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 (I), Graphic Systems specification (available on request)

BUGS

NAME

dc – DC-11 communications interface

DESCRIPTION

The discussion of typewriter I/O given in tty (IV) 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 (IV) device (enumerated in stty (II)); impossible speed changes are ignored.

FILES

/dev/tty[01234567abcd] 113B Dataphones (not currently connected– see dh (IV))

SEE ALSO

tty (IV), stty (II), dh (IV)

BUGS

NAME

dh – DH-11 communications multiplexer

DESCRIPTION

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

FILES

/dev/tty[f-u]

SEE ALSO

tty (IV), stty (II)

BUGS

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 (IV)

BUGS

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 (IV), gerts (III)

BUGS

-

NAME

hp – RH-11/RP04 moving-head disk

DESCRIPTION

The files *hp0 ... hp7* refer to sections of RP disk drive 0. The files *hp8 ... hp15* refer to drive 1 etc. This is done since the size of a full RP drive is 170,544 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	9614
1	18392	65535
2	48018	65535
3	149644	20900
4	0	40600
5	41800	40600
6	83600	40600
7	125400	40600

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.

The *hp* 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 *rhp* and end with a number which selects the same disk section as the corresponding *hp* 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/hp?, /dev/rhp?

BUGS

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'' inteface 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?

BUGS

NAME

ht – RH-11/TU-16 magtape interface

DESCRIPTION

The files *mt0, ..., mt7* refer to the DEC RH/TM/TU16 magtape. When opened for reading or writing, the tape is rewound. When closed, it is rewound; if it was open for writing, a double end-of-file is written first.

A standard tape consists of a series of 512 byte records terminated by a double 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 (I)

BUGS

Raw I/O doesnt work yet. 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.

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

kl – KL-11 or DL-11 asynchronous interface

DESCRIPTION

The discussion of typewriter I/O given in tty (IV) applies to these devices.

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

The on-line console typewriter 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 typewriter.

FILES

/dev/tty8 console

SEE ALSO

tty (IV), init (VIII)

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 (I)

BUGS

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

-

NAME

mem, kmem, null − core memory

DESCRIPTION

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

A memory address is an 18-bit quantity which is used directly as a UNIBUS address. 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. In particular, the I/O area of *kmem* is located beginning at 160000 (octal) rather than at 760000. The 1K region beginning at 140000 (octal) is the system's data for the current process.

The file *null* returns end-of-file on *read* and ignores *write.*

FILES

/dev/mem, /dev/kmem, /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×(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 RK03 disk as a single sequentially-addressed file. Its 256-word blocks are numbered 0 to 4871.

Drive numbers (minor devices) of eight and larger are treated specially. Drive 8+*x* is the *x*+1 way interleaving of devices rk0 to rk*x*. Thus blocks on rk10 are distributed alternately among rk0, rk1, and rk2.

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

Care should be taken in using the interleaved files. First, the same drive should not be accessed simultaneously using the ordinary name and as part of an interleaved file, because the same physical blocks have in effect two different names; this fools the system's buffering strategy. Second, the combined files cannot be used for swapping or raw I/O.

-

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?

BUGS

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 (I)

BUGS

-

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 (I)

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 typewriter interface

DESCRIPTION

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

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

As for typewriters 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; the KL, DC, and DH writeups (IV) describe peculiarities of the individual devices.

When a typewriter 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 typewriter file open in a process becomes the *control typewriter* for that process. The control typewriter plays a special role in handling quit or interrupt signals, as discussed below. The control typewriter is inherited by a child process during a *fork.*

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 which 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.

These special files have a number of modes which can be changed by use of the *stty* system call (II). When first opened, the interface mode is 300 baud; either parity accepted; 10 bits/character (one stop bit); and newline action character. Modes that can be changed by *stty* include the interface speed (if the hardware permits); acceptance of even parity, odd parity, or both; a raw mode in which all characters may be read one at a time; a carriage return (CR) mode in which CR is mapped into newline on input and either CR or line feed (LF) cause echoing of the sequence LF-CR; mapping of upper case letters into lower case; suppression of echoing; a variety of delays after function characters; and the printing of tabs as spaces. See *getty* (VIII) for the way that terminal speed and type are detected.

Normally, typewriter 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.

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
`	\´
|	\!
~	\^
{	\(
}	\)

In raw mode, the program reading is awakened on each character. No erase or kill processing is done; and the EOT, quit and interrupt characters are not treated specially. The input parity bit is passed back to the reader, but parity is still generated for output characters.

The ASCII EOT (control-D) character may be used to generate an end of file from a typewriter. 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. The EOT is passed back unchanged in raw mode.

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 typewriter as control typewriter. 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 which read a typewriter and test for end-of-file on their input can terminate appropriately when hung up on.

Two characters have a special meaning when typed. The ASCII DEL character (sometimes called 'rubout') is not passed to a program but generates an *interrupt* signal which is sent to all processes with the associated control typewriter. 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* (II).

The ASCII character FS 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. If you find it hard to type this character, try control-\ or control-shift-L.

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 which respond to it from hanging up.

FILES

/dev/tty

SEE ALSO

dc (IV), kl (IV), dh (IV), getty (VIII), stty (I, II), gtty (I, II), signal (II)

BUGS

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