRIPTION

Makenewfs is a C shell script which constructs the default file system layout.

Makenewfs is intended to be run during the initial boot from tape. It should be run in single user mode after the root and /usr are restored from tape. **Makenewfs** makes the file systems /tmp and /z. It then labels all the file systems. Finally, it copies the secondary bootstrapper, boot, from the root file system to the first filesystem on the disk.

The program may be run to restore the disk after severe damage to it has occurred. It is important to note that if file systems /z or /tmp, are already on the disk THEY WILL BE COMPLETELY DESTROYED. Therefore this program should be run only on initial boot or when the file systems have been irreparably damaged.

In order to safeguard against malicious or unintentional usage, the program is owned by zeus with mode 000. It is necessary to enter "chmod 700 /etc/makenewfs" before executing this program. The program will revert to mode 000 and owner zeus upon termination.

SEE ALSO

mkfs(M), restor(M).
The System Administrator Manual

LIMITATIONS

This program applies only to the default disk layout, and standard Zilog distribution tape as specified in the Zeus System Administrators Manual. It would have to be changed for non-standard disk layouts or other dump tapes.

NAME

makewhatis - remake the data base for the whatis and apropos commands

SYNOPSIS

makewhatis

DESCRIPTION

Makewhatis remakes the data base for the whatis and apropos commands from the manual sections in /usr/man/*. It uses **getNAME**(1) to extract relevant text from each manual entry and then rearranges and sorts this output. Manuals sections may be in either the old (version 6) or version 7 manual formats.

FILES

/usr/lib/whatis	constructed data base
/etc/makewhatis	the program

SEE ALSO

getNAME(1).

NAME

mfs - mount all file systems
umfs - unmount all file systems

SYNOPSIS

mfs
umfs

DESCRIPTION

Mfs is a shell script used to mount all file systems in the current configuration. It is intended that mfs be run after **fsck**(M) has run successfully. Note that if the system's disk configuration is changed from the default, /etc/mfs must be changed accordingly. **Umfs** is a shell script used to un-mount all file systems in the current configuration.

FILES

/etc/mtab

/dev/zd*

/etc/mfs the program

/etc/umfs the program

SEE ALSO

fsck(M), mount(M), mtab(5).

LIMITATIONS

If **mfs** is not run before bringing the system up multi-user, /usr will not mount correctly. This is because the system creates several files in /usr after going multi-user and keeps them open as long as the system is running. So in order to mount /usr, the system must be brought back to single user mode.

Mfs/umfs will not mount/unmount /usr if **cron**(M) **update**(M) or **dequeuer**(M) are running, or if accounting (**acct**(M)) is on.

NAME

mkfs - construct a file system

SYNOPSIS

mkfs special proto
mkfs special blocks interleave sectors

DESCRIPTION

Mkfs constructs a file system by writing on the special file special according to the directions found in the remainder of the command line. If the second argument is proto **mkfs** constructs a filesystem according to the prototype file proto. The prototype file contains tokens separated by spaces or new lines.

The first token is the name of a file to be copied onto block zero as the bootstrap program. The second token is a number specifying the size of the created file system. Typically it will be the number of blocks on the device, perhaps diminished by space for swapping.

The next token is the number of i-nodes in the i-list. The next set of tokens comprise the specification for the root file. File specifications consist of tokens giving the mode, the user-id, the group id, and the initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a 6 character string. The first character specifies the type of the file. (The characters **-bcd** specify regular, block special, character special and directory files respectively.) The second character of the type is either **u** or **-** to specify set-user-id mode or not. The third is **g** or **-** for the set-group-id mode. The rest of the mode is a three digit octal number giving the owner, group, and other read, write, execute permissions, see **chmod**(1).

Two decimal number tokens come after the mode; they specify the user and group ID's of the owner of the file.

If the file is a regular file, the next token is a pathname whence the contents and size are copied.

If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers.

If the file is a directory, **mkfs** makes the entries **.** and **..** and then reads a list of names and (recursively) file specifications for the entries in the directory. The scan is terminated with the token **$**.

If the second argument is blocks, **mkfs** builds a file system with a single empty directory on it. The size of the file system is the value of blocks interpreted as a decimal number. The number of i-nodes is calculated as a function of the filesystem size.

A sample prototype specification follows:

```
/usr/diag/uboot
4872 55
d--777 3 1
usr  d--777 3 1
     sh   ---755 3 1 /bin/sh
     ken  d--755 6 1
          $
     bØ   b--644 3 1 Ø Ø
     cØ   c--644 3 1 Ø Ø
     $
$
```

FILES

/etc/mkfs the program

SEE ALSO

dir(5), filsys(5).

LIMITATIONS

There should be some way to specify links.

NAME

mkmt - make special files for magnetic tape devices

SYNOPSIS

mkmt [**mtØ**] [**mt1**] [**mt2**] [**mt3**]

DESCRIPTION

Mkmt makes appropriate special files in /dev for the various kinds of tape access associated with a particular nine track magnetic tape unit. It can be used to simplify the task of creating all the tape special files. For example, **mkmt mtØ** would make the following special files:

/dev/mtØ /dev/rmtØ /dev/nmtØ /dev/fmtØ /dev/smtØ
/dev/frmtØ /dev/nrmtØ /dev/srmtØ /dev/fnmtØ /dev/snmtØ
/dev/fsmtØ /dev/fnrmtØ /dev/snrmtØ /dev/fsrmtØ /dev/fsnmtØ
/dev/fsnrmtØ

FILES

/etc/mkmt the program

SEE ALSO

mknod(M), mt(4).

NAME

mknod - build special file

SYNOPSIS

mknod name [**c**] [**b**] major minor
mknod name **p**

DESCRIPTION

Mknod makes a directory entry and corresponding i-node for a special file.

Mknod can also be used to create fifo's (a.k.a named pipes) (second case in **SYNOPSIS** above). This use of **Mknod** can be used by any user. The first case can be used only by members of the 'system' group. It is used to create special device files.

The first argument is the name of the entry. In the first case, the second is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the major device type and the minor device (e.g. unit, drive, or line number), which may be either decimal or octal. A leading Ø for the major and minor device numbers mean that they are in octal.

The assignment of major device numbers depends on the position of the driver in dispatch tables in the kernel. The major device numbers for current drivers is as follows:

Device	Character Dev. Major Number	Block Dev. Major Number
ZD	Ø	Ø
CT	1	1
SMD	2	8
MT	3	9
MD	4	1Ø
ERR	5	--
MEM	6	--
TTY	7	--
SIO	8	--
LP	9, 1Ø	--
PTC (unet)	11	--
PTS (unet)	12	--
UP (unet)	13	--
UU (unet)	14	--
UD (unet)	15	--
U1 (user defined dev)	16	2
U2 (user defined dev)	17	3
U3 (user defined dev)	18	4
U4 (user defined dev)	19	5
U5 (user defined dev)	2Ø	6

U6 (user defined dev) 21 7

The minor device number is device dependent. For disks, the minor device number is the number of the file system. The first digit (decimal) corresponds to the drive number and the second digit (decimal) is the order of the file system on the disk. For ttys, the minor device number is the number of the port. For other devices the numbers represent options passed to the drivers (eg. no-rewind for tapes).

FILES

/etc/mknod the program

SEE ALSO

mknod(1), mknod(2).

NAME

mktape - make special files for cartridge tape devices

SYNOPSIS

mktape [**Ø**] [**1**] [**2**] [**3**]

DESCRIPTION

Mktape makes appropriate special files in /dev for the various kinds of tape access associated with a particular unit. It can be used to simplify the task of creating all the tape special files since a total of 2Ø are required per tape unit. For example the command:

mktape 1

would make the following special files:

- non-raw
/dev/ct1 /dev/ct1a /dev/ct1b /dev/ct1c /dev/ct1d

- raw
/dev/rct1 /dev/rct1a /dev/rct1b /dev/rct1c /dev/rct1d

- no rewind
/dev/nct1 /dev/nct1a /dev/nct1b /dev/nct1c /dev/nct1d

- no rewind raw
/dev/nrct1 /dev/nrct1a /dev/nrct1b /dev/nrct1c /dev/nrct1d

FILES

/etc/mktape the program

SEE ALSO

mknod(M).

NAME

mount, umount - mount and dismount file system

SYNOPSIS

mount [special directory [**-r**]]

umount special

DESCRIPTION

Mount announces to the system that a removable file system is present on the device special. The directory must exist already; it becomes the name of the root of the newly mounted file system.

These commands maintain a table of mounted devices. If invoked with no arguments, **mount** prints the table.

The optional last argument indicates that the file is to be mounted read-only. Physically write-protected and magnetic tape file systems must be mounted in this way or errors will occur when access times are updated, whether or not any explicit write is attempted.

Umount announces to the system that the removable file system previously mounted on device special is to be removed.

FILES

/etc/mnttab	mount table
/etc/mount	the program
/etc/umount	the program

SEE ALSO

mount(2), mnttab(5).

DIAGNOSTICS

Mount issues a warning if the file system to be mounted is currently mounted under another name.

Umount complains if the special file is not mounted or if it is busy. The file system is busy if it contains an open file or some user's working directory.

LIMITATIONS

Some degree of validation is done on the file system, however it is generally unwise to mount garbage file systems.

NAME

mvdir - move a directory

SYNOPSIS

mvdir dirname name

DESCRIPTION

Mvdir renames directories within a file system. Dirname must be a directory; name must not exist. Neither name can be a sub-set of the other (**/x/y** cannot be moved to **/x/y/z**, nor vice versa).

Only the super-user can use **mvdir**.

FILES

/etc/mvdir the program

SEE ALSO

mkdir(1).

LIMITATIONS

Directories cannot be moved across filesystems

NAME

ncheck - generate names from i-numbers

SYNOPSIS

ncheck [**-i** numbers] [**-as**] [filesystem]

DESCRIPTION

Ncheck generates a pathname vs. i-number list of all files on the file system specified. Names of directory files are followed by `/.'. The **-i** option reduces the report to only those files whose i-numbers follow. The **-a** option allows printing of the names `.' and `..', which are ordinarily suppressed. The **-s** option reduces the report to special files and files with set-user-ID mode; it is intended to discover concealed violations of security policy.

The report is in no special order, so sorting may be desirable.

SEE ALSO

sort(1), dcheck(M), icheck(M).

DIAGNOSTICS

When the filesystem structure is improper, `??' denotes the `parent' of a parentless file and a pathname beginning with `...' denotes a loop.

NAME

pstat - print system facts

SYNOPSIS

pstat [**-aixptuf**] [suboptions] [file

DESCRIPTION

Pstat interprets the contents of certain system tables. If file is given, the tables are sought there, otherwise in /dev/mem and /dev/kmem. (System core images not currently supported). The required namelist is taken from /zeus. Options are

-a Under **-p**, describe all process slots rather than just active ones.

-i Print the inode table with the these headings:

LOC The core location of this table entry.
FLAGS Miscellaneous state variables encoded thus:
- L locked
- U update time (**filsys**(5)) must be corrected
- A access time must be corrected
- M file system is mounted here
- W wanted by another process (L flag is on)
- T contains a text file
- C changed time must be corrected

CNT Number of open file table entries for this inode.
DEV Major and minor device number of file system in which this inode resides.
INO I-number within the device.
MODE Mode bits, see **chmod**(2).
NLK Number of links to this inode.
UID User ID of owner.
SIZ/DEV Number of bytes in an ordinary file, or major and minor device of special file.

-x Print the text table with these headings:

LOC The core location of this table entry.
FLAGS Miscellaneous state variables encoded thus:
- T **ptrace**(2) in effect
- W text not yet written on swap device
- L loading in progress
- K locked
- w wanted (L flag is on)

DADDR Disk address in swap, measured in multiples of 512 bytes.

CADDR Core address, measured in multiples of 256 bytes.

SIZE Size of text segment, measured in multiples of 256 bytes.

IPTR Core location of corresponding inode.

CNT Number of processes using this text segment.

CCNT Number of processes in core using this text segment.

-p Print process table for active processes with these headings:

LOC The core location of this table entry.
S Run state encoded thus:
- Ø no process
- 1 waiting for some event
- 3 runnable
- 4 being created
- 5 being terminated
- 6 stopped under trace

F Miscellaneous state variables, or-ed together:
- ØØ1 loaded
- ØØ2 the scheduler process
- ØØ4 locked
- ØØ8 swapped out
- Ø1Ø traced
- Ø2Ø used in tracing
- Ø4Ø locked in by **lock**(2).

PR Scheduling priority, see **nice**(2).
SG Signals received (signals 1-16 coded in bits Ø-15),
UID Real user ID.
TIM Time resident in seconds; times over 127 coded as 127.
CPU Weighted integral of CPU time, for scheduler.
NI Nice level, see **nice**(2).
PGRP Process number of root of process group (the opener of the controlling terminal).
PID The process ID number.
PPID The process ID of parent process.
ADDR If in core, the physical address of the `u-area' of the process measured in multiples of 256 bytes. If swapped out, the position in the swap area measured in multiples of 512 bytes.
SZE Size of process image in multiples of 256 bytes.
WCHN Wait channel number of a waiting process.
LINK Link pointer in list of runnable processes.
TEXTP If text is pure, pointer to location of text table entry.
CLKT Countdown for **alarm**(2) measured in seconds.

-t Print table for terminals (sio ports) with these headings:

RAW Number of characters in raw input queue.
CAN Number of characters in canonicalized input queue.
OUT Number of characters in output queue.
IMODE Input mode flags
OMODE Output mode flags
CMODE Control mode flags
LDMODE
Line discipline mode flags
IOADDR
IO address.
CTCADDR
CTC address.
DEL Number of delimiters (newlines) in canonicalized input queue.
COL Calculated column position of terminal.
STATE Miscellaneous state variables encoded thus:
W waiting for open to complete
O open
S has special (output) start routine
C carrier is on
B busy doing output
A process is awaiting output
X open for exclusive use
H hangup on close
PGRP Process group for which this is controlling terminal.

-u print information about a user process; the next argument is its address as given by **ps**(1). The process must be in main memory, or the file used can be a user core image and the address Ø.

-f Print the open file table with these headings:

LOC The core location of this table entry.
FLG Miscellaneous state variables encoded thus:
R open for reading
W open for writing
P pipe
CNT Number of processes that know this open file.
IPTR The location of the inode table entry for this file.
OFFS The file offset, see **lseek**(2).

FILES

/zeus namelist
/dev/mem default source of user structure
/dev/kmem default source of kernel data tables

SEE ALSO

ps(1), stat(2), filsys(5)

NAME

quot - summarize file system ownership

SYNOPSIS

quot [**-ncf**] filesystem

DESCRIPTION

Quot prints the number of blocks in the named filesystem currently owned by each user. If no filesystem is named, no default name is assumed.

OPTIONS

-n To produce a list of all files and their owners, use the pipeline:

ncheck filesystem | sort +Øn |

Note that the -n option is meaningless without this command string.

-c Print three columns giving file size in blocks, number of files of that size, and cumulative total of blocks in that size or smaller file.

-f Print count of number of files as well as space owned by each user.

FILES

/etc/passwd to get user names

SEE ALSO

ls(1), du(1).

LIMITATIONS

Holes in files are counted as if they actually occupied space.

Quot will ignore any uids greater than 1ØØØ.

NAME

rc - "read command" startup control script
rc_csh - "read command" Cshell multi-user startup script

SYNOPSIS

rc
rc_csh

DESCRIPTION

Rc is invoked by **INIT** when the system changes states (e.g. from multi-user mode to single-user mode, and vise versa). **Rc_csh** in invoked by **rc** when the system goes to multi-user mode. **Rc_csh** is a shell script, and as such can easily be altered to suit the particular needs of a system. Generally, commands in **rc_csh** perform miscellaneous functions such as checking the consistency of the disk, mounting file systems, etc.

INIT invokes **rc** with three arguments: target state, the number of times the state has been entered and the last state. This allows **rc** to invoke other programs selectively depending on the specified state transition (e.g. **rc_csh** should be invoked when the system goes multi-user). If the system is booted to a multi-user state automatically, the last state is "Ø".

FILES

/etc/rc the program

/etc/rc_csh the program

SEE ALSO

INIT(M).

THE PROGRAMS

The **rc** file:

```
: "@[$]rc 2.11  Ø3/28/83 Ø9:21:Ø3 - Zilog Inc"
: "This is the rc control script. Init invokes this script
: "and then for the other important states where real work
: "occurs we use csh(1) scripts"

TZ=PST8PDT
export TZ
trap "INIT 1" 2

: "In all cases i/o is re-directed to the console"

: "zero out the mount table if this is a bootup"
if [ $3 = "Ø" ]
     then /bin/cat /dev/null > /etc/mnttab
          /etc/devnm / | grep root | /etc/setmnt
```

```
(
case ${1-2} in
1)
     echo
     uname -sn
     echo Single-User Mode
     if [ $3 = "2" ]
          then /etc/killall
     fi
     /etc/umfs > /dev/null 2>&1
     ;;
2)
     if [ $1 != $3 ]
          then exec rc_csh
     fi
     ;;

*)
     echo "Unknown state for init:" $*
     ;;
esac
)
```

The **rc_csh** file:

```
# "@[$]rc_csh  2.18  03/28/83 09:21:04 - Zilog Inc"
     echo
     uname -sn
     echo "Multi-user Startup"
     echo

# Check for filesystem consistency
     /etc/fsck -y /dev/root
     /etc/fsck -y /dev/rusr /dev/rtmp
     /etc/fsck -y -t /scratch /dev/rz

# Set the date
     /etc/datem

# Now mount the filesystems
     /etc/mfs

# Give the user a chance to interrupt, then go multi-user
     /bin/echo -n 'The date the system knows is ' ; date
     echo 'Going multi-user in 30 seconds'

# configure system for correct modem/tty configuration
     /etc/ttyconfig -t 0-7

# Remove the remote lock file
```

```
        cat /dev/null > /usr/spool/uucp/LCK..tmp
        rm -f /usr/spool/uucp/LCK..*

    # Save the su and cron log files
        if ( -e /usr/adm/sulog ) then
             mv /usr/adm/sulog /usr/adm/osulog
        endif
        if ( -e /usr/lib/cronlog ) then
             mv /usr/lib/cronlog /usr/lib/ocronlog
        endif

    # zero out utmp file
        cat /dev/null > /etc/utmp

        exit 0
```

NAME

reservrc - reserv install and remove utility

SYNOPSIS

reservrc [-i][-r]

DESCRIPTION

Reservrc installs or removes the **reserv**(1) utility for the system

Reserv runs under the Zeus operating system as a utility. The actions of reserv depend on what keys the user entered on the command line. There is no other user interaction after the program has been invoked. **Reserve** lets the user know what has happened both by messages sent to stdout and by values left in the 'status' shell variable.

The primary function of reserv is to reserve the tape drive for a user. With programs such as **tar**(1) or **dump**(M), the device must first be opened to use a tape. There are 20 virtual tape devices for a given drive. For example, tar could open /dev/nrct0 (no rewind, raw, cartridge tape #0) or /dev/ct0a (cooked cartridge tape #0, track a).

Before tar can open any of these, **reserv** must be run to reserve drive #0.

The unreserved state of the device is permissions set to 000 and owner set to 'bin' for each virtual device. This prohibits anyone from opening the device. **Reserv** reserves the device by changing (for all virtual devices) permissions to 0600 and owner to the person running **reserv** at the time. This allows only the owner to open the device. A **fork**(2) is performed, the parent is killed and the child continues to execute. It is the child who does all of the queueing and abandoning. Thus for every person on the queue and the person who has possession of the tape drive, there is a process running.

KEYS

-i Installs reserv into the system. The line "/etc/reservrc -i" should be added to the /etc/rc script.

-r Removes reserv from the system. The line "/etc/reservrc -i" should be removed from the /etc/rc script. All tape drives must be free before this is executed.

FILES

/etc/reservrc

/etc/rc

/etc/rc_csh

/usr/spool/reserv/*

SEE ALSO

rc(M), reserv(1).

NAME

restor, srestor - incremental file system restore

SYNOPSIS

restor key [argument ...]

DESCRIPTION

Restor is used to read magtapes dumped with the **dump** command. **Srestor** is the segmented version. It allows the 9 track tape drive to run at a higher rate if reading and writing. The default source device is /dev/dumpdev. If /dev/dumpdev does not exist, then /dev/rctØ is used. The key specifies what is to be done.

KEYS

Key is one of the characters **rRtxl** optionally combined with **f, w, d** and **v.**

f Use the first argument as the name of the tape instead of the /dev/rctØ.

r or **R** The tape is read and loaded into the file system specified in argument. This should not be done lightly (see below). If the key is R **restor** asks which tape of a multi volume set to start on. This allows restor to be interrupted and then restarted (an **icheck -s** must be done before restart).

x Each file on the tape named by an argument is extracted. The file name has all `mount' prefixes removed; for example, /usr/bin/lpr is named /bin/lpr on the tape. The file extracted is placed in a file with the same name. All necessary directories are automatically created with reference to the present working directory. In order to keep the amount of tape read to a minimum, the following procedure is recommended:

Mount volume 1 of the set of dump tapes.

Type the **restor** command.

Restor will announce whether or not it found the files, and rewind the tape.

It then prompts you to "mount desired volume that contains file of the last volume if you aren't sure". It then asks for the volume number that you just mounted. Type the number of the volume you choose. On a multivolume dump the recommended procedure is to mount the last through the first volume in that order. **Restor** checks to see if any of the files requested are on the mounted tape (or a later tape, thus the reverse order)

dump or the number of files being restored is large, respond to the query with `l' and **restor** will read the tapes in sequential order.

If you have a hierarchy to restore you can use the '-l' option to produce the list of names and a tape.

w Causes **restor** to prompt the user with the filename to be restored immediatly before it is extracted. It waits for a confirmation. If a word beginning with 'y' is given, the file is extracted into the original filename; if a word beginning with 'y' followed by another word, this second word is taken as the filename to which the file will be extracted. Any other response is taken as negative, that file won't be extracted.

t Print the filesystem name, user comments made at dump time, the date the tape was written and the date the filesystem was dumped from.

l Does everything the '-t' option does plus lists the names of the files that are on the dump.

v Only meaningful when used with the '-l' option. This gives a verbose listing of the filenames including the owner group, permissions, and date of last modification.

d Take the next argument as the position (starting with 1) of the file on the tape with respect to the current position of the tape. This enables the user to restor from a dump even though it isn't the first file on the tape. For example the command 'restor -dx 3 filename' will cause restor to space to the third file on the tape (assuming the tape was rewound prior to the command) to do the extracting.

The **r** option should only be used to restore a complete dump tape onto a clear file system or to restore an incremental dump tape onto this. Thus

```
/etc/mkfs /dev/zdØ 7ØØØ
restor r /dev/zdØ
```

is a typical sequence to restore a complete dump. Another **restor** can be done to get an incremental dump in on top of this.

A **dump** followed by a **mkfs** and a **restor** is used to change the size of a file system.

FILES

/tmp/rst*

/etc/restor the program

SEE ALSO

dump(M), mkfs(M).

DIAGNOSTICS

There are various diagnostics involved with reading the tape and writing the disk. There are also diagnostics if the i-list or the free list of the file system is not large enough to hold the dump.

LIMITATIONS

There is redundant information on the tape that could be used in case of tape reading problems. Unfortunately, **restor** doesn't use it.

NAME

rmuser - remove a user from the system

SYNOPSIS

rmuser name

DESCRIPTION

This shell procedure aids the system administrator in removing a user to the system. It handles the updating of /etc/passwd and /etc/group and produces a list of all the files in the system owned by the departing user for screening and eventual disposition. Only the user's mailbox file is deleted, all other files are untouched. The procedure is interactive and prompts for all needed information.

FILES

/etc/group group file, name is removed from all groups

/etc/passwd password file, name's password is changed

files.name file containing list of all files owned by name /etc/rmuser the program

SEE ALSO

passwd(1).

DIAGNOSTICS

If name is not found in the procedure exits with a complaint.

LIMITATIONS

If the departing user has access to other user's passwords (e.g. bin, root) these should be changed also.

Files owned by name in unmounted file systems are not included in files.name for obvious reasons.

No check is made on the success of changing the user's password.

NAME

runacct - run daily accounting

SYNOPSIS

runacct [mmdd [state]]

DESCRIPTION

Runacct is the main daily accounting shell procedure. It is normally initiated via **cron**(M). **Runacct** processes connect, fee, disk, and process accounting files. It also prepares summary files for **prdaily** or billing purposes.

Runacct takes care not to damage active accounting files or summary files in the event of errors. It records its progress by writing descriptive diagnostic messages into active. When an error is detected, a message is written to /dev/console, mail (see **mail**(1)) is sent to **zeus** and **adm**, and **runacct** terminates. **Runacct** uses a series of lock files to protect against re-invocation. The files lock and lock1 are used to prevent simultaneous invocation, and lastdate is used to prevent more than one invocation per day.

Runacct breaks its processing into separate, restartable states using statefile to remember the last state completed. It accomplishes this by writing the state name into **statefile**.

Runacct then looks in statefile to see what it has done and to determine what to process next. States are executed in the following order:

SETUP Move active accounting files into working files.

WTMPFIX Verify integrity of **wtmp** file, correcting date changes if necessary.

CONNECT1 Produce connect session records in ctmp.h format.

CONNECT2 Convert ctmp.h records into tacct.h format.

PROCESS Convert process accounting records into tacct.h format.

MERGE Merge the connect and process accounting records.

FEES Convert output of **chargefee** into tacct.h format and merge with connect and process accounting records.

DISK Merge disk accounting records with connect,

process, and fee accounting records.

MERGETACCT Merge the daily total accounting records in daytacct with the summary total accounting records in /usr/adm/acct/sum/tacct.

CMS Produce command summaries.

USEREXIT Any installation-dependent accounting programs can be included here.

CLEANUP Cleanup temporary files and exit.

To restart **runacct** after a failure, first check the **active** file for diagnostics, then fix up any corrupted data files such as pacct or wtmp. The **lock** files and **lastdate** file must be removed before **runacct** can be restarted. The argument mmdd is necessary if **runacct** is being restarted, and specifies the month and day for which **runacct** will rerun the accounting. Entry point for processing is based on the contents of statefile; to override this, include the desired state on the command line to designate where processing should begin.

EXAMPLES

To start **runacct**.

```
nohup runacct 2> /usr/adm/acct/nite/fd2log &
```

To restart **runacct**.

```
nohup runacct 0601 2>> /usr/adm/acct/nite/fd2log &
```

To restart **runacct** at a specific state.

```
nohup runacct 0601 MERGE 2>> /usr/adm/acct/nite/fd2log
&
```

FILES

/usr/lib/acct/runacct
/usr/adm/wtmp
/usr/adm/pacct[1-9]
/usr/src/cmd/acct/tacct.h
/usr/src/cmd/acct/ctmp.h
/usr/adm/acct/nite/active
/usr/adm/acct/nite/daytacct
/usr/adm/acct/nite/lock
/usr/adm/acct/nite/lock1
/usr/adm/acct/nite/lastdate
/usr/adm/acct/nite/statefile
/usr/adm/acct/nite/ptacct[1-9]. mmdd

SEE ALSO

acctcom(M), acct(2), acct(5), utmp(5), acct(M), acctcms(M), acctcon(M), acctmerg(M), acctprc(M), acctsh(M), cron(M),

fwtmp(M).
The PWB/ZEUS Accounting System in the ZEUS Utilities Manual

DIAGNOSTICS

Self explanatory.

LIMITATIONS

Normally it is not a good idea to restart **runacct** in the **SETUP** state. Run **SETUP** manually and restart via:

runacct mmdd **WTMPFIX**

If **runacct** failed in the **PROCESS** state, remove the last **ptacct** file because it will not be complete.

NAME

setlp - set line printer parameters

SYNOPSIS

setlp [**-lwsd**]

DESCRIPTION

Setlp displays and changes the line printer parameters: number of columns, lines per page, and indent (number of blanks before first column). If no options are given, **setlp** displays the current parameter settings.

These are the options.

-l number
The number of lines is set to the decimal number given.

-w number
The number of columns is set to the decimal number given.

-s number
The starting column (numbered from Ø) is set to the decimal number given.

-d devname
The device file set or examined is devname. If this option is missing, the device file used is /dev/lp.

SEE ALSO

ioctl(2), stty(1).

NAME

setmnt - establish mnttab table

SYNOPSIS

setmnt

DESCRIPTION

Setmnt creates the /etc/mnttab table (see **mnttab**(5)), which is needed for both the **mount**(M) and **umount**(M) commands. **Setmnt** reads standard input and creates a **mnttab** entry for each line. Input lines have the format:

filesys node

where filesys is the name of the file system's special file (e.g., /dev/rz??) and node is the root name of that file system. Filesys and node become the first two strings in the **mnttab**(5) entry.

FILES

/etc/mnttab

/etc/setmnt the program

SEE ALSO

mnttab(5).

LIMITATIONS

Do not make filesys or node longer than 32 characters. **Setmnt** silently enforces an upper limit on the maximum number of **mnttab** entries; this limit is based on the system parameter **Nmount** in the kernel

NAME

shut - warns of system shutdown

SYNOPSIS

shut [dtime btime]

DESCRIPTION

Shut is called by the shell file "down". It sends warning messages to users. The optional arguments, dtime and btime, specify the time, in minutes, until system shutdown and the time between error messages, respectively. If the arguments are not supplied the user is prompted for them. At the appropriate times messages are broadcast to all logged-on users until the final warning message is sent then the program exits.

At the five minute mark **shut** begins to issue warnings at 1 minute intervals.

FILES

/etc/utmp

SEE ALSO

down(M).

DIAGNOSTICS

Diagnostics are intended to be self-explanatory.

NAME

str - software trouble report input program

SYNOPSIS

str [**-l**]

DESCRIPTION

This program is used for the initial capture of software trouble reports. It prompts the user for all needed information. The user also has the option of entering the editor, **vi**(1) is the default editor. The user may enter the editor at the beginning of the Problem Description field or just before submitting the trouble report.

Each submitted trouble report is assigned a sequence number, and is reflected in a master tracking file.

OPTIONS

-l Produce a line printer listing upon completion of the trouble report.

FILES

/usr/lib/problems/strfile
Data base for tracking trouble report progress

/usr/lib/problems/seqno
File containing the last used sequence number

/usr/lib/problems
Directory in which problem reports are kept

/usr/lib/problems/scrb_date
File containing date of next review meeting

SEE ALSO

strfile(5), strprint(M).

NAME

strprint - software trouble report listing program

SYNOPSIS

strprint

DESCRIPTION

This shell procedure is used for printing a copy of all software trouble reports on the line printer. Each trouble report is printed on a separate page. Each installation should periodically send a a copy of the output from this program to Zilog. Also, this listing should be made available to users so that they become aware of any software trouble reports about the system.

FILES

/etc/strfile
data base for tracking trouble report progress

/etc/seqno
file containing the last used sequence number

/etc/problems
directory in which problem reports are kept

/etc/scrb_date
File containing date of next review meeting

SEE ALSO

strfile(5), str(M).

NAME

sync - update the super block

SYNOPSIS

sync

DESCRIPTION

Sync executes the **sync** system primitive. If the system is to be stopped, **sync** must be called to insure file system integrity. See **sync**(2) for details.

SEE ALSO

sync(2), update(M).

NAME

sysgen - generate a Zeus kernel

SYNOPSIS

sysgen [-f file] [**-d** {11 21 31}]

DESCRIPTION

Sysgen is a Zeus kernel generation utility. **Sysgen** allows a user to interactively modify certain kernel parameters, configure up to 4 disk drives, include user written device drivers, add line printers, tape drives, or UNET into the Zeus kernel that is generated. This is a System Administrator's function that is outlined in more detail in the System Administrator Manual. The resulting kernel file is named zeus unless the -f option is used.

The **-d** option tells sysgen to use default answers to all the questions normally asked. The **-d 11** option is for the Model 11, the **-d 21** option is for the Model 21, and the **-d 31** option is for the Model 31.

FILES

/etc/sysgen

/usr/sys/conf/z.save.c

/usr/sys/h/sysparm.h

DIAGNOSTICS

The user must have write permission in /usr/sys/conf.

SEE ALSO

Zeus System Administrator Manual

NAME

lp, text - service line printer spooler print requests

SYNOPSIS

/usr/lib/lp [options]
/usr/lib/text [options]

DESCRIPTION

These programs are used by the spooling system to do the actual copy of data from a print request to a printer device. Lp is used to print requests on line printers. Text is used to print requests on text quality printers.

Each backend is started by dqueuer(M). The way backends are started is controlled by the configuration file (see the System Adminstrator Manual). A backend program inherits three open file descriptors from the dqueuer: RFFD (read file), SFFD (device) and STATUSFD (status log). Actual definitions of these variables can be found in /usr/include/spool.h. The lp and text backends copy data from file RFFD to file SFFD with minimal processing of input. Statuses are written to file STATUSFD.

In addition to printing, the backends perform functions required by the specific device being used. For instance, lp filters backspaces and text can perform stty(1) calls before printing. They also provide banner pages with appropriate information.

When invoked, backends are passed a parameter list generated by dqueuer(M). This list contains options generated by dqueuer(M) and those options that may have been specified in the configuration file. The options that dqueuer generates and that both backends recognize are:

-B Indicates that the file the dqueuer was to print could not be found. In this case, RFFD is not open and should not be used. This option is used to generate banner pages that say the file could not be found, so it won't appear that the file disappeared.

-c n
Indicates that the file must be printed multiple times. The backend should create n copies of the output.

-d dest
Used in a destination portion of the banner page. Dest is a string of no other importance to the backend.

-f filename
Filename, printed on the banner page.

-F from
Indicates a 'from' string, for use on the banner page.

-s spool time
A string in ctime(3) format, indicating when the file was spooled to be printed. This value is printed on banner page.

-t title
A string used as a title on banner pages.

The backend also looks in a file called '/etc/sitefile' to find a string to be printed near the page breaks on the banner page. It uses the string of characters on the first line of this file, up to the first white space. If this file does not exist, an appropriate default is used.

The text backend accepts one additional option, that can only specified through the configuration file. Its description follows:

-T [stty parameters]
Set terminal parameters. When a tty device is used as a printer, it is disabled in the /etc/inittab file. When this is done, the device is closed except when being used to print upon. This causes ZEUS to reset all terminal characteristics. The -T option is used to set these characteristics so that printing can occur. The quoted string is passed, unchanged except for removal of the quotes, to stty(1). Thus, in the configuration file the line

```
D1,R,/dev/tty0,/usr/lib/text -T "1200"
```

would cause device 1 on the most recent queue to use stty to set /dev/tty0 to 1200 BAUD before being used.

SEE ALSO
dqueuer(M)

NAME

ttyconfig - configure ports for terminal or modem line

SYNOPSIS

ttyconfig [**-m** ports] [**-t** ports]

DESCRIPTION

Ttyconfig is used by the System Administrator to configure ports for use with a terminal or modem. Typical use of **ttyconfig** is in the startup script /etc/rc.

If **ttyconfig** is invoked with no arguments, it reports the current settings of the ports. The options have the following meanings:

-m Configure the user supplied list of ports for use with a modem.

-t Configure the user supplied list of ports for use with a terminal.

The syntax for specifying a list of ports for the above options is a range of ports or a comma separated list. For example, to configure the ports 1 to 6 for use with a terminal the command line would read:

ttyconfig -t 1-6

To configure the ports Ø to 4 and 15 for use with a modem the command line would read:

ttyconfig -m Ø-4,15

The initial state when the system is booted, is that all ports are configured for use with terminals.

SEE ALSO

mdmctl(2).

DIAGNOSTICS

Diagnostics are issued for invalid options, invalid ports or ranges, and not being able to get or set the port configuration.

NAME

update - periodic buffer flush

SYNOPSIS

update

DESCRIPTION

Update causes a **sync** every 30 seconds. Since **update** never exits, it should only be executed once. This is best done by running **update** from the file /etc/rc_csh see **init**(M).

FILES

/etc/update the program

SEE ALSO

sync(2).

NAME

upkeep - directory maintenance

SYNOPSIS

upkeep [-mdfsli] [directory **]**

DESCRIPTION

upkeep maintains a .contents file and its corresponding directory. The .contents file is used as a master file. It contains information on what files (and their corresponding mode, user ID, group ID) should be in the directory. The flags have the following meanings:

-i Initialize the .contents file with current specified directory.

-l List .contents file

-d Report the difference between .contents file and its corresponding directory.

-f Fast update of directory, change the directory to match .contents file, produces a change report.

-s Slow update, same action as fast update, except before each change action, a confirmation request will be display on the terminal and user can confirm or deny action interactively.

-m Modification of .contents file and directory. In this mode upkeep accepts the user specified file name and performs one of the following requested actions:

add - add file to .contents file. File must be in the directory before this operation can be performed.

delete
- delete file from .contents and directory.

change
- change mode, uid and/or gid of file in .contents to be same as in directory.

correct
- correct the mode, uid and/or gid of file in directory to be same as in .contents.

A ctl-D answer to an interactive prompt terminates the program. Only owner or super user can make .contents/directory changes.

FILES

.contents
/tmp/upk.aXXXXX

NAME

wall - write to all users

SYNOPSIS

wall

DESCRIPTION

Wall reads its standard input until an end-of-file. It then sends this message, preceded by `Broadcast Message ...', to all logged in users.

The sender should be super-user to override any protections the users may have invoked.

FILES

/etc/wall the program

/dev/tty?

/etc/utmp

SEE ALSO

mesg(1), write(1).

DIAGNOSTICS

"Cannot send to " name" if the terminal is write-protected (see **mesg**(1)).

LIMITATIONS

Some protection should be possible to avoid interfering with communications occurring over a terminal line.

NAME

xq - examine or delete requests from the line printer spooler

SYNOPSIS

xq option [option]

DESCRIPTION

Xq is the part of the general queuing system that allows examination and deletion of items in the queue area. The system administrator and the system group are capable of executing all commands described in **xq**(1) with the added capability of affecting requests that are not their own. In order to allow this the user should use the following modifier on the command line before his request, to indicate he wishes to exercise a system command:

-o When used on the xq command line before a '-d' or '-s' option, to another user. The user must be super-user or system group.

With no options, **xq** acts as described in **xq** giving a status list of all the queues and devices.

Additional options can be used to control operation of **dqueuer**(M) and the queues and devices involved. In cases where the option acts upon a queue or queue:device pair, the '-q' modifier described in **xq** must be given. The default queue and device will not be assumed. Only one option may be given per **xq** command. Additional options are:

-b backup device. Causes a device to backup its output one page (used when a printer has become fouled, or needs new paper loaded.)

-Dd

Down device. Sets a device to the 'DOWN' state. This is commonly used if a printer is temporarily incapacitated, or if it desired to interrupt normal service. A device designated as 'DOWN' will continue to print a file in progress, but will not be selected for printing additional requests.

-Dq

Down queue. Sets a queue to 'DOWN' state. No requests to this queue will be accepted while its status is 'DOWN', nor can any devices on this queue be accessed. This step should only be used if all devices on a queue are likely to be down for an extended period of time.

-s[kd]

Quit printing. Causes printing on specified device to

stop immediately. If 'k' is specified, the file currently printing will be kept, otherwise it is deleted from the queue. If 'd' is specified, the status of the device affected will be set to 'DOWN' to temporarily prevent future printing.

-r Restart. Restart indicated queue:device. Causes the current file on the indicated device to restart printing at the beginning. This command is used to start over requests when a printer has become fouled.

-Ud
Up device. Makes a device available for print selection. (see '-Dd')

-Uq
Up queue. Makes a queue available for print requests. (see '-Dq')

FILES

/usr/spool/queuer/activeconfig	active configuration file
/usr/spool/queuer/logfile	
/usr/spool/queuer/statusdir	
/usr/spool/queuer/requestdir	
/tmp/queuer	

SEE ALSO
nq(1), xq(1), lp(M), dqueuer(M).

DIAGNOSTICS
If any part of the spooling system appears to have inconsistencies, **xq** will print an error and log it in the spooler error log file.

PERMUTED INDEX

This alphabetical index is based on the NAME line of the ZEUS Reference Manual. It contains one Index entry for each significant word in each NAME line.

For example, the NAME line for the **csh**(1) command reads:

csh, - a command interpreter with C-like syntax

The permuted index shows 5 entries for the **csh** command listed alphabetically under the listings:

C-like,
command,
csh,
interpreter,
and syntax.

the listings are shown below:

To use the permuted index, look for the desired word or topic at the left-most side of the page. The command entry is listed at the right-most side of the page. The number in parenthesis to the right of the word indicates the ZEUS Reference Manual section that the entry appears in.

Permuted Index

su -
substitute user ID temporarily..... su(1)

seisigl, setsig, sname, strend,
substr, trnslat, un...../satoi, pwb(3)

sum - sum and count blocks in a..... sum(1)
file

sum -
sum and count blocks in a file..... sum(1)

du -
summarize disk usage..... du(1)

quot -
summarize file system ownership..... quot(M)

acctcms - command
summary from per-process..... acctcms(M)
accounting/

sync - update the
super block..... sync(M)

sync - update
super-block..... sync(2)

sleep -
suspend execution for an interval..... sleep(1)

sleep -
suspend execution for interval..... sleep(3)

swab - swap bytes..... swab(3)

printf, putc, putchar, read,
swab, swap, write - Z8000/...../open, dm(3)

swap swapsegt - swap routines -
swap...../longswap - swap routines - swap(3)

swap - swap routines - swap..... swap(3)
longswap - swap routines - swap/

swab -
swap bytes..... swab(3)

swap - swap routines -
swap longswap - swap routines -..... swap(3)
swap/

troff, nroff -
text formatting and typesetting..... troff(1)

tgetent, tgetnum, tgetflag,..... termlib(3)
tgetstr, tgoto, tputs - terminal/
tgetent, tgetnum,
tgetflag, tgetstr, tgoto, tputs -..... termlib(3)
terminal/
tgetent,
tgetnum, tgetflag, tgetstr,..... termlib(3)
tgoto, tputs - terminal/
tgetent, tgetnum, tgetflag,
tgetstr, tgoto, tputs - terminal/..... termlib(3)

tgetnum, tgetflag, tgetstr,
tgoto, tputs - terminal/.....tgetent, termlib(3)

l3tol, ltol3 - convert between
three-byte integers and long/..... l3tol(3)

- schedule signal after specified
time.....alarm alarm(2)

or shell script file at a later
time.....at - execute command at(1)

up an environment at login
time...../cshprofile, login - setting cshrc(5)

date - print the date and
time..... date(1)

stime - set
time..... stime(2)

time, ftime - get date and
time..... time(2)

time - time a command..... time(1)

time -
time a command..... time(1)

timex -
time a command and generate a..... timex(1)
system activity report

time, ftime - get date and time..... time(2)

Systems Publications

Reader's Comments

Your feedback about this document helps us ascertain your needs and fulfill them in the future. Please take the time to fill out this questionaire and return it to us. this information will be helpful to us and, in time, to future users of Zilog products.

Your Name: _____

Company Name: _____

Address: _____

Title of this document: _____

Briefly describe application: _____

_____

_____

Does this publication meet your needs? ☐ Yes ☐ No If not, why not? _____

_____

_____

How are you using this publication?

☐ As an introduction to the subject?

☐ As a reference manual?

☐ As an instructor or student?

How do you find the material?

	Excellent	Good	Poor
Technicality	☐	☐	☐
Organization	☐	☐	☐
Completeness	☐	☐	☐

What would have improved the material? _____

_____

_____

Other comments and suggestions: _____

_____

_____

If you found any mistakes in this document, please let us know what and where they are: _____

_____

_____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 35 CAMPBELL, CA.

POSTAGE WILL BE PAID BY ADDRESSEE

Zilog

Systems Publications
1315 Dell Avenue
Campbell, California 95008
Attn: Publications Manager