NAME

tty – general interface for terminals

DESCRIPTION

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

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

As for terminals in general: all of the 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; *dh* (4) and *kl* (4) describe peculiarities of the individual devices.

When a terminal file is opened, it causes a wait to take place at the first read or write. In practice, user's programs seldom open these files; they are opened by *init* (1M) and become a user's input and output files. 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 and interrupt signals, as discussed below. The control terminal is inherited by a child process during a *fork* (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.

These special files have a number of modes that can be changed by use of the *ioctl* (2) system call. When first opened, the interface mode is 300 baud, either parity accepted, and 10 bits/character (one stop bit). Subsequent opens do not change the modes or speeds even if all the processes referencing the line have closed the line. Modes that can be changed by *ioctl* include the interface speed (if the hardware permits); number of data and stop bits; acceptance of even parity, odd parity, or both; a raw mode in which all characters may be read, and all 8 bits are sent on output (see *ioctl* (2)); a carriage return (CR) mode in which CR is mapped into new-line on input and in which both CR and line-feed (LF) cause the echoing of the sequence CR-LF; 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.

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 ASCII EOT. By default, the character @ kills the entire line up to the point where it was typed, but not beyond an EOT, and causes a carriage return. Both these characters operate on a key-stroke basis, independently of any backspacing or tabbing that may have been done. Both @ and # may be entered literally by preceding them with a \; the @ and # remain, but the \ disappears. These erase and kill characters may be changed.

When desired, all upper-case letters are mapped into the corresponding lower-case letters. In this made, an upper-case letter may be generated by preceding it with \. In addition, the

following escape sequences are generated on output and accepted on input:

for:	*use:*
`	\'
|	\!
~	\^
{	\(
}	\)

In *raw* mode, no erase or kill processing is done for the reading program; and the EOT, quit, and interrupt characters are not treated specially. Control is returned to the reading program only when the *read* (2) character count has been satisfied (as well as if an *alarm* (2) signal occurs, or if the line hangs up). The input parity bit is passed back to the reader. On output, all 8-bits are sent if parity is set to *even* and *odd* and the number of data bits is set to 8.

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.

The ASCII DC3 (control-s) character can be used to temporarily stop output. It is useful with CRT terminals to prevent output from disappearing before it can be read. Output is resumed when the ASCII DC1 (control-q) character is typed. These start/stop characters are not passed to any other program when used in this manner. Output may also be stopped by typing an ESC character. In this case output is resumed by typing any character. A BREAK character is treated like an ESC character when not 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 that have this terminal as the control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hangup signal is ignored, any 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.

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 that 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* (2).

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 (called **core**) will be generated in the working directory.

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.

FILES

/dev/ln*

SEE ALSO

ioctl(2), signal(2), dh(4), kl(4).