Bell Laboratories

subject: UNIX Release 4.0

date: April 1, 1981

from: M. J. Petrella
PY 45174
2G-213 x6401

To All Recipients of UNIX Release 4.0:

Enclosed please find two tapes that contain Release 4.0 of the UNIX* Time-sharing System. The tapes are marked PDP-11 or VAX-11; make sure that you have the appropriate set.

In addition, you will find the following documentation:

1. UNIX Product Release Description with the following attachments:

   a. Differences Between Releases 3.0 and 4.0
   b. UNIX Release 4.0 Changes to Text Processing Software
   c. MRs Resolved
   d. Western Electric Company Right-to-Use Fee Announcement

2. Setting up UNIX
3. Administrative Advice for UNIX
4. LP Administrator's Guide
5. MM -- Memorandum Macros
6. UNIX Remote Job Entry Administrator's Guide
7. Release 1.0 of the UNIX Virtual Protocol Machine
8. Release 2.0 of the UNIX Virtual Protocol Machine
9. The UNIX Accounting System
10. Selected Manual Pages

Sites should install Release 4.0 by October 1, 1981; support of Release 3.0.1 will be discontinued as of April 1, 1982.

M. J. Petrella

M. J. Petrella
UNIX Release Coordinator

---

* UNIX is a trademark of Bell Laboratories.

NAME

accept, reject − allow/prevent LP requests

SYNOPSIS

/usr/lib/accept destinations
/usr/lib/reject [−**r**[reason]] destinations

DESCRIPTION

Accept allows *lp*(1) to accept requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use *lpstat*(1) to find the status of *destinations*.

Reject prevents *lp*(1) from accepting requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use *lpstat*(1) to find the status of *destinations*. The following option is useful with *reject*.

−**r**[*reason*] Associates a *reason* with preventing *lp* from accepting requests. This *reason* applies to all printers mentioned up to the next −**r** option. *Reason* is reported by *lp* when users direct requests to the named *destinations* and by *lpstat*(1). If the −**r** option is not present or the −**r** option is given without a *reason*, then a default *reason* will be used.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1), lpadmin(1M), lpsched(1M), lpstat(1).

NAME

asa − interpret ASA carriage control characters

SYNOPSIS

asa [files]

DESCRIPTION

Asa interprets the output of FORTRAN programs that utilize ASA carriage control characters. It processes either the *files* whose names are given as arguments or the standard input if no file names are supplied. The first character of each line is assumed to be a control character; their meanings are:

′ ′ single new line before printing

0 double new line before printing

1 new page before printing

\+ overprint previous line.

Lines beginning with other than the above characters are treated as if they began with ′ ′. The first character of a line is *not* printed. If any such lines appear, an appropriate diagnostic will appear on standard error. This program forces the first line of each input file to start on a new page.

SEE ALSO

efl(1), f77(1), fsplit(1), ratfor(1).

NAME

cflow − generate C flow graph

SYNOPSIS

cflow [**−r**] [**−ix**] [**−i_**] [**−d**num] files

DESCRIPTION

Cflow analyzes a collection of C, YACC, LEX, assembler, and object files and attempts to build a graph charting the external references. Files suffixed in **.y**, **.l**, **.c**, and **.i** are YACC'd, LEX'd, and C-preprocessed (bypassed for **.i** files) as appropriate and then run through the first pass of *lint*(1). (The **.SM−I, .SM−D,** and **.SM−U** options of the C-preprocessor are also understood.) Files suffixed with **.s** are assembled and information is extracted (as in **.o** files) from the symbol table. The output of all this non-trivial processing is collected and turned into a graph of external references which is displayed upon the standard output.

Each line of output begins with a reference (i.e., line) number, followed by a suitable number of tabs indicating the level. Then the name of the global (normally only a function not defined as an external or beginning with an underscore; see below for the **−i** inclusion option) a colon and its definition. For information extracted from C source, the definition consists of an abstract type declaration (e.g., **char ***), and, delimited by angle brackets, the name of the source file and the line number where the definition was found. Definitions extracted from object files indicate the file name and location counter under which the symbol appeared (e.g., *text*). Leading underscores in C-style external names are deleted.

Once a definition of a name has been printed, subsequent references to that name contain only the reference number of the line where the definition may be found. For undefined references, only < > is printed.

As an example, given the following in *file.c*:

```
int     i;

main()
{
        f();
        g();
        f();
}

f()
{
        i = h();
}
```

the command

```
cflow file.c
```

produces the the output

```
1       main: int(), <file.c 4>
2               f: int(), <file.c 11>
3                       h: <>
4                       i: int, <file.c 1>
5               g: <>
```

When the nesting level becomes too deep, the **−e** option of *pr*(1) can be used to compress the tab expansion to something less than every eight spaces.

The following options are interpreted by *cflow*:

−r Reverse the "caller:callee" relationship producing an inverted listing showing the callers of each function. The listing is also sorted in collating sequence.

−ix Include external and static names which are (by default) eliminated from the output listing.

−i_ Include names that begin with an underscore which are (by default) eliminated from the output listing.

−dnum The *num* decimal integer indicates the depth at which the flow graph is cut off. By default this is a very large number. Attempts to set the cutoff depth to a nonpositive integer will be met with contempt.

DIAGNOSTICS

Complains about bad options. Complains about multiple definitions and only believes the first. Other messages may come from the various programs used (e.g., the C-preprocessor).

SEE ALSO

as(1), cc(1), lex(1), lint(1), nm(1), pr(1), yacc(1).

BUGS

Files produced by *lex*(1) and *yacc*(1) cause the reordering of line number declarations which can confuse *cflow*. To get proper results, feed *cflow* the *yacc* or *lex* input.

NAME

checkall − faster file system checking procedure

SYNOPSIS

/etc/checkall

DESCRIPTION

The *checkall* procedure is a prototype and must be modified to suit local conditions. The following will serve as a example:

```
# check the root file system by itself
fsck /dev/rp0

# dual fsck of drives 0 and 1
dfsck /dev/rrp[12345] − /dev/rrp11
```

In the above example (where **/dev/rrp11** is 320K blocks and **/dev/rrp[12345]** are each 65K or less), a previous sequential *fsck* took 19 minutes. The *checkall* procedure takes 11 minutes.

Dfsck is a program that permits an operator to interact with two *fsck*(1M) programs at once. To aid in this, *dfsck* will print the file system name for each message to the operator. When answering a question from *dfsck*, the operator must prefix the response with a **1** or a **2** (indicating that the answer refers to the first or second file system group).

Due to the file system load balancing required for dual checking, the *dfsck* command should always be executed through the *checkall* shell procedure.

In a practical sense, the file systems are divided up as follows:

```
dfsck file_systems_on_drive_0 − file_systems_on_drive_1
dfsck file_systems_on_drive_2 − file_systems_on_drive_3
  . . .
```

A three drive system can be handled by this more concrete example (assumes two large file systems per drive):

```
dfsck /dev/rrp31 /dev/rrp[14] − /dev/rrp1[14] /dev/rrp34
```

Note that the first drive 3 file system is first in the *filesystems1* list and is last in the *filesystems2* list assuring that references to that drive will not overlap at execution time.

WARNINGS

1. Do not use *dfsck* to check the *root* file system.
2. On a check that requires a scratch file (see **−t** above), be careful not to use the same temporary file for the two groups (this is sure to scramble the file systems).
3. The *dfsck* procedure is useful only if the system is set up for multiple physical I/O buffers.

SEE ALSO

fsck(1M).
Setting up UNIX.

NAME

config − configure a UNIX system

SYNOPSIS

/etc/config [**−n**] [**−t**] [**−l** file] [**−c** file] [**−m** file] dfile

DESCRIPTION

Config is a program that takes a description of a UNIX system and generates two files. One file provides information regarding the interface between the hardware and device handlers. The other file is a C program defining the configuration tables for the various devices on the system.

The **−n** option produces a non-separated **I** and **D** space **low.s** core image for the PDP-11 (this is for small systems, i.e., PDP11/23 and 11/34).

The **−l** option specifies the name of the hardware interface file; **low.s** is the default name on the PDP-11; **univec.c** is the default name on the VAX-11/780.

The **−c** option specifies the name of the configuration table file; **conf.c** is the default name.

The **−m** option specifies the name of the file that contains all the information regarding supported devices; **/etc/master** is the default name. This file is supplied with the UNIX system and should *not* be modified unless the user *fully* understands its construction.

The **−t** option requests a short table of major device numbers for character and block type devices. This can facilitate the creation of special files.

The user must supply *dfile*; it must contain device information for the user's system. This file is divided into two parts. The first part contains physical device specifications. The second part contains system-dependent information. Any line with an asterisk (*) in column 1 is a comment.

All configurations are assumed to have the following devices:

one DL11 (for the system console)
one KW11-L line clock or KW11-P programmable clock

with standard interrupt vectors and addresses. These two devices *must not* be specified in *dfile*. Note that UNIX needs only one clock, but can handle both types.

First Part of *dfile*

Each line contains four or five fields, delimited by blanks and/or tabs in the following format:

devname vector address bus number

where *devname* is the name of the device (as it appears in the **/etc/master** device table), *vector* is the interrupt vector location (octal), *address* is the device address (octal), *bus* is the bus request level (4 through 7), and *number* is the number (decimal) of devices associated with the corresponding controller; *number* is optional, and if omitted, a default value which is the maximum value for that controller is used.

There are certain drivers that may be provided with the system, that are actually *pseudo-device* drivers; that is, there is no real hardware associated with the driver. Drivers of this type are identified on their respective manual entries. When these devices are specified in the description file, the interrupt *vector*, device *address*, and *bus* request level must all be zero.

If the device is a VAX-11/780 massbus adapter, then *vector* is the adapter nexus number, and *address* must be zero.

Second Part of *dfile*

The second part contains three different types of lines. Note that *all* specifications of this part *are required*, although their order is arbitrary.

1. *Root/pipe/dump device specification*

Three lines of three fields each:

root	devname	minor
pipe	devname	minor
dump	devname	minor

where *minor* is the minor device number (in octal).

2. *Swap device specification*

   One line that contains five fields as follows:

   swap devname minor swplo nswap

   where *swplo* is the lowest disk block (decimal) in the swap area and *nswap* is the number of disk blocks (decimal) in the swap area.

3. *Parameter specification*

   Several lines of two fields each as follows (*number* is decimal):

buffers	number	
sabufs	number	(zero on the VAX-11/780)
inodes	number	
files	number	
mounts	number	
coremap	number	(PDP-11 only)
swapmap	number	
calls	number	
procs	number	
maxproc	number	
texts	number	
clists	number	
hashbuf	number	
physbuf	number	
x25links	number	
x25bufs	number	
x25map	number	
x25bytes	number	
iblocks	number	(PDP-11 only)
smbufs	number	(VAX-11/780 only)
smmsgs	number	(VAX-11/780 only)
power	0 or 1	
msg	0 or 1	
sema	0 or 1	
maus	0 or 1	(PDP-11 only)

EXAMPLE

To configure a PDP-11/70 system with the following devices:

one RP06 disk drive controller with 6 drives
one DH11 asynchronous multiplexer with 16 lines (default number)
one DM11 modem control with 16 lines (for the DH11)
one DH11 asynchronous multiplexer with 8 lines
one DM11 modem control with 8 lines (for the DH11)
one LP11 line printer
one TU16 tape drive controller with 2 drives
one DL11 asynchronous interface

Note that UNIX only supports DH11 units that require corresponding DM11 units. It is wise to specify them in DH-DM pairs to facilitate understanding the configuration. Note also that, in the preceding case, the DL11 that is specified is *in addition* to the DL11 that was part of the initial system. We must also specify the following parameter information:

root device is an RP06 (drive 0, section 0)
pipe device is an RP06 (drive 0, section 0)
swap device is an RP06 (drive 1, section 4),
    with a swplo of 6000 and an nswap of 2000
dump device is a TU16 (drive 0)
number of buffers is 35
number of *system addressable* buffers is 12
number of processes is 150
maximum number of processes per user ID is 25
number of mounts is 8
number of inodes is 120
number of files is 120
number of calls is 30
number of texts is 35
number of character buffers is 150
number of coremap entries is 50
number of swapmap entries is 50
power fail recovery is to be included
messages are to be included
semaphores are to be included
one psuedo device driver for the Operating System Profiler

The actual system configuration would be specified as follows:

```
rp06      254     776700    5       6
dh11      320     760020    5
dm11      300     770500    4
dh11      330     760060    5       8
dm11      304     770510    4       8
lp11      200     775514    5
tu16      224     772440    5       2
dl11      350     775610    5
prf       0       0         0
root      rp06    0
pipe      rp06    0
swap      rp06    14        6000    2000
dump      tu16    0
* Comments may be inserted in this manner
buffers   35
sabufs    12
procs     150
maxproc   25
mounts    8
inodes    120
files     120
calls     30
texts     35
clists    150
coremap   50
swapmap   50
power     1
msg       1
```

sema 1

FILES

/etc/master	default input master device table
low.s	default output hardware interface file for PDP-11
univec.c	default output hardware interface file for the VAX-11/780
conf.c	default output configuration table file

SEE ALSO

sysdef(1M), master(5).
Setting up UNIX.

DIAGNOSTICS

Diagnostics are routed to the standard output and are self-explanatory.

BUGS

The **−t** option does not know about devices that have aliases. For example, an TE16 (an alias for an TU16) will show up as an TU16; however, the major device numbers are always correct.

NAME

cxref − generate C program cross reference

SYNOPSIS

cxref [options] files

DESCRIPTION

Cxref analyzes a collection of C files and attempts to build a cross reference table. *Cxref* utilizes a special version of *cpp* to include **#define**'d information in its symbol table. It produces a listing on standard output of all symbols (auto, static, and global) in each file separately, or with the **−c** option, in combination. Each symbol contains an asterisk (*) before the declaring reference.

In addition to the **.SM−D, .SM −I** and **.SM −U** options (which are identical to their interpretation by *cc*(1), the following *options* are interpreted by *cxref*:

−c Print a combined cross-reference of all input files.

−t Format listing for 72-column (terminal) width.

−o file Direct output to named *file*.

−s Operate silently; does not print input file names.

FILES

/usr/lib/xcpp special version of C-preprocessor.

SEE ALSO

cc(1).

DIAGNOSTICS

Error messages are unusually cryptic, but usually mean that you can't compile these files, anyway.

NAME

dcopy − copy file systems for optimal access time

SYNOPSIS

/etc/dcopy [**−c**n] [**−x**] [**−s**X] [**−d**] [**−p**] inputfs outputfs

DESCRIPTION

Dcopy copies file system *inputfs* to *outputfs*. *Inputfs* is the existing file system; *outputfs* is an appropriate sized section of disk to hold the reorganized result. Both *inputfs* and *outputfs* should be raw devices. *Dcopy* should be run on unmounted file systems (in the case of the root file system, copy to a new pack). With no arguments, *dcopy* copies files from *inputfs* compressing directories by removing vacant entries, and spacing consecutive blocks in a file by the optimal rotational gap. The possible options are

−s*X* Allows the user to supply the device information for creating an optimal organization of blocks in a file. The forms of *X* are the same as the **−s** option of *fsck*(1M).

−c*n* Make every *n* consecutive blocks in a file contiguous, separating each group of *n* blocks by the appropriate interval.

−a*n* place the files not accessed in *n* days after the free blocks of the destination file system (default for *n* is 7). If no *n* is specified then no movement occurs.

−d leave order of directory entries as is (default is to move sub-directories to the beginning of directories).

−v currently reports how many files were processed, and how big the source and destination freelists are.

−f*fsize*[**:isize**]
allows the user to specify the destination file system size and inode list size (as in *mkfs*(1M)). If not given, the size of the destination file system is taken from the size of *inputfs*.

Dcopy catches interrupts and quits and reports on its progress. To terminate *dcopy*, send a quit signal and *dcopy* will no longer catch interrupts or quits. *Dcopy* also attempts to modify its command line arguments so its progress can be monitored with *ps*(1).

SEE ALSO

fsck(1M), mkfs(1M), ps(1).

NAME

enable, disable — enable/disable LP printers

SYNOPSIS

enable printers
disable [**−c**] [**−r**[reason]] printers

DESCRIPTION

Enable activates the named *printers*, enabling them to print requests taken by *lp*(1). Use *lpstat*(1) to find the status of printers.

Disable deactivates the named *printers*, disabling them from printing requests taken by *lp*(1). By default, any requests that are currently printing on the designated printers will be reprinted in their entirety either on the same printer or on another member of the same class. Use *lpstat*(1) to find the status of printers. Options useful with *disable* are:

−c Cancel any requests that are currently printing on any of the designated printers.

−r[*reason*] Associates a *reason* with the deactivation of the printers. This reason applies to all printers mentioned up to the next **−r** option. If the **−r** option is not present or the **−r** option is given without a reason, then a default reason will be used. *Reason* is reported by *lpstat*(1).

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), lp(1), lpadmin(1M), lpsched(1M), lpstat(1).

NAME

fsck, dfsck − file system consistency check and interactive repair

SYNOPSIS

/etc/fsck [**−y**] [**−n**] [**−sX**] [**−SX**] [**−t**] file [**−r**] [**−q**] [**−D**] [file-systems]

/etc/dfsck [**options1**] filsys1... − [**options2**] filsys2...

DESCRIPTION

Fsck audits and interactively repairs inconsistent conditions for UNIX file systems. If the 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. It should be noted that most corrective actions will result in some loss of data. The amount and severity of data lost may be determined from the diagnostic output. The default action for each consistency correction is to wait for the operator to respond **yes** or **no**. If the operator does not have write permission *fsck* will default to a **−n** action.

Fsck has more consistency checks than its predecessors *check*, *dcheck*, *fcheck*, and *icheck* combined.

The following flags are interpreted by *fsck*.

−y Assume a yes response to all questions asked by *fsck*.

−n Assume a no response to all questions asked by *fsck*; do not open the file system for writing.

−s*X* Ignore the actual free list and (unconditionally) reconstruct a new one by rewriting the super-block of the file system. The file system should be unmounted while this is done; if this is not possible, 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 **−s***X* option allows for creating an optimal free-list organization. The following forms of *X* are supported for the following devices:

−s3 (RP03)
−s4 (RP04, RP05, RP06)
−sBlocks-per-cylinder:Blocks-to-skip (for anything else)

If *X* is not given, the values used when the file system was created are used. If these values were not specified, then the value *400:9* is used.

−S*X* Conditionally reconstruct the free list. This option is like **−s***X* above except that the free list is rebuilt only if there were no discrepancies discovered in the file system. Using **−S** will force a no response to all questions asked by *fsck*. This option is useful for forcing free list reorganization on uncontaminated file systems.

−t If *fsck* cannot obtain enough memory to keep its tables, it uses a scratch file. If the **−t** option is specified, the file named in the next argument is used as the scratch file, if needed. Without the **−t** flag, *fsck* will prompt the operator for the name of the scratch file. The file chosen 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.

−r Silently remove files under **/tmp** and **/usr/tmp**.

−q Quiet *fsck*. Do not print size-check messages in Phase 1. Unreferenced **fifos** will silently be removed. If *fsck* requires it, counts in the superblock will be automatically fixed and the free list salvaged.

−D Directories are checked for bad blocks. Useful after system crashes.

If no *file-systems* are specified, *fsck* will read a list of default file systems 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.
   Directory size not 16-byte aligned.
5. Bad inode format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   File pointing to unallocated inode.
   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, if the files are nonempty. The user will be notified if the file or directory is empty or not. If it is empty, *fsck* will silently remove them. *Fsck* will force the reconnection of nonempty directories. The name assigned is the inode number. The only restriction is that the directory **lost+found** must preexist in the root of the file system being checked and must have empty slots in which entries can be made. This is accomplished by making **lost+found**, copying a number of files to the directory, and then removing them (before *fsck* is executed).

Checking the raw device is almost always faster and should be used with everything but the *root* file system.

Dfsck allows two file system checks on two different drives simultaneously. *options1* and *options2* are used to pass options to *fsck* for the two sets of file systems. A **'−'** is the separator between the file system groups.

The *dfsck* program permits an operator to interact with two *fsck*(1M) programs at once. To aid in this, *dfsck* will print the file system name for each message to the operator. When answering a question from *dfsck*, the operator must prefix the response with a **1** or a **2** (indicating that the answer refers to the first or second file system group).

Do not use *dfsck* to check the *root* file system.

FILES

/etc/checklist	contains default list of file systems to check.
/etc/checkall	optimizing *dfsck* shell file.

SEE ALSO

checkall(1M), clri(1M), ncheck(1M), checklist(5), fs(5), crash(8).
Setting up UNIX.

BUGS

Inode numbers for **.** and **..** in each directory should be checked for validity.

DIAGNOSTICS

The diagnostics produced by *fsck* are intended to be self-explanatory.

NAME

fsplit − split f77, ratfor, or efl files

SYNOPSIS

fsplit options files

DESCRIPTION

Fsplit splits the named *file(s)* into separate files, with one procedure per file. Procedure *X* is put in file *X*.**f**, *X*.**r**, or *X*.**e** depending on the language option chosen.

The following *options* pertain:

−f (default) Input files are *f77*.

−r Input files are *ratfor*.

−e Input files are *Efl*.

−s Strip *f77* procedure lines to 72 or fewer characters with trailing blanks removed.

SEE ALSO

csplit(1), efl(1), f77(1), ratfor(1), split(1).

NAME

fuser — identify processes using a file or file structure

SYNOPSIS

fuser [**−ku**] files [−] [[**−ku**] files]

DESCRIPTION

Fuser lists the process IDs of the processes using the *files* specified as arguments. For block special devices, all processes using any file on that device are listed. The process ID is followed by **c**, **p** or **r** if the process is using the file as its current **d**irectory, the **p**arent of its current directory (only when in use by the system), or its **r**oot directory, respectively. If the **−u** option is specified, the login name, in parentheses, also follows the process ID. In addition, if the **−k** option is specified, the **SIGKILL** signal is sent to each process. Only the super-user can terminate another user's process (see *kill*(2)). Options may be respecified between groups of files. The new set of options replaces the old set, with a lone dash canceling any options currently in force.

The process IDs are printed as a single line on the standard output, separated by spaces and terminated with a single new line. All other output is written on standard error.

EXAMPLES

fuser −ku /dev/rp1?

will terminate all processes that are preventing disk drive one from being unmounted if typed by the super-user, listing the process ID and login name of each as it is killed.

fuser −u /etc/passwd

will list process IDs and login names of processes that have the password file open.

fuser −ku /dev/rp1? −u /etc/passwd

will do both of the above examples in a single command line.

FILES

/unix	for namelist
/dev/kmem	for system image
/dev/mem	also for system image

SEE ALSO

ps(1), mount(1M), kill(2), signal(2).

NAME

killall — kill all active processes

SYNOPSIS

/etc/killall [signal]

DESCRIPTION

Killall is is a procedure used by **/etc/shutdown** to kill all active processes not directly related to the shut down procedure.

Killall is chiefly used to terminate all processes with open files so that the mounted file systems will be unbusied and can be unmounted.

Killall sends *signal* (see *kill*(1)) to all remaining processes not belonging to the above group of exclusions. If no *signal* is specified, a default of **9** is used.

FILES

/etc/shutdown

SEE ALSO

fuser(1M), kill(1), ps(1), shutdown(1M), signal(2).

NAME

lp, cancel − send/cancel requests to an LP line printer

SYNOPSIS

lp [**−c**] [**−d**dest] [**−m**] [**−n**number] [**−o**option] [**−t**title] [**−w**] files

cancel [ids] [printers]

DESCRIPTION

Lp arranges for the named files and associated information (collectively called a *request*) to be printed by a line printer. If no file names are mentioned, the standard input is assumed. The file name − stands for the standard input and may be supplied on the command line in conjunction with named *files*. The order in which *files* appear is the same order in which they will be printed.

Lp associates a unique *id* with each request and prints it on the standard output. This *id* can be used later to cancel (see *cancel*) or find the status (see *lpstat*(1)) of the request.

The following options to *lp* may appear in any order and may be intermixed with file names:

−c Make copies of the *files* to be printed immediately when *lp* is invoked. Normally, *files* will not be copied, but will be linked whenever possible. If the **−c** option is not given, then the user should be careful not to remove any of the *files* before the request has been printed in its entirety. It should also be noted that in the absence of the **−c** option, any changes made to the named *files* after the request is made but before it is printed will be reflected in the printed output.

−d*dest* Choose *dest* as the printer or class of printers that is to do the printing. If *dest* is a printer, then the request will be printed only on that specific printer. If *dest* is a class of printers, then the request will be printed on the first available printer that is a member of the class. Under certain conditions (printer unavailability, file space limitation, etc.), requests for specific destinations may not be accepted (see *accept*(1) and *lpstat*(1)). By default, *dest* is taken from the environment variable **LPDEST** (if it is set). Otherwise, a default destination (if one exists) for the computer system is used. Destination names vary between systems (see *lpstat*(1)).

−m Send mail (see *mail*(1)) after the files have been printed. By default, no mail is sent upon normal completion of the print request.

−n*number* Print *number* copies (default of 1) of the output.

−o*option* Specify printer-dependent or class-dependent *options*. Several such *options* may be collected by specifying the **−o** keyletter more than once. For more information about what is valid for *options*, see *Models* in *lpadmin*(1M).

−t*title* Print *title* on the banner page of the output.

−w Write a message on the user's terminal after the *files* have been printed. If the user is not logged in, then mail will be sent instead.

Cancel cancels line printer requests that were made by the *lp* command. The command line arguments may be either request *ids* (as returned by *lp*) or *printer* names (for a complete list, use *lpstat*(1)). Specifying a request *id* cancels the associated request even if it is currently printing. Specifying a *printer* cancels the request which is currently printing on that printer. In either case, the cancellation of a request that is currently printing frees the printer to print its next available request.

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), enable(1), lpadmin(1M), lpsched(1M), lpstat(1), mail(1).

NAME

lpadmin − configure the LP spooling system

SYNOPSIS

/usr/lib/lpadmin −pprinter [options]
/usr/lib/lpadmin −xdest
/usr/lib/lpadmin −d[dest]

DESCRIPTION

Lpadmin configures LP spooling systems to describe printers, classes and devices. It is used to add and remove destinations, change membership in classes, change devices for printers, change printer interface programs and to change the system default destination. *Lpadmin* may not be used when the LP scheduler, *lpsched*(1M), is running, except where noted below.

Exactly one of the **−p**, **−d** or **−x** options must be present for every legal invocation of *lpadmin*.

−d[*dest*] makes *dest*, an existing destination, the new system default destination. If *dest* is not supplied, then there is no system default destination. This option may be used when *lpsched*(1M) is running. No other *options* are allowed with **−d**.

−x*dest* removes destination *dest* from the LP system. If *dest* is a printer and is the only member of a class, then the class will be deleted, too. No other *options* are allowed with **−x**.

−p*printer* names a *printer* to which all of the *options* below refer. If *printer* does not exist then it will be created.

The following *options* are only useful with **−p** and may appear in any order. For ease of discussion, the printer will be refered to as *P* below.

−c*class* inserts printer *P* into the specified *class*. *Class* will be created if it does not already exist.

−e*printer* copies an existing *printer's* interface program to be the new interface program for *P*.

−h indicates that the device associated with *P* is hardwired. This *option* is assumed when creating a new printer unless the **−l** *option* is supplied.

−i*interface* establishes a new interface program for *P*. *Interface* is the path name of the new program.

−l indicates that the device associated with *P* is a login terminal. The LP scheduler, *lpsched*, disables all login terminals automatically each time it is started. Before re-enabling *P*, its current *device* should be established using *lpadmin*.

−m*model* selects a model interface program for *P*. *Model* is one of the model interface names supplied with the LP software (see *Models* below).

−r*class* removes printer *P* from the specified *class*. If *P* is the last member of the *class*, then the *class* will be removed.

−v*device* associates a new *device* with printer *P*. *Device* is the pathname of a file that is writable by the LP administrator, *lp*. Note that there is nothing to stop an administrator from associating the same *device* with more than one *printer*. If only the **−p** and **−v** *options* are supplied, then *lpadmin* may be used while the scheduler is running.

Restrictions.

When creating a new printer, the **−v** option and one of the **−e**, **−i** or **−m** options must be supplied. Only one of the **−e**, **−i** or **−m** options may be supplied. The **−h** and **−l** keyletters are mutually exclusive. Printer and class names may be no longer than 14 characters and must consist entirely of the characters **A-Z**, **a-z**, **0-9** and _ (underscore).

Models.

Model printer interface programs are supplied with the LP software. They are shell procedures which interface between *lpsched* and devices. All models reside in the directory **/usr/spool/lp/model** and may be used as is with *lpadmin* **−m**. Alternatively, LP administrators may modify copies of models and then use *lpadmin* **−i** to associate them with printers. The following list describes the *models* and lists the options which they may be given on the *lp* command line using the **−o** keyletter:

dumb interface for a line printer without special functions and protocol. Form feeds are assumed. This is a good model to copy and modify for printers which do not have models.

1640 Diablo 1640 terminal running at 1200 baud, using XON/XOFF protocol. Options:

- **−12** 12-pitch (10-pitch is the default)
- **−f** don't use the *450*(1) filter. The output has been pre-processed by either *450*(1) or the *nroff* 450 driving table.

hp Hewlett Packard 2631A line printer at 2400 baud. Options:

- **−c** compressed print
- **−e** expanded print

prx Printronix P300 printer using XON/XOFF protocol at 1200 baud.

EXAMPLES

1. Assuming there is an existing Hewlett Packard 2631A line printer named *hp2*, it will use the **hp** model interface after the command:

   /usr/lib/lpadmin −php2 −mhp

2. To obtain compressed print on *hp2*, use the command:

   lp −dhp2 −o−c files

3. A Diablo 1640 printer called *st1* can be added to the LP configuration with the command:

   /usr/lib/lpadmin −pst1 −v/dev/tty20 −m1640

4. An *nroff* document may be printed on *st1* in any of the following ways:

   nroff −T450 files | lp −dst1 −of
   nroff −T450−12 files | lp −dst1 −of
   nroff −T37 files | col | lp −dst1

5. The following command prints the password file on *st1* in 12-pitch:

   lp −dst1 −o12 /etc/passwd

   NOTE: the **−12** option to the **1640** model should never be used in conjunction with *nroff*.

FILES

/usr/spool/lp/*

SEE ALSO

450(1), accept(1M), enable(1), lp(1), lpsched(1M), lpstat(1).

NAME

lpsched, lpshut, lpmove — start/stop the LP request scheduler and move requests

SYNOPSIS

/usr/lib/lpsched
/usr/lib/lpshut
/usr/lib/lpmove requests dest
/usr/lib/lpmove dest1 dest2

DESCRIPTION

Lpsched schedules requests taken by *lp*(1) for printing on line printers.

Lpshut shuts down the line printer scheduler. All printers that are printing at the time *lpshut* is invoked will stop printing. Requests that were printing at the time a printer was shut down will be reprinted in their entirety after *lpsched* is started again. All LP commands perform their functions even when *lpsched* is not running.

Lpmove moves requests that were queued by *lp*(1) between LP destinations. This command may be used only when *lpsched* is not running.

The first form of the command moves the named *requests* to the LP destination, *dest*. *Requests* are request ids as returned by *lp*. The second form moves all requests for destination *dest1* to destination *dest2*. As a side effect, *lp* will reject requests for *dest1*.

Note that *lpmove* never checks the acceptance status (see *accept*(1)) for the new destination when moving requests.

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), enable(1), lp(1), lpadmin(1M), lpstat(1).

NAME

lpstat − print LP status information

SYNOPSIS

lpstat [options]

DESCRIPTION

Lpstat prints information about the current status of the LP line printer system.

If no *options* are given, then *lpstat* prints the status of all requests made to *lp*(1) by the user. Any arguments that are not *options* are assumed to be request *ids* (as returned by *lp*). *Lpstat* prints the status of such requests. *Options* may appear in any order and may be repeated and intermixed with other arguments. Some of the keyletters below may be followed by an optional *list* that can be in one of two forms: a list of items separated from one another by a comma, or a list of items enclosed in double quotes and separated from one another by a comma and/or one or more spaces. For example:

−u"user1, user2, user3"

The omission of a *list* following such keyletters causes all information relevant to the keyletter to be printed, for example:

lpstat −o

prints the status of all output requests.

−a[*list*] Print acceptance status (with respect to *lp*) of destinations for requests. *List* is a list of intermixed printer names and class names.

−c[*list*] Print class names and their members. *List* is a list of class names.

−d Print the system default destination for *lp*.

−o[*list*] Print the status of output requests. *List* is a list of intermixed printer names, class names, and request *ids*.

−p[*list*] Print the status of printers. *List* is a list of printer names.

−r Print the status of *lpsched*(1M).

−s Print a status summary, including the status of the line printer scheduler, the system default destination, a list of class names and their members, and a list of printers and their associated devices.

−t Print all status information.

−u[*list*] Print status of output requests for users. *List* is a list of login names.

−v[*list*] Print the names of printers and the pathnames of the devices associated with them. *List* is a list of printer names.

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), enable(1), lp(1), lpadmin(1M), lpsched(1M).

NAME

mkfs − construct a file system

SYNOPSIS

/etc/mkfs special blocks[:inodes] [gap blocks]
/etc/mkfs special proto [gap blocks]

DESCRIPTION

Mkfs constructs a file system by writing on the special file according to the directions found in the remainder of the command line. If the second argument is given as a string of digits, *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 boot program is left uninitialized. If the optional number of inodes is not given, the default is the number of blocks divided by 4.

If the second argument is a file name that can be opened, *mkfs* assumes it to be a prototype file *proto*, and will take its directions from that file. 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 (see *unixboot*(8)). 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 i-list size in blocks (remember there are eight i-nodes per block). 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 path name 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 **$**.

A sample prototype specification follows:

```
/stand/diskboot
4872 110
d--777 3 1
usr     d--777 3 1
        sh      ---755 3 1 /bin/sh
        ken     d--755 6 1
                $
        b0      b--644 3 1 0 0
        c0      c--644 3 1 0 0
        $
$
```

In both command syntaxes, the rotational *gap* and the number of *blocks* can be specified. The following values are recommended:

Device	Gap Size	Blks/Cyl
RP03	5	200
RP04/05/06	7	418
RM05	7	608
RL01/02	12	20
default	3	1000

The *default* will be used if the supplied *gap* and *block* are considered illegal values or if a short argument count occurs.

SEE ALSO

dir(5), fs(5), unixboot(8).

BUGS

If a prototype is used, it is not possible to initialize a file larger than 64K bytes, nor is there a way to specify links.

NAME

net − execute a command on the PCL network

SYNOPSIS

net system [command [args]]

DESCRIPTION

Net provides a bi-directional connection to another UNIX. The first argument is the name of the remote *system*. The second argument is a *command* to be executed. If *command* is not given, then an interactive shell (**/bin/sh −i**) on the remote system is created and an initial working directory of / is established. Any remaining arguments are passed to the given command as arguments.

Net reads the standard input, thus allowing *command* to be part of a "pipeline" if *command* reads the standard input also.

EXAMPLES

Execute the *who*(1) command on system A and return the output to your terminal:

```
net A who
```

Copy a directory structure from system A to the local system:

```
cd /dir/on/localsys
net A "cd /dir/on/A; find . −print | cpio −oc" | cpio −icda
```

Copy one file from system A to the local system:

```
net A "cat /file/on/A" > /file/on/localsys
```

Send a directory structure from the local system to system A (this uses the command's ability to read standard input):

```
find . −print | cpio −o | net A "cd /dir/on/A; cpio −id"
```

FILES

/dev/pcl/*?*[0-7]	PCL channel interfaces for system *?*.
/dev/pcl/ctrl	PCL control channel.
/usr/adm/pcllog	activity log.

SEE ALSO

cpio(1), find(1), pcldaemon(1C), sh(1), pcl(4).

DIAGNOSTICS

net: cannot open channel to system
A connection can't be made to the requested system.

connection broken
A non-recoverable write error occurred.

write error
A recoverable write error occurred. The write will be retried until it completes successfully without losing data.

cannot fork reader process
Net is unable to create a reader process and a writer process.

WARNINGS

A successful invocation of *net* reads at least 2 blocks of the standard input, if present, even if *command* does not use standard input. The standard input must be explicitly closed (via <&−) or redirected (such as from **/dev/null**) if this feature is not desired.

BUGS

Only the first character of a *system* name is recognized. Remaining characters are silently

discarded.
The user's command environment is not carried forward to the remote system except for the effective user ID.
Executing commands that do "funny" things with your terminal (i.e., *cu*(1C), *passwd*(1), *su*(1), etc.) don't work as expected.

NAME

newform − change the format of a text file

SYNOPSIS

newform [−**s**] [−**i**tabspec] [−**o**tabspec] [−**b**n] [−**e**n] [−**p**n] [−**a**n] [−**f**] [−**c**char] [−**l**n] [files]

DESCRIPTION

Newform reads lines from the named *files*, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect.

Except for −**s**, command line options may appear in any order, may be repeated, and may be intermingled with the optional *files*. Command line options are processed in the order specified. This means that option sequences like "−**e**15 −**l**60" will yield results different from "−**l**60 −**e**15". Options are applied to all *files* on the command line.

−**i***tabspec* Input tab specification: expands tabs to spaces, according to the tab specifications given. *Tabspec* recognizes all tab specification forms described in *tabs*(1). In addition, *tabspec* may be −−, in which *newform* assumes that the tab specification is to be found in the first line read from the standard input (see *fspec*(5)). If no *tabspec* is given, *tabspec* defaults to −**8**. A *tabspec* of −**0** expects no tabs; if any are found, they are treated as −**1**.

−**o***tabspec* Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for −**i***tabspec*. If no *tabspec* is given, *tabspec* defaults to −**8**. A *tabspec* of −**0** means that no spaces will be converted to tabs on output.

−**l***n* Set the effective line length to *n* characters. If *n* is not entered, −**l** defaults to 72. The default line length without the −**l** option is 80 characters. Note that tabs and backspaces are considered to be one character (use −**i** to expand tabs to spaces).

−**b***n* Truncate *n* characters from the beginning of the line when the line length is greater than the effective line length (see −**l***n*). Default is to truncate the number of characters necessary to obtain the effective line length. The default value is used when −**b** with no *n* is used. This option can be used to delete the sequence numbers from a COBOL program as follows:

```
newform −l1 −b7 file-name
```

The −**l1** must be used to set the effective line length shorter than any existing line in the file so that the −**b** option is activated.

−**e***n* Same as −**b***n* except that characters are truncated from the end of the line.

−**c***k* Change the prefix/append character to *k*. Default character for *k* is a space.

−**p***n* Prefix *n* characters (see −**c***k*) to the beginning of a line when the line length is less than the effective line length. Default is to prefix the number of characters necessary to obtain the effective line length.

−**a***n* Same as −**p***n* except characters are appended to the end of a line.

−**f** Write the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the *last* −**o** option. If no −**o** option is specified, the line which is printed will contain the default specification of −**8**.

−**s** Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by a * and any characters to the right of it are discarded. The first tab is always discarded.

An error message and program exit will occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.

For example, to convert a file with leading digits, one or more tabs, and text on each line, to a file beginning with the text, all tabs after the first expanded to spaces, padded with spaces out to column 72 (or truncated to column 72), and the leading digits placed starting at column 73, the command would be:

newform −s −i −l −a −e file-name

DIAGNOSTICS

All diagnostics are fatal.

usage: ...	*Newform* was called with a bad option.
not −s format	There was no tab on one line.
can't open file	Self explanatory.
internal line too long	A line exceeds 512 characters after being expanded in the internal work buffer.
tabspec in error	A tab specification is incorrectly formatted, or specified tab stops are not ascending.
tabspec indirection illegal	A *tabspec* read from a file (or standard input) may not contain a *tabspec* referencing another file (or standard input).

EXIT CODES

0 − normal execution
1 − for any error

SEE ALSO

csplit(1), tabs(1), fspec(5).

BUGS

Newform normally only keeps track of physical characters; however, for the **−i** and **−o** options, *newform* will keep track of backspaces in order to line up tabs in the appropriate logical columns.

Newform will not prompt the user if a *tabspec* is to be read from the standard input (by use of **−i**−− or **−o**−−).

If the **−f** option is used, and the last **−o** option specified was **−o**−−, and was preceded by either a **−o**−− or a **−i**−−, the tab specification format line will be incorrect.

NAME

nscrpt − NSC local network monthly availability report generator

SYNOPSIS

nscrpt [−**m** month] [−**y** year] [−**d** dir] [−**f** file] [−**acelpqrsv**] [units]

DESCRIPTION

Using raw data files as produced by *nscgath*(1C), *nscrpt* produces a monthly report on NSC local network hardware availability. The report uses raw data files generated on every host configured on the network. If the raw data file from any configured host does not exist, an error message will be printed and no report will be generated.

A separate report is produced for each NSC processor adapter. Each report is divided into two similar parts. The first part reports on detected host/adapter interface problems. Errors indicate that the adapter did not respond to an outgoing network transfer request. The second part reports on adapter/trunk interface problems. Errors indicate that the adapter did not complete an incoming trunk transfer from some remote host.

Each part of the report is subdivided into several sections. Each part contains a day by hour graph section illustrating exactly when each error was detected. Each error is flagged by a severity code (1-6). The lower the severity code, the more likely the error is a transient one. The higher the severity code, the more likely the error indicates a hardware or microcode problem. Each part also lists the daily adapter availability percentages for extended-prime and non-prime hours. The last section lists every error detected during the month.

The final part of the report gives a monthly availability summary for all reported adapters. This is a condensed report of the previous two sections.

The following options select the month, the adapter(s), and and where to find the raw data files for the report.

−**m** month Generate a report for *month*. The default is the previous month.

−**y** year Generate a report for the indicated month in *year*. The default is the year of the previous month.

−**d** dir Look for the raw data files in directory *dir*. The default search directory is **/usr/adm/rje**.

−**f** file Use *file* as the network configuration file. The default configuration file is **/etc/nsc**.

The following options select the sections and the parts of the report. One or more of **lr** may be optionally concatenated with one or more of **acepv**. That is, every **acepv** specified applies to every **lr** in effect. If one or more from either of the two groups are specified, the ones not specified are automatically disabled.

−**a** Enable options **cepsv**.

−**c** Print the host connection configuration report. This section is included in the host/adapter interface part.

−**e** Print a detailed list of all errors detected. Each error will be sorted by day and time of the occurrence.

−**l** Print the host/adapter interface report.

−**p** Print the day by hour plot indicating when the errors occurred and the severity of each one.

−**q** Do not complain when one or more raw data files do not exist.

−**r** Print the adapter/trunk interface report.

−s Print the monthly summary report for overall adapter availability.

−v Print the daily availability percentages for extended-prime and non-prime hours.

By default, options **lrcpsv** are enabled. If *unit* is specified, a report will be generated for only those adapters whose *unit* number is given.

FILES

/etc/nsc	default network configuration file
/usr/adm/rje/mmm.nnnnn	raw data file for machine *nnnnn* during month *mmm*.

SEE ALSO

nscgath(1C), nscmon(1M).

DIAGNOSTICS

All error messages are designed to be self-explanatory.

NAME

nscstat − query the operation status of the NSC network

SYNOPSIS

nscstat [**−ludqa**] [names]

DESCRIPTION

Nscstat, without arguments, gives a short operational status report of the NSC network from the viewpoint of the local node. This includes the status of the NSC network and the total number of files queued for transmission. *Nscstat* recognizes the following arguments:

−l Output a long listing. This option indicates the status of the printed node (online or offline), the total number of files queued waiting transmission to this node, and the time when the first job was queued for transmission.

−u List the status of all nodes that are online (up).

−d List the status of all nodes that are offline (down).

−q List the status of all nodes that have files queued for transmission.

−a List the status of all nodes configured on the network.

Each of the above arguments may be used singly or together with several others. When used together, the output is the intersection of the sets of nodes matching each option. *Name* specifies that status is requested for this node only. If this node is not in the intersection set of the specified options, the status of the node will not be reported.

EXAMPLE

To get a long listing of all nodes that are currently offline and have files queued for transmission:

```
nscstat −ldq
```

FILES

/usr/nsc/rvchan	list of nodes currently configured on the network
/usr/nsc/cons/*	nodes that are considered online
/usr/nsc/jobs/C*	jobs queued for transmission
/usr/nsc/cons/ACTIVE	whether the NSC network is active or not

BUGS

Nscstat tries to interpret the specified options intelligently. If none of the options specified apply to any of the specified nodes, no detailed status will be reported.

NAME

nsctorje − re-route jobs from the NSC network to RJE

SYNOPSIS

nsctorje [**−d** names]

DESCRIPTION

Nsctorje will resubmit jobs queued on the NSC local network (via *nusend*(1C) across the RJE link (if it exists). *Nsctorje* submits a *nusend*(1C) command to re-route each queued job. By default, jobs will be re-routed if either the remote host is marked down locally or if the NSC network on the local host is inactive. *Nsctorje* recognizes the following options:

−d names re-route all jobs queued only to the remote machine *name*.

SEE ALSO

nscmon(1M), rje(8).

FILES

/usr/nsc/NORJE	file indicating that no RJE connection exists on this machine
/usr/nsc/rvchan	*nusend*(1C) network configuration file
/usr/asp/udest	nodes accessible through RJE

BUGS

Any file larger than 190,000 bytes will not be re-routed across the RJE link. It will remain queued on the NSC network until the remote node becomes available.

NAME

nusend − send files to another UNIX on the NSC network

SYNOPSIS

nusend **−d** dest [**−a** acct] [**−m**] [**−e**] [**−s**] [**−c**] [**−u** destuser] [[**−f** destfile] srcfile] [−!cmd [cmdfile]] ...

DESCRIPTION

Nusend sends copies of the named files or command to another UNIX system via the NSC network. If the file name − is given, the standard input is read at that point.

−d dest Destination. *Dest* can be any one of the UNIX systems on the NSC local network. See **/usr/nsc/rvchan** for an up-to-date list of valid NSC destinations.

−a acct Use *acct* as the account number for the job. By default, the account number is read from the password file.

−s Silent. Suppress the one-line message which contains the submitted job name.

−c Copy. Make a copy of the file. Default is to set up a pointer to the file in the user's directory. If any changes are made to the file before transmission, the changes will be sent to the destination unless the **−c** option is used.

Mail will normally be sent to the receiving login(s) to report the receipt of the file(s). Mail will be sent to both sending and receiving logins if there were errors in transmission. The default may be overridden with the following switches:

−m Report by *mail*(1) when the file transfer is complete. The mail is sent from the remote system via *nusend*.

−e Report by *mail*(1) only when an error occurred during the transfer. No other mail will be sent.

Normally, the login name under which the new file will appear on the destination system is the same as the login name of the person who issues the command.

The following options, each as a separate argument, may be interspersed with file name arguments:

−u Use the next argument as the destination user's login name for all succeeding files.

−f Use the next argument as the destination file name for the succeeding file. *Srcfile* must be specified. The destination path name is assumed to be relative to the destination login directory if there is no leading /. In either case, the target directory must be mode 777, or if the file already exists, the file must be writable by others. By default, files are delivered to directory **rje** under the destination login directory. **Rje** must have been previously created in mode 777 for everything to work. The name of the destination file is ordinarily the same as the last component of the original file. When the standard input is sent, the destination file name is normally taken to be **pipe.end**. If − is used, the standard input is taken.

−!cmd *Cmd* is sent to the remote machine for execution. A file name or − can be used as standard input to the command. If no file is specified, **/dev/null** is used.

EXAMPLES

Assuming XXAAA, XXBBB and XXCCC are machines on the NSC network, then:

To send files *file1*, *file2*, and *file3* to XXAAA (assuming the source and destination logins are the same):

```
nusend −d XXAAA file1 file2 file3
```

To send file **cprog.c** to login name **dave** on XXBBB and to get confirmation mail returned:

nusend −d XXBBB −m −u dave cprog.c

To send file **myfile** to XXCCC and rename it to **yourfile** (assuming the source and destination logins are the same):

nusend −d XXCCC −f yourfile myfile

To send file **a.out** from XXAAA to login name **debbie** on XXBBB via remote execution:

nusend −d XXAAA .RS −!´nusend −d XXBBB −u debbie `logdir debbie`/a.out´

FILES

/etc/passwd	account number for NSC job
/usr/nsc/jobs/C*	job queue area
/usr/nsc/rvchan	table of known destinations
/usr/nsc/log/usendlog	usage log

SEE ALSO

mail(1), nscstat(1C).

NAME

pcldaemon − PCL link monitor

SYNOPSIS

/usr/lib/pcldaemon

DESCRIPTION

Pcldaemon monitors the *pcl*(4) control channel, servicing requests as they arrive. Requests are transmitted via *net*(1C).

FILES

/dev/pcl/*?*[0-7] PCL channel interfaces for system *?*.
/dev/pcl/ctrl PCL control channel.
/usr/adm/pcllog activity log.

SEE ALSO

net(1C), pcl(4).

DIAGNOSTICS

cannot open pcl control channel
Another *pcldaemon* is running.

WARNINGS

Running *pcldaemon* may present security hazards. A super-user may *net*(1C) to any system on the PCL bus that is running *pcldaemon* and execute any command on that system.

NAME

prerr − print a trace of detected NSC adapter errors

SYNOPSIS

prerr [**−f** file]

DESCRIPTION

Prerr prints a trace of detected NSC adapter errors during the NSC adapter to adapter protocol operation. NSC protocol error logging is enabled and disabled by the *nscmon*(1M) command. The contents of *file* (default *nscerr)* is printed as the tracing information.

Each log entry is printed sequentially from the time of log initiation. The format of a typical entry is:

time id status errno errcod

where each of the fields are defined as follows:

time The relative time from log initiation that this event occurred.

id The event identifier. At the time of this writing, the recognized entries are:

RTSE	error while sending a request to send
WMSGE	error while sending a message
WADE	error while sending a data block
RACK	error while reading a remote ACK
RDATAE	error while reading a data block
RTST	timer expired waiting for a RTS ACK
ACKT	timer expired waiting for a data ACK
DATAT	timer expired waiting for a data block
RMSGE	error while reading a message
RADE	error while reading a data block

status the eight (8) bytes of adapter status are printed. These bytes are printed according to the returned adapter status function, *adptrst* (see *nsc*(4)).

errno the UNIX system returned error code (see *intro*(2)).

errcod the NSC network returned error code).

FILES

nscerr the default binary error log file

SEE ALSO

nscmon(1M), prnsc(1C).

DIAGNOSTICS

If the file has a bad format, the id entries will be printed as **????**.

NAME

prnsc − print a trace of the NSC protocol operation

SYNOPSIS

prnsc [**−f** file] [**−l** len]

DESCRIPTION

Prnsc will print a trace of the NSC adapter to adapter protocol operation. Tracing is enabled and disabled using the *nscmon*(1M) command. *Prnsc* recognizes the following options:

−f file Print a trace from the log information found in *file*. The default file name is *nsclog*.

−l len Set the length of the output device to *len* column positions. The default line size is 80 columns.

The print format for a particular trace observation is as follows:

time id arg1 arg2 arg3

where *time* is the relative time from the beginning of the trace that the event *id* occurred. *Arg1* is normally the address of the remote destination (printed in hex), *arg2* is normally the size of the transmission data block (printed in decimal), and *arg3* is normally a queue pointer (printed in octal).

FILES

nsclog default binary trace file

SEE ALSO

nscmon(1M), prerr(1C).

NAME

sadp − disk access profiler

SYNOPSIS

sadp [**−th**] [**−d** device[−drive]] s [n]

DESCRIPTION

Sadp reports disk access location and seek distance, in tabular or histogram form. It samples disk activity once every second during an interval of **s** seconds. This is done repeatedly if **n** is specified. Cylinder usage and disk distance are recorded in units of eight cylinders.

Valid values of *device* are **rp06, rm05,** and **disk.** *Drive* specifies the disk drives and it may be:

> a drive number in the range supported by *device;*
> two numbers separated by a minus (indicating an inclusive range); or
> a list of drive numbers separated by commas.

Up to eight disk drives may be reported. The **−d** option may be omitted, if only one *device* is present.

The **−t** flag causes the data to be reported in tabular form. The **−h** flag produces a histogram on the printer of the data. Default is **−t.**

EXAMPLE

The command:

```
sadp −d rp06 −0 900 4
```

will generate 4 tabular reports, each describing cylinder usage and seek distance of rp06 disk drive 0 during a 15 minute interval.

FILES

/dev/kmem

NAME

sag − system activity graph

SYNOPSIS

sag [options]

DESCRIPTION

Sag displays graphically the system activity data of *sar*(1). Command options permit display of any of the *sar* data singly, or in combination; as cross plots, or versus time. Simple arithmetic combinations of data may be specified. *Sag* invokes *sar* and finds the desired data by string-matching the data column header (run *sar* to see what's available).

These options are passed thru to *sar* :

−s *time* Select data later than *time* in the form hh:mm. Default is 08:00.

−e *time* Select data up to *time*. Default is 18:00.

−i *sec* Select data at intervals as close as possible to *sec* seconds.

−f *file* Use *file* as the data source for *sar*. Default is the current daily data file **/usr/adm/sa/sa***dd*.

Other options:

−T *term* Produce output suitable for terminal *term*. See *tplot*(1G) for known terminals. If *term* is **vpr**, output is processed by **vpr −p** and queued to a Versatec printer. Default for *term* is **$TERM**.

−x *spec*

−y *spec* x and y axis specifications with *spec* in the form "*name*[*op name*]...[*lo hi*]".

Name is either a string that will match a column header in the *sar* report, with an optional device name in square brackets, e. g.: **r+w/s[rp06−1]**, or an integer value. *Op* is + − * or / surrounded by blanks. Up to five names may be specified. Parentheses are not recognized. Contrary to custom, + and − have precedence over * and /. Evaluation is left to right. Thus A / A + B * 100 is evaluated (A/(A+B))*100, and A + B / C + D is (A+B)/(C+D). *Lo* and *hi* are optional numeric scale limits. If unspecified, they are deduced from the data.

A single *spec* is permitted for the x axis. If unspecified, time is used.

Up to 5 *spec*'s separated by ; may be given for **−y**. Enclose the **−x** and **−y** arguments in "" if blanks or \<new-line> are included. The **−y** default is:
−y "%usr 0 100; %usr + %sys 0 100; %usr + %sys + %wio 0 100"

EXAMPLES

To see today's CPU utilization:

```
sag
```

To see activity over 15 minutes of all rp06 disk drives:

```
sar −o tempfile 60 15
sag −f tempfile −y "r+w/s[rp06]"
```

FILES

/usr/adm/sa/sa*dd* daily data file for day *dd*.

SEE ALSO

sar(1), tplot(1G), sar(8).

NAME

sar − system activity reporter

SYNOPSIS

sar [**−ubdycwaqvm**] [**−o** file] t [n]

sar [**−ubdycwaqvm**] [**−s** time] [**−e** time] [**−i** sec] [**−f** file]

DESCRIPTION

Sar, in the first instance, samples cumulative activity counters in the operating system at *n* intervals of *t* seconds. If the **−o** option is specified, it saves the samples in *file* in binary format. The default value of *n* is 1. In the second instance, with no sampling interval specified, **sar** extracts data from a previously recorded *file,* either the one specified by **−f** option or by default, the standard system activity daily data file **/usr/adm/sa/sa***dd* for the current day *dd.* The starting and ending times of the data report can be restricted via the **−s**, and **−e** time arguments in the form *hh*[:*mm*]. The interval of data report can be selected by **−i** option in seconds. Otherwise, all intervals found in the data file will be reported.

In either case, subsets of data to be printed are specified by option:

−u Report CPU % time in each of user mode, system mode, idle and waiting for I/O to complete, idle with no I/O activity.

−b Report buffer usage: block reads/second, block writes/second, logical reads/second, logical writes/second, cache hit ratios for read and for write, physical reads/second and physical writes/second.

−d Report activity for each disk drive or tape drive: percentage of time when disk or tape is busy, average disk or tape queue length while device is busy, I/O requests/second, blocks transferred/second, average wait time on queue in milliseconds, and average service time in milliseconds.

−y Report TTY device activity: characters/second in three queues; raw queue, canonical queue and output queue, receiving interrupts/second, transmitting interrupts/second and modem interrupts/second.

−c Report system calls: total system calls/second, read/second, write/second, fork/second, exec/second, bytes read/second by system call read and bytes written/second by system call write.

−w Report system swapping and switching activity: I/O swapins/second, blocks swapped in/second, I/O swapouts/second, blocks swapped out/second, and process switches/second.

−a Report file access activity: iget/second, namei/second, directory block read requests/second.

−q Report average queue size and the percentage of time occupied of each of the two queues; the run queue of processes in core and runnable and the swap queue of processes on swap device but marked runnable.

−v Report number of entries, size and overflows for each of the four system tables: text, process, inode and file. The numbers of entries are instantaneous sampled values, the sizes are values fixed at system generation time and the overflows are occurrences during the reporting interval.

−m Report message and semaphore activities: message primitives/second, and semaphore operations/second.

EXAMPLES

To see today's CPU activity so far:

```
sar
```

To watch CPU activity evolve for 10 minutes and save data:

```
sar −o temp 60 10
```

To later review disk and tape activity from that period:

sar −d −f temp

FILES

/usr/adm/sa/sa*dd* daily data file, where *dd* are digits representing the day of the month.

SEE ALSO

sag(1G) and sar(8).

NAME

sysfix — fix UNIX image for I/D space

SYNOPSIS

sysfix [**−v**] file1 file2 [name]

DESCRIPTION

Sysfix converts *file1* produced by the link editor *ld*(1) to a file *file2* suitable for booting on a CPU having separate I and D space. *File1* must have been created with the **−r** option of *ld* but not with the **−i** or **−n** options. If *name* is specified, then the first eight characters of *name* are stored in the character array **utsname** in *file2*.

To produce *file2* from *file1*, text addresses are relocated to virtual 020000, data addresses are relocated to virtual 0, and the positions of the text and the data segments in *file2* are interchanged.

The **−v** option is a verbose flag.

File2 is given the erroneous magic number 0407 (0437 from overlay system).

SEE ALSO

ld(1), uname(2), a.out(5).

DIAGNOSTICS

Sysfix complains when the data, text or both exceed allowable limits.

NAME

trouble − log a trouble report

SYNOPSIS

trouble

DESCRIPTION

The *trouble* command is a front end for the Piscataway Change Management Tracking System (CMTS). It is used to log trouble reports on, or request enhancements to UNIX. Trouble reports will be forwarded to Piscataway via *uucp*(1C), where they are transformed into Modification Requests (MRs).

The command will prompt for the following mandatory fields:

Name:	The originator's name (F. M. Last, F. Last, or First Last); (3 to 6 letter ID, if they are in the **names** file)
Location:	The external or internal mailing address
Phone:	The telephone number (aaaa, aaa-bbb-cccc, 8aaa-bbbb, or aaa-bbb-cccc xdddd)
Type:	**sw** (software), **hdw** (hardware), **doc** (documentation), **enh** (enhancement), **unk** (unknown)
System:	The product under discussion (usually **unix**)
Machine:	The CPU on which the trouble was found; **na** if not applicable
Release:	The product release number; **na** if not applicable
Severity:	**1** (out of commission, no circumvention), **2** (severity 1 if not fixed by due date (mo/da/yr)), **3** (needed), **4** (can be deferred)
Date required:	The due date for a severity **2** trouble report
Trouble Area:	The command or area in which the trouble was found
Abstract:	A one-line summary of the problem
Description:	The exact description of the problem; *ed*(1) is the entry mechanism, so an **a** (append) must first be typed. Once the description has been entered and edited, a **w** (write) followed by a **q** (quit) is required. Since *nroff* is used to format these reports, all examples should be enclosed in **.ES** and **.EE** *trouble*ˆ macros. In addition, any backslashes should be entered using the **\e** construct.

A response of **?** will cause the expected format of the response to be displayed.

Unless the description states otherwise, the trouble report may be selected to appear in the *MINI-SYSTEM NEWSLETTER*.

FILES

/usr/lib/trouble/tr.a	archived trouble reports
/usr/lib/trouble/instruct	instructions
/usr/lib/trouble/trsh	trouble report shell
/usr/lib/trouble/trxmit	re-transmission shell
/usr/lib/trouble/names	letter ID data base

SEE ALSO

uucp(1C), trouble(8).

NAME

vpmc − compiler for the virtual protocol machine

SYNOPSIS

vpmc [**−m**] [**−r**] [**−c**] [**−x**] [**−s** sfile] [**−l** lfile] [**−i** ifile] [**−o** ofile] file

DESCRIPTION

Vpmc is the compiler for a language that is used to describe communications link protocols. The output of *vpmc* is a load module for the virtual protocol machine (VPM), which is a software construct for implementing communications link protocols (e.g., BISYNC) on the DEC KMC11-B microprocessor. VPM is implemented by an interpreter in the KMC which cooperates with a driver in the UNIX host computer to transfer data over a communications link in accordance with a specified link protocol. UNIX user processes transfer data to or from a remote terminal or computer system through VPM using normal UNIX *open*, *read*, *write*, and *close* operations. The VPM program in the KMC provides error control and flow control using the conventions specified in the protocol.

The language accepted by *vpmc* is essentially a subset of C; the implementation of *vpmc* uses the RATFOR preprocessor (*ratfor*(1)) as a front end; this leads to a few minor differences, mostly syntactic.

There are two versions of the interpreter. The appropriate version for a particular application is selected by means of the **−i** option. The BISYNC version (**−i bisync**) supports half-duplex, character-oriented protocols such as the various forms of BISYNC. The HDLC version (**−i hdlc**) supports full-duplex, bit-oriented protocols such as HDLC. The communications primitives used with the BISYNC version are character-oriented and blocking; the primitives used with the HDLC version are frame-oriented and non-blocking.

Options

The meanings of the command-line options are:

−m	Use *m4*(1) instead of *cpp* as the macro preprocessor.
−r	Produce RATFOR output on the standard output and suppress the remaining compiler phases.
−c	Compile only (suppress the assembly and linking phases).
−x	Retain the intermediate files used for communication between passes.
−s *sfile*	Save the generated VPM assembly language on file *sfile*.
−l *lfile*	Produce a VPM assembly-language listing on file *lfile*.
−i *ifile*	Use the interpreter version specified by *ifile* (default **bisync**).
−o *ofile*	Write the executable object file on file *ofile* (default **a.out**).

These options may be given in any order.

Programs

Input to *vpmc* consists of a (possibly null) sequence of array declarations, followed by one or more function definitions. The first defined function is invoked (on command from the UNIX VPM driver) to begin program execution.

Functions

A function definition has the following form:

> function *name*( )
> *statement_list*
> end

Function arguments (formal parameters) are not allowed. The effect of a function call with arguments can be obtained by invoking the function via a macro that first assigns the value of each argument to a global variable reserved for that purpose. See *EXAMPLES* below.

A *statement_list* is a (possibly null) sequence of labeled statements. A *labeled_statement* is a statement preceded by a (possibly null) sequence of labels. A *label* is either a name followed by a colon (:) or a decimal integer optionally followed by a colon.

The statements that make up a statement list must be separated by semicolons (;). (A semicolon at the end of a line can usually be omitted; refer to the description of RATFOR for details.) Null statements are allowed.

Statement Syntax

The following types of statements are allowed:

expression
lvalue=*expression*
lvalue+=*expression*
lvalue−=*expression*
lvalue |=*expression*
lvalue&=*expression*
lvalue^=*expression*
lvalue<<=*expression*
lvalue>>=*expression*
if(*expression*)*statement*
if(*expression*)*statement* else *statement*
while(*expression*)*statement*
for(*statement*; *expression*; *statement*)*statement*
repeat *statement*
repeat *statement* until *expression*
break
next
switch(*expression*){*case_list*}
return(*expression*)
return
goto *name*
goto *decimal_constant*
{*statement_list*}

repeat is equivalent to the **do** keyword in C; **next** is equivalent to **continue**.

A *case_list* is a sequence of statement lists, each of which is preceded by a label of the form:

case *constant*:

The label for the last *statement_list* in a *case_list* may be of the form:

default:

Unlike C, RATFOR supplies an automatic **break** preceding each new case label.

Expression Syntax

A *primary_expression* (abbreviated *primary*) is an lvalue or a constant. An *lvalue* is one of the following:

name
name[*constant*]

A *unary_expression* (abbreviated *unary*) is one of the following:

primary
name( )
system_call
++*lvalue*
−−*lvalue*

(*expression*)
!*unary*
~*unary*

The following types of expressions are allowed:

unary
unary + *primary*
unary − *primary*
unary | *primary*
unary & *primary*
unary &~ *primary*
unary ^ *primary*
unary << *primary*
unary >> *primary*
unary == *primary*
unary != *primary*
unary > *primary*
unary < *primary*
unary >= *primary*
unary <= *primary*

Note that the right operand of a binary operator can only be a constant, a name, or a name with a constant subscript.

System Calls

A VPM program interacts with a communications device and a driver in the host computer by means of system calls (primitives).

The following primitives are available only in the BISYNC version of the interpreter:

atoe(*primary*)
: Translate ASCII to EBCDIC. The returned value is the EBCDIC character that corresponds to the ASCII character represented by the value of the primary expression. The translation tables reflect the prejudices of a particular installation.

crc16(*primary*)
: The value of the primary expression is combined with the cyclic redundancy check-sum at the location passed by a previous **crcloc** system call. The CRC-16 polynomial ($x^{16}+x^{15}+x^2+1$) is used for the check-sum calculation.

crcloc(*name*)
: The two-byte array starting at the location specified by *name* is cleared. The address of the array is recorded as the location to be updated by subsequent **crc16** system calls.

etoa(*primary*)
: Translate EBCDIC to ASCII. The returned value is the ASCII character that corresponds to the EBCDIC character represented by the value of the primary expression. The translation tables reflect the prejudices of a particular installation.

get(*lvalue*)
: Get a byte from the current *transmit* buffer. The next available byte, if any, is copied into the location specified by *lvalue*. The returned value is zero if a byte was obtained, otherwise it is non-zero.

getrbuf(*name*)
: Get (open) a *receive* buffer. The returned value is zero if a buffer is available, otherwise it is non-zero. If a buffer is obtained, the buffer parameters are copied into the array specified by *name*. The array should be large enough to hold at least three bytes.

The meaning of the buffer parameters is driver-dependent. If a receive buffer has previously been opened via a **getrbuf** call but has not yet been closed via a call to **rtnrbuf**, that buffer is reinitialized and remains the current buffer.

getxbuf(*name*)
Get (open) a *transmit* buffer. The returned value is zero if a buffer is available, otherwise it is non-zero. If a buffer is obtained, the buffer parameters are copied into the array specified by *name*. The array should be large enough to hold at least three bytes. The meaning of the buffer parameters is driver-dependent. If a transmit buffer has previously been opened via a **getxbuf** call but has not yet been closed via a call to **rtnxbuf**, that buffer is reinitialized and remains the current buffer.

put(*primary*)
Put a byte into the current *receive* buffer. The value of the primary expression is inserted into the next available position, if any, in the current receive buffer. The returned value is zero if a byte was transferred, otherwise it is non-zero.

rcv(*lvalue*)
Receive a character. The process delays until a character is available in the input silo. The character is then moved to the location specified by *lvalue* and the process is reactivated.

rsom(*constant*)
Skip to the beginning of a new *receive* frame. The receiver hardware is cleared and the value of *constant* is stored as the receive sync character. This call is used to synchronize the local receiver and remote transmitter when the process is ready to accept a new receive frame.

rtnrbuf(*name*)
Return a *receive* buffer. The original values of the buffer parameters for the current receive buffer are replaced with values from the array specified by *name*. The current receive buffer is then released to the driver.

rtnxbuf(*name*)
Return a *transmit* buffer. The original values of the buffer parameters for the current transmit buffer are replaced with values from the array specified by *name*. The current transmit buffer is then released to the driver.

xeom(*constant*)
Transmit end-of-message. The value of the constant is transmitted, then the transmitter is shut down.

xmt(*primary*)
Transmit a character. The value of the primary expression is transmitted over the communications line. If the output silo is full, the process waits until there is room in the silo.

xsom(*constant*)
Transmit start-of-message. The transmitter is cleared, then the value of *constant* is transmitted six times. This call is used to synchronize the local transmitter and the remote receiver at the beginning of a frame.

The following primitives are available only with the HDLC version of the interpreter:

abtxfrm( )
The current transmission, if any, is aborted, if possible, by sending a frame-abort sequence (seven one bits, followed immediately by a terminating flag). This operation is not feasible with some hardware interfaces, in which case this primitive is a no-operation.

getxfrm(*primary*)
Get a transmit buffer. If the transmit-buffer queue is *not* empty, the buffer at the head of the queue is removed from the queue and attached to the sequence number specified by the value of the primary expression If the sequence number is greater than seven or the sequence number already has a buffer attached, the process is terminated in error. The returned value is zero if a buffer was obtained, otherwise non-zero.

norbuf()
Test for the availability of an empty receive buffer. The returned value is **true** (non-zero) if the queue of empty receive buffers is currently empty; otherwise the returned value is **false** (zero).

rcvfrm(*name*)
Get a completed receive frame. If the queue of completed receive frames is non-empty, the frame at the head of the queue is removed and becomes the current receive frame. If a frame is obtained, the first five bytes of the frame are copied into the array specified by *name*. The returned value is **true** (non-zero) if a frame was obtained; otherwise, it is **false** (zero). The rightmost four bits of the returned value indicate the frame length as follows: if the value of the rightmost four bits is equal to fifteen, the frame length is greater than or equal to 15; otherwise the frame length is equal to the value of the rightmost four bits. The frame length includes the two CRC bytes at the end of the frame and any control information at the beginning of the frame. Bytes following the first two bytes of the frame, but not including the two CRC bytes, are copied into a receive buffer, if one is available at the time the frame is received. Bit 020 of the returned value is zero if a receive buffer was available, otherwise non-zero. The values of the leftmost three bits of the returned value are currently unspecified. If a frame was obtained, the first five bytes of the frame are copied into the array specified by *name*. Frames with errors are discarded; a count is kept for each type of error. Frames may be discarded for any of the following reasons: (1) CRC error, (2) frame too short (less than four bytes), (3) frame too long (buffer size exceeded), or (4) no receive buffer available. If a frame with a buffer attached was previously obtained with **rcvfrm**, but the buffer has not been released to the driver with **rtnrfrm**, that buffer is returned to the queue of empty receive buffers. At most one receive frame with no buffer attached is retained by the interpreter; if a new frame arrives before the frame with no buffer attached has been obtained with **rcvfrm**, the new frame is discarded.

rtnrfrm( )
Return a receive buffer. The current receive buffer (the one obtained by the most recent **rcvfrm** primitive) is returned to the driver. If there is no current receive buffer, the process is terminated in error.

rsxmtq( )
Reset the transmit-buffer queue. The sequence number assignment is removed from all transmit buffers. If a transmission is currently in progress, the transmission is aborted, if possible.

rtnxfrm(*primary*)
Return a transmit buffer. The transmit buffer currently attached to the sequence number specified by the value of the primary expression is returned to the driver and the sequence number assignment is removed from that buffer. If the specified sequence number does not have a buffer attached, the process is terminated in error. Transmit buffers must be returned in the same sequence in which they were obtained, otherwise the process is terminated in error.

setctl(*name*,*primary*)
Specify transmit-control information. The number of bytes specified by the primary

expression are copied from the array specified by *name* and saved for use with subsequent **xmtfrm** or **xmtctl** primitives. If the transmitter is currently busy, the process is terminated in error.

xmtbusy( )
Test for transmitter busy. If a frame is currently being transmitted, the returned value is **true** (non-zero); otherwise the returned value is **false** (zero).

xmtctl( )
Transmit a control frame. If a transmission is not already in progress, a new transmission is initiated. The transmitted frame will contain the control information specified by the most recent **setctl** primitive, followed by a two-byte CRC. The CRC-CCITT polynomial ($x^{16}+x^{12}+x^5+1$) is used for the CRC calculation. The returned value is zero if a new transmission was initiated, otherwise non-zero.

xmtfrm(*primary*)
Transmit an information frame. If a transmission is not already in progress, a new transmission is initiated. The transmitted frame will contain the control information specified by the most recent **setctl** primitive, followed by the contents of the buffer which is currently attached to the sequence number specified by the value of the primary expression followed by a two-byte CRC. The CRC-CCITT polynomial ($x^{16}+x^{12}+x^5+1$) is used for the CRC calculation. The returned value is zero if a new transmission was initiated, otherwise non-zero. If the sequence number is greater than seven or the sequence number does not have a buffer attached, the process is terminated in error.

The following primitives are available with all versions of the interpreter:

dsrwait( )
Wait for modem-ready and then set modem-ready mode. The process delays until the modem-ready signal from the modem interface is asserted. If the modem-ready signal subsequently drops, the process is terminated. If **dsrwait** is never invoked, the modem-ready signal is ignored.

exit(*primary*)
Terminate execution. The process is halted and the value of the primary expression is passed to the driver.

getcmd(*name*)
Get a command from the driver. If a command has been received from the driver since the last call to **getcmd**, four bytes of command information are copied into the array specified by *name* and a value of **true** (non-zero) is returned. If no command is available, the returned value is **false** (zero).

pause( )
Return control to the dispatcher. This primitive informs the dispatcher that the virtual process may be suspended until the next occurrence of an event that might affect the state of the protocol for this line. Examples of such events are: (1) completion of an output transfer, (2) completion of an input transfer, (3) timer expiration, and (4) a buffer-in command from the driver. In a multi-line implementation, the **pause** primitive allows the process for a given line to give up control to allow the processor to service another line. In a single-line implementation this primitive has no effect.

snap(*name*)
Create a *snap* event record. Four bytes from the array specified by *name* are passed to the driver, which prefixes a time stamp and sequence number and creates a trace event record containing the data. If minor device 1 of the trace driver is currently open, the record is placed on the read queue for that device; otherwise the event record is

discarded. The information passed via the *snap* primitive can be displayed using the *vpmsnap* command (see *vpmstart*(1C)).

rtnrpt(*name*)
: Return a report to the driver. Four bytes from the array specified by *name* are transferred to the driver. The process delays until the transfer is complete.

testop(*primary*)
: Test for odd parity. The returned value is **true** (non-zero) if the value of the primary expression has odd parity, otherwise the returned value is **false** (zero).

timeout(*primary*)
: Schedule or cancel a timer interrupt. If the value of the primary expression is non-zero, the current values of the program counter and stack pointer are saved and a timer is loaded with the value of the primary expression. The system call then returns immediately with a value of **false** (zero) as the returned value. The timer is decremented each tenth of a second thereafter. If the timer is decremented to zero, the saved values of the program counter and stack pointer are restored and the system call returns with a value of **true** (non-zero). The effect of the timer interrupt is to return control to the code immediately following the **timeout** system call, at which point a non-zero return value indicates that the timer has expired. The **timeout** system call with a non-zero argument is normally written as the condition part of an **if** statement. A **timeout** system call with a zero argument value cancels all previous **timeout** requests, as does a **return** from the function in which the **timeout** system call was made. A **timeout** system call with a non-zero argument value overrides all previous **timeout** requests. The maximum permissible value for the argument is 255, which gives a timeout period of 25.5 seconds.

timer(*primary*)
: Start a timer or test for timer expiration. If the value of the primary expression is non-zero, a software timer is loaded with the value of the primary expression and a value of **true** (non-zero) is returned. The timer is decremented each tenth of a second thereafter until it reaches zero. If the value of the primary expression is zero, the returned value is the current value of the timer; this will be **true** (non-zero) if the value of the timer is currently non-zero, otherwise **false** (zero). The timer used by this primitive is different from the timer used by the **timeout** primitive.

trace(*primary*[,*primary*])
: The values of the two primary expressions and the current value of the script location counter are passed to the driver, which prefixes a sequence number and creates a trace event record containing the data. If minor device 0 of the trace driver is currently open, the record is placed on the read queue for that device; otherwise the event record is discarded. The information passed via the *trace* primitive can be displayed using the *vpmtrace* command (see *vpmstart*(1C)). If the second argument is omitted, a zero is used instead. The process delays until the values have been accepted by the host computer.

Constants

A *constant* is a decimal, octal, or hexadecimal integer, or a single character enclosed in single quotes. A token consisting of a string of digits is taken to be an octal integer if the first digit is a zero, otherwise the string is interpreted as a decimal integer. If a token begins with **0x** or **0X**, the remainder of the token is interpreted as a hexadecimal integer. The hexadecimal digits include **a** through **f** or, equivalently, **A** through **F**.

Variables

Variable names may be used without having been previously declared. All names are global. All values are treated as 8-bit unsigned integers.

Arrays of contiguous storage may be allocated using the **array** declaration:

array *name*[*constant*]

where *constant* is a decimal integer. Elements of arrays can be referenced using constant subscripts:

name[*constant*]

Indexing of arrays assumes that the first element has an index of zero.

Names

A *name* is a sequence of letters and digits; the first character must be a letter. Upper- and lower-case letters are considered to be distinct. Names longer than 31 characters are truncated to 31 characters. The underscore (_) may be used within a name to improve readability, but is discarded by RATFOR.

Preprocessor Commands

If the **−m** option is omitted, comments, macro definitions, and file inclusion statements are written as in C. Otherwise, the following rules apply:

1. If the character **#** appears in an input line, the remainder of the line is treated as a comment.

2. A statement of the form:

   define(*name*,*text*)

   causes every subsequent appearance of *name* to be replaced by *text*. The defining text includes everything after the comma up to the balancing right parenthesis; multi-line definitions are allowed. Macros may have arguments. Any occurrence of **$***n* within the replacement text for a macro will be replaced by the *n*th actual argument when the macro is invoked.

3. A statement of the form:

   include(*file*)

   inserts the contents of *file* in place of the **include** command. The contents of the included file is often a set of definitions.

EXAMPLES

These examples require the use of the **−m** option.

```
# The function defined below transmits a frame in transparent BISYNC.
# A transmit buffer must be obtained with getxbuf before the function
# is invoked.
#
# Define symbolic constants:
#
define(DLE,0x10)
define(ETB,0x26)
define(PAD,0xff)
define(STX,0x02)
define(SYNC,0x32)
#
# Define a macro with an argument:
#
define(xmtcrc,{crc16($1); xmt($1);})
#
# Declare an array:
#
```

```
array crc[2];
#
# Define the function:
#
function xmtblk( )
        crcloc(crc);
        xsom(SYNC);
        xmt(DLE);
        xmt(STX);
        while(get(byte)==0){
                if(byte == DLE)
                        xmt(DLE);
                xmtcrc(byte);
        }
        xmt(DLE);
        xmtcrc(ETB);
        xmt(crc[0]);
        xmt(crc[1]);
        xeom(PAD);
end
#
# The following example illustrates the use of macros to simulate a
# function call with arguments.
#
# The macro definition:
#
define(xmtctl,{c=$1;d=$2;xmtctl1( )})
#
# The function definition:
#
function xmtctl1( )
        xsom(SYNC);
        xmt(c);
        if(d!=0)
                xmt(d);
        xeom(PAD);
end
#
# Sample invocation:
#
function test( )
        xmtctl(DLE,0x70);
end
```

FILES

sas_temp*	temporaries
/tmp/sas_ta??	temporary
/tmp/sas_tb??	temporary
/usr/lib/vpm/pass*	compiler phases
/usr/lib/vpm/pl	compiler phase
/usr/lib/vpm/vratfor	compiler phase
/lib/cpp	preprocessor

/usr/bin/m4	preprocessor
/bin/kasb	KMC11-B assembler
/usr/lib/vpm/bisync/*	interpreter source for the BISYNC interpreter
/usr/lib/vpm/hdlc/*	interpreter source for the HDLC interpreter

SEE ALSO

m4(1), ratfor(1), vpmstart(1C), vpm(4).
C Reference Manual by D. M. Ritchie.
RATFOR—A Preprocessor for a Rational Fortran by B. W. Kernighan.
The M4 Macro Processor by B. W. Kernighan and D. M. Ritchie.
Software Tools by B. W. Kernighan and P. J. Plauger (pp. 28-30).

NAME

vpmsave, vpmsnap, vpmtrace, vpmfmt — save and print VPM event traces

SYNOPSIS

/etc/vpmsave mask device

/etc/vpmsnap mask

/etc/vpmtrace mask

/etc/vpmfmt

DESCRIPTION

Vpmsave opens the minor device of the trace driver specified by *device*, enables the channels specified by *mask* (octal), and then reads event records and writes them to its standard output (unformatted) until killed.

Vpmsnap opens **/dev/snap** (assumed to be minor device 1 of the trace driver), enables the channels specified by *mask* (octal). It then reads time-stamped event records and prints them until killed.

Vpmtrace opens **/dev/trace** (assumed to be minor device 0 of the trace driver), enables the channels specified by *mask* (octal). It then reads event records and prints them until killed.

Vpmfmt reads its standard input, which it assumes was generated by *vpmsave*, and prints it (formatted) to its standard output.

Note that:

    vpmsave mask device | vpmfmt

is equivalent to

    vpmtrace mask

where *device* is the name of minor device 0 of the trace driver. If *device* is the name of minor device 1 of the trace driver, it is almost equivalent to

    vpmsnap mask

the only difference being that the time stamps are not normalized to zero.

Vpmsave and *vpmfmt* when used as follows:

    vpmsave mask device > t &

    vpmfmt < t

lose fewer event records than *vpmtrace* and *vpmsnap*.

SEE ALSO

vpmc(1C), kmc(4), trace(4), vpm(4).

NAME

vpmset, vpmstart — connect/load VPM drivers and KMCs

SYNOPSIS

/etc/vpmset pdev idev kdev [lineno]

/etc/vpmstart device n [filen]

DESCRIPTION

The *vpmset* command provides a means for associating dynamically a VPM protocol driver minor device with a particular KMC11 microcomputer or a particular line on a KMS11 communications multiplexor. Each such connection requires the use of a separate VPM interface driver minor device as an intermediary. *Pdev* is the protocol minor device name; *idev* is the interface minor device name; *kdev* is the KMC or KMS minor device name; and *lineno* is the line number if a KMS was specified. Until these connections have been made, a user program cannot open the VPM protocol minor device for reading or writing. These connections can be changed, provided that the VPM protocol minor device is not open for reading or writing and the protocol associated with the interface driver is not running.

Example:

```
/etc/vpmset /dev/vpm2 /dev/vpb3 /dev/kmc1 4
```

Vpmstart writes *filen* (**a.out** by default) to the KMC11-B specified by *device* and initiates execution.

The argument *n* is a magic number that the KMC driver saves to identify the running program. This number is checked when the VPM driver is opened to provide some assurance that the program running in the KMC is the one expected. The magic number for VPM interpreters is 6. When *filen* has been loaded into to the KMC, its execution is begun. *Filen* may be any file executable by the KMC.

If *filen* was made using *vpmc*(1C), the VPM interpreter will be started by *vpmstart*. The VPM interpreter waits for a RUN command from the VPM interface driver before beginning execution of the protocol script. The RUN command is sent by the VPM interface driver when the VPM protocol minor device is opened.

SEE ALSO

kmc(4), vpm(4).

NAME

x25pvc, x25lnk — install, remove, or get status for a PVC or BX.25 link

SYNOPSIS

x25pvc options

x25lnk options

DESCRIPTION

X25pvc may be used to install or remove a BX.25 Permanent Virtual Circuit (PVC) on a specified BX.25 interface (*link*), or to display the status of a specified BX.25 minor device (*slot*). Exactly one of the following options must be used:

−i [**−S**] [**−R**] [**−N**] *slotname chno linkno*

Slotname is a path name that specifies a BX.25 minor device (slot). If the minor device is currently connected to some logical channel on some BX.25 interface (link), then first that minor device will be removed, if possible (see the **−r** option). If that minor device is available, it is connected to logical channel *chno* on link number *linkno*. *Chno* must be in the range of 1 to 4,095 and must not be currently in use for any other BX.25 minor device associated with that link. Exactly one of three options must be used to indicate the session-establishment protocol to be used. The **−S** option, which is the preferred option, indicates that session-layer connect/accept/disconnect qualified data messages are to be used. The **−R** option indicates that RESET in-order/out-of-order packets will be recognized. The **−N** option indicates that the "no protocol" session mode is used.

−r *slotname*

Remove BX.25 minor device *slotname*. The command will fail if the slot is open, if packets are waiting to be transmitted, or if there are unacknowledged packets outstanding.

−s *slotname*

Print the status of BX.25 minor device *slotname*. The information printed consists of *slotname*, the logical channel number, the link number, and the session-establishment option.

X25lnk is used to activate or deactivate a specified BX.25 link. Exactly one of the following options must be used:

−i [**−b**] [**−p** *pktsize*] *linkno vpmb kmc* [*lineno*]

Activate the BX.25 link that is specified by *linkno*. The **−b** option specifies that the link-level protocol will use Address B. The default is Address A. The **−p** option defines the packet size; if it is used, *pktsize* must be a number that is a power of 2 between 16 and 1,024 inclusive. The default packet size is 128. *Linkno* is the number of the BX.25 link to be installed. *Vpmb* is a path name that specifies the minor device number of the VPM interface driver that is to be used for the link. *Kmc* is a path name that specifies the minor device of the KMC driver and *lineno* specifies one of the eight lines (0-7) of a KMS11 multiple-line hardware interface. The default for *lineno* is 0. The **−i** option makes the necessary connections between data structures and starts the BX.25 protocol on the link.

−h *linkno*

Halt the link specified by *linkno*.

−s *linkno*

Print the status of the link specified by *linkno*. The information printed consists of the link number, the packet size used on that link, the minor device number of the VPM interface driver, the minor device number of the KMC driver, and the line number of the KMC or KMS.

SEE ALSO

nc(4), vpm(4), x25(4).
Operations Systems Network Protocol Specification: BX.25 Issue 2, Bell Laboratories.

NAME

maus — multiple-access-user-space (shared memory) operations

SYNOPSIS

```
#include <sys/fcntl.h>

int getmaus (path, oflag)
char *path;
int oflag;

int freemaus (mausdes)
int mausdes;

char *enabmaus (mausdes)
int mausdes;

int dismaus (saddr)
char *saddr;

char *switmaus (mausdes, saddr)
int mausdes;
char *saddr;
```

DESCRIPTION

MAUS (Multiple Access User Space) is a dedicated portion of physical memory that is subdivided into logical subsections. These subsections can be attached to the calling process's data segment or released from its data segment with the following calls.

Path points to a path name naming a special file that is one of the MAUS logical subsections. *Getmaus* opens a maus descriptor for the named file and sets the file status flag according to the value of *oflag*. *Oflag* is one of the following:

- **O_RDONLY** Open for reading only.
- **O_WRONLY** Open for writing only.
- **O_RDWR** Open for reading and writing.

No process may have more than eight (8) maus descriptors open simultaneously.

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]
- The named file does not exist. [ENOENT]
- The named file is not a maus special file. [EINVAL]
- A component of the path prefix denies search permission. [EACCES]
- *Oflag* permission is denied for the named file. [EACCES]
- Eight (8) maus descriptors are currently open. [EMFILE]
- The MAUS area associated with the special file does not exist. [ENXIO]
- *Path* points to an illegal address. [EFAULT]

Freemaus closes the maus descriptor specified by *mausdes*. Note that if a maus descriptor has been enabled (see *enabmaus* below) it may still be closed: a MAUS file remains attached to a process's data segment until a *dismaus* (see below) is used to free it.

Freemaus will fail if *mausdes* is not a valid open maus descriptor. [EBADF]

Enabmaus attaches the MAUS file associated with *mausdes* to the data segment of the calling process. The file is attached starting at the first available 8k-byte boundary address beyond the current break value (see *brk*(2)). Note that multiple *enabmaus* calls can be made with the same maus descriptor. Each call will attach the file at a different 8k-byte boundary address.

Enabmaus will fail and not attach the MAUS file if one or more of the following are true:

> *Mausdes* is not a valid open maus descriptor. [EBADF]
>
> No more 8k-byte boundary starting addresses are available. [ENOMEM]

Dismaus frees from the calling process's data segment the MAUS file that starts at the data segment address given by (*saddr* - (*saddr* modulus 8192)).

Dismaus will fail and not free the MAUS file if (*saddr* - (*saddr* modulus 8192)) is not the data segment starting address of a MAUS file. [EINVAL]

Switmaus attaches the MAUS file associated with *mausdes* to the data segment of the calling process. The file is attached starting at the address given by (*saddr* - (*saddr* modulus 8192)).

Switmaus will fail if one or more of the following are true:

> *Mausdes* is not a valid open maus descriptor. [EBADF]
>
> The value of (*saddr* - (*saddr* modulus 8192)) is not a legal 8k-byte boundary address above the current break value. [EINVAL]

RETURN VALUES

Upon successful completion, the return value is as follows:

> *Getmaus* returns a non-negative integer, namely a maus descriptor.
>
> *Freemaus* returns a value of 0.
>
> *Enabmaus* returns the data segment starting address of the attached MAUS file.
>
> *Dismaus* and *switmaus* return the maus descriptor previously associated with the data segment starting address given by (*saddr* - (*saddr* modulus 8192)) if one exists. Otherwise, a value of −2 is returned.

On other than successful completion, a value of −1 is returned with *errno* set to indicate the error.

NAME

msg − send and receive messages

SYNOPSIS

```
#include <sys/omsg.h>

int send (buf, size, topid, type)
char *buf;
int size, topid, type;

int sendw (buf, size, topid, type)
char *buf;
int size, topid, type;

int msgenab ()

int msgdisab ()

int recv (buf, size, mstructp, type)
char *buf;
struct mstruct *mstructp;
int size, type;

int recvw (buf, size, mstructp, type)
char *buf;
struct mstruct *mstructp;
int size, type;
```

DESCRIPTION

These calls facilitate inter-processes communication by allowing the calling processes to send messages to another process and receive messages from other processes. In this scheme processes are identified by their process IDs.

Send allows the calling process to send a message to another process. *Buf* points to a buffer that contains the message that is to be sent. *Size* is the size of the message in bytes. *Size* can range from 0 to the system-imposed maximum MAXMLEN. *Topid* is the process ID to which the message is to be sent. *Type* is a number ranging from 1 to 128. It can be used by the receiving process for message selection (see below) and is used by *msgdisab* (see below).

Send will fail and no message will be sent if one or more of the following are true:

> The process to which the message is to be sent does not exist or has not enabled message reception (see *msgenab* below). [ESRCH]
>
> The process to which the message is to be sent already has more messages outstanding (e.g., by count) than is allowed by the system-imposed maximum MAXMSG. [EAGAIN]
>
> *Size* is less than 0 or greater than the system-imposed maximum MAXMLEN. [EINVAL]
>
> *Type* is less than 1 or greater than 128. [EINVAL]
>
> *Buf* points to an illegal address. [EFAULT]

Sendw behaves just like *send*, except the EAGAIN error can never occur. Instead the calling process suspends execution until one of the following two events occurs:

> The process to which the message is to be sent reduces its number of outstanding messages below the system-imposed maximum MAXMSG. When this occurs the message is sent and the calling process resumes execution.
>
> The calling process receives a signal that is to be caught. In this case the message is not sent and the calling process resumes execution in the manner prescribed in *signal*(2).

Msgenab enables message reception. No messages may be sent to a process until it enables message reception through this call.

Msgenab will fail and not enable message reception if the total number of processes that have already enabled messages is equal to the system-imposed limit NMENAB. [EAGAIN]

Msgdisab disables message reception; messages may no longer be sent to the calling process. Depending on their *type*, all of the calling process's outstanding messages are either discarded or returned to the sending process. Those of type 64 to 128 are discarded. Those of type 1 to 63 are changed to type 128 and returned to the sending process if the following are true:

the sending process still exists
the sending process has message reception enabled
the sending process has less than MAXMSGs outstanding.

If the above conditions are not true, the messages are discarded.

Msgdisab will fail if message reception is not currently enabled for the calling process. [ESRCH]

Recv allows the calling process to receive the first outstanding message of the requested *type*. For this call *type* may equal 0, in which case the first outstanding message, regardless of its *type*, is returned. *Buf* points to the buffer into which the message is to be placed. *Size* is the desired size in bytes of the message to be received. The received message is truncated to *size* bytes if it is larger than *size*. The truncated part of the message is lost and no indication of the truncation is given to the calling process. *Mstructp* points to a buffer into which information about the received message is placed as follows:

```
struct mstruct {
        int     frompid;        /* processid of sender */
        int     mtype;          /* message type */
};
```

Recv will fail and no message will be received if one or more of the following are true:

Message reception has not been enabled. [ESRCH]

Size is less than 0. [EINVAL]

Type is less than 0 or greater than 128. [EINVAL]

No message of the desired *type* exists. [ENOMSG]

Buf points to an illegal address. [EFAULT]

Mstructp points to an illegal address. [EFAULT]

Recvw behaves just like *recv*, except when no message of the desired *type* exists (i.e., the ENOMSG error) the calling process suspends execution until one of the following two events occurs:

A message of the desired *type* is received.

The calling process receives a signal that is to be caught. In this case the calling process resumes execution without having received a message.

RETURN VALUES

If *sendw* or *recvw* return due to the receipt of a signal, a value of −1 is returned to the calling process and *errno* is set to EINTR. Upon successful completion, *send*, *sendw*, *msgenab* and *msgdisab* return a value of 0, and *recv*, and *recvw* return a value equal to the actual number of bytes received. Otherwise, a value of −1 is returned and *errno* is set to indicate the error.

BUGS

Some implementations do not allow the user to specify messages of *type* 128 to *recv* or *recvw*, in which case *errno* is set to EINVAL.

SEE ALSO

exec(2), exit(2), fork(2), signal(2).

NAME

plock – lock process, text, or data in memory

SYNOPSIS

#include <sys/lock.h>

int plock (op)
int op;

DESCRIPTION

Plock allows the calling process to lock its text segment (text lock), its data segment (data lock), or both its text and data segments (process lock) into memory. Locked segments are immune to all routine swapping. *Plock* also allows these segments to be unlocked. The effective user ID of the calling process must be super-user to use this call. *Op* specifies the following:

PROCLOCK – lock text & data segments into memory (process lock)

TXTLOCK – lock text segment into memory (text lock)

DATLOCK – lock data segment into memory (data lock)

UNLOCK – remove locks

Plock will fail and not perform the requested operation if one or more of the following are true:

The effective user ID of the calling process is not super-user. [EPERM]

Op is equal to **PROCLOCK** and a process lock, a text lock, or a data lock already exists on the calling process. [EINVAL]

Op is equal to **TXTLOCK** and a text lock, or a process lock already exists on the calling process. [EINVAL]

Op is equal to **DATLOCK** and a data lock, or a process lock already exists on the calling process. [EINVAL]

Op is equal to **UNLOCK** and no type of lock exists on the calling process. [EINVAL]

RETURN VALUE

Upon successful completion, a value of 0 is returned to the calling process. Otherwise, a value of −1 is returned and *errno* is set to indicate the error.

SEE ALSO

exec(2), exit(2), fork(2).

NAME

sema — semaphore operations

SYNOPSIS

```
#include <sys/sema.h>

int p (sd)
int v (sd)
int test (sd)
int setsem (sd,sval)
int rdsem (sd)

int post (sd)
int block (sd)
int setsem (sd,sval)
int rdsem (sd)

int lock (sd)
int unlock (sd)
int tlock (sd)
int noulk ( )
int setsem (sd,sval)
int rdsem (sd)

int sd, sval;
```

DESCRIPTION

Sd is a semaphore descriptor naming a semaphore. Semaphore descriptors are integers that range from **SEMAMIN** to **SEMAMAX**. An intermediate semaphore descriptor limit exists called **SEMASU**. **SEMASU** is a number between **SEMAMIN** and **SEMAMAX** and is used to reserve the semaphore descriptors between **SEMAMIN** and **SEMASU** for super-user processes. Each semaphore descriptor has an integer semaphore value associated with it.

There are three groups of semaphore operations defined as follows:

GROUP I:

p(sd) — Suspend execution of the calling process until the the value associated with *sd* becomes nonzero or the calling process receives a signal that is to be caught (see *signal*(2)). If the former occurs, decrement the value associated with *sd*.

v(sd) — Increment the value associated with *sd* (if the incremented value would be greater than or equal to **MAXSVAL**, instead set it equal to 1) and cause one process that suspended execution on *sd* with a *p* operation to resume execution.

test(sd) — If the value associated with *sd* is not equal to 0, decrement the value associated with *sd*.

setsem(sd, sval) — Set the value associated with *sd* to *sval*.

rdsem(sd) — Return the current value associated with *sd*.

The above semaphore operations will fail and not perform the requested operation if one or more of the following are true:

Sd is less than **SEMAMIN** or greater than **SEMAMAX**. [EINVAL]

Sd is less than or equal to **SEMASU** and the effective user ID of the calling process is not super-user. [EPERM]

The current value associated with *sd* is less than zero and the operation is *p*, *v*, or *test*. [EACCES]

GROUP II:

block(sd)
: Suspend execution of the calling process until the the value associated with *sd* changes from its current value or the calling process receives a signal that is to be caught (see *signal*(2)).

post(sd)
: Increment the value associated with *sd* (if the incremented value would be greater than or equal to **MAXSVAL**, instead set it equal to 1) and cause all processes that suspended execution on *sd* with a *block* operation to resume execution.

setsem(sd, sval)
: Set the value associated with *sd* to *sval*.

rdsem(sd)
: Return the current value associated with *sd*.

The above semaphore operations will fail and not perform the requested operation if one or more of the following are true:

Sd is less than **SEMAMIN** or greater than **SEMAMAX**. [EINVAL]

Sd is less than or equal to **SEMASU** and the effective user ID of the calling process is not super-user. [EPERM]

The current value associated with *sd* is less than zero and the operation is *block*, or *post*. [EACCES]

GROUP III:

lock(sd)
: Suspend execution of the calling process until the the value associated with *sd* becomes zero or the calling process receives a signal that is to be caught (see *signal*(2)). If the former occurs, set the value associated with *sd* to the negative of the calling process's process ID.

unlock(sd)
: Set the value associated with *sd* to zero and causes one process that suspended execution on *sd* with a *lock* operation to resume execution.

tlock(sd)
: If the value associated with *sd* is equal to zero, set the value equal to the negative of the calling process's process ID.

setsem(sd, sval)
: Set the value associated with *sd* to *sval*.

rdsem(sd)
: Return the current value associated with *sd*.

noulk()
: Set the calling process's no-sema-unlock flag (see *exit*(2) and *signal*(2)). With this flag set, the semaphores locked with the negative of the calling process's process ID will not be unlocked (i.e., not have their value set equal to 0) when the process terminates.

The above semaphore operations will fail and not perform the requested operation if one or more of the following are true:

Sd is less than **SEMAMIN** or greater than **SEMAMAX**. [EINVAL]

Sd is less than or equal to **SEMASU** and the effective user ID of the calling process is not super-user. [EPERM]

The current value associated with *sd* is greater than zero and the operation is *lock*, or *tlock*. [EINVAL]

The current value associated with *sd* is not the negative of the calling process's process ID, and the operation is *unlock*. [EINVAL]

WARNING

Mixing semaphore operations between groups gives undefined results and should not be done.

RETURN VALUE

If *p*, *block*, or *lock* return due to the receipt of a signal, a value of −1 is returned to the calling process and *errno* is set to EINTR. Upon successful completion, the return value is as follows:

> If the operation is *p*, the value associated with *sd* at the time the process resumes execution and before it is decremented is returned.
>
> If the operation is *lock* or *noulk* a value of 0 is returned.
>
> For all other operations, the value associated with *sd* at the time of the call is returned.

On other than successful completion, a value of −1 is returned and *errno* is set to indicate the error.

SEE ALSO

exec(2), exit(2), fork(2), signal(2).

NAME

drand48, lrand48, mrand48, srand48, seed48, lcong48 — uniform random number generators

SYNOPSIS

double drand48( )

long lrand48( )

long mrand48( )

srand48(**seedval**)
long seedval;

seed48(**seed16v**)
short seed16v[**3**];

lcong48(**param**)
short param[**7**]

DESCRIPTION

This is a family of interfaces to a good pseudo-random number generator based upon the linear congruential algorithm and 48 bit integer arithmetic.

Drand48 returns the next pseudo-random number as a non-negative valued double-precision binary fraction. The values returned are uniformly distributed over the interval [0, 1).

Lrand48 returns the next pseudo-random number as a non-negative valued long integer. The values returned are uniformly distributed over the interval [0, 2**31).

Mrand48 returns the next pseudo-random number as a signed long integer. The values returned are uniformly distributed over the interval [−2**31, 2**31).

Srand48 initializes the generator according to the argument *seedval*.

Seed48 initializes the generator according to the 48 bit integer represented in the array *seed16v*. Suitable defaults are supplied for the two initializing functions, or if no initialization is used.

Lcong48 permits the parameters of the linear congruential generator to be specified. The array of seven short integers passed to *lcong48* is interpreted as two 48 bit integers (the initial value, and the multiplier) and a short integer (the addend) used to construct the linear congruential algorithm.

SEE ALSO

rand(3C).

NAME

getcwd — get path-name of current working directory

SYNOPSIS

char *getcwd (buf, size)
char *buf;
int size;

DESCRIPTION

Getcwd returns a pointer to the current directory path-name. The value of *size* must be at least 2 greater than the length of the path-name to be returned.

If *buf* is given as zero, then *getcwd* will obtain *size* bytes of space using *malloc* (see malloc(3C)). In this case, the pointer returned by *getcwd* may be used as the argument in a subsequent call to *free*.

The routine is implemented by using *popen* to pipe the output of the *pwd* command into the specified string space.

EXAMPLE

```
char *cwd, *getcwd();
.
.
.
if((cwd = getcwd(0, 64)) == NULL) {
        perror("pwd");
        exit(1);
}
printf("%s\n", cwd);
```

SEE ALSO

malloc(3C), pclose(3S), popen(3S), string(3C).

DIAGNOSTICS

Returns NULL with *errno* set if *size* is not large enough, or if an error ocurrs in a lower level routine.

NAME

hsearch, hcreate, hdestroy — manage hash search tables

SYNOPSIS

```
typedef struct entry {
        char *key;
        char *data;
} ENTRY;

typedef enum {
        FIND,
        ENTER
} ACTION;

ENTRY *hsearch (item, action)
        ENTRY item;
        ACTION action;

int hcreate (nel)
        int nel;

void hdestroy ()
```

DESCRIPTION

Hsearch is a hash table search routine generalized from Knuth (6.4) Algorithm D. It returns a pointer into a hash table indicating the location at which an entry can be found. The first argument is a structure containing two pointers, one to the comparison key and one to any other data to be associated with that key. The second argument is a flag indicating the disposition of the entry if it cannot be found in the table. **ENTER** indicates that the item should be inserted in the table at an appropriate point. **FIND** indicates that no entry should be made. Unsuccessful resolution is indicated by the return of a NULL pointer.

The user must allocate sufficient space for the table by calling *hcreate* before *hsearch* is used. *Hcreate* takes one argument, an estimate of the maximum number of entries that the table will contain. This number may be adjusted upward by the algorithm in order to obtain certain mathematically favorable circumstances.

If the user wishes to reinitialize the search table, *hdestroy* can be called followed by another call to *hcreate*. *Hdestroy* takes no arguments and returns no results.

Hsearch uses *open addressing* with a *multiplicative* hash function. Its source code, however, has many other options available which the user may select by compiling the *hsearch* source with the following symbols defined to the preprocessor.

DIV use the *remainder modulo table size* as the hash function insteadof the multiplicative algorithm.

USCR use a User Supplied Comparison Routine for ascertaining table membership. The routine should be named *hcompar* and should behave in a mannner similar to *strcmp*.

CHAINED use a linked list to resolve collisions. If this option is selected, the following other options become available.

- **START** place new entries at the beginning of the linked list (default is at the end).
- **SORTUP** keep the linked list sorted by key in ascending order.
- **SORTDOWN** keep the linked list sorted by key in descending order.

Additionally, there are preprocessor flags for obtaining debugging printout (**−DDEBUG**) and for including a test driver in the calling routine (**−DDRIVER**). The source code should be consulted for further details.

SEE ALSO

bsearch(3C), lsearch(3C), qsort(3C), string(3C), tsearch(3C).

DIAGNOSTICS

Hsearch returns a NULL pointer if either the action is **FIND** and the item could not be found or the action is **ENTER** and the table is full.

Hcreate returns a NULL pointer if it cannot allocate sufficient space for the table.

NAME

strtol — convert ascii to long integer, specify base

SYNOPSIS

long strtol (str, ptr, base)
char *str;
char **ptr;
int base;

DESCRIPTION

Strtol returns in a long the integer represented by the character string *str* . The string is scanned up to the first character inconsistent with the base. Leading "white-space" characters are ignored.

If the value of *ptr* is not (char**)NULL, a pointer to the character terminating the scan is returned in **ptr* . If no integer can be formed, **ptr* is set to *str* , and zero is returned.

If *base* is positive (and not greater than 36), it is used as the base for conversion. After an optional leading sign, leading zeros are ignored, and "0x" or "0X" is ignored if *base* is 16.

If *base* is zero, the string itself determines the base thus: after an optional leading sign, a leading zero indicates octal conversion, and a leading "0x" or "0X" hexadecimal conversion. Otherwise, decimal conversion is used.

Truncation from long to int can, of course, take place upon assignment, or by an explicit cast.

SEE ALSO

atof(3C), scanf(3S).

BUGS

Overflow conditions are ignored.

NAME

tsearch, tdelete, twalk — manage binary search trees

SYNOPSIS

```
char *tsearch (key, rootp, compar)
char *key;
char **rootp;
int (*compar)();

char *tdelete (key, rootp, compar)
char *key;
char **rootp;
int (*compar)();

twalk (root, funct)
char *root;
int (*funct)();
```

DESCRIPTION

Tsearch is a binary tree search routine generalized from Knuth (6.2.2) Algorithm T. *Tsearch* returns a pointer into a tree indicating the location at which a datum may be found. If the item does not occur, it is added at an appropriate point in the tree. The first argument is a pointer to the datum to be located in the tree. The second argument is the address of a variable that points to the root of the tree. The variable should be set to 0 before the initial call to *tsearch*. The last argument 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, negative if the first argument is to be considered less than the second, and positive otherwise.

Tdelete deletes nodes from a binary search tree. It is generalized from Knuth (6.2.2) algorithm D. The arguments are the same as for *tsearch*. *Tdelete* returns a pointer to the parent of the deleted node.

Twalk traverses a binary search tree. It is called with two arguments. The first is the root of tree to be traversed. The second is the address of a function to be applied at each node. This function is, in turn, called with three arguments. The first is the address of the node being visited. The second is an enumeration data type defined with enum {preorder, postorder, endorder, leaf} depending on whether this is the first, second, or third time that the node has been visited, (during a depth-first, left-to-right traversal of the tree) or whether the node is a leaf. The third is the level of the node in the tree, with the root being level zero.

NOTES

The root of a tree may be changed by *tsearch* and *tdelete* during the course of execution.
Attempts to delete a non-existent node from a tree causes *tdelete* to return the address of the root of the tree and to leave the tree unaltered. Warning: the root argument to *twalk* is one level of indirection less than the rootp arguments to *tsearch* and *tdelete*.

BUGS

Awful things can happen if the calling function alters the pointer to the root.

DIAGNOSTICS

Zero is returned by *tsearch* if there is not enough space available to create a new node.
Zero is returned by *tsearch* and *tdelete* if rootp is zero on entry.

SEE ALSO

qsort(3C), bsearch(3C), lsearch(3C)

NAME

gd − general driver for moving-head disks

DESCRIPTION

Gd provides a general interface to the RM05, RP04, RP05, and RP06 moving head disks. In addition to the capability of mixing these mediums on the same controller, the driver will handle up to four controllers.

The driver reads the disk hardware drive-type register to determine access partitioning and other drive dependent attributes. Thus, the manual entries describing the above disk drives should be used for information regarding that particular drive.

The configuration name of *disk* should be specified when generating a system with *config*(1M).

FILES

/dev/rp*, /dev/rrp*

SEE ALSO

config(1M), hm(4), hp(4), rp(4), master(5).

NAME

hm − RM05 moving-head disk

DESCRIPTION

The files **rp0**, **...**, **rp7** refer to sections of the RM05 disk drive 0. The files **rp10**, **...**, **rp17** refer to drive 1, etc. This slicing allows the pack to be broken up into more manageable pieces.

The origin and size of the sections on each drive are as follows:

section	start	length
0	0	24320
1	40	476064
2	160	403104
3	280	330144
4	400	257184
5	520	184224
6	640	111264
7	0	500384

The start address is a cylinder address, with each cylinder containing 608 blocks. It is extremely unwise for all of these files to be present in one installation, since there is overlap in addresses and protection becomes a sticky matter.

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

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise *lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/rp*, /dev/rrp*

SEE ALSO

gd(4), gd(4), hp(4).

NAME

nc − network control

DESCRIPTION

The network control pseudo-device provides a means by which a privileged user program can install, remove, and get the status of a BX.25 Permanent Virtual Circuit (PVC), and start, stop, and get the status of a BX.25 link. Additional functions are planned for this driver when the virtual-call feature and additional layers of BX.25 are added to the UNIX BX.25 implementation. This driver supports *open*(2), *close*(2), and *ioctl*(2) system calls.

The network-control *ioctl* system call has the following form:

```
ioctl (fildes, cmd, arg)
```

where *fildes* is the file descriptor returned by the *open* of the **nc** device and *cmd* is one of the following constants (defined in **/usr/include/sys/nc.h**):

NCPVCI −Install a PVC. This command creates one end of a PVC by connecting a minor device of the X25 driver (slot) to a particular logical channel on a specified link. *Arg* is a pointer to a structure defined as follows:

```
struct pvc {
        unsigned short  slot;
        unsigned short  chno;
        unsigned short  link;
        unsigned short  options;
}
```

where *slot* is the minor device number of the slot to be used as the end point of the PVC, *chno* is the logical channel number to be used, and *link* is the number of the BX.25 link to be used. Links are numbered starting with 0. *Chno* must be in the range 1 to 4,095 and must not be in use currently on the link. The two low-order bits of *options* specify one of three possible session-establishment protocols:

PVC_SESS	session-layer open/close protocol
PVC_RST	reset in-order/out-of-order protocol
PVC_NONE	"no-protocol" session mode

These protocols are explained in *x25*(4).

The constants PVC_SESS, PVC_RST, and PVC_NONE are defined in **/usr/include/sys/x25u.h** .

If the link on which the PVC is installed is currently active (i.e., not in the halted state), the reset procedure will be initiated for the logical channel. When the reset procedure is completed, the PVC is ready for data transfer.

NCPVCR −Remove a PVC. If *arg* is the minor device number of a slot that is currently associated with a PVC and is not open, the local end of that PVC is removed, i.e., disconnected. The slot and logical channel number become available for reuse.

NCSTART −Start a link. *Arg* is a pointer to a structure defined as follows:

```
struct start {
        unsigned short  link;
        unsigned short  vpb;
        unsigned short  kmc;
        unsigned short  line;
        unsigned short  options;
        unsigned char   cmd[4];
}
```

where *link* is the number of a BX.25 link, *vpb* specifies a minor device number of the VPM interface driver, and *kmc* specifies a minor device number of the KMC driver. If a KMS is being used, *line* specifies which one of the eight synchronous lines (0 through 7) of the KMS that is to be used. If a single-line synchronous interface is being used, this argument must be zero. The four bytes of *cmd* are passed to the protocol script via the *vpmcmd* function.

This command connects the BX.25 link to a VPM interface driver minor device, and the VPM interface driver minor device to a synchronous line on the KMC. The restart procedure is then initiated for the link.

NCSTOP −Stop a BX.25 link. This command stops the link specified by *arg*. The link data structure is initialized. The link, the associated VPM interface driver, and the KMC synchronous line become available for reuse. While in the halted state packets received for this link are discarded.

NCPVCSTAT −Get the status of a PVC. This command gets the connections and status information for slot *slot* and places it in the *pvc* data structure pointed to by *arg*.

NCLINKSTAT −Get the status of a link. This command gets the connections and status information for link *link* and places it in the *start* data structure pointed to by *arg*.

SEE ALSO

x25lnk(1C), x25pvc(1C), vpm(4), x25(4).

NAME

nsc — NSC adapter interface specification

DESCRIPTION

The special files **nsc0, ..., nscn** refer to the control of a Network Systems Corporation (NSC) A-410 processor adapter. Each special file multiplexes across the transmission medium the full-duplex network operations of twenty (20) simultaneous opens.

Physical NSC network transmissions occur in two parts and in the following order: a 64-byte message block and an n-byte associated data block. The 64-byte message contains network control and routing information. The network message has the following structure:

```
struct nmsg
{
        char nm_adata;       /* associated data flag */
        char nm_trunk;       /* trunk selection */
        char nm_acode;       /* access code */
        char nm_vchan;       /* virtual channel */
        short nm_tonad;      /* "to" network address */
        short nm_frnad;      /* "from" network address */
        char nm_fnc;         /* protocol function */
        char nm_opcod;       /* adapter operation code */
        char nm_data[54];    /* control info */
};
```

The associated data block transfers large, variable-length data blocks. The NSC driver currently limits the associated data block size to 4096 bytes.

The driver issues the proper function code sequences to the A-410 adapter. The available function codes are defined as follows:

```
ATM      0005   /* Transmit Message */
ATD      0010   /* Transmit Data */
ATLSTD   0014   /* Transmit Last Data */
ATLM     0021   /* Transmit Local Message */
AIM      0045   /* Input Message */
AID      0050   /* Input Data   */
ASTAT    0101   /* Status       */
AMDP0    0140   /* Mark Down Port 0 */
AMDP1    0144   /* Mark Down Port 1 */
AMDP2    0150   /* Mark Down Port 2 */
AMDP3    0154   /* Mark Down Port 3 */
AMDR0    0160   /* Mark Down Port 0 & Reroute Msgs */
AMDR1    0164   /* Mark Down Port 1 & Reroute Msgs */
AMDR2    0170   /* Mark Down Port 2 & Reroute Msgs */
AMDR3    0174   /* Mark Down Port 3 & Reroute Msgs */
ARST     0241   /* Read Statistics */
ARCST    0245   /* Read & Clear Statistics */
ASTST    0300   /* Set Test     */
ASAL     0305   /* Set Address & Length */
AWA      0310   /* Write Adapter */
ARA      0314   /* Read Adapter         */
ACA      0340   /* Clear Adapter */
AEOP     0344   /* End Operation */
ACLWM    0346   /* Clear Wait For Message State */
```

```
AWAITM   0350   /* Wait Message        */
```

The driver always saves the network status bytes on failed or aborted transfer attempts. The user may retrieve the eight bytes of adapter status and perform the appropriate error recovery procedures. The eight adapter status bytes are defined by the following structure:

```
struct adptrst
{
     char st_afc;         /* last function code */
     char st_gsw;         /* general status word */
     char st_trkst;       /* trunk status */
     char st_trkrsp;      /* trunk response */
     char st_err;         /* adapter error code */
     char st_internal;    /* reserved for adapter use */
     char st_rtnrsp;      /* remote returned response */
     char st_spare[3];
};
```

After successful *open*(2) completion and before reading and writing to the network, the user must establish a virtual channel. Both the local and the remote machines must agree on this virtual channel to properly transfer data. A virtual channel is defined to be a destination network address and a virtual channel number. The driver enforces mutually exclusive virtual channels to properly route incoming network transmissions. There are currently 256 virtual channel numbers (0-255) supported. If the user specifies a zero destination address in the virtual channel, that process will receive the incoming transmissions from all remote nodes on the specified virtual channel number. If, however, another process establishes a virtual connection with the same virtual channel number but with a specific (non-zero) destination address, the specific connection will preempt the non-specific (zero destination address) connection and receive all incoming message transmissions from the particular remote node.

The NSC driver supports two modes of transfer: *data* and *control. Data* mode is the default mode. After virtual channel configuration, the user process performs simple reads and writes. The process does not need to know that it is transmitting across the NSC bus. Placement of the data into the message, the associated data block, or both is completely transparent. The user may optionally specify, however, that the first n-bytes of the transfer buffer be always placed into the NSC message (0 <= n <= 52). This may be particularly useful when transferring combinations of binary (i.e protocol headers) and character data (i.e. files) between heterogeneous machines (see the NSCIOASMB NSC *ioctl*(2) command). Reads and writes in data mode return the total number of user bytes transferred.

In *control* mode, the user has direct access to the NSC control information within the NSC message. The user process specifies into the driver pointers to static buffer areas, one for reads and one for writes. For write operations, the user builds a modified version of the NSC message in the write static buffer area. When the *write*(2) system call is made, the driver retrieves the data from this buffer to build the outgoing NSC message. The structure for the write static buffer area is:

```
struct nsctrl {
      short cn_flags;      /* associated data flag */
      short cn_tonad;      /* destination address */
      char  cn_fnc;        /* adapter function code */
      char  cn_opcode;     /* adapter operation code */
      char  cn_data[54];   /* unused data area */
};
```

If the user sets the associated data flag in **cn_flags** (NSCADATA), the buffer pointer in the *write*(2) call is sent in the associated data block. Otherwise, the driver transfers a message alone.

On *control* mode read operations, the driver places the entire 64-byte NSC message into the read static buffer area. If the message has an associated data block, the received data is placed into the buffer area specified in the *read*(2) system call.

For both *control* mode reads and writes, the NSC driver returns the number of bytes transferred in the associated data block. The driver returns a count of **one** (1) if a message alone was transmitted or received.

User processes configure parameters into the driver through *ioctl*(2). The driver recognizes the following *ioctl* requests:

NSCIODATA (struct datam *) argp

NSCIODATA places the virtual connection into *data* mode (the default mode for successful opens). The user specifies the number of data bytes always contained in the NSC message on reads and writes. *Argp* is a pointer to the following structure:

```
struct datam {
        short i_mbytes;   /* bytes in msg on reads */
        short o_mbytes;   /* bytes in msg on write */
};
```

NSCIOCTRL (struct ctrlm *) argp

NSCIOCTRL places the virtual connection into *control* mode. *Argp* is a pointer to the following structure:

```
struct ctrlm {
        struct nmsg   *i_mptr;   /* read static area */
        struct nsctrl *o_mptr;   /* write static area */
};
```

NSCIOVCHAN (struct nscvchan *) argp

NSCIOVCHAN configures the virtual channel for the specified open. *Argp* is a pointer to the following structure:

```
struct nscvchan {
        short v_tonad;  /* destination network addr */
        char  v_vchan;  /* virtual channel number */
        char  v_tmsk;   /* trunk mask */
        char  v_acode;  /* access code */
};
```

V_tonad and *v_vchan* configure the virtual channel for all subsequent reads and writes to the network. *V_tmsk* sets the trunk transfer mask for network transfers. For trunk transfers, each bit in the trunk mask is cleared in the trunk specification of the NSC network message *(nm_trunk)*. *V_acode* is the hardware access code place into the outgoing NSC network message *(nm_acode)*.

NSCIOESTAT (struct adptrst *) argp

NSCIOESTAT retrieves into the user buffer specified by *argp* the 8 bytes of adapter status from the last failed network operation. After the status bytes are retrieved, the buffer area in the driver is cleared.

NSCIOGETP (struct nscgetty *) argp

NSCIOGETP retrieves network connection parameters from the driver. *Argp* is a pointer to the following structure:

```
struct nscgetty {
        short g_flags;   /* connection flags */
        char  g_openm;   /* file open mode */
        char  g_vchan;   /* virtual channel number */
        short g_taddr;   /* destination network addr */
        char  g_acode;   /* access code */
        char  g_tmsk;    /* trunk mask */
        struct nscasmb g_asm;   /* assembly modes */
};
```

NSCIOASMB (struct nscasmb *) argp

NSCIOASMB selects the assembly/disassembly modes for network message and associated data block transmission and reception. Assembly modes are not necessary unless data is transferred between heterogeneous processors. Assembly mode 0 causes the hardware to swap incoming and outgoing bytes. This one is used primarily when transferring character data. Assembly mode 1 causes no swapping. This one is used primarily when transferring binary data. Assembly mode 1 is defaulted for the NSC message; assembly mode 0 is defaulted for the associated data block. *Argp* is a pointer to the following structure:

```
struct nscasmb {
        unsigned i_msg : 2;     /* input msg */
        unsigned i_data : 2;    /* input data */
        unsigned o_msg : 2;     /* output msg */
        unsigned o_data : 2;    /* output data */
};
```

NSCIOADDR (short *) argp

NSCIOADDR returns to the calling process the network address of the local node. The local address is generated from data retrieved from the adapter.

NSCIOBYE (char *) 0

NSCIOBYE disconnects the user process from the driver. This function performs the necessary cleanup to ensure proper driver operation.

NSCIOFCODE (struct nscfcode *) argp

NSCIOFCODE allows the user to issue any function to the adapter. The super-user is allowed to issue any function; others may issue only the status function (ASTAT). *Argp* is a pointer to the following structure:

```
struct nscfcode {
        char  *f_base;   /* buffer area */
        short f_cnt;     /* no. of bytes to xfer */
        short f_fcode;   /* func. code to issue */
};
```

NSCIOCANCEL (char *) 0

NSCIOCANCEL is a super-user only function. This command cancels the currently active adapter operation and returns an error to the effected user process. This command is used to clear hung processes.

NSCIOOFFLINE (char *) 0

NSCIOOFFLINE is a super-user only function. This command inhibits via software all function code issuance. Opens will occur normally, but all reads and writes will block. There is one exception: the super-user (by an NSCIOFCODE command) may issue any function.

NSCIOONLINE (char *) 0

NSCIOONLINE is a super-user only function. This command enables via software all function code issuance. This command is the converse of NSCIOOFFLINE.

SEE ALSO

nusend(1C), ioctl(2), read(2), write(2).

FILES

/usr/src/cmd/nusend/nscdef.h
/usr/include/sys/nsc.h

DIAGNOSTICS

Read(2) and *write*(2) both return the number of bytes successfully transferred. A −1 is returned on error.

BUGS

An error return does not necessarily mean that the network is down. Whenever an error occurs, adapter status should be retrieved from the driver. Most failed operations should be retried several times before giving up.

In *control* mode, 1 is returned if a message alone is transmitted or received.

NAME

pcl — parallel communications link interface

DESCRIPTION

Pcl provides the interface for up to two Digital Equipment Corporation PCL-11B network buses. Each bus can be used to interconnect up to 16 CPU's, providing relatively fast communication without individual point-to-point connections.

The interface permits simultaneous bi-directional communication between any machines on a single bus. Additionally, each such path is further subdivided into 8 independent channels. A single control interface is provided to reduce the line monitoring overhead for a daemon process.

The minor device number for a PCL channel is constructed as follows:

the low order 3 bits	specify a channel number.
the next 4 bits	specify one of 16 machines. (This number must be one less than the PCL Time Division Multiplexed bus number set in the hardware.)
the next bit	specifies one of 2 PCL's.

FILES

/dev/pcl/?[0-7]	normal machine and subchannel interface.
/dev/pcl/ctrl	control interface.

SEE ALSO

net(1C), pcldaemon(1C).

NAME

vpm, vpb — Virtual Protocol Machine Protocol and Interface Drivers

DESCRIPTION

This entry describes the *vpm* and *vpb* drivers and gives an introduction to the Virtual Protocol Machine (VPM).

VPM is a software package for implementing link-level protocols on the DEC KMC11 microcomputer in a high-level language. This is accomplished by a compiler that runs on UNIX and translates a high-level language description of a protocol into an intermediate language that is executed by an interpreter running in the KMC. VPM also provides a framework for implementing higher levels of protocols (levels 3 and above) as UNIX drivers.

The VPM software consists of the following components:

1. A compiler (*vpmc*(1C)) for the protocol description language; it runs on UNIX.
2. An interpreter that controls the overall operation of the KMC and interprets the protocol script.
3. Two UNIX drivers: A Protocol driver and an Interface driver.
4. *vpmstart*(1C): a UNIX command that copies a load module into the KMC and starts it.
5. *vpmset*(1C): a UNIX command that logically connects VPM minor devices and KMC synchronous links.
6. *vpmsnap*(1C): a UNIX command that prints a time-stamped event trace while the protocol is running.
7. *vpmtrace*(1C): a UNIX command that prints an event trace for debugging purposes while the protocol is running.
8. *vpmsave*(1C): a UNIX command that writes unformatted trace data to its standard output.
9. *vpmfmt*(1C): a UNIX command that formats the output of *vpmsave*.

The VPM protocol driver provides a simple user interface to a synchronous line controlled by a link-level protocol executed by the VPM interpreter in the KMC. It supports the following UNIX system calls: *open*, *read*, *write*, *close*, and *ioctl*. If higher levels of protocol are required, the VPM protocol driver may be modified or replaced. The VPM interface driver provides a common interface to a synchronous line controlled by a link-level protocol executed by the VPM interpreter. This common interface can be shared by several different protocol modules (see *x25*(4)).

Before a protocol driver minor device can be used, it must be logically connected to a VPM interface driver; the interface driver minor device must in turn be logically connected to a synchronous line of a KMC microprocesser or a KMS11 communication multiplexor. These corrections can be made by means of *ioctl* commands (see below). The command *vpmset*(1C) uses these *ioctl* commands to make these connections.

The VPM Interface Driver

The VPM interface driver provides a general purpose interface between level 3 protocols executing in the UNIX kernel and level 2 protocols being executed by the VPM interpreter in the KMC. This interface is used by the VPM Protocol driver as well as by other protocol drivers such as LEAP, X25, and ST.

The interface driver supports *open*, *close*, and *ioctl* systems calls. These calls are used to set up connections between the interface driver and a synchronous line on a KMC or KMS and to set interpreter options. The system *ioctl* call has the following form:

ioctl (fildes, cmd, arg)

Possible values for the *cmd* argument are:

VPMSDEV—Connect an interface driver minor device to a synchronous line on a KMC or KMS. Bits 6 and 7 of *arg* contain the minor device number of the KMC or KMS. Bits 0-2 of *arg* contain the line number (0-7) if a KMS is being used; for a single-line KMC they must be zero.

VPMGETM—Get the interpreter modes. The currently available modes are the normal mode and the X.25 mode. The normal mode is indicated by an *arg* of all zeros. In this mode, the entire information field of an I frame (see O. S. N. BX.25 Protocol Specification) is copied by the interpreter to the assigned buffer. The X.25 mode is indicated by a 1 in bit 0. In this mode, the VPM interpreter copies the first three bytes of the information field of level 2 I frames into the buffer descriptor of the buffer assigned to receive the frame. These three bytes are the X.25 level 3 header. The remaining bytes, if any, are copied to the buffer pointed to by the buffer descriptor.

VPMSETM—Set the interpreter modes. The modes specified by a 1 in the mask *arg* are set.

VPMCLRM—Clear interpreter modes. The modes specified by a 1 in the mask *arg* are cleared.

Operation of the Standard Protocol Driver

This driver allows UNIX user processes to transfer data to or from a remote terminal or computer system through VPM using normal *open*, *read*, *write*, and *close* operations. Flow control and error recovery are provided by the level 2 protocol executed by the interpreter in the KMC.

The VPM protocol driver *open* for reading-and-writing is exclusive; *open*s for reading-only or writing-only are not exclusive. The *open* checks that the correct interpreter is running in the KMC and then sends a command to the interpreter which causes it to start interpreting the protocol script. The driver then supplies one or more 512-byte receive buffers to the interpreter.

The *read* returns either the number of bytes requested or the number remaining in the current receive buffer, whichever is less; any bytes remaining in the current receive buffer are used to satisfy subsequent reads. The *write* copies the user data into 512-byte system buffers and passes them to the VPM interpreter in the KMC for transmission.

The *close* arranges for the return of system buffers and for a general cleanup when the last transmit buffer has been returned by the interpreter. It also stops the execution of the protocol script.

The VPM protocol driver supports the following *ioctl* commands:

VPMCMD—Send a command to the protocol script. The first four bytes of the array pointed to by *arg* are passed to the VPM interpreter, which saves them and passes them to the protocol script when it executes a *getcmd* primitive. Only the most recent command is kept by the VPM interpreter.

VPMERRS—Get and then reset the error counters maintained by the interpreter. The error counters are copied to the array pointed to by *arg*, which must be large enough to contain eight bytes. The interpreter's copy of the counters is then set to zero.

VPMRPT—Get the latest script report. When the protocol script executes a *rtnrpt* primitive, a four-byte report is passed from the protocol script to the VPM protocol driver. Only the most recent script report is kept by the driver. If there is a script report that has not previously been passed to a user via this *ioctl* command, that report is copied to the array pointed to by *arg* and a 1 is passed as the return value. If no script report is available, a 0 is passed as the return value.

VPMSDEV—Connect a protocol driver minor device to an interface driver minor device. *Arg* is the minor device number of the interface driver to be connected to this protocol driver. To invoke this *ioctl* command, the file status flag, O_NDELAY must be set.

The VPM Event Trace

The VPM drivers generate a number of event records to allow the activity of the drivers and protocol script to be monitored for debugging purposes. If a program such as *vpmtrace*(1C) or *vpmsave*(1C) has opened minor device 0 of the trace driver and has enabled the appropriate channels on that device, these event records are queued for reading; otherwise the event records are discarded by the trace driver. Event records associated with interface-driver minor device *n* are put on the read queue for minor device 0 of the trace driver with a channel number of *n*. Calls to the system functions *vpmopen*, *vpmread*, *vpmwrite*, and *vpmclose* generate event records identified respectively by **o**, **r**, **w**, and **c**. Calls to the *vpmc*(1C) primitive *trace*(*arg1*, *arg2*) cause the VPM interpreter to pass *arg1* and *arg2* along with the current value of the script location counter to the VPM driver, which generates an event record identified by a **T**. Each event record is structured as follows:

```
struct event {
        short   e_seqn;         /* sequence number */
        char    e_type;         /* record identifier */
        char    e_dev;          /* minor device number */
        short   e_short1;       /* data */
        short   e_short2;       /* data */
}
```

When the script terminates for any reason, the driver is notified and generates an event record identified by an **E**. This record also contains the minor device number, the script location counter, and a termination code; the termination codes are defined as follows:

0 Normal termination; the interpreter received a HALT command from the driver.
1 Undefined virtual-machine operation code.
2 Script program counter out of bounds.
3 Interpreter stack overflow or underflow.
4 Jump address not even.
5 UNIBUS error.
6 Transmit buffer has an odd address; or the driver tried to give the interpreter too many transmit buffers; or a *get* or *rtnxbuf* was executed while no transmit buffer was open, i.e., no *getxbuf* was executed prior to the *get* or *rtnxbuf*.
7 Receive buffer has an odd address; or the driver tried to give the interpreter too many receive buffers; or a *put* or *rtnrbuf* was executed while no receive buffer was open, i.e., no *getrbuf* was executed prior to the *get* or *rtnxbuf*.
8 The script executed an *exit* primitive.
9 A *crc16* was executed without a preceding *crcloc* execution.
10 The interpreter detected loss of the modem-ready signal at the modem interface.
11 Transmit-buffer sequence-number error.
12 Command error: an invalid command or an improper sequence of commands was received from the driver.
13 Not used.
14 Invalid transmit state (internal error).
15 Invalid receive state (internal error).
16 Not used.
17 *Xmtctl* or *setctl* attempted while transmitter was still busy.
18 Not used.
19 Same as error code 6.
20 Same as error code 7.
21 Script too large.
22 Used for debugging the interpreter.
23 The driver's OK-check has timed out.

NAME

X25 — BX.25 network interface

DESCRIPTION

The X25 driver provides multiplexed channels over one or more synchronous communications lines using the Bell System standard BX.25 Level 3 protocol. The current release supports permanent virtual circuits (PVCs) only; the call set-up features needed to support virtual calls have not yet been implemented. There is a separate and independent Level 3 interface for each communications line. Point-to-point connections between hosts are supported as well as connections via an X.25 network.

The X25 driver is implemented as a VPM protocol module (see *vpm*(4)). The X25 driver uses the VPM interface module to access communications lines controlled by KMC11-B microprocessors. Level 2 of BX.25, the link level, is implemented by a VPM protocol script in the KMC.

The special files **/dev/x25/s***?* refer to the minor devices of the X25 driver. Each such minor device, also referred to as a *slot*, can be connected by means of a *network control* device (see *nc*(4)) to an arbitrary logical channel (1-4095) on a specified X25 interface. When the other end of the logical channel has been connected in an analogous fashion, each slot so connected is the terminus of a *permanent virtual circuit*, which is a full-duplex connection over a BX.25 logical channel between a set of user processes on the local host and another set of user processes on a remote host. A logical channel is a connection which may be multiplexed with other channels over a physical link to a remote host or an X.25 network. Each X25 interface (also referred to as a *link*) must be connected via the network-control device to a particular KMC microprocessor or to a particular line on a KMS11 communications multiplexor.

A user process accesses a BX.25 minor device (slot) using *open*, *close*, *read*, *write*, and *ioctl* system calls.

There are several internal flags that are maintained by the X25 driver for each slot. The values of these flags can be read and in some cases modified by means of the *ioctl* system call (see below).

An *open* will fail and return the error EIO if the specified slot does not exist, if the slot is not currently connected to a logical channel on some link, or if the link to which the slot is connected is not currently active. The user may request the normal *open* options O_RDONLY, O_WRONLY, and O_RDWR. The user may also request that reads with no data available should not sleep, using the O_NDELAY *open* flag, or that the *open* is to be exclusive, using the O_EXCL *open* flag. If an exclusive *open* is requested and cannot be granted, the error EBUSY will be returned. A successful *open* will clear the *isreset* status bit (see the discussion of *ioctl* below).

An *open* may or may not block until the far end is also open, depending on the session-establishment protocol requested. There are three choices for the session-establishment protocol. The choice is made by means of the network-control device at the time the permanent virtual circuit is installed. The first mode, referred to as the "no-protocol" session mode, is for the *open* to return immediately. This puts the burden on the user program to determine whether the far end is actually open. The reset session mode, designed mainly for compatibility with certain non-UNIX implementations of BX.25, uses a RESET in-order packet to indicate to the far end that a slot has been opened and a RESET out-of-order packet to indicate to the far end that the slot has been closed. In the current implementation, the RESET in-order and RESET out-of-order packets are recognized when they are received, but are not transmitted (so-called "passive" mode). The third mode, which is the one recommended for most applications, uses session-layer *Connect/Accept* qualified data messages to indicate that a slot has been opened and session-layer *Disconnect* qualified data messages to indicate that a *close* has occurred. (This mode is not likely to be compatible with any other implementations of X.25 or BX.25). In the last two modes, an *open* will block until the indication that the far end is open has been received, unless the O_NDELAY *open* option was specified.

Regardless of the session mode in effect, data which is received while a slot is not open will be acknowledged and silently discarded. However, if the session-layer *open/close* protocol is selected, no data can be transmitted until both ends of the PVC have been opened.

The data specified by each *write* is transmitted as a single BX.25 message, possibly multi-packet. The user has the option of waiting for acknowledgement of the last packet of each message before the *write* returns; this feature is called *delivery confirmation* (see the discussion of *ioctl* below). Even if the O_NDELAY mode was requested, the user process will be put to sleep if the amount of data in the transmit queue for the slot exceeds some high-water mark; the process will be given a wake-up when the transmit queue has been drained to the low-water mark.

A user reads in record mode, which means that each *read* will return data from a single message only. If the count specified on the read request is not large enough to accommodate the entire message, the remainder of the message will be returned on subsequent reads. The message-continued flag (*messcont*) will be set when a partial read occurs; this flag will be cleared when the last byte of the message is finally read. If no data is available, the user process will be put to sleep, unless the O_NDELAY option was specified on the *open* or the equivalent mode was set via an *fcntl* system call. If O_NDELAY was specified and no data is available, the *read* will return zero bytes. If a partial message is available, the *read* will block until the complete message has been received or the count has been satisfied, even if the O_NDELAY mode was specified.

Zero-length messages will be sent and received as such (but see *BUGS* below).

If the *faropen* flag is not set, a *write* will return immediately with a count of zero. A *read* will return a zero-length record (indicating end-of-file) if *faropen* is not set and the receive queue is empty. If the end of a message is on the input queue, a *read* will *not* return a zero indicating end-of-file, regardless of the state of the slot. Note that this means that the flags returned by *ioctl*, indicating that the far end has closed or a RESET has occurred, may be set before they take effect on user reads.

Except as just noted, once a RESET has occurred all reads and writes will fail, returning the characteristic error EIO, until the *isreset* flag is cleared by an *ioctl* or a successful *open*.

If the *active* flag is not set (see the discussion of *ioctl* below), all calls to the BX.25 subsystem for that channel will fail with error ENXIO. This flag is set when the permanent virtual circuit is installed and the level 2 protocol has been started on a link and is only cleared when a link dies or a severe error causes the protocol to be stopped on the channel or link. Only the network control device can reset this flag.

Signals will cause reads and writes to return the amount actually read or written, unless it was zero bytes, in which case the error EINTR will be returned. However, if O_NDELAY was specified, the amount actually read or written will be returned, whether zero or greater. If a *write* is interrupted by a signal, the data already packetized will be transmitted as a BX.25 message; that is, a subsequent *write* will always begin a new message. The return value from the *write* will indicate the number of bytes actually queued for transmission.

When the last user closes, any unread data in the receive queue will be discarded. Data in the transmit queue will *not* be discarded, but will be transmitted normally. When the transmit queue has been drained, the session take-down protocol, if any, is then followed (either to send a session-layer *Disconnect* message or to mark the channel as being out of use, so that incoming packets can be discarded).

Several options and actions can be requested via the *ioctl* system call, which takes three arguments: *fildes*, *request*, and *arg*. Possible values for the *request* argument are:

X25GET Return the status of the channel. The *ushort* pointed to by *arg* will receive the values of the flag bits described in the X25SET command.

X25SET Set the flags and options for the slot or channel.

The structure of the flag word pointed to by *arg* is:

```
struct x25ctl {
        ushort
                c_delconf : 1;  /* delivery confirmation */
                c_excl : 1;     /* exclusive use */
                c_faropen : 1;  /* far end is open */
                c_messcont : 1; /* last message continued */
                c_sesspr : 2;   /* session protocol being used */
                c_isreset : 1;  /* channel reset not cleared */
                c_active : 1;   /* channel protocol functioning */
};
```

The above structure and commands may be defined by

#include <sys/x25u.h>

Only the *delconf* and *isreset* flags can be changed by *ioctl*. If a user attempts to set or clear any of the other flags, the system call will not fail, but that flag will not be changed.

The flag bits are further explained as follows:

c_delconf

While this bit is set, each *write* system call will block until the last packet of the corresponding message has been acknowledged; another writer, if one exists, will also be blocked until the previous writer's message is acknowledged. If this bit is not set, a user *write* can return immediately after the message has been completely packetized and queued for transmission, possibly allowing several out-going messages to be unacknowledged at the same time.

c_faropen

This flag reflects the X25 driver's estimate of the *open/close* status at the far end of the virtual channel. The flag is set when a session-layer *Connect/Accept* message or a RESET in-order packet is received from the far end. The flag is cleared when a session-layer *Disconnect/Reject* message or a RESET out-of-order packet is received from the far end. This flag cannot be changed by the user. If the "no-protocol" session mode is being used, this flag is set automatically when the near-end *open* occurs and cleared when the last *close* occurs.

c_messcont

This flag is set whenever a message was not completely consumed by the last user *read*. The flag is cleared when the last byte of the message is finally read. The user cannot clear this flag using *ioctl*.

c_sesspr

The value of this cell determines the session-establishment protocol in effect. The possible values for this cell are:

PVC_SESS	session-layer open/close protocol
PVC_RST	reset in-order/out-of-order protocol
PVC_NONE	"no-protocol" session mode

These constants are defined by

#include <sys/x25u.h>

This cell can only be read; it cannot be set, except via the network-control device.

c_isreset

This flag if set indicates that a RESET has occurred. The user may clear this flag by setting the corresponding flag bit to 1 in the value passed by X25SET. (The user may not set this flag). If the value for this flag passed by X25SET is 0, the internal value of this flag will *not* be changed. This flag is always cleared by a successful *open*.

c_active

If a channel appears to be dead due to a failure detected at either level 2 or level 3 of the protocol, this flag will be cleared by the system; all outstanding or subsequent *read* or *write* system calls will fail with error ENXIO until the flag has been cleared. This flag can only be set via the network-control device, by restarting the link and/or channel.

SEE ALSO

x25pvc(1C), fcntl(2), ioctl(2), open(2), read(2), write(2), nc(4), vpm(4).
Operations Systems Network Protocol Specification: BX.25 Issue 2.
Design Specification for the UNIX Implementation of BX.25 Level 3—Permanent Virtual Circuits by V. H. Rosenthal, E. W. Leggett, Jr., C. Mee III, and P. F. Long.

BUGS

The X25 driver is a new and relatively untried piece of software. It should therefore be approached with considerable caution. It is likely that some of the details of the user interface will change in the future, possibly in ways that will not be compatible with the current implementation.

The multiplicity of options for the *open/close* protocol reflects a lack of standardization and a certain amount of confusion. It is possible that a preferred *open/close* protocol for permanent virtual circuits will be established in the future; this may or may not require that user programs that use the *open/close* protocols described here will need to be changed. The session-layer connect/disconnect protocol and the "passive" reset in-order/out-of-order protocol are relatively transparent to the user. Programs which use one of these protocols are less likely to need to be changed if the *open/close* protocol is changed.

It is not clear that the treatment of the O_NDELAY flag is correct; this is an area that is particularly likely to change.

At present, there is no way to tell whether a return value of zero for a no-delay *read* is due to a zero-length message or to the lack of anything to read.

It would be dangerous to assume that zero-length messages will be propagated correctly through an X25 network or that they will be treated in a compatible manner by other implementations of BX.25.

There is no way to send an INTERRUPT packet. An INTERRUPT packet received from the far end will be discarded.

Caveat emptor.

NAME

master − master device information table

DESCRIPTION

This file is used by the *config*(1M) program to obtain device information that enables it to generate the configuration files. The file consists of 3 parts, each separated by a line with a dollar sign (**$**) in column 1. Part 1 contains device information; part 2 contains names of devices that have aliases; part 3 contains tunable parameter information. Any line with an asterisk (*) in column 1 is treated as a comment.

Part 1 contains lines consisting of at least 10 fields and at most 13 fields, with the fields delimited by tabs and/or blanks:

Field 1: device name (8 chars. maximum).
Field 2: interrupt vector size (decimal, in bytes).
Field 3: device mask (octal)—each "on" bit indicates that the handler exists:
- 000100 initialization handler
- 000040 power-failure handler
- 000020 open handler
- 000010 close handler
- 000004 read handler
- 000002 write handler
- 000001 ioctl handler.

Field 4: device type indicator (octal):
- 000400 VAX-11/780 massbus adapter
- 000200 allow only one of these devices
- 000100 suppress count field in the **conf.c** file
- 000040 suppress interrupt vector
- 000020 required device
- 000010 block device
- 000004 character device
- 000002 floating vector
- 000001 fixed vector.

Field 5: handler prefix (4 chars. maximum).
Field 6: device address size (decimal).
Field 7: major device number for block-type device.
Field 8: major device number for character-type device.
Field 9: maximum number of devices per controller (decimal).
Field 10: maximum bus request level (4 through 7).
Fields 11-13: optional configuration table structure declarations (8 chars. maximum).

Part 2 contains lines with 2 fields each:

Field 1: alias name of device (8 chars. maximum).
Field 2: reference name of device (8 chars. maximum; specified in part 1).

Part 3 contains lines with 2 or 3 fields each:

Field 1: parameter name (as it appears in description file; 20 chars. maximum)
Field 2: parameter name (as it appears in the **conf.c** file; 20 chars. maximum)
Field 3: default parameter value (20 chars. maximum; parameter specification is required if this field is omitted)

Devices that are not interrupt-driven have an interrupt vector size of zero. The 040 bit in Field 4 causes *config*(1M) to record the interrupt vector although the **low.s (univec.c** on the VAX-11/780) file will show no interrupt vector assignment at those locations (interrupts here will be treated as strays).

NAME

jotto − secret word game

SYNOPSIS

/usr/games/jotto [**−p**]

DESCRIPTION

Jotto is a word guessing game. You try to guess the computer's secret word before it guesses yours. Clues are obtained by entering probe words. For example, if the computer's secret word is "brown" and you probe with "stare", it will reply "1" indicating that there is one letter in common between your probe and the secret word. Double letters count only once unless they appear in both words. For example, if the hidden word is "igloo" and you probe with "broke", the computer will reply "1". But if you probe with "gloom", the computer will respond "4". All secret words and probe words should be non-proper English five-letter words. If the computer guesses your word exactly, please respond with "y". It will then tell you what its secret word was. The **−p** flag instructs the computer to report its progress in guessing your word.

BUGS

The dictionary contains some unusual words and lacks some common ones.

NAME

mosd − the OSDD adapter macro package for formatting documents

SYNOPSIS

osdd [options] [files]

mm −mosd [options] [files]

nroff −mm −mosd [options] [files]

nroff −cm −mosd [options] [files]

mmt −mosd [options] [files]

troff −mm −mosd [options] [files]

troff −cm −mosd [options] [files]

DESCRIPTION

The OSDD adapter macro package is a tool used in conjunction with the MM macro package to prepare Operations Systems Deliverable Documentation. Many of the OSDD Standards are different than the default format provided by MM. The OSDD adapter package sets the appropriate MM options for automatic production of the OSDD Standards. The OSDD adapter package also generates the correct OSDD page headers and footers, heading styles, Table of Contents format, etc.

OSDD document (input) files are prepared with the MM macros. Additional information which must be given at the beginning of the document file is specified by the following string definitions:

```
.ds H1 document-number
.ds H2 section-number
.ds H3 issue-number
.ds H4 date
.ds H5 rating
```

The *document-number* should be of the standard 10 character format. The words "Section" and "Issue" should not be included in the string definitions; they will be supplied automatically when the document is printed. For example:

```
.ds H1 OPA−1P135−01
.ds H2 4
.ds H3 2
```

automatically produces

```
OPA-1P135-01
Section 4
Issue 2
```

as the document page header. Quotation marks are not used in string definitions.

If certain information is not to be included in a page header, then the string is defined as null; e.g.,

```
.ds H2
```

means that there is no *section-number*.

The OSDD Standards require that the *Table of Contents* be numbered beginning with *Page 1*. By default, the first page of text will be numbered *Page 2*. If the *Table of Contents* has more than one page, for example *n*, then either **−rP***n+1* must be included as a command line option or **.nr P n** must be included in the document file. For example, if the *Table of Contents* is four pages then use **−rP5** on the command line or **.nr P 4** in the document file.

The OSDD Standards require that certain information such as the document *rating* appear on the *Document Index* or on the *Table of Contents* page if there is no index. By default, it is assumed that an index has been prepared separately. If there is no index, the following must be included in the document file:

.nr Di 0

This will ensure that the necessary information is included on the *Table of Contents* page.

The OSDD Standards require that all numbered figures be placed at the end of the document. The **.Fg** macro is used to produce full page figures. This macro produces a blank page with the appropriate header, footer, and figure caption. Insertion of the actual figure on the page is a manual operation. The macro usage is

.Fg page-count "figure caption"

where *page-count* is the number of pages required for a multi-page figure (default 1 page).

The *Table of Contents* for OSDD documents (see Figure 4 in Section 4.1 of the OSDD Standards) is produced with:

```
.Tc
System Type
System Name
Document Type
.Td
```

The **.Tc/.Td** macros are used instead of the **.TC** macro from MM.

By default, the adapter package causes the **NOTICE** disclosure statement to be printed. The **.PM** macro may be used to suppress the **NOTICE** or to replace it with the **PRIVATE** disclosure statement as follows:

.PM	none printed
.PM P	PRIVATE printed
.PM N	NOTICE printed (default)

The **.P** macro is used for paragraphs. The **Np** register is set automatically to indicate the paragraph numbering style. It is very important that the **.P** macro be used correctly. All paragraphs (including those immediately following a **.H** macro) must use a **.P** macro. Unless there is a **.P** macro, there will not be a number generated for the paragraph. Similarly, the **.P** macro should not be used for text which is not a paragraph. The **.SP** macro may be appropriate for these cases, e.g., for "paragraphs" within a list item.

The page header format is produced automatically in accordance with the OSDD Standards. The OSDD Adapter macro package uses the **.TP** macro for this purpose. Therefore the **.TP** macro normally available in MM is not available for users.

Figure captions are produced by the **.Fg** macro using the **.BS/.BE** macros. Thus the **.BS/.BE** macros are also not available for users.

The OSDD Standards require that all numbered figures be placed at the end of the document. The **.Fg** macro cannot be used within the document unless the final **.Fg** in a series of figures is followed by a **.SK** macro to force out the last figure page.

FILES

/usr/lib/tmac/tmac.osd

SEE ALSO

mm(1), mmt(1), troff(1), mm(7).
MM—Memorandum Macros by D. W. Smith and J. R. Mashey.
Operations Systems Deliverable Documentation Standards, June 1980.

NAME

mptx — the macro package for formatting a permuted index

SYNOPSIS

nroff −mptx [options] [files]

troff −mptx [options] [files]

DESCRIPTION

This package provides a definition for the **.xx** macro used for formatting a permuted index as produced by *ptx*(1). This package does not provide any other formatting capabilities such as headers and footers. If these or other capabilities are required, the *mptx* macro package may be used in conjuction with the *MM* macro package. In this case, the **−mptx** option must be invoked *after* the **−mm** call. For example:

nroff −cm −mptx file

or

mm −mptx file

FILES

/usr/lib/tmac/tmac.ptx	pointer to the non-compacted version of the package
/usr/lib/macros/ptx	non-compacted version of the package

SEE ALSO

mm(1), ptx(1), troff(1), mm(7).

NAME

sar − system activity report package

DESCRIPTION

Sar provides UNIX system-wide measurements.

The operating system contains a number of counters that are incremented as various system actions occur. These includes CPU utilization counters, buffer usage counters, disk and tape I/O activity counters, TTY device activity counters, switching and system-call counters, file-access counters, and queue activity counters. Shell procedures are provided by the system activity package to write system activity parameters periodically on a binary file and to generate a daily system activity report.

The shell program sa1 [t n] is used to collect and store data in binary file **/usr/adm/sa/sa***dd* where *dd* is the current day. The arguments **t** and **n** cause records to be written **n** times at an interval of **t** seconds, or once if omitted. Entries in **crontab** (see *cron*(1M)) such as

```
0 * * * 0,6 su sys -c "/usr/lib/sa/sa1"
0 8−17 * * 1−5 su sys −c "/usr/lib/sa/sa1 1200 3"
0 18−7 * * 1−5 su sys −c "/usr/lib/sa/sa1"
```

will produce records every 20 minutes during working hours and hourly otherwise.

Whenever the system is rebooted, the counters restart from zero. To mark this discontinuity in the data file, a special record is written, triggered by an entry in the **/etc/rc** file which is executed during the reboot:

```
su sys −c "/usr/lib/sa/sadc /usr/adm/sa/sa`date +%d`&"
```

A daily summary report may be generated by a **crontab** entry that invokes the shell procedure sa2 late in the day. Sa2 is equivalent to the *sar* command (see *sar(1)*), except its output is directed to file **/usr/adm/sa/sar***dd*. Any of the *sar* options may be used. The **crontab** entry

```
5 18 * * 1−5 su adm −c "/usr/lib/sa/sa2 −s 8:00 −e 18:01 −i 3600 −ubcwy-
daqvm"
```

will report important activities hourly during the working day.

The structure of the binary daily data file is:

```
struct sa {
        struct sysinfo si;          /* defined in /usr/include/sys/sysinfo.h */
        int  szinode;               /* current size of inode table */
        int  szfile;                /* current size of file table */
        int  sztext;                /* current size of text table */
        int  szproc;                /* current size of proc table */
        int  mszinode;              /* maximum size of inode table */
        int  mszfile;               /* maximum size of file table */
        int  msztext;               /* maximum size of text table */
        int  mszproc;               /* maximum size of proc table */
        long  inodeovf;             /* cumulative overflows of inode table since boot */
        long  inodeovf;             /* cumulative overflows of file table since boot */
        long  textovf;              /* cumulative overflows of text tablt since boot */
        long  procovf;              /* cumulative overflows of proc table since boot */
        time_t  ts;                 /* time stamp */
        long  devio[NDEVS][4];/* device unit information */
#define IO_OPS  0                   /* number of I/O requests since boot */
#define IO_BCNT  1                  /* number of blocks transferred since boot */
#define IO_ACT  2                   /* cumulative drive busy time in ticks */
#define IO_RESP  3                  /* cumulative I/O response time in ticks */
};
```

FILES

/usr/adm/sa/sa*dd*	daily data file
/usr/adm/sa/sar*dd*	daily report file
/tmp/sa.adrfl	address file

NAME

trouble — trouble reporting system

DESCRIPTION

The first line of the **/usr/lib/trouble/trsh** file must have the correct company code for your site; a local modification is necessary. The *trouble* command will not run until this change is made.

The official company codes are as follows:

at	AT&T
bl	Bell Labs
cb	Cincinnati Bell
cd	C&P of Washington
cm	C&P of Maryland
cv	C&P of Virginia
cw	C&P of West Virginia
lb	Illinois Bell
ll	AT&T Long Lines
mb	Michigan Bell
ms	Mountain States Telephone
nb	Indiana Bell
ne	New England Telephone
nj	New Jersey Bell
nv	Nevada Bell
nw	Northwestern Bell
ny	New York Telephone
ob	Ohio Bell
pa	Bell of Pennsylvania
pn	Pacific Northwest Bell
pt	Pacific Telephone & Telegraph
sb	Southern Bell
sc	South Central Bell
sn	Southern New England Telephone
sw	Southwestern Bell
we	Western Electric
wt	Wisconsin Telephone

All trouble reports are archived in **/usr/lib/trouble/tr.a**; this file should be checked weekly to ensure that it does not get too large. If it gets too large, it should be moved to **/usr/lib/trouble/otr.a**; after a week or so, the old archive can be thrown away.

The **trouble** login is intended to be used only for administering the *trouble* system. If *uucp* cannot deliver a trouble report, *mail*(1) will be returned to the **trouble** login. Any trouble reports not delivered may be retransmitted by using the *trxmit* command with the trouble report numbers as arguments. (Hence, the reason for saving the archive for a while.)

The **/usr/lib/trouble/names** file can be expanded to include the names of additional people at your site.

The per-line format of this file is as follows:

letter-IDs(3-6) location phone name(with appropriate blanks)

The above fields are separated by blanks and/or tabs. When the *letter-ID* is identified, the *name*, *location* and *phone* will be taken from this file (provided they have legal formats). Note that the *name* field is the only one that can have blanks.

FILES

/usr/lib/trouble/tr.a	archived trouble reports
/usr/lib/trouble/instruct	instructions
/usr/lib/trouble/trsh	trouble report shell
/usr/lib/trouble/trxmit	re-transmission shell
/usr/lib/trouble/names	letter ID data base

SEE ALSO

trouble(1), uucp(1C).

NAME

vaxops − VAX-11/780 console operations

DESCRIPTION

The procedures described here include the major operational sequences involved in running UNIX on the VAX-11/780 system. The following notation is used:

1. Special characters are enclosed in <> (e.g., **<ctl>** represents the "control" key, and **<cr>** stands for the "carriage return" key).
2. Items within { }s are mandatory substitutions.

DAILY PROCEDURES

Disk Boot

This procedure can be used only on a system with a floppy disk updated for use with UNIX. If the floppy disk has not been so updated, the sequences shown below under *UNIX Floppy Update* must be performed.

When the system is first turned on, the console prompt >>> is printed. If UNIX has been shut down, but not halted (see *Bringing the System Down*), the operator must type **<ctl>p** to get into console mode. After the prompt, type **H<cr>** to halt the system.

With the system halted, any of the console commands may be executed as described below under *Console Operation*.

To boot the stand-alone shell *(sash)* from the default disk drive, the operator types **B<cr>**. The following is an example of this operation as seen on the console, picking up after the **<ctl>p**:

```
>>>H<cr>
        HALTED AT nnnnnnnn

>>>B<cr>
        CPU HALTED
        INIT SEQ DONE
        HALT INST EXECUTED
        HALTED AT nnnnnnnn
        LOAD DONE, nnnnnnnn BYTES LOADED

$$
```

At this point two alternate boot sequences are provided in the console floppy
package.
®will boot from disk drive 0, and
®.fi

The
$$
prompt indicates that the stand-alone shell
(*sash*)
is ready to accept commands.
If it is desired to run stand-alone
fsck(1M)
(or other stand-alone functions), this is the time to do it.
The commands have the form
/stand/*program*
where
program
can be any name from a limited list of UNIX commands
found in the directory

/stand.
To perform a file system consistency check, type:

```
$$ /stand/fsck /dev/rp0
```

To bring up UNIX, the operator must type **unix<cr>**. The system will come up through **init 1** (see *init*(8)).

This is the appropriate time to do file system backups, and *fsck*(1M) should be executed if it was not executed in the stand-alone section of the boot. One must never operate the system with a defective file system.

After successful completion of *fsck*(1M) and setting the date and time (see *date*(1)), the operator can bring the system to multi-user operation by executing **init 2**.

Bringing the System Down

The shutdown procedure is designed to gracefully turn off all processes and bring the system back to single user state with all buffers flushed. To do this the operator can execute *shutdown*(1M) or the following sequence of commands:

```
killall
sync
init 1
fsck (optional)
```

The system may then be halted by typing the **<ctl>p** and **H<cr>** sequence.

System Dumps

After a system crash, the following procedure should be used to get a system dump on tape.

1. Mount a tape with write ring and bring it on-line.
2. Enter console mode with **<ctl>p**.
3. After the >>> prompt, halt the system with **H<cr>**.
4. Issue the following command sequence, each command followed by **<cr>**:

E R0/N:F	(*Examine R0 thru R15*)
E SP	(*Get the stack pointer for the next command*)
E/V @/N:3F	(*Examine virtual memory beginning at the address from the previous instruction, and continuing for the next 63 locations; i.e., examine the stack*)
ST 400	(*Start execution at 400, i.e., dump to tape*)

5. Before returning to UNIX, execute the stand-alone *fsck*(1M).

System Faults

On occasion, the UNIBUS or its devices fail in such a manner as to flood the console with error messages and suspend operations on UNIBUS devices. It may be possible under these conditions to bring the system down gracefully from an internal point-of-view, by inhibiting UNIBUS interrupts and running a normal shutdown. The following sequence can be executed:

```
<ctl>p
>>> H
>>> E 20006004       (Look at UBA control register)
>>> D * 1            (Clear the UBA)
>>> C                (Return to UNIX)
```

You should now be able to login as root and run a normal shutdown sequence. Reboot the system by normal means, ensuring *fsck*(1M) is performed.

INSTALLATION BOOT PROCEDURES

Tape Boot

The floppy disk delivered with the VAX-11/780 does not have tape-boot capability. The user must type in the following program to read the first record on tape drive 0. Type **<cr>** at the end of each input line:

```
>>> H
>>> U
>>> I

        INIT SEQ DONE

>>> D 20000 20008FD0
>>> D + D0502001
>>> D + 3204A001
>>> D + C003C08F
>>> D + A0D40424
>>> D + 8FD00C
>>> D + C0800000
>>> D + 8F320800
>>> D + 10A0FE00
>>> D + C007D0
>>> D + C039D004
>>> D + 400
>>> S 20000          (Start tape load)

        HALT INST EXECUTED
        HALTED AT 0002002F

>>> S 0              (Execute boot program loaded from tape)
```

From this point the loader initiates a question and answer sequence to control the remainder of the load process.

Disk Boot

The floppy disk delivered with the VAX-11/780 does not have UNIX disk-boot capability. The user must type in the following program to read the first block on disk drive 0. Type **<cr>** at the end of each line.

```
>>> H
>>> LINK                  (Save the following sequence on the floppy)
                          (The prompt should change to <<<)
<<< H                     Halt processor
<<< U                     Unjam the SBI
<<< I                     Initialize the processor
<<< D/I 11 20003800       Register initialization
```

<<<D R0 0	*Device type code*
<<<D R1 8	*NEXUS number of MBA*
<<<D R2 0	*drive number*
<<<D R3 0	*drive number*
<<<D R4 0	
<<<D R5 8	*Software boot flags*
<<<D FP 0	*Set "no machine check expected"*
<<<S 20003000	*Start rom program*
<<<WAIT DONE	
<<<E SP	*Show address of working memory +0x200*
<<<L VMB.EXE/S:@	*Load primary bootstrap*
<<<S @	*and start it*
<<< <ctl>C	*(Exit* **LINK** *mode)*
>>>	

You are now ready to boot UNIX. Each time it is necessary to boot (or reboot) UNIX, simply follow the sequence:

>>> P<cr>	*(Execute the commands saved in floppy link file; the console should echo each command in the file.)*
$$ unix<cr>	*(Load and execute* **/unix***)*

UNIX Floppy Update

To update the console floppy for UNIX operation, one must have brought UNIX up by one of the initial-load procedures described above. The following sequence can then be executed.

```
# cd /stand/conflp
# sh update
```

Update prints commentary during the update operation indicating the files that are being replaced or added. Finally, a new table of contents is printed and the available space is indicated.

CONSOLE OPERATION

The following is condensed from Chapter 2 of the *VAX-11/780 Hardware Handbook*, DEC, 1978.

The following are the standard console commands. The most abbreviated form is shown in parentheses.

<ctl>P — Causes console to exit Program I/O mode (talking to the VAX-11/780 program). This does *not* halt the VAX CPU.

<ctl>U — Deletes the current input line.

<del> — Deletes the previous character.

<ctl>C — Interrupts printout.

(HE)LP — Prints "help" file of which this is a part.

(E)XAMINE {address}
: Displays 8-digit hexadecimal address and its contents. See "help" file for qualifiers.

(D)EPOSIT {address} {data}
: Enters data to address.

(I)NITIALIZE — Initializes CPU.

(U)NJAM — Unjams the SBI.

(SH)OW — Displays console and CPU state.

(H)ALT — Halts execution of VAX CPU instructions.

(S)TART {address}
: Initializes CPU, enters address to PC, issues CONTINUE to CPU, and puts console into Program I/O mode.

(C)ONTINUE — Starts execution of VAX CPU instructions.

(SE)T (T)ERMINAL (P)ROGRAM
: Puts console into Program I/O mode.

@{file} — Causes the named floppy file to be printed and executed.

WARNINGS

Only **<ctl>p** can be executed from Program I/O mode. It *does not* stop the VAX CPU from running. Only **HALT** can be executed while the VAX CPU is running and not in Program I/O mode; therefore, the sequence to stop the VAX-11/780 while running UNIX (Program I/O mode) is:

```
<ctl>p
>>>H<cr>
```

FILES

/etc/shutdown
/stand/*

SEE ALSO

fsck(1M), shutdown(1M), filesave(8), init(8), tapeboot(8).

NAME

vcf − VAX-11/780 configuration verification program

DESCRIPTION

This program scans hardware registers and software configuration tables in order to verify device availability and addressing.

With the system halted, any of the console commands may be executed as described in *vaxops*(8) under *Console Operation*. The following is an example of execution of *vcf* as seen on the console, starting with a halted system:

```
>>>H<cr>
        HALTED AT nnnnnnnn

>>>B<cr>
        CPU HALTED
        INIT SEQ DONE
        HALT INST EXECUTED
        HALTED AT nnnnnnnn
        LOAD DONE, nnnnnnnn BYTES LOADED

$$
```

The **$$** prompt indicates that the stand-alone shell (*sash*) is ready to accept commands. To execute the configuration verification program, type:

$$ stand/vcf [*unix_a.out*]

Default for the *unix_a.out* file is **/unix**.

Vcf will scan the VAX machine registers looking for Memory, MASSBUS Adapters (MBAs), and UNIBUS Adapters (UBAs). For memory and MASSBUS devices, hardware status information is reported. Information on UNIBUS devices is obtained from configuration information in the UNIX executable, and an attempt is made to verify device address and interrupt vectors.

FILES

/unix (or other UNIX executable)
/stand/vcf

SEE ALSO

vaxops(8).