

NAME

cat — phototypesetter interface

DESCRIPTION

Cat provides the interface to a Graphic Systems C/A/T phototypesetter. Bytes written on the file specify font, size, and other control information as well as the characters to be flashed. The coding will not be described here.

Only one process may have this file open at a time. It is write-only.

FILES

/dev/cat

SEE ALSO

troff(1)

Phototypesetter interface specification

DC (4)

DC (4)

NAME

dc — DC-11 communications interface

DESCRIPTION

The discussion of typewriter I/O given in *ty*(4) applies to these devices.

The DC-11 typewriter interface operates at any of four speeds, independently settable for input and output. The speed is selected by the same encoding used by the *dh*(4) device (enumerated in *stty*(2)); impossible speed changes are ignored.

FILES

/dev/tty[01234567abcd] 113B Dataphones (not currently connected — see *dh*(4))

SEE ALSO

tty(4), *stty*(2), *dh*(4)

NAME

dh — DH-11 communications multiplexer

DESCRIPTION

Each line attached to the DH-11 communications multiplexer behaves as described in *tty(4)*. Input and output for each line may independently be set to run at any of 16 speeds; see *stty(2)* for the encoding.

FILES

/dev/tty[f-u]

SEE ALSO

tty(4), *stty(2)*

NAME

dn — DN-11 ACU interface

DESCRIPTION

The *dn?* files are write-only. The permissible codes are:

0-9	dial 0-9
:	dial *
;	dial #
—	4 second delay for second dial tone
<	end-of-number

The entire telephone number must be presented in a single *write* system call.

It is recommended that an end-of-number code be given even though not all ACU's actually require it.

FILES

/dev/dn0	connected to 801 with dp0
/dev/dn1	not currently connected
/dev/dn2	not currently connected

SEE ALSO

dp(4)

NAME

dp — DP-11 201 data-phone interface

DESCRIPTION

The *dp0* file is a 201 data-phone interface. *Read* and *write* calls to *dp0* are limited to a maximum of 512 bytes. Each write call is sent as a single record. Seven bits from each byte are written along with an eighth odd parity bit. The sync must be user supplied. Each read call returns characters received from a single record. Seven bits are returned unaltered; the eighth bit is set if the byte was not received in odd parity. A 10 second time out is set and a zero-byte record is returned if nothing is received in that time.

FILES

/dev/dp0

SEE ALSO

dn(4), gerts(3)

NAME

dz — DZ-11 communications multiplexer

DESCRIPTION

Each line attached to the DZ-11 communications multiplexer behaves as described in *tty(4)*.
Each line may be set to run at any of 16 speeds; see *stty(2)* for the encoding.

FILES

/dev/tty*

SEE ALSO

tty(4), *stty(2)*

NAME

hp — RH-11/RP04, RP05, RP06, RM03 moving-head disk

DESCRIPTION

Files with minor device numbers 0 through 7 refer to various portions of drive 0; minor devices 8 through 15 refer to drive 1, etc.

The origin and size of the pseudo-disks on each drive are as follows:

RP partitions

disk	start	length
0	0	9614
1	9614	8778
2	0	0
3	0	0
4	0	0
5	0	0
6	10450	330220
7	18392	322278

RM partitions

disk	start	length
0	0	9600
1	9600	8800
2	0	0
3	0	0
4	0	0
5	0	0
6	9600	122080
7	18400	113280

It is unwise for all of these files to be present in one installation, since there is overlap in addresses and protection becomes a sticky matter. Ordinarily devices 0 and 6 (or 0 and 7) will be used.

The block 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 files conventionally begin with an extra 'r.'

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 *seek* calls should specify a multiple of 512 bytes.

FILES

/dev/rp?, /dev/rp?

SEE ALSO

rp(4)

NAME

hs — RH11/RS03-RS04 fixed-head disk file

DESCRIPTION

The files *hs0* ... *hs7* refer to RJS03 disk drives 0 through 7. The files *hs8* ... *hs15* refer to RJS04 disk drives 0 through 7. The RJS03 drives are each 1024 blocks long and the RJS04 drives are 2048 blocks long.

The *hs* 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 HS files begin with *rhs*. The same minor device considerations hold for the raw interface as for the normal interface.

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 *seek* calls should specify a multiple of 512 bytes.

FILES

/dev/hs?, /dev/rhs?

NAME

ht — RH-11/TE-16 magtape interface

DESCRIPTION

The files *mt0*, ..., *mt15* refer to the DEC RH/TM/TE-16 magtape. The files *mt0*, ..., *mt7* are 800 bpi, and *mt8*, ..., *mt15* are 1600bpi. The files *mt0*, ..., *mt3* and *mt8*, ..., *mt11* are rewound when closed; the others are not. When a file open for writing is closed, a double end-of-file is written.

A standard tape consists of a series of 512 byte records terminated by an end-of-file. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time. Writing in very small units is inadvisable, however, because it tends to create monstrous record gaps.

The *mt* files discussed above are useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, and especially when long records are to be read or written, the 'raw' interface is appropriate. The associated files may be named *rmt0*, ..., *rmt15*, but the same minor-device considerations as for the regular files still apply.

Each *read* or *write* call reads or writes the next record on the tape. In the write case the record has the same length as the buffer given. During a read, the record size is passed back as the number of bytes read, provided it is no greater than the buffer size; if the record is long, an error is indicated. In raw tape I/O, the buffer must begin on a word boundary and the count must be even. Seeks are ignored. A zero count is returned when a tape mark is read; another read will fetch the first record of the next tape file.

FILES

/dev/mt?, /dev/rmt?

SEE ALSO

tp(1)

BUGS

The magtape system is supposed to be able to take 64 drives. Such addressing has never been tried. These bugs will be fixed when we get more experience with this device.

The driver is limited to four transports.

If any non-data error is encountered, it refuses to do anything more until closed. In raw I/O, there should be a way to perform forward and backward record and file spacing and to write an EOF mark explicitly.

NAME

kl — KL-11 or DL-11 asynchronous interface

DESCRIPTION

The discussion of terminal I/O given in *tty(4)* applies to these devices.

Since they run at a constant speed, attempts to change the speed via *stty(2)* are ignored.

The on-line console terminal is interfaced using a KL-11 or DL-11. By appropriate switch settings during a reboot, UNIX will come up as a single-user system with I/O on the console terminal.

FILES

/dev/console console

SEE ALSO

tty(4), *init(8)*

BUGS

Modem control for the DL-11E is not implemented.

NAME

lp — line printer

DESCRIPTION

Lp provides the interface to any of the standard DEC line printers. When it is opened or closed, a suitable number of page ejects is generated. Bytes written are printed.

An internal parameter within the driver determines whether or not the device is treated as having a 96- or 64-character set. In half-ASCII mode, lower case letters are turned into upper case and certain characters are escaped according to the following table:

{	+
}	+
,	+
	+
-	+

The driver correctly interprets carriage returns, backspaces, tabs, and form feeds. A sequence of newlines which extends over the end of a page is turned into a form feed. All lines are indented 8 characters. Lines longer than 80 characters are truncated. These numbers are parameters in the driver; another parameter allows indenting all printout if it is unpleasantly near the left margin.

FILES

/dev/lp

SEE ALSO

lpr(1)

BUGS

Half-ASCII mode, the indent and the maximum line length should be settable by a call analogous to *sty*(2).

NAME

mem, kmem — main memory

DESCRIPTION

Mem is a special file that is an image of the main memory of the computer. It may be used, for example, to examine (and even to patch) the system.

Byte addresses in *mem* are interpreted as physical memory addresses. References to non-existent locations cause errors to be returned.

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

The file *kmem* is the same as *mem* except that kernel virtual memory rather than physical memory is accessed.

On PDP11's, the I/O page begins at location 0160000 of *kmem* and per-process data for the current process begins at 0140000. On VAX11's the I/O space begins at physical address 20000000(16) and per-process data for the current process is at virtual 80020000(16).

FILES

/dev/mem, /dev/kmem

BUGS

On PDP11's and VAX11's, memory files are accessed one byte at a time, an inappropriate method for some device registers.

NULL (4)

NULL (4)

NAME

null — data sink

DESCRIPTION

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

NAME

pc — PC-11 paper tape reader/punch

DESCRIPTION

Ppt refers to the PC-11 paper tape reader or punch, depending on whether it is read or written.

When *ppt* is opened for writing, a 100-character leader is punched. Thereafter each byte written is punched on the tape. No editing of the characters is performed. When the file is closed, a 100-character trailer is punched.

When *ppt* is opened for reading, the process waits until tape is placed in the reader and the reader is on-line. Then requests to read cause the characters read to be passed back to the program, again without any editing. This means that several null leader characters will usually appear at the beginning of the file. Likewise several nulls are likely to appear at the end. End-of-file is generated when the tape runs out.

Seek calls for this file are meaningless.

FILES

/dev/ppt

BUGS

If both the reader and the punch are open simultaneously, the trailer is sometimes not punched. Sometimes the reader goes into a dead state in which it cannot be opened.

NAME

rf — RF11/RS11 fixed-head disk file

DESCRIPTION

This file refers to the concatenation of all RS-11 disks.

Each disk contains 1024 256-word blocks. The length of the combined RF file is $1024 \times (\text{minor} + 1)$ blocks. That is minor device zero is taken to be 1024 blocks long; minor device one is 2048, etc.

The *rf0* file accesses 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 name of the raw RF file is *rrf0*. The same minor device considerations hold for the raw interface as for the normal interface.

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 *seek* calls should specify a multiple of 512 bytes.

FILES

/dev/rf0, /dev/rrf0

BUGS

The 512-byte restrictions on the raw device are not physically necessary, but are still imposed.

NAME

rk — RK-11/RK03 or RK05 disk

DESCRIPTION

Rk? refers to an entire disk as a single sequentially-addressed file. Its 256-word blocks are numbered 0 to 4871. Minor device numbers are drive numbers on one controller.

The *rk* files discussed above 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 RK files begin with *rrk* and end with a number which selects the same disk as the corresponding *rk* 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 *seek* calls should specify a multiple of 512 bytes.

FILES

/dev/*rk?*, /dev/*rrk?*

BUGS

In raw I/O *read* and *write*(2) truncate file offsets to 512-byte block boundaries, and *write* scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, *read*, *write* and *lseek*(2) should always deal in 512-byte multiples.

NAME

rp — RP-11/RP03 moving-head disk

DESCRIPTION

The files *rp0* ... *rp7* refer to sections of RP disk drive 0. The files *rp8* ... *rp15* refer to drive 1 etc. This is done since the size of a full RP drive is 81200 blocks and internally the system is only capable of addressing 65536 blocks. Also since the disk is so large, this allows it to be broken up into more manageable pieces.

The origin and size of the pseudo-disks on each drive are as follows:

disk	start	length
0	0	40600
1	40600	40600
2	0	9200
3	72000	9200
4	0	65535
5	15600	65535
6-7	unassigned	

It is unwise for all of these files to be present in one installation, since there is overlap in addresses and protection becomes a sticky matter. Here is a suggestion for two useful configurations: If the root of the file system is on some other device and the RP used as a mounted device, then *rp0* and *rp1*, which divide the disk into two equal size portions, is a good idea. Other things being equal, it is advantageous to have two equal-sized portions since one can always be copied onto the other, which is occasionally useful.

If the RP is the only disk and has to contain the root and the swap area, the root can be put on *rp2* and a mountable file system on *rp5*. Then the swap space can be put in the unused blocks 9200 to 15600 of *rp0* (or, equivalently, *rp4*). This arrangement puts the root file system, the swap area, and the i-list of the mounted file system relatively near each other and thus tends to minimize head movement.

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 *seek* calls should specify a multiple of 512 bytes.

FILES

/dev/rp?, /dev/rrp?

SEE ALSO

hp(4)

TC (4)

TC (4)

NAME

tc — TC-11/TU56 DECtape

DESCRIPTION

The files *tap0* ... *tap7* refer to the TC-11/TU56 DECtape drives 0 to 7.

The 256-word blocks on a standard DECtape are numbered 0 to 577.

FILES

/dev/tap?

SEE ALSO

tp(1)

NAME

tm — TM-11/TU-10 magtape interface

DESCRIPTION

The files *mt0*, ..., *mt7* refer to the DEC TU10/TM11 magtape. When opened for reading or writing, the tape is rewound. When closed, it is rewound; if it was open for writing, an end-of-file is written first.

A standard tape consists of a series of 512 byte records terminated by an end-of-file. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time. Writing in very small units is inadvisable, however, because it tends to create monstrous record gaps.

The *mt* files discussed above are useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, and especially when long records are to be read or written, the 'raw' interface is appropriate. The associated files are named *rmt0*, ..., *rmt7*. Each *read* or *write* call reads or writes the next record on the tape. In the write case the record has the same length as the buffer given. During a read, the record size is passed back as the number of bytes read, provided it is no greater than the buffer size; if the record is long, an error is indicated. In raw tape I/O, the buffer must begin on a word boundary and the count must be even. Seeks are ignored. An error is returned when a tape mark is read, but another read will fetch the first record of the new tape file.

FILES

/dev/mt?, /dev/rmt?

SEE ALSO

tp(1)

BUGS

If any non-data error is encountered, it refuses to do anything more until closed. In raw I/O, there should be a way to perform forward and backward record and file spacing and to write an EOF mark.

NAME

tty — general terminal interface

DESCRIPTION

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

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

As for terminals in general: all of the low-speed asynchronous communications ports use the same general interface, no matter what hardware is involved. The remainder of this section discusses the common features of the interface and differences resulting from the particular device, KL-11 or DH-11, on which the interface is implemented.

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

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

Normally, terminal input is processed in units of lines. This means that a program attempting to read will be suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not however necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information. There are special modes, discussed below, that permit the program to read each character as typed without waiting for a full line.

During input, erase and kill processing is normally done. By default, the character '#' erases the last character typed, except that it will not erase beyond the beginning of a line or an EOT. By default, the character '@' kills the entire line up to the point where it was typed, but not beyond an EOT. Both these characters operate on a keystroke basis independently of any backspacing or tabbing that may have been done. Either '@' or '#' may be entered literally by preceding it by '\'; the erase or kill character remains, but the '\' disappears. These two characters may be changed to others.

When desired, all upper-case letters are mapped into the corresponding lower-case letter. The upper-case letter may be generated by preceding it by '\'. In addition, the following escape sequences are generated on output and accepted on input:

for	use
`	\`
	\
~	\~
{	\{

```
} \)
```

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

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

DEL (Rubout) is not passed to a program but generates an *interrupt* signal which is sent to all processes with the associated control terminal. Normally each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location. See *signal(2)*.

FS (Control-\ or control-shift-L) generates the *quit* signal. Its treatment is identical to the interrupt signal except that unless a receiving process has made other arrangements it will not only be terminated but a core image file will be generated.

DC3 (Control-S) delays all printing on the terminal until something is typed in. A second DC3 may be used to restart printing without causing any input.

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

When one or more characters are written, they are actually transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed by putting them in the output queue as they arrive. When a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold the program is resumed. Even parity is always generated on output. The EOT character is not transmitted (except in raw mode) to prevent terminals that respond to it from hanging up.

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

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

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

B0	0	(hang up dataphone)
B50	1	50 baud
B75	2	75 baud
B110	3	110 baud
B134	4	134.5 baud
B150	5	150 baud

B200	6	200 baud
B300	7	300 baud
B600	8	600 baud
B1200	9	1200 baud
B1800	10	1800 baud
B2400	11	2400 baud
B4800	12	4800 baud
B9600	13	9600 baud
EXTA	14	External A
EXTB	15	External B

In the current configuration, only 110, 150, 300 and 1200 baud are really supported on dial-up lines. Code conversion and line control required for IBM 2741's (134.5 baud) must be implemented by the user's program. The half-duplex line discipline required for the 202 dataset (1200 baud) is not supplied; full-duplex 212 datasets work fine.

The *sg_erase* and *sg_kill* fields of the argument structure specify the erase and kill characters respectively. (Defaults are # and @.)

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

ALLDELAY	0177400	Delay algorithm selection
BSDELAY	0100000	Select backspace delays (not implemented):
BS0	0	
BS1	0100000	
VTDELAY	0040000	Select form-feed and vertical-tab delays:
FF0	0	
FF1	0100000	
CRDELAY	0030000	Select carriage-return delays:
CR0	0	
CR1	0010000	
CR2	0020000	
CR3	0030000	
TBDELAY	0006000	Select tab delays:
TAB0	0	
TAB1	0001000	
TAB2	0004000	
XTABS	0006000	
NLDELAY	0001400	Select new-line delays:
NL0	0	
NL1	0000400	
NL2	0001000	
NL3	0001400	
EVENP	0000200	Even parity allowed on input (most terminals)
ODDP	0000100	Odd parity allowed on input
RAW	0000040	Raw mode: wake up on all characters, 8-bit interface
CRMOD	0000020	Map CR into LF; echo LF or CR as CR-LF
ECHO	0000010	Echo (full duplex)
LCASE	0000004	Map upper case to lower on input
CBREAK	0000002	Return each character as soon as typed
HUPCL	0000001	Unused

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 indicates no delay.

Backspace delays are currently ignored but might be used for Terminet 300's.

If a form-feed/vertical tab delay is specified, it lasts for about 2 seconds.

Carriage-return delay type 1 lasts about .08 seconds and is suitable for the Terminet 300. Delay type 2 lasts about .16 seconds and is suitable for the VT05 and the TI 700. Delay type 3 is unimplemented and is 0.

New-line delay type 1 is dependent on the current column and is tuned for Teletype model 37's. Type 2 is useful for the VT05 and is about .10 seconds. Type 3 is unimplemented and is 0.

Tab delay type 1 is dependent on the amount of movement and is tuned to the Teletype model 37. Type 3, called XTABS, is not a delay at all but causes tabs to be replaced by the appropriate number of spaces on output.

Characters with the wrong parity, as determined by bits 200 and 100, are ignored.

In raw mode, every character is passed immediately to the program without waiting until a full line has been typed. No erase or kill processing is done; the end-of-file indicator (EOT), the interrupt character (DEL) and the quit character (FS) are not treated specially. There are no delays and no echoing, and no replacement of one character for another; characters are a full 8 bits for both input and output (parity is up to the program).

Mode 020 causes input carriage returns to be turned into new-lines; input of either CR or LF causes LF-CR both to be echoed (used for GE TermiNet 300's and other terminals without the newline function).

CBREAK is a sort of half-cooked mode. Programs can read each character as soon as typed, instead of waiting for a full line, but quit and interrupt work, and output delays, case-translation, CRMOD, XTABS, ECHO, and parity work normally. On the other hand there is no erase or kill, and no special treatment of \ or EOT.

Several *ioctl* calls have the form:

```
#include <sgtty.h>
```

```
ioctl(fildes, code, arg)
```

```
struct sgttyb *arg;
```

The applicable codes are:

TIOCGETP

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

TIOCSETP

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

TIOCSETN

Set the parameters but do not delay of flush input.

With the following codes the *arg* is ignored.

TIOCEXCL

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

TIOCNXCL

Turns off "exclusive-use" mode.

TIOCHPCL

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

Speed cannot be changed on terminals attached to a KL-11; the UNIX console, whose special use is described in *boot*(8), is such a terminal. Other terminals, called */dev/tty**, are attached to DZ-11's.

FILES

/dev/tty
*/dev/tty**
/dev/console

SEE ALSO

getty(8), *stty* (1), *signal*(2), *ioctl*(2)

BUGS

Half-duplex terminals are not supported. On raw-mode output, parity should be transmitted as specified in the characters written.

NAME

vp — Versatec printer-plotter

DESCRIPTION

vp0 is the interface to a Versatec printer-plotter. Ordinarily bytes written on it are interpreted as ASCII characters and printed. As a printer, it writes 64 lines of 132 characters each on 11 by 8.5 inch paper. Only some of the ASCII control characters are interpreted.

NL performs the usual new-line function, i.e. spaces up the paper and resets to the left margin. It is ignored however following a CR which ends a non-empty line.

CR is ignored if the current line is empty but is otherwise like NL.

FF resets to the left margin and then to the top of the next page.

EOT resets to the left margin, advances 8 inches, and then performs a FF.

The *stty(2)* system call may be used to change the mode of the device. Only the first word of the 3-word argument vector is used. The bits mean:

0400	Enter simultaneous print/plot mode.
0200	Enter plot mode.
0100	Enter print mode (default on open).
040	Send remote terminate.
020	Send remote form-feed.
010	Send remote EOT.
04	Send remote clear.
02	Send remote reset.

On open a reset, clear, and form-feed are performed automatically. Notice that the mode bits are not encoded, so that it is required that exactly one be set. The *gtty(2)* call fetches the current mode.

In plot mode each byte is interpreted as 8 bits of which the high-order is plotted to the left; a '1' leaves a visible dot. A full line of dots is produced by 264 bytes; lines are terminated only by count or by a remote terminate function. There are 200 dots per inch both vertically and horizontally.

When simultaneous print-plot mode is entered exactly one line of characters, terminated by NL, CR, or the remote terminate function, should be written. Then the device enters plot mode and at least 20 lines of plotting bytes should be sent. As the line of characters (which is 20 dots high) is printed, the plotting bytes overlay the characters. Notice that it is impossible to print characters on baselines that differ by fewer than 20 dot-lines.

In print mode lines may be terminated either with an appropriate ASCII character or by using the remote terminate function.

FILES

/dev/vp0

SEE ALSO

opr(1), vpr(1)

DIAGNOSTICS

