NAME

termios - general terminal line discipline

SYNOPSIS

#include <termios.h>

DESCRIPTION

This describes a general terminal line discipline that is supported on tty asynchronous communication ports.

Opening a Terminal Device File

When a terminal file is opened, it normally causes the process to wait until a connection is established. For most hardware, the presence of a connection is indicated by the assertion of the hardware CARRIER line. If the termios structure associated with the terminal file has the CLOCAL flag set in the cflag, or if the O_NONBLOCK flag is set in the open(2) call, then the open will succeed even without a connection being present. In practice, applications seldom open these files; they are opened by special programs, such as getty(8), and become an application’s standard input, output, and error files.

Job Control in a Nutshell

Every process is associated with a particular process group and session. The grouping is hierarchical: every member of a particular process group is a member of the same session. This structuring is used in managing groups of related processes for purposes of *job control*; that is, the ability from the keyboard (or from program control) to simultaneously stop or restart a complex command (a command composed of one or more related processes). The grouping into process groups allows delivering of signals that stop or start the group as a whole, along with arbitrating which process group has access to the single controlling terminal. The grouping at a higher layer into sessions is to restrict the job control related signals and system calls to within processes resulting from a particular instance of a "login". Typically, a session is created when a user logs in, and the login terminal is setup to be the controlling terminal; all processes spawned from that login shell are in the same session, and inherit the controlling terminal.

A job control shell operating interactively (that is, reading commands from a terminal) normally groups related processes together by placing them into the same process group. A set of processes in the same process group is collectively referred to as a "job". When the foreground process group of the terminal is the same as the process group of a particular job, that job is said to be in the "foreground". When the process group of the terminal is different from the process group of a job (but is still the controlling terminal), that job is said to be in the "background". Normally the shell reads a command and starts the job that implements that command. If the command is to be started in the foreground (typical), it sets the process group of the terminal to the process group of the started job, waits for the job to complete, and then sets the process group of the terminal back to its own process group (it puts itself into the foreground). If the job is to be started in the background (as denoted by the shell operator "&"), it never

changes the process group of the terminal and does not wait for the job to complete (that is, it immediately attempts to read the next command). If the job is started in the foreground, the user may type a key (usually '^Z') which generates the terminal stop signal (SIGTSTP) and has the effect of stopping the entire job. The shell will notice that the job stopped, and will resume running after placing itself in the foreground. The shell also has commands for placing stopped jobs in the background, and for placing stopped or background jobs into the foreground.

Orphaned Process Groups

An orphaned process group is a process group that has no process whose parent is in a different process group, yet is in the same session. Conceptually it means a process group that does not have a parent that could do anything if it were to be stopped. For example, the initial login shell is typically in an orphaned process group. Orphaned process groups are immune to keyboard generated stop signals and job control signals resulting from reads or writes to the controlling terminal.

The Controlling Terminal

A terminal may belong to a process as its controlling terminal. Each process of a session that has a controlling terminal has the same controlling terminal. A terminal may be the controlling terminal for at most one session. The controlling terminal for a session is allocated by the session leader by issuing the TIOCSCTTY ioctl. A controlling terminal is never acquired by merely opening a terminal device file. When a controlling terminal becomes associated with a session, its foreground process group is set to the process group of the session leader.

The controlling terminal is inherited by a child process during a fork(2) function call. A process relinquishes its controlling terminal when it creates a new session with the setsid(2) function; other processes remaining in the old session that had this terminal as their controlling terminal continue to have it. A process does not relinquish its controlling terminal simply by closing all of its file descriptors associated with the controlling terminal if other processes continue to have it open.

When a controlling process terminates, the controlling terminal is disassociated from the current session, allowing it to be acquired by a new session leader. Subsequent access to the terminal by other processes in the earlier session will be denied, with attempts to access the terminal treated as if modem disconnect had been sensed.

Terminal Access Control

If a process is in the foreground process group of its controlling terminal, read operations are allowed. Any attempts by a process in a background process group to read from its controlling terminal causes a SIGTTIN signal to be sent to the process's group unless one of the following special cases apply: if the reading process is ignoring or blocking the SIGTTIN signal, or if the process group of the reading process is orphaned, the read(2) returns -1 with *errno* set to EIO and no signal is sent. The default action of the SIGTTIN signal is to stop the process to which it is sent.

If a process is in the foreground process group of its controlling terminal, write operations are allowed. Attempts by a process in a background process group to write to its controlling terminal will cause the process group to be sent a SIGTTOU signal unless one of the following special cases apply: if TOSTOP is not set, or if TOSTOP is set and the process is ignoring or blocking the SIGTTOU signal, the process is allowed to write to the terminal and the SIGTTOU signal is not sent. If TOSTOP is set, and the process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU, the write(2) returns -1 with errno set to EIO and no signal is sent.

Certain calls that set terminal parameters are treated in the same fashion as write, except that TOSTOP is ignored; that is, the effect is identical to that of terminal writes when TOSTOP is set.

Input Processing and Reading Data

A terminal device associated with a terminal device file may operate in full-duplex mode, so that data may arrive even while output is occurring. Each terminal device file has associated with it an input queue, into which incoming data is stored by the system before being read by a process. The system imposes a limit, {MAX_INPUT}, on the number of bytes that may be stored in the input queue. The behavior of the system when this limit is exceeded depends on the setting of the IMAXBEL flag in the termios *c_iflag*. If this flag is set, the terminal is sent an ASCII BEL character each time a character is received while the input queue is full. Otherwise, the input queue is flushed upon receiving the character.

Two general kinds of input processing are available, determined by whether the terminal device file is in canonical mode or noncanonical mode. Additionally, input characters are processed according to the *c_iflag* and *c_lflag* fields. Such processing can include echoing, which in general means transmitting input characters immediately back to the terminal when they are received from the terminal. This is useful for terminals that can operate in full-duplex mode.

The manner in which data is provided to a process reading from a terminal device file is dependent on whether the terminal device file is in canonical or noncanonical mode.

Another dependency is whether the O_NONBLOCK flag is set by open(2) or fcntl(2). If the O_NONBLOCK flag is clear, then the read request is blocked until data is available or a signal has been received. If the O_NONBLOCK flag is set, then the read request is completed, without blocking, in one of three ways:

1. If there is enough data available to satisfy the entire request, and the read completes successfully the number of bytes read is returned.

2. If there is not enough data available to satisfy the entire request, and the read completes successfully, having read as much data as possible, the number of bytes read is returned.

3. If there is no data available, the read returns -1, with errno set to EAGAIN.

When data is available depends on whether the input processing mode is canonical or noncanonical.

Canonical Mode Input Processing

In canonical mode input processing, terminal input is processed in units of lines. A line is delimited by a newline '\n' character, an end-of-file (EOF) character, or an end-of-line (EOL) character. See the *Special Characters* section for more information on EOF and EOL. This means that a read request will not return until an entire line has been typed, or a signal has been received. Also, no matter how many bytes are requested in the read call, at most one line is returned. It is not, however, necessary to read a whole line at once; any number of bytes, even one, may be requested in a read without losing information.

{MAX_CANON} is a limit on the number of bytes in a line. The behavior of the system when this limit is exceeded is the same as when the input queue limit {MAX_INPUT}, is exceeded.

Erase and kill processing occur when either of two special characters, the ERASE and KILL characters (see the *Special Characters* section), is received. This processing affects data in the input queue that has not yet been delimited by a newline NL, EOF, or EOL character. This un-delimited data makes up the current line. The ERASE character deletes the last character in the current line, if there is any. The KILL character deletes all data in the current line, if there is any. The ERASE and KILL characters have no effect if there is no data in the current line. The ERASE and KILL characters themselves are not placed in the input queue.

Noncanonical Mode Input Processing

In noncanonical mode input processing, input bytes are not assembled into lines, and erase and kill processing does not occur. The values of the VMIN and VTIME members of the *c_cc* array are used to determine how to process the bytes received.

MIN represents the minimum number of bytes that should be received when the read(2) function successfully returns. TIME is a timer of 0.1 second granularity that is used to time out bursty and short term data transmissions. If MIN is greater than { MAX_INPUT}, the response to the request is undefined. The four possible values for MIN and TIME and their interactions are described below.

Case A: MIN > 0, TIME > 0

In this case TIME serves as an inter-byte timer and is activated after the first byte is received. Since it is an inter-byte timer, it is reset after a byte is received. The interaction between MIN and TIME is as follows: as soon as one byte is received, the inter-byte timer is started. If MIN bytes are received before the inter-byte timer expires (remember that the timer is reset upon receipt of each byte), the read is satisfied. If the timer expires before MIN bytes are received, the characters received to that point are

returned to the user. Note that if TIME expires at least one byte is returned because the timer would not have been enabled unless a byte was received. In this case (MIN > 0, TIME > 0) the read blocks until the MIN and TIME mechanisms are activated by the receipt of the first byte, or a signal is received. If data is in the buffer at the time of the **read**(), the result is as if data had been received immediately after the **read**().

Case B: MIN > 0, TIME = 0

In this case, since the value of TIME is zero, the timer plays no role and only MIN is significant. A pending read is not satisfied until MIN bytes are received (i.e., the pending read blocks until MIN bytes are received), or a signal is received. A program that uses this case to read record-based terminal I/O may block indefinitely in the read operation.

Case C: MIN = 0, TIME > 0

In this case, since MIN = 0, TIME no longer represents an inter-byte timer. It now serves as a read timer that is activated as soon as the read function is processed. A read is satisfied as soon as a single byte is received or the read timer expires. Note that in this case if the timer expires, no bytes are returned. If the timer does not expire, the only way the read can be satisfied is if a byte is received. In this case the read will not block indefinitely waiting for a byte; if no byte is received within TIME*0.1 seconds after the read is initiated, the read returns a value of zero, having read no data. If data is in the buffer at the time of the read, the timer is started as if data had been received immediately after the read.

Case D: MIN = 0, TIME = 0

The minimum of either the number of bytes requested or the number of bytes currently available is returned without waiting for more bytes to be input. If no characters are available, read returns a value of zero, having read no data.

Writing Data and Output Processing

When a process writes one or more bytes to a terminal device file, they are processed according to the *c_oflag* field (see the *Output Modes* section). The implementation may provide a buffering mechanism; as such, when a call to **write**() completes, all of the bytes written have been scheduled for transmission to the device, but the transmission will not necessarily have been completed.

Special Characters

Certain characters have special functions on input or output or both. These functions are summarized as follows:

INTR Special character on input and is recognized if the ISIG flag (see the *Local Modes* section) is enabled. Generates a SIGINT signal which is sent to all processes in the foreground process group for which the terminal is the controlling terminal. If ISIG is set, the INTR character is discarded when processed.

QUIT Special character on input and is recognized if the ISIG flag is enabled. Generates a SIGQUIT signal which is sent to all processes in the foreground process group for which the terminal is the controlling terminal. If ISIG is set, the QUIT character is discarded when processed.

ERASE
Special character on input and is recognized if the ICANON flag is set. Erases the last character in the current line; see *Canonical Mode Input Processing*. It does not erase beyond the start of a line, as delimited by an NL, EOF, or EOL character. If ICANON is set, the ERASE character is discarded when processed.

KILL Special character on input and is recognized if the ICANON flag is set. Deletes the entire line, as delimited by a NL, EOF, or EOL character. If ICANON is set, the KILL character is discarded when processed.

EOF Special character on input and is recognized if the ICANON flag is set. When received, all the bytes waiting to be read are immediately passed to the process, without waiting for a newline, and the EOF is discarded. Thus, if there are no bytes waiting (that is, the EOF occurred at the beginning of a line), a byte count of zero is returned from the **read**(), representing an end-of-file indication. If ICANON is set, the EOF character is discarded when processed.

NL Special character on input and is recognized if the ICANON flag is set. It is the line delimiter '\n'.

EOL Special character on input and is recognized if the ICANON flag is set. Is an additional line delimiter, like NL.

SUSP If the ISIG flag is enabled, receipt of the SUSP character causes a SIGTSTP signal to be sent to all processes in the foreground process group for which the terminal is the controlling terminal, and the SUSP character is discarded when processed.

STOP Special character on both input and output and is recognized if the IXON (output control) or IXOFF (input control) flag is set. Can be used to temporarily suspend output. It is useful with fast terminals to prevent output from disappearing before it can be read. If IXON is set, the STOP character is discarded when processed.

START
Special character on both input and output and is recognized if the IXON (output control) or IXOFF (input control) flag is set. Can be used to resume output that has been suspended by a STOP character. If IXON is set, the START character is discarded when processed.

CR Special character on input and is recognized if the ICANON flag is set; it is the '\r', as denoted in the C Standard {2}. When ICANON and ICRNL are set and IGNCR is not set, this character is translated into a NL, and has the same effect as a NL character.

The following special characters are extensions defined by this system and are not a part of IEEE Std 1003.1 ("POSIX.1") termios.

EOL2 Secondary EOL character. Same function as EOL.

WERASE
Special character on input and is recognized if the ICANON flag is set. Erases the last word in the current line according to one of two algorithms. If the ALTWERASE flag is not set, first any preceding whitespace is erased, and then the maximal sequence of non-whitespace characters. If ALTWERASE is set, first any preceding whitespace is erased, and then the maximal sequence of alphabetic/underscores or non alphabetic/underscores. As a special case in this second algorithm, the first previous non-whitespace character is skipped in determining whether the preceding word is a sequence of alphabetic/underscores. This sounds confusing but turns out to be quite practical.

REPRINT
Special character on input and is recognized if the ICANON flag is set. Causes the current input edit line to be retyped.

DSUSP
Has similar actions to the SUSP character, except that the SIGTSTP signal is delivered when one of the processes in the foreground process group issues a **read**() to the controlling terminal.

LNEXT
Special character on input and is recognized if the IEXTEN flag is set. Receipt of this character causes the next character to be taken literally.

DISCARD
Special character on input and is recognized if the IEXTEN flag is set. Receipt of this character toggles the flushing of terminal output.

STATUS
Special character on input and is recognized if the ICANON flag is set. Receipt of this character causes a SIGINFO signal to be sent to the foreground process group of the terminal. Also, if the NOKERNINFO flag is not set, it causes the kernel to write a status message to the terminal that displays the current load average, the name of the command in the foreground, its process ID, the

symbolic wait channel, the number of user and system seconds used, the percentage of cpu the process is getting, and the resident set size of the process.

In case the sysctl(8) variable *kern.tty_info_kstacks* is set to a non-zero value, the running thread's kernel stack is written to the terminal (e.g., for debugging purposes).

The NL and CR characters cannot be changed. The values for all the remaining characters can be set and are described later in the document under Special Control Characters.

Special character functions associated with changeable special control characters can be disabled individually by setting their value to {_POSIX_VDISABLE}; see *Special Control Characters*.

If two or more special characters have the same value, the function performed when that character is received is undefined.

Modem Disconnect

If a modem disconnect is detected by the terminal interface for a controlling terminal, and if CLOCAL is not set in the *c_cflag* field for the terminal, the SIGHUP signal is sent to the controlling process associated with the terminal. Unless other arrangements have been made, this causes the controlling process to terminate. Any subsequent call to the **read**() function returns the value zero, indicating end of file. Thus, processes that read a terminal file and test for end-of-file can terminate appropriately after a disconnect. Any subsequent **write**() to the terminal device returns -1, with *errno* set to EIO, until the device is closed.

General Terminal Interface

Closing a Terminal Device File

The last process to close a terminal device file causes any output to be sent to the device and any input to be discarded. Then, if HUPCL is set in the control modes, and the communications port supports a disconnect function, the terminal device performs a disconnect.

Parameters That Can Be Set

Routines that need to control certain terminal I/O characteristics do so by using the termios structure as defined in the header <*termios.h*>. This structure contains minimally four scalar elements of bit flags and one array of special characters. The scalar flag elements are named: *c_iflag*, *c_oflag*, *c_cflag*, and *c_lflag*. The character array is named *c_cc*, and its maximum index is NCCS.

Input Modes

Values of the *c_iflag* field describe the basic terminal input control, and are composed of following masks:

```
IGNBRK   /* ignore BREAK condition */
BRKINT   /* map BREAK to SIGINTR */
IGNPAR   /* ignore (discard) parity errors */
PARMRK   /* mark parity and framing errors */
INPCK    /* enable checking of parity errors */
ISTRIP   /* strip 8th bit off chars */
INLCR    /* map NL into CR */
IGNCR    /* ignore CR */
ICRNL    /* map CR to NL (ala CRMOD) */
IXON     /* enable output flow control */
IXOFF    /* enable input flow control */
IXANY    /* any char will restart after stop */
IMAXBEL  /* ring bell on input queue full */
IUTF8    /* assume input is UTF-8 encoded */
```

In the context of asynchronous serial data transmission, a break condition is defined as a sequence of zero-valued bits that continues for more than the time to send one byte. The entire sequence of zero-valued bits is interpreted as a single break condition, even if it continues for a time equivalent to more than one byte. In contexts other than asynchronous serial data transmission the definition of a break condition is implementation defined.

If IGNBRK is set, a break condition detected on input is ignored, that is, not put on the input queue and therefore not read by any process. If IGNBRK is not set and BRKINT is set, the break condition flushes the input and output queues and if the terminal is the controlling terminal of a foreground process group, the break condition generates a single SIGINT signal to that foreground process group. If neither IGNBRK nor BRKINT is set, a break condition is read as a single ‘\0’, or if PARMRK is set, as ‘\377’, ‘\0’, ‘\0’.

If IGNPAR is set, a byte with a framing or parity error (other than break) is ignored.

If PARMRK is set, and IGNPAR is not set, a byte with a framing or parity error (other than break) is given to the application as the three-character sequence ‘\377’, ‘\0’, X, where ‘\377’, ‘\0’ is a two-character flag preceding each sequence and X is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of ‘\377’ is given to the application as ‘\377’, ‘\377’. If neither PARMRK nor IGNPAR is set, a framing or parity error (other than break) is given to the application as a single character ‘\0’.

If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is disabled, allowing output parity generation without input parity errors. Note that whether input parity checking is enabled or disabled is independent of whether parity detection is enabled or disabled (see *Control*

Modes). If parity detection is enabled but input parity checking is disabled, the hardware to which the terminal is connected recognizes the parity bit, but the terminal special file does not check whether this bit is set correctly or not.

If ISTRIP is set, valid input bytes are first stripped to seven bits, otherwise all eight bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). If IGNCR is not set and ICRNL is set, a received CR character is translated into a NL character.

If IXON is set, start/stop output control is enabled. A received STOP character suspends output and a received START character restarts output. If IXANY is also set, then any character may restart output. When IXON is set, START and STOP characters are not read, but merely perform flow control functions. When IXON is not set, the START and STOP characters are read.

If IXOFF is set, start/stop input control is enabled. The system shall transmit one or more STOP characters, which are intended to cause the terminal device to stop transmitting data, as needed to prevent the input queue from overflowing and causing the undefined behavior described in *Input Processing and Reading Data*, and shall transmit one or more START characters, which are intended to cause the terminal device to resume transmitting data, as soon as the device can continue transmitting data without risk of overflowing the input queue. The precise conditions under which STOP and START characters are transmitted are implementation defined.

If IMAXBEL is set and the input queue is full, subsequent input shall cause an ASCII BEL character to be transmitted to the output queue.

The initial input control value after **open**() is implementation defined.

Output Modes

Values of the *c_oflag* field describe the basic terminal output control, and are composed of the following masks:

```
OPOST    /* enable following output processing */
ONLCR    /* map NL to CR-NL (ala CRMOD) */
OCRNL    /* map CR to NL */
TABDLY   /* tab delay mask */
TAB0     /* no tab delay and expansion */
TAB3     /* expand tabs to spaces */
ONOEOT   /* discard EOT's '^D' on output) */
ONOCR    /* do not transmit CRs on column 0 */
```

```
ONLRET   /* on the terminal NL performs the CR function */
```

If OPOST is set, the remaining flag masks are interpreted as follows; otherwise characters are transmitted without change.

If ONLCR is set, newlines are translated to carriage return, linefeeds.

If OCRNL is set, carriage returns are translated to newlines.

The TABDLY bits specify the tab delay. The *c_oflag* is masked with TABDLY and compared with the values TAB0 or TAB3. If TAB3 is set, tabs are expanded to the appropriate number of spaces (assuming 8 column tab stops).

If ONOEOT is set, ASCII EOT's are discarded on output.

If ONOCR is set, no CR character is transmitted when at column 0 (first position).

If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer will be set to 0.

Control Modes

Values of the *c_cflag* field describe the basic terminal hardware control, and are composed of the following masks. Not all values specified are supported by all hardware.

```
CSIZE       /* character size mask */
CS5         /* 5 bits (pseudo) */
CS6         /* 6 bits */
CS7         /* 7 bits */
CS8         /* 8 bits */
CSTOPB      /* send 2 stop bits */
CREAD       /* enable receiver */
PARENB      /* parity enable */
PARODD      /* odd parity, else even */
HUPCL       /* hang up on last close */
CLOCAL      /* ignore modem status lines */
CCTS_OFLOW
            /* CTS flow control of output */
CRTSCTS     /* same as CCTS_OFLOW */
CRTS_IFLOW  /* RTS flow control of input */
MDMBUF      /* flow control output via Carrier */
```

CNO_RTSDTR

/* Do not assert RTS or DTR automatically */

The CSIZE bits specify the byte size in bits for both transmission and reception. The *c_cflag* is masked with CSIZE and compared with the values CS5, CS6, CS7, or CS8. This size does not include the parity bit, if any. If CSTOPB is set, two stop bits are used, otherwise one stop bit. For example, at 110 baud, two stop bits are normally used.

If CREAD is set, the receiver is enabled. Otherwise, no character is received. Not all hardware supports this bit. In fact, this flag is pretty silly and if it were not part of the **termios** specification it would be omitted.

If PARENB is set, parity generation and detection are enabled and a parity bit is added to each character. If parity is enabled, PARODD specifies odd parity if set, otherwise even parity is used.

If HUPCL is set, the modem control lines for the port are lowered when the last process with the port open closes the port or the process terminates. The modem connection is broken.

If CLOCAL is set, a connection does not depend on the state of the modem status lines. If CLOCAL is clear, the modem status lines are monitored.

Under normal circumstances, a call to the **open**() function waits for the modem connection to complete. However, if the O_NONBLOCK flag is set or if CLOCAL has been set, the **open**() function returns immediately without waiting for the connection.

The CCTS_OFLOW (CRTSCTS) flag is currently unused.

If MDMBUF is set then output flow control is controlled by the state of Carrier Detect.

If CNO_RTSDTR is set then the RTS and DTR lines will not be asserted when the device is opened. As a result, this flag is only useful on initial-state devices.

If the object for which the control modes are set is not an asynchronous serial connection, some of the modes may be ignored; for example, if an attempt is made to set the baud rate on a network connection to a terminal on another host, the baud rate may or may not be set on the connection between that terminal and the machine it is directly connected to.

Local Modes

Values of the *c_lflag* field describe the control of various functions, and are composed of the following masks.

```
ECHOKE      /* visual erase for line kill */
ECHOE       /* visually erase chars */
ECHO        /* enable echoing */
ECHONL      /* echo NL even if ECHO is off */
ECHOPRT     /* visual erase mode for hardcopy */
ECHOCTL     /* echo control chars as ^(Char) */
ISIG        /* enable signals INTR, QUIT, [D]SUSP */
ICANON      /* canonicalize input lines */
ALTWERASE   /* use alternate WERASE algorithm */
IEXTEN      /* enable DISCARD and LNEXT */
EXTPROC     /* external processing */
TOSTOP      /* stop background jobs from output */
FLUSHO      /* output being flushed (state) */
NOKERNINFO  /* no kernel output from VSTATUS */
PENDIN      /* XXX retype pending input (state) */
NOFLSH      /* don't flush after interrupt */
```

If ECHO is set, input characters are echoed back to the terminal. If ECHO is not set, input characters are not echoed.

If ECHOE and ICANON are set, the ERASE character causes the terminal to erase the last character in the current line from the display, if possible. If there is no character to erase, an implementation may echo an indication that this was the case or do nothing.

If ECHOK and ICANON are set, the KILL character causes the current line to be discarded and the system echoes the '\n' character after the KILL character.

If ECHOKE and ICANON are set, the KILL character causes the current line to be discarded and the system causes the terminal to erase the line from the display.

If ECHOPRT and ICANON are set, the system assumes that the display is a printing device and prints a backslash and the erased characters when processing ERASE characters, followed by a forward slash.

If ECHOCTL is set, the system echoes control characters in a visible fashion using a caret followed by the control character.

If ALTWERASE is set, the system uses an alternative algorithm for determining what constitutes a word when processing WERASE characters (see WERASE).

If ECHONL and ICANON are set, the '\n' character echoes even if ECHO is not set.

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL, as described in *Canonical Mode Input Processing*.

If ICANON is not set, read requests are satisfied directly from the input queue. A read is not satisfied until at least MIN bytes have been received or the timeout value TIME expired between bytes. The time value represents tenths of seconds. See *Noncanonical Mode Input Processing* for more details.

If ISIG is set, each input character is checked against the special control characters INTR, QUIT, and SUSP (job control only). If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, no checking is done. Thus these special input functions are possible only if ISIG is set.

If IEXTEN is set, implementation-defined functions are recognized from the input data. How IEXTEN being set interacts with ICANON, ISIG, IXON, or IXOFF is implementation defined. If IEXTEN is not set, then implementation-defined functions are not recognized, and the corresponding input characters are not processed as described for ICANON, ISIG, IXON, and IXOFF.

If NOFLSH is set, the normal flush of the input and output queues associated with the INTR, QUIT, and SUSP characters are not be done.

If TOSTOP is set, the signal SIGTTOU is sent to the process group of a process that tries to write to its controlling terminal if it is not in the foreground process group for that terminal. This signal, by default, stops the members of the process group. Otherwise, the output generated by that process is output to the current output stream. Processes that are blocking or ignoring SIGTTOU signals are excepted and allowed to produce output and the SIGTTOU signal is not sent.

If NOKERNINFO is set, the kernel does not produce a status message when processing STATUS characters (see STATUS).

Special Control Characters

The special control characters values are defined by the array *c_cc*. This table lists the array index, the corresponding special character, and the system default value. For an accurate list of the system defaults, consult the header file <*sys/ttydefaults.h*>.

Index Name	Special Character	Default Value
VEOF	EOF	^D
VEOL	EOL	_POSIX_VDISABLE
VEOL2	EOL2	_POSIX_VDISABLE
VERASE	ERASE	^? ‘\177’

VWERASE	WERASE	^W
VKILL	KILL	^U
VREPRINT	REPRINT	^R
VINTR	INTR	^C
VQUIT	QUIT	^\\ ‘\34’
VSUSP	SUSP	^Z
VDSUSP	DSUSP	^Y
VSTART	START	^Q
VSTOP	STOP	^S
VLNEXT	LNEXT	^V
VDISCARD	DISCARD	^O
VMIN	---	1
VTIME	---	0
VSTATUS	STATUS	^T

If the value of one of the changeable special control characters (see *Special Characters*) is {_POSIX_VDISABLE}, that function is disabled; that is, no input data is recognized as the disabled special character. If ICANON is not set, the value of {_POSIX_VDISABLE} has no special meaning for the VMIN and VTIME entries of the *c_cc* array.

The initial values of the flags and control characters after **open**() is set according to the values in the header <*sys/ttydefaults.h*>.

SEE ALSO

stty(1), tcgetsid(3), tcgetwinsize(3), tcsendbreak(3), tcsetattr(3), tcsetsid(3), tty(4), stack(9)