NAME

termio – general terminal interface

SYNOPSIS

#include <termio.h>

ioctl(int *fildes,* **int** *request,* **struct termio** **arg***);**

ioctl(int *fildes,* **int** *request,* **int** *arg***);**

#include <termios.h>

ioctl(int *fildes,* **int** *request,* **struct termios** **arg***);**

DESCRIPTION

This release supports a general interface for asynchronous communications ports that is hardware-independent. The user interface to this functionality is using function calls (the preferred interface) described in **termios**(3) or **ioctl** commands described in this section. This section also discusses the common features of the terminal subsystem which are relevant with both user interfaces.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, users' programs seldom open terminal files; they are opened by the system and become a user's standard input, output, and error files. The first terminal file opened by the session leader that is not already associated with a session becomes the controlling terminal for that session. The controlling terminal plays a special role in handling quit and interrupt signals, as discussed below. The controlling terminal is inherited by a child process during a **fork**(2). A process can break this association by changing its session using **setsid( )** (see **getsid**(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 character input buffers of the system become completely full, which is rare (for example, if the number of characters in the line discipline buffer exceeds **{MAX_CANON}** and **IMAXBEL** (see below) is not set), or when the user has accumulated {**MAX_INPUT**} number of input characters that have not yet been read by some program. When the input limit is reached, all the characters saved in the buffer up to that point are thrown away without notice.

Session management (Job Control)

A control terminal will distinguish one of the process groups in the session associated with it to be the foreground process group. All other process groups in the session are designated as background process groups. This foreground process group plays a special role in handling signal-generating input characters, as discussed below. By default, when a controlling terminal is allocated, the controlling process's process group is assigned as foreground process group.

Background process groups in the controlling process's session are subject to a job control line discipline when they attempt to access their controlling terminal. Process groups can be sent signals that will cause them to stop, unless they have made other arrangements. An exception is made for members of orphaned process groups.

The operating system will not normally send **SIGTSTP, SIGTTIN,** or **SIGTTOU.** signals to a process that is a member of an orphaned process group.

These are process groups which do not have a member with a parent in another process group that is in the same session and therefore shares the same controlling terminal. When a member's orphaned process group attempts to access its controlling terminal, errors will be returned. since there is no process to continue it if it should stop.

If a member of a background process group attempts to read its controlling terminal, its process group will be sent a **SIGTTIN** signal, which will normally cause the members of that process group to stop. If, however, the process is ignoring or holding **SIGTTIN**, or is a member of an orphaned process group, the read will fail with **errno** set to **EIO**, and no signal will be sent.

If a member of a background process group attempts to write its controlling terminal and the **TOSTOP** bit is set in the **c_lflag** field, its process group will be sent a **SIGTTOU** signal, which will normally cause the members of that process group to stop. If, however, the process is ignoring or holding **SIGTTOU**, the write will succeed. If the process is not ignoring or holding **SIGTTOU** and is a member of an orphaned process group, the write will fail with **errno** set to **EIO**, and no signal will be sent.

If **TOSTOP** is set and a member of a background process group attempts to **ioctl** its controlling terminal, and that **ioctl** will modify terminal parameters (for example, **TCSETA**, **TCSETAW, TCSETAF,** or **TIOCSPGRP**), its process group will be sent a **SIGTTOU** signal, which will normally cause the members of that process group to stop. If, however, the process is ignoring or holding **SIGTTOU**, the ioctl will succeed. If the process is not ignoring or holding **SIGTTOU** and is a member of an orphaned process group, the write will fail with **errno** set to **EIO**, and no signal will be sent.

Canonical mode input processing

Normally, terminal input is processed in units of lines. A line is delimited by a newline (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. 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 necessary, however, 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. The ERASE character (by default, the character DEL) erases the last character typed. The WERASE character (the character control-W) erases the last "word" typed in the current input line (but not any preceding spaces or tabs). A "word" is defined as a sequence of non-blank characters, with tabs counted as blanks. Neither ERASE nor WERASE will erase beyond the beginning of the line. The KILL character (by default, the character NAK) kills (deletes) the entire input line, and optionally outputs a newline character. All these characters operate on a key stroke basis, independent of any backspacing or tabbing that may have been done. The **REPRINT** character (the character control-R) prints a newline followed by all characters that have not been read. Reprinting also occurs automatically if characters that would normally be erased from the screen are fouled by program output. The characters are reprinted as if they were being echoed; consequencely, if **ECHO** is not set, they are not printed.

The ERASE and KILL characters may be entered literally by preceding them with the escape character (\ ). In this case, the escape character is not read. The erase and kill characters may be changed.

Non-canonical mode input processing

In non-canonical mode input processing, input characters are not assembled into lines, and erase and kill processing does not occur. The **MIN** and **TIME** values are used to determine how to process the characters received.

MIN represents the minimum number of characters that should be received when the read is satisfied (that is, when the characters are returned to the user). **TIME** is a timer of 0.10-second granularity that is used to timeout bursty and short-term data transmissions. 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 intercharacter timer and is activated after the first character is received. Since it is an intercharacter timer, it is reset after a character is received. The interaction between **MIN** and **TIME** is as follows: as soon as one character is received, the intercharacter timer is started. If **MIN** characters are received before the intercharacter timer expires (note that the timer is reset upon receipt of each character), the read is satisfied. If the timer expires before **MIN** characters are received, the characters received to that point are returned to the user. Note that if **TIME** expires, at least one character will be returned because the timer would not have been enabled unless a character was received. In this case (**MIN** > 0, **TIME** > 0), the read sleeps until the **MIN** and **TIME** mechanisms are activated by the receipt of the first character. If the number of characters read is less than the number of characters available, the timer is not reactivated and the subsequent read is satisfied immediately.

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** characters are received (the pending read sleeps until **MIN** characters are 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 intercharacter timer: it now serves as a read timer that is activated as soon as a **read** is done. A read is satisfied as soon as a single character is received or the read timer expires. Note that, in this case, if the timer expires, no character is returned. If the timer does not expire, the only way the read can be satisfied is if a character is received. In this case, the read will not block indefinitely waiting for a character; if no character is received within **TIME**∗.10 seconds after the read is initiated, the read returns with zero characters.

Case D: **MIN** = 0, **TIME** = 0

In this case, return is immediate. The minimum of either the number of characters requested or the number of characters currently available is returned without waiting for more characters to be input.

Comparison of the different cases of MIN, TIME interaction

Some points to note about **MIN** and **TIME**:

1. In the following explanations, note that the interactions of **MIN** and **TIME** are not symmetric. For example, when **MIN** > 0 and **TIME** = 0, **TIME** has no effect. However, in the opposite case, where **MIN** = 0 and **TIME** > 0, both **MIN** and **TIME** play a role in that **MIN** is satisfied with the receipt of a single character.
2. Also note that in case A (**MIN** > 0, **TIME** > 0), **TIME** represents an intercharacter timer, whereas in case C (**MIN** = 0, **TIME** > 0), **TIME** represents a read timer.

These two points highlight the dual purpose of the **MIN/TIME** feature. Cases A and B, where **MIN** > 0, exist to handle burst mode activity (for example, file transfer programs), where a program would like to process at least **MIN** characters at a time. In case A, the intercharacter timer is activated by a user as a safety measure; in case B, the timer is turned off.

Cases C and D exist to handle single character, timed transfers. These cases are readily adaptable to screen-based applications that need to know if a character is present in the input queue before refreshing the screen. In case C, the read is timed, whereas in case D, it is not.

Another important note is that **MIN** is always just a minimum. It does not denote a record length. For example, if a program does a read of 20 bytes, **MIN** is 10, and 25 characters are present, then 20 characters will be returned to the user.

Writing characters

When one or more characters are written, they are transmitted to the terminal as soon as previously written characters have finished typing. Input characters are echoed as they are typed if echoing has been enabled. If 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 is drained down to some threshold, the program is resumed.

Special Characters

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR
: (CTRL-C or ASCII ETX) generates a SIGINT signal. SIGINT is sent to all frequent processes associated with the controlling 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**(5)).

QUIT
: (CTRL-| or ASCII FS) generates a SIGQUIT 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 created in the current working directory.

ERASE
: (DEL) erases the preceding character. It does not erase beyond the start of a line, as delimited by a NL, EOF, EOL, or EOL2 character.

WERASE
: (CTRL-W or ASCII ETX) erases the preceding “word”. It does not erase beyond the start of a line, as delimited by a NL, EOF, EOL, or EOL2 character.

KILL
: (CTRL-U or ASCII NAK) deletes the entire line, as delimited by a NL, EOF,

EOL, or EOL2 character.

REPRINT	(CTRL-R or ASCII DC2) reprints all characters, preceded by a newline, that have not been read.
EOF	(CTRL-D or ASCII EOT) may be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a newline, and the EOF is discarded. Thus, if no characters are waiting (that is, the EOF occurred at the beginning of a line) zero characters are passed back, which is the standard end-of-file indication. Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.
NL	(ASCII LF) is the normal line delimiter. It cannot be changed or escaped.
EOL	(ASCII NULL) is an additional line delimiter, like NL. It is not normally used.
EOL2	is another additional line delimiter.
SWTCH	(CTRL-Z or ASCII EM) is used only when **shl** layers is invoked.
SUSP	(CTRL-Z or ASCII SUB) generates a SIGTSTP signal. SIGTSTP stops all processes in the foreground process group for that terminal.
DSUSP	(CTRL-Y or ASCII EM) It generates a SIGTSTP signal as SUSP does, but the signal is sent when a process in the foreground process group attempts to read the DSUSP character, rather than when it is typed.
STOP	(CTRL-S or ASCII DC3) can be used to suspend output temporarily. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.
START	(CTRL-Q or ASCII DC1) is used to resume output. Output has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read.
DISCARD	(CTRL-O or ASCII SI) causes subsequent output to be discarded. Output is discarded until another DISCARD character is typed, more input arrives, or the condition is cleared by a program.
LNEXT	(CTRL-V or ASCII SYN) causes the special meaning of the next character to be ignored. This works for all the special characters mentioned above. It allows characters to be input that would otherwise be interpreted by the system (for example KILL, QUIT).

The character values for INTR, QUIT, ERASE, WERASE, KILL, REPRINT, EOF, EOL, EOL2, SWTCH, SUSP, DSUSP, STOP, START, DISCARD, and LNEXT may be changed to suit individual tastes. If the value of a special control character is _POSIX_VDISABLE (0), the function of that special control character is disabled. The ERASE, KILL, and EOF characters may be escaped by a preceding \ character, in which case no special function is done. Any of the special characters may be preceded by the LNEXT character, in which case no special function is done.

Modem disconnect

When a modem disconnect is detected, a **SIGHUP** signal is sent to the terminal's controlling process. Unless other arrangements have been made, these signals cause the process to terminate. If **SIGHUP** is ignored or caught, any subsequent read returns with an end-of-file indication until the terminal is closed.

If the controlling process is not in the foreground process group of the terminal, a **SIGTSTP** is sent to the terminal's foreground process group. Unless other arrangements have been made, these signals cause the processes to stop.

Processes in background process groups that attempt to access the controlling terminal after modem disconnect while the terminal is still allocated to the session will receive appropriate **SIGTTOU** and **SIGTTIN** signals. Unless other arrangements have been made, this signal causes the processes to stop.

The controlling terminal will remain in this state until it is reinitialized with a successful open by the controlling process, or deallocated by the controlling process.

Terminal parameters

The parameters that control the behavior of devices and modules providing the **termios** interface are specified by the **termios** structure defined by **<termios.h>**. Several **ioctl**(2) system calls that fetch or change these parameters use this structure that contains the following members:

```
tcflag_t    c_iflag;      /* input modes */
tcflag_t    c_oflag;      /* output modes */
tcflag_t    c_cflag;      /* control modes */
tcflag_t    c_lflag;      /* local modes */
cc_t        c_cc[NCCS];   /* control chars */
```

The special control characters are defined by the array **c_cc**. The symbolic name **NCCS** is the size of the control-character array and is also defined by **<termios.h>**. The relative positions, subscript names, and typical default values for each function are as follows:

0	**VINTR**	ETX
1	**VQUIT**	FS
2	**VERASE**	DEL
3	**VKILL**	NAK
4	**VEOF**	EOT
5	**VEOL**	NUL
6	**VEOL2**	NUL
7	**VSWTCH**	NUL
8	**VSTART**	DC1
9	**VSTOP**	DC3
10	**VSUSP**	SUB
11	**VDSUSP**	EM
12	**VREPRINT**	DC2
13	**VDISCARD**	SI
14	**VWERASE**	ETB
15	**VLNEXT**	SYN

16-19 reserved

Input modes

The **c_iflag** field describes the basic terminal input control:

IGNBRK	Ignore break condition.
BRKINT	Signal interrupt on break.
IGNPAR	Ignore characters with parity errors.
PARMRK	Mark parity errors.
INPCK	Enable input parity check.
ISTRIP	Strip character.
INLCR	Map NL to CR on input.
IGNCR	Ignore CR.
ICRNL	Map CR to NL on input.
IUCLC	Map upper-case to lower-case on input.
IXON	Enable start/stop output control.
IXANY	Enable any character to restart output.
IXOFF	Enable start/stop input control.
IMAXBEL	Echo BEL on input line too long.

If **IGNBRK** is set, a break condition (a character framing error with data all zeros) 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 shall flush 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 ASCII NULL character (´\0´), or if **PARMRK** is set, as ´\377´, ´\0´, ´\0´.

If **IGNPAR** is set, a byte with framing or parity errors (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 X is the data of the byte 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 **IGNPAR** nor **PARMRK** is set, a framing or parity error (other than break) is given to the application as a single ASCII NULL character (´\0´).

If **INPCK** is set, input parity checking is enabled. If I**NPCK** is not set, input parity checking is disabled. This allows 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. If parity detection is enabled but input parity checking is disabled, the hardware to which the terminal is connected will recognize the parity bit, but the terminal special file will not check whether this is set correctly or not.

If **ISTRIP** is set, valid input characters 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). Otherwise, if **ICRNL** is set, a received CR character is translated into a NL character.

If **IUCLC** is set, a received upper case, alphabetic character is translated into the corresponding lower case character.

If **IXON** is set, start/stop output control is enabled. A received STOP character suspends output and a received START character restarts output. The STOP and START characters will not be read, but will merely perform flow control functions. If **IXANY** is set, any input character restarts output that has been suspended.

If **IXOFF** is set, the system transmits a STOP character when the input queue is nearly full, and a START character when enough input has been read so that the input queue is nearly empty again.

If **IMAXBEL** is set, the ASCII BEL character is echoed if the input stream overflows. Further input is not stored, but any input already present in the input stream is not disturbed. If **IMAXBEL** is not set, no BEL character is echoed, and all input present in the input queue is discarded if the input stream overflows.

Output modes

The **c_oflag** field specifies the system treatment of output:

OPOST	Post-process output.
OLCUC	Map lower case to upper on output.
ONLCR	Map NL to CR-NL on output.
OCRNL	Map CR to NL on output.
ONOCR	No CR output at column 0.
ONLRET	NL performs CR function.
OFILL	Use fill characters for delay.
OFDEL	Fill is DEL, else **NULL**.
NLDLY	Select newline delays:
NL0	
NL1	
CRDLY	Select carriage-return delays:
CR0	
CR1	
CR2	
CR3	
TABDLY	Select horizontal tab delays:
TAB0	or tab expansion:
TAB1	
TAB2	
TAB3	Expand tabs to spaces.
XTABS	Expand tabs to spaces.
BSDLY	Select backspace delays:
BS0	
BS1	
VTDLY	Select vertical tab delays:
VT0	
VT1	
FFDLY	Select form feed delays:

FF0
FF1

If **OPOST** is set, output characters are post-processed as indicated by the remaining flags; otherwise, characters are transmitted without change.

If **OLCUC** is set, a lower case alphabetic character is transmitted as the corresponding upper case character. This function is often used in conjunction with **IUCLC**.

If **ONLCR** is set, the NL character is transmitted as the CR-NL character pair. If **OCRNL** is set, the CR character is transmitted as the NL character. If **ONOCR** is set, no CR character is transmitted when at column 0 (first position). If **ONRET** is set, the NL character is assumed to do the carriage-return function; the column pointer is set to 0 and the delays specified for CR are used. Otherwise, the NL character is assumed to do just the line-feed function; the column pointer remains unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.

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. If **OFILL** is set, fill characters are transmitted for delay instead of a timed delay. This is useful for high baud rate terminals that need only a minimal delay. If **OFDEL** is set, the fill character is DEL; otherwise it is **NULL**.

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

Newline delay lasts about 0.10 seconds. If **ONLRET** is set, the carriage-return delays are used instead of the newline delays. If **OFILL** is set, two fill characters are transmitted.

Carriage-return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If **OFILL** is set, delay type 1 transmits two fill characters, and type 2 transmits four fill characters.

Horizontal-tab delay type 1 is dependent on the current column position. Type 2 is about 0.10 seconds. Type 3 specifies that tabs are to be expanded into spaces. If **OFILL** is set, two fill characters are transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If **OFILL** is set, one fill character is transmitted.

The actual delays depend on line speed and system load.

Control Modes

The **c_cflag** field describes the hardware control of the terminal:

CBAUD	Baud rate:
B0	Hang up
B50	50 baud
B75	75 baud
B110	110 baud
B134	134 baud
B150	150 baud
B200	200 baud
B300	300 baud
B600	600 baud
B1200	1200 baud
B1800	1800 baud
B2400	2400 baud
B4800	4800 baud
B9600	9600 baud
B19200	19200 baud
EXTA	External A
B38400	38400 baud
EXTB	External B
B57600	57600 baud
B76800	76800 baud
B115200	115200 baud
B153600	153600 baud
B230400	230400 baud
B307200	307200 baud
B460800	460800 baud

CSIZE	Character size:
CS5	5 bits
CS6	6 bits
CS7	7 bits
CS8	8 bits

CSTOPB	Send two stop bits, else one
CREAD	Enable receiver
PARENB	Parity enable
PARODD	Odd parity, else even
HUPCL	Hang up on last close
CLOCAL	Local line, else dial-up
CIBAUD	Input baud rate, if different from output rate
PAREXT	Extended parity for mark and space parity
CRTSXOFF	Enable inbound hardware flow control
CRTSCTS	Enable outbound hardware flow control

CBAUDEXT Bit to indicate output speed > B38400
CIBAUDEXTBit to indicate input speed > B38400

The **CBAUD** bits together with the **CBAUDEXT** bit specify the output baud rate. To retrieve the output speed from the **termios** structure pointed to by **termios_p** see the following code segment.

```
unsigned long ospeed;

if (termios_p->c_cflag & CBAUDEXT)
        ospeed = (termios_p->c_cflag & CBAUD) + CBAUD + 1;
else
        ospeed = termios_p->c_cflag & CBAUD;
```

To store the output speed in the termios structure pointed to by **termios_p** see the following code segment.

```
unsigned long ospeed;

if (ospeed > CBAUD) {
        termios_p->c_cflag |= CBAUDEXT;
        ospeed -= (CBAUD + 1);
} else
        termios_p->c_cflag &= ~CBAUDEXT;

termios_p->c_cflag =
        (termios_p->c_cflag & ~CBAUD) | (ospeed & CBAUD);
```

The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal is not asserted. Normally, this disconnects the line.

If the **CIBAUDEXT** or **CIBAUD** bits are not zero, they specify the input baud rate, with the **CBAUDEXT** and **CBAUD** bits specifying the output baud rate; otherwise, the output and input baud rates are both specified by the **CBAUDEXT** and **CBAUD** bits. The values for the **CIBAUD** bits are the same as the values for the **CBAUD** bits, shifted left **IBSHIFT** bits. For any particular hardware, impossible speed changes are ignored. To retrieve the input speed in the **termios** structure pointed to by **termios_p** see the following code segment.

```
unsigned long ispeed;

if (termios_p->c_cflag & CIBAUDEXT)
        ispeed = ((termios_p->c_cflag & CIBAUD) >> IBSHIFT)
         + (CIBAUD >> IBSHIFT) + 1;
else
        ispeed = (termios_p->c_cflag & CIBAUD) >> IBSHIFT;
```

To store the input speed in the **termios** structure pointed to by **termios_p** see the following code segment.

```
unsigned long ispeed;

if (ispeed == 0) {
    ispeed = termios_p->c_cflag & CBAUD;
    if (termios_p->c_cflag & CBAUDEXT)
            ispeed += (CBAUD + 1);
}

if ((ispeed << IBSHIFT) > CIBAUD) {
            termios_p->c_cflag |= CIBAUDEXT;
            ispeed -= ((CIBAUD >> IBSHIFT) + 1);
} else
            termios_p->c_cflag &= ˜CIBAUDEXT;
termios_p->c_cflag =
            (termios_p->c_cflag & ˜CIBAUD) |
          ((ispeed << IBSHIFT) & CIBAUD);
```

The **CSIZE** bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If **CSTOPB** is set, two stop bits are used; otherwise, one stop bit is used. For example, at 110 baud, two stops bits are required.

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

If **CREAD** is set, the receiver is enabled. Otherwise, no characters are received.

If **HUPCL** is set, the line is disconnected when the last process with the line open closes it or terminates. That is, the data-terminal-ready signal is not asserted.

If **CLOCAL** is set, the line is assumed to be a local, direct connection with no modem control; otherwise, modem control is assumed.

If **CRTSCTS** is set, inbound hardware flow control is enabled.

If **CRTSCTS** is set, outbound hardware flow control is enabled.

The four possible combinations for the state of **CRTSCTS** and **CRTSXOFF** bits and their interactions are described below.

Case A: **CRTSCTS** off, **CRTSXOFF** off.
In this case the hardware flow control is disabled.

Case B: **CRTSCTS** on, **CRTSXOFF** off.
In this case only outbound hardware flow control is enabled. The state of CTS signal is used to do outbound flow control. It is expected that output will be suspended if CTS is low and resumed when CTS is high.

Case C: **CRTSCTS** off, **CRTSXOFF** on.
In this case only inbound hardware flow control is enabled. The state of RTS

signal is used to do inbound flow control. It is expected that input will be suspended if RTS is low and resumed when RTS is high.

Case D: **CRTSCTS** on, **CRTSXOFF** on.
In this case both inbound and outbound hardware flow control are enabled. Uses the state of CTS signal to do outbound flow control and RTS signal to do inbound flow control.

Local modes

The **c_lflag** field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline provides the following:

ISIG	Enable signals.
ICANON	Canonical input (erase and kill processing).
XCASE	Canonical upper/lower presentation.
ECHO	Enable echo.
ECHOE	Echo erase character as BS-SP-BS.
ECHOK	Echo NL after kill character.
ECHONL	Echo NL.
NOFLSH	Disable flush after interrupt or quit.
TOSTOP	Send **SIGTTOU** for background output.
ECHOCTL	Echo control characters as ^*char*, delete as ^?.
ECHOPRT	Echo erase character as character erased.
ECHOKE	BS-SP-BS erase entire line on line kill.
FLUSHO	Output is being flushed.
PENDIN	Retype pending input at next read or input character.
IEXTEN	Enable extended (implementation-defined) functions.

If **ISIG** is set, each input character is checked against the special control characters INTR, QUIT, SWTCH, SUSP, STATUS, and DSUSP. 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 **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, EOL, and EOL. If **ICANON** is not set, read requests are satisfied directly from the input queue. A read is not satisfied until at least **MIN** characters have been received or the timeout value **TIME** has expired between characters. This allows fast bursts of input to be read efficiently while still allowing single character input. The time value represents tenths of seconds.

If **XCASE** is set, and if **ICANON** is set, an upper case letter is accepted on input by preceding it with a \ character, and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted on input:

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

For example, **A** is input as **\a**, **\n** as **\\n**, and **\N** as **\\\n**.

If **ECHO** is set, characters are echoed as received.

When **ICANON** is set, the following echo functions are possible.

1. If **ECHO** and **ECHOE** are set, and **ECHOPRT** is not set, the ERASE and WERASE characters are echoed as one or more ASCII BS SP BS, which clears the last character(s) from a CRT screen.
2. If **ECHO**, **ECHOPRT**, and **IEXTEN** are set, the first ERASE and WERASE character in a sequence echoes as a backslash (\\), followed by the characters being erased. Subsequent ERASE and WERASE characters echo the characters being erased, in reverse order. The next non-erase character causes a slash (/) to be typed before it is echoed. **ECHOPRT** should be used for hard copy terminals.
3. If **ECHOKE** and **IEXTEN** are set, the kill character is echoed by erasing each character on the line from the screen (using the mechanism selected by **ECHOE** and **ECHOPRT**).
4. If **ECHOK** is set, and **ECHOKE** is not set, the NL character is echoed after the kill character to emphasize that the line is deleted. Note that an escape character (\\) or an LNEXT character preceding the erase or kill character removes any special function.
5. If **ECHONL** is set, the NL character is echoed even if **ECHO** is not set. This is useful for terminals set to local echo (so called half-duplex).

If **ECHOCTL** and **IEXTEN** are set, all control characters (characters with codes between 0 and 37 octal) other than ASCII TAB, ASCII NL, the START character, and the STOP character, ASCII CR, and ASCII BS are echoed as ^**X**, where **X** is the character given by adding 100 octal to the code of the control character (so that the character with octal code 1 is echoed as **^A**), and the ASCII DEL character, with code 177 octal, is echoed as ^?.

If **NOFLSH** is set, the normal flush of the input and output queues associated with the INTR, QUIT, and SUSP characters is not done. This bit should be set when restarting system calls that read from or write to a terminal (see **sigaction**(2)).

If **TOSTOP** and **IEXTEN** are set, the signal **SIGTTOU** is sent to a process that tries to write to its controlling terminal if it is not in the foreground process group for that terminal. This signal normally stops the process. 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, if any.

If **FLUSHO** and **IEXTEN** are set, data written to the terminal is discarded. This bit is set when the FLUSH character is typed. A program can cancel the effect of typing the FLUSH character by clearing **FLUSHO**.

If **PENDIN** and **IEXTEN** are set, any input that has not yet been read is reprinted when the next character arrives as input. **PENDIN** is then automatically cleared.

If **IEXTEN** is set, the following implementation-defined functions are enabled: special characters (**WERASE, REPRINT, DISCARD**, and **LNEXT**) and local flags (**TOSTOP, ECHOCTL, ECHOPRT, ECHOKE, FLUSHO**, and **PENDIN**).

Minimum and Timeout

The **MIN** and **TIME** values are described above under *Non-canonical mode input processing*. The initial value of **MIN** is 1, and the initial value of **TIME** is 0.

Terminal size

The number of lines and columns on the terminal's display is specified in the **winsize** structure defined by **<sys/termios.h>** and includes the following members:

```
unsigned   short   ws_row;      /* rows, in characters */
unsigned   short   ws_col;      /* columns, in characters */
unsigned   short   ws_xpixel;   /* horizontal size, in pixels */
unsigned   short   ws_ypixel;   /* vertical size, in pixels */
```

Termio structure

The SunOS/SVR4 **termio** structure is used by some **ioctl**s; it is defined by **<sys/termio.h>** and includes the following members:

```
unsigned   short   c_iflag;     /* input modes */
unsigned   short   c_oflag;     /* output modes */
unsigned   short   c_cflag;     /* control modes */
unsigned   short   c_lflag;     /* local modes */
char               c_line;      /* line discipline */
unsigned   char    c_cc[NCC];   /* control chars */
```

The special control characters are defined by the array **c_cc**. The symbolic name **NCC** is the size of the control-character array and is also defined by **<termio.h>**. The relative positions, subscript names, and typical default values for each function are as follows:

0	**VINTR**	EXT
1	**VQUIT**	FS
2	**VERASE**	DEL
3	**VKILL**	NAK
4	**VEOF**	EOT
5	**VEOL**	NUL
6	**VEOL2**	NUL
7	reserved	

The **MIN** values is stored in the **VMIN** element of the **c_cc** array; the **TIME** value is stored in the **VTIME** element of the **c_cc** array. The **VMIN** element is the same element as the **VEOF** element; the **VTIME** element is the same element as the **VEOL** element.

The calls that use the **termio** structure only affect the flags and control characters that can be stored in the **termio** structure; all other flags and control characters are unaffected.

Modem lines

On special files representing serial ports, the modem control lines supported by the hardware can be read, and the modem status lines supported by the hardware can be changed. The following modem control and status lines may be supported by a device; they are defined by **<sys/termios.h>**:

TIOCM_LE	line enable
TIOCM_DTR	data terminal ready
TIOCM_RTS	request to send
TIOCM_ST	secondary transmit
TIOCM_SR	secondary receive
TIOCM_CTS	clear to send
TIOCM_CAR	carrier detect
TIOCM_RNG	ring
TIOCM_DSR	data set ready

TIOCM_CD is a synonym for **TIOCM_CAR**, and **TIOCM_RI** is a synonym for **TIOCM_RNG**. Not all of these are necessarily supported by any particular device; check the manual page for the device in question.

Default values

The initial **termios** values upon driver open is configurable. This is accomplished by setting the “ttymodes” property in the file **/kernel/drv/options.conf**. Note: This property is assigned during system initialization, therefore any change to the “ttymodes” property will not take effect until the next reboot. The string value assigned to this property should be in the same format as the output of the stty command with the -g option. See **stty**(1).

If this property is undefined, the following **termios** modes are in effect. The initial input control value is **BRKINT**, **ICRNL**, **IXON**, **IMAXBEL**. The initial output control value is **OPOST**, **ONLCR**, **TAB3**. The initial hardware control value is **B9600**, **CS8**, **CREAD**. The initial line-discipline control value is **ISIG**, **ICANON**, **IEXTEN**, **ECHO**, **ECHOK**, **ECHOE**, **ECHOKE**, **ECHOCTL**.

IOCTLS

The **ioctl**s supported by devices and STREAMS modules providing the **termios** interface are listed below. Some calls may not be supported by all devices or modules. The functionality provided by these calls is also available through the preferred function call interface specified on **termios**(3).

TCGETS The argument is a pointer to a **termios** structure. The current terminal parameters are fetched and stored into that structure.

TCSETS The argument is a pointer to a **termios** structure. The current terminal parameters are set from the values stored in that structure. The change is immediate.

TCSETSW The argument is a pointer to a **termios** structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that affect output.

TCSETSF The argument is a pointer to a **termios** structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and then the change occurs.

TCGETA The argument is a pointer to a **termio** structure. The current terminal parameters are fetched, and those parameters that can be stored in a **termio** structure are stored into that structure.

TCSETA The argument is a pointer to a **termio** structure. Those terminal parameters that can be stored in a **termio** structure are set from the values stored in that structure. The change is immediate.

TCSETAW The argument is a pointer to a **termio** structure. Those terminal parameters that can be stored in a **termio** structure are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that affect output.

TCSETAF The argument is a pointer to a **termio** structure. Those terminal parameters that can be stored in a **termio** structure are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and then the change occurs.

TCSBRK The argument is an **int** value. Wait for the output to drain. If the argument is 0, then send a break (zero valued bits for 0.25 seconds).

TCXONC Start/stop control. The argument is an **int** value. If the argument is 0, suspend output; if 1, restart suspended output; if 2, suspend input; if 3, restart suspended input.

TCFLSH The argument is an **int** value. If the argument is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

TIOCGPGRP The argument is a pointer to a **pid_t**. Set the value of that **pid_t** to the process group ID of the foreground process group associated with the terminal. See **termios**(3) for a description or **TCGETPGRP**.

TIOCSPGRP The argument is a pointer to a **pid_t**. Associate the process group whose process group ID is specified by the value of that **pid_t** with the terminal. The new process group value must be in the range of valid process group ID values. Otherwise, the error **EPERM** is returned. See **termios**(3) for a description of **TCSETPGRP**.

TIOCGSID The argument is a pointer to a **pid_t**. The session ID of the terminal is fetched and stored in the **pid_t**.

TIOCGWINSZ The argument is a pointer to a **winsize** structure. The terminal driver's notion of the terminal size is stored into that structure.

TIOCSWINSZ The argument is a pointer to a **winsize** structure. The terminal driver's notion of the terminal size is set from the values specified in that structure. If the new sizes are different from the old sizes, a **SIGWINCH** signal is set to the process group of the terminal.

TIOCMBIS The argument is a pointer to an **int** whose value is a mask containing modem control lines to be turned on. The control lines whose bits are set in the argument are turned on; no other control lines are affected.

TIOCMBIC The argument is a pointer to an **int** whose value is a mask containing modem control lines to be turned off. The control lines whose bits are set in the argument are turned off; no other control lines are affected.

TIOCMGET The argument is a pointer to an **int**. The current state of the modem status lines is fetched and stored in the **int** pointed to by the argument.

TIOCMSET The argument is a pointer to an **int** containing a new set of modem control lines. The modem control lines are turned on or off, depending on whether the bit for that mode is set or clear.

FILES

files in or under /**dev**

SEE ALSO

fork(2), **getsid**(2), **ioctl**(2), **termios**(3), **signal**(3C), **streamio**(7I)