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)

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?

SEE ALSO

tty(IV), stty(II)

NAME

dn − DN-11 ACU interface

DESCRIPTION

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

0-9	dial 0-9
:	dial *
;	dial #
−	3 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/dn?

SEE ALSO

dp(IV)

NAME

dp − DP-11, DU-11 synchronous line interface

DESCRIPTION

The *dp0* file is a data-set interface. *Read* and *write* calls to dp0 are unlimited, but this works best when restricted to less than 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 characters 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. An error is returned if data-set ready is not present.

FILES

/dev/dp0

SEE ALSO

dn(IV)

NAME

hp − RP04/RP05/RP06 moving-head disk

DESCRIPTION

The files *rp0 ... rp7* refer to sections of the RP04/RP05/RP06 disk drive 0. The files *rp10 ... rp17* refer to drive 1, etc. This is done since the size of of a full pack is over 100,000 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 sections on each drive are as follows:

section	start	length
0	0	11286
1	27	53504
2	155	53504
3	283	53504
4	27	65535
5	184	65535
6	341	29260
7	unassigned	

The start address is a cylinder address, with each cylinder containing 418 blocks. For the RP06 drives, this table in the system must be changed to allow full addressing. 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 *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*

NAME

hs − RS03/RS04 fixed-head disk

DESCRIPTION

The files *rs0* ... *rs7* refer to RS03 disk drives 0 through 7. The files *rs10* ... *rs17* refer to RS04 disk drives 0 through 7. The RS03 drives are each 1024 blocks long and the RS04 drives are 2048 blocks long.

The *rs* 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 *rrs*. 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/rs*, /dev/rrs*

NAME

ht − TU16 magtape interface

DESCRIPTION

The files *mt0, ..., mt15* refer to the Digital Equipment Corporation TU16 magnetic tape control and transports. The files *mt0, ..., mt7* are 800bpi, and the files *mt8, ..., mt15* are 1600bpi. The files *mt0, ..., mt3, mt8, ..., mt11* are designated normal-rewind on close, and the files *mt4, ..., mt7, mt12, ..., mt15* are no-rewind on close. When opened for reading or writing, the tape is assumed to be positioned as desired. When a file is closed, a double end-of-file (double tape mark) is written if the file was opened for writing. If the file was normal-rewind, the tape is rewound. If it is no-rewind and the file was open for writing, the tape is positioned before the second EOF just written. If the file was no-rewind and opened read-only, the tape is positioned after the EOF following the data just read. Once opened, reading is restricted to between the position when opened and the next EOF or the last write. The EOF is returned as a zero-length read. By judiciously choosing *mt* files, it is possible to read and write multi-file tapes.

A standard tape consists of several 512 byte records terminated by an EOF. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time (although very inadvisable).

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, ..., rmt15.* Each *read* or *write* call reads or writes the next record on the tape. In the write case the record has the same length as the buffer given. During a read, the record size is passed back as the number of bytes read, up to the buffer size specified. In raw tape I/O, the buffer must begin on a word boundary and the count must be even. Seeks are ignored. An EOF is returned as a zero-length read, with the tape positioned after the EOF, so that the next read will return the next record.

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. The driver is limited to four transports.

NAME

kl − KL-11 or DL-11 asynchronous interface

DESCRIPTION

The discussion of terminal 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 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/tty8console

SEE ALSO

tty(IV), init(VIII)

BUGS

Full modem control for the DL-11E is not implemented.

NAME

lp − line printer

DESCRIPTION

Lp provides the interface to any of the standard Digital Equipment Corporation 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 output if it is unpleasantly near the left margin.

FILES

/dev/lp

BUGS

In 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 a 22-bit quantity used to set up memory management to address the full memory space. 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. Especially since reads and writes are a byte at a time.

The file *kmem* is the same as *mem* except that the kernel virtual data address space 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

rje − DQS-11B interface for remote job entry

DESCRIPTION

The **rje** interface defines a special file that looks like a concatenation of Binary Synchronous Communication (BSC) text blocks. This file may be both written to and read from, but not simultaneously. Data transfer with the two-point BSC discipline is strictly half-duplex.

The device can be opened by only one process at a time. It is expected that a process that successfully opens the DQS will spawn separate subprocesses to handle reading and writing. However, no distinction is made among the several processes that may have the DQS open. For example, reads within a message, even from a single block, may be executed by several processes in sequence. The overriding constraint is that a complete message must be read from or written to the DQS before any transfer of data in the opposite direction can begin. A process that tries to write while the DQS is reading, or vice versa, will be put to sleep until the transfer of the currently active message has been completed.

A complete message consists of one or more text blocks. A message being written to the DQS is terminated by a write of zero bytes, which causes an EOT to be transmitted. A message being read from the DQS is terminated by the reception of an EOT (which is not passed on to the reader, but is registered as a read of zero bytes). By convention, an EOT follows each block which ends in an ETX.

The length of a text block cannot exceed 512 bytes, including the line prefix and appendix. These two sequences, which must be present in blocks being written and will be passed on in blocks read, are constructed from the control bytes SOH, STX, ETB, ETX, DLE. The DQS itself will supply leading SYN bytes and trailing block check and pad bytes. The interface examines only the last byte of each text block received and so is unaware of the presence of headings or transparent text. The selection and interpretation of these features is the user's responsibility.

Line control functions, such as the alternating affirmative responses (ACK0 and ACK1), are automatically interspersed with text blocks as required by the line discipline. The interface handles the initial line bid and the EOT reset at the end of a transmission. A 3-second time-out is also respected. The interface will send TTD's and respond WACK's if its buffers are not serviced fast enough. When receiving, expiration of the time-out will cause the interface to abort the active message by sending EOT. When transmitting, the failure to send a block successfully after seven tries will cause the interface to terminate the active message prematurely. Such aborts cannot be appealed.

Reads on the DQS will return bytes from a single text block. If one read does not exhaust a text block, successive reads will return additional bytes from the same block. A returned count of zero indicates the end of a message. Until the remote station bids for the line, all reads will return zero bytes. The error bit will never be set by the interface itself. The DQS must be read to the end of a message before it will accept writes.

Writes to the DQS must consist of a single, entire text block. A write that specifies a count of zero bytes defines the end of a message. The count returned by a write call must be checked. A count of zero for the first write of a new message indicates that it was not possible to acquire the line. Otherwise, the DQS should return exactly the count specified in the write call. However, the error bit is set when a line error requires that the message be aborted. Notification of the error is not punctual, because data blocks are buffered for transmission. A write of zero bytes must be issued, or an error must occur, before the DQS will accept reads.

An open returns with the error bit set if the DQS is already open or not ready. The DQS should be opened in mode 2 to allow both reading and writing.

The DQS interface steals a number of buffers from UNIX (currently two) for the duration of each message. This number is specified at system generation time and may be tuned to influence overall system throughput.

FILES

/dev/rjei DQS11-B communicating with IBM 370

SEE ALSO

General Information−Binary Synchronous Communication, IBM Systems Reference Library #GA27-3004.
DQS11-A/B PDP-11 Communications Controller Option Description, Digital Equipment Corporation.

NAME

rp − RP-11/RP03 moving-head disk

DESCRIPTION

The files *rp0 ... rp7* refer to sections of the RP03 disk drive 0. The files *rp10 ... rp17* refer to drive 1, etc. This is done since the size of a full pack 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 sections on each drive are as follows:

section	start	length
0	0	7600
1	38	36200
2	219	36200
3	40	65535
4	22	36200
5	203	40600
6-7	unassigned	

The start address is a cylinder address, with each cylinder containing 200 blocks. 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 *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*

NAME

tm − TM11/TU10 magtape interface

DESCRIPTION

The files *mt0, ..., mt7* refer to the Digital Equipment Corporation TM11/TU10 magnetic tape control and transports at 800bpi. The files *mt0, ..., mt3* are designated normal-rewind on close, and the files *mt4, ..., mt7* are no-rewind on close. When opened for reading or writing, the tape is assumed to be positioned as desired. When a file is closed, a double end-of-file (double tape mark) is written if the file was opened for writing. If the file was normal-rewind, the tape is rewound. If it is no-rewind and the file was open for writing, the tape is positioned before the second EOF just written. If the file was no-rewind and opened read-only, the tape is positioned after the EOF following the data just read. Once opened, reading is restricted to between the position when opened and the next EOF or the last write. The EOF is returned as a zero-length read. By judiciously choosing *mt* files, it is possible to read and write multi-file tapes.

A standard tape consists of several 512 byte records terminated by an EOF. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time (although very inadvisable).

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, up to the buffer size specified. In raw tape I/O, the buffer must begin on a word boundary and the count must be even. Seeks are ignored. An EOF is returned as a zero-length read, with the tape positioned after the EOF, so that the next read will return the next record.

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. The driver is limited to four transports.

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 or Shell sequences which wish to be sure of writing messages on the terminal 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 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; *kl(IV)* and *dh(IV)* describe peculiarities of the individual devices.

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(VIII)* 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.*

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(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 interf ace 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, and all 8-bits are sent on output; 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, 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.

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. On output, all 8-bits are sent.

The ASCII EOT (control-D) character 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. The EOT is passed back unchanged in raw mode.

When the carrier signal from the data-set 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 which read a terminal 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 associated with the 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*(II).

The ASCII FS character 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.

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. 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

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

BUGS

Half-duplex terminals are not supported.