NAME

tty - general terminal interface

SYNOPSIS

#include <sys/ioctl.h>

DESCRIPTION

This section describes the interface to the terminal drivers in the system.

Terminal Special Files

Each hardware terminal port on the system usually has a terminal special device file associated with it in the directory ‘‘/dev/’’ (for example, ‘‘/dev/tty03’’). When a user logs into the system on one of these hardware terminal ports, the system has already opened the associated device and prepared the line for normal interactive use (see getty(8).) There is also a special case of a terminal file that connects not to a hardware terminal port, but to another program on the other side. These special terminal devices are called *ptys* and provide the mechanism necessary to give users the same interface to the system when logging in over a network (using telnet(1) for example). Even in these cases the details of how the terminal file was opened and set up is already handled by special software in the system. Thus, users do not normally need to worry about the details of how these lines are opened or used. Also, these lines are often used for dialing out of a system (through an out-calling modem), but again the system provides programs that hide the details of accessing these terminal special files (see tip(1)).

When an interactive user logs in, the system prepares the line to behave in a certain way (called a *line discipline*), the particular details of which is described in stty(1) at the command level, and in termios(4) at the programming level. A user may be concerned with changing settings associated with his particular login terminal and should refer to the preceding man pages for the common cases. The remainder of this man page is concerned with describing details of using and controlling terminal devices at a low level, such as that possibly required by a program wishing to provide features similar to those provided by the system.

Terminal File Operations

All of the following operations are invoked using the ioctl(2) system call. Refer to that man page for a description of the *request* and *argp* parameters. In addition to the ioctl *requests* defined here, the specific line discipline in effect will define other *requests* specific to it (actually termios(4) defines them as function calls, not ioctl *requests*.) The following section lists the available ioctl requests. The name of the request, a description of its purpose, and the typed *argp* parameter (if any) are listed. For example, the first entry says

*TIOCSPGRP int *tpgrp*

and would be called on the terminal associated with file descriptor zero by the following code fragment:

```
int pgrp;

pgrp = getpgrp();
ioctl(0, TIOCSPGRP, &pgrp);
```

Terminal File Request Descriptions

TIOCSETD *int *ldisc*
: This call is obsolete but left for compatibility. Before FreeBSD 8.0, it would change to the new line discipline pointed to by *ldisc*.

TIOCGETD *int *ldisc*
: Return the current line discipline in the integer pointed to by *ldisc*.

TIOCSBRK *void*
: Set the terminal hardware into BREAK condition.

TIOCCBRK *void*
: Clear the terminal hardware BREAK condition.

TIOCSDTR *void*
: Assert data terminal ready (DTR).

TIOCCDTR *void*
: Clear data terminal ready (DTR).

TIOCGPGRP *int *tpgrp*
: Return the current process group with which the terminal is associated in the integer pointed to by *tpgrp*. This is the underlying call that implements the termios(4) **tcgetattr**() call.

TIOCSPGRP *int *tpgrp*
: Associate the terminal with the process group (as an integer) pointed to by *tpgrp*. This is the underlying call that implements the termios(4) **tcsetattr**() call.

TIOCGETA *struct termios *term*
: Place the current value of the termios state associated with the device in the termios structure pointed to by *term*. This is the underlying call that implements the termios(4) **tcgetattr**() call.

TIOCSETA *struct termios *term*

Set the termios state associated with the device immediately. This is the underlying call that implements the termios(4) **tcsetattr**() call with the TCSANOW option.

TIOCSETAW *struct termios *term*

First wait for any output to complete, then set the termios state associated with the device. This is the underlying call that implements the termios(4) **tcsetattr**() call with the TCSADRAIN option.

TIOCSETAF *struct termios *term*

First wait for any output to complete, clear any pending input, then set the termios state associated with the device. This is the underlying call that implements the termios(4) **tcsetattr**() call with the TCSAFLUSH option.

TIOCOUTQ *int *num*

Place the current number of characters in the output queue in the integer pointed to by *num*.

TIOCSTI *char *cp*

Simulate typed input. Pretend as if the terminal received the character pointed to by *cp*.

TIOCNOTTY *void*

In the past, when a process that did not have a controlling terminal (see *The Controlling Terminal* in termios(4)) first opened a terminal device, it acquired that terminal as its controlling terminal. For some programs this was a hazard as they did not want a controlling terminal in the first place, and this provides a mechanism to disassociate the controlling terminal from the calling process. It *must* be called by opening the file */dev/tty* and calling TIOCNOTTY on that file descriptor.

The current system does not allocate a controlling terminal to a process on an **open**() call: there is a specific ioctl called TIOCSCTTY to make a terminal the controlling terminal. In addition, a program can **fork**() and call the **setsid**() system call which will place the process into its own session - which has the effect of disassociating it from the controlling terminal. This is the new and preferred method for programs to lose their controlling terminal.

However, environmental restrictions may prohibit the process from being able to **fork**() and call the **setsid**() system call to disassociate it from the controlling terminal. In this case, it must use TIOCNOTTY.

TIOCSTOP *void*

Stop output on the terminal (like typing ^S at the keyboard).

TIOCSTART *void*

Start output on the terminal (like typing ^Q at the keyboard).

TIOCSCTTY *void*

Make the terminal the controlling terminal for the process (the process must not currently have a controlling terminal).

TIOCDRAIN *void*

Wait until all output is drained, or until the drain wait timeout expires.

TIOCGDRAINWAIT *int *timeout*

Return the current drain wait timeout in seconds.

TIOCSDRAINWAIT *int *timeout*

Set the drain wait timeout in seconds. A value of zero disables timeouts. The default drain wait timeout is controlled by the tunable sysctl(8) OID *kern.tty_drainwait*.

TIOCEXCL *void*

Set exclusive use on the terminal. No further opens are permitted except by root. Of course, this means that programs that are run by root (or setuid) will not obey the exclusive setting - which limits the usefulness of this feature.

TIOCNXCL *void*

Clear exclusive use of the terminal. Further opens are permitted.

TIOCFLUSH *int *what*

If the value of the int pointed to by *what* contains the FREAD bit as defined in <*sys/file.h*>, then all characters in the input queue are cleared. If it contains the FWRITE bit, then all characters in the output queue are cleared. If the value of the integer is zero, then it behaves as if both the FREAD and FWRITE bits were set (i.e., clears both queues).

TIOCGWINSZ *struct winsize *ws*

Put the window size information associated with the terminal in the *winsize* structure pointed to by *ws*. The window size structure contains the number of rows and columns (and pixels if appropriate) of the devices attached to the terminal. It is set by user software and is the means by which most full-screen oriented programs determine the screen size. The *winsize* structure is defined in <*sys/ioctl.h*>.

TIOCSWINSZ *struct winsize *ws*

Set the window size associated with the terminal to be the value in the *winsize* structure pointed to by *ws* (see above).

TIOCCONS *int *on*

If *on* points to a non-zero integer, redirect kernel console output (kernel printf's) to this terminal. If *on* points to a zero integer, redirect kernel console output back to the normal console. This is usually used on workstations to redirect kernel messages to a particular window.

TIOCMSET *int *state*

The integer pointed to by *state* contains bits that correspond to modem state. Following is a list of defined variables and the modem state they represent:

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_CD	Carrier Detect (synonym).
TIOCM_RNG	Ring Indication.
TIOCM_RI	Ring Indication (synonym).
TIOCM_DSR	Data Set Ready.

This call sets the terminal modem state to that represented by *state*. Not all terminals may support this.

TIOCMGET *int *state*

Return the current state of the terminal modem lines as represented above in the integer pointed to by *state*.

TIOCMBIS *int *state*

The bits in the integer pointed to by *state* represent modem state as described above, however the state is OR-ed in with the current state.

TIOCMBIC *int *state*

The bits in the integer pointed to by *state* represent modem state as described above, however each bit which is on in *state* is cleared in the terminal.

IMPLEMENTATION NOTES

The total number of input and output bytes through all terminal devices are available via the *kern.tty_nin* and *kern.tty_nout* read-only sysctl(8) variables.

SEE ALSO

stty(1), ioctl(2), ng_tty(4), pty(4), termios(4), getty(8)

HISTORY

A console typewriter device */dev/tty* and asynchronous communication interfaces */dev/tty[0-5]* first appeared in Version 1 AT&T UNIX.