

Terminal Control

- Many special files represent devices that are platform dependent, making standardization difficult
- POSIX standards committee decided to include library functions to manipulate special files representing terminals and asynchronous communications ports

stty

- `stty`: Minimum listing of terminal settings
- `stty -a`: More complete listing of terminal settings
- `stty -g`: Produces terminal settings in a form that can be used by a program and allows you to change them

stty sane

- Sets all modes to reasonable values
- Useful if you terminate a program that has set the modes in an inconvenient way – i.e., local echo has been turned off and you can't see what you are typing
- If return has been set to carriage return rather than newline, you may have to terminate stty with Ctrl-j rather than return

tcgetattr and tcsetattr

SYNOPSIS

```
#include <termios.h>
```

```
int tcgetattr(int fildes, struct termios *termios_p)
```

```
int tcsetattr(int fildes, int optional_actions, const struct  
termios *termios_p);
```

POSIX

Returns 0 if successful. If unsuccessful, returns -1 and sets
errno

tcgetattr and tcsetattr errno

errno	cause
EBADF	Fildes is not a valid file descriptor
EINVAL	Optional_actions is not a supported value, or attempt to change attribute represented in termios to unsupported value
ENOTTY	File associated with fildes is not a terminal

tcgetattr

- Retrieves attributes associated with the terminal referenced by the open file descriptor `fd`
- Attributes are returned in `struct termios` structure pointed to by `termios_p`

tcsetattr

- Sets parameters of the terminal referenced by the open file descriptor `fdes` from the `termios` structure pointed to by `termios_p`
- `Optional_actions` parameter controls the point at which the changes take
 - `TCSANOW` – Changes occur immediately
 - `TCSADRAIN` – Changes occur after all output to `fdes` is transmitted
 - `TCSAFLUSH` – Changes occur after all output to `fdes` is transmitted (all input received but not read is discarded)

struct termios

Programs access characteristics through the struct termios structure with 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 characters */

c_cc array

- Holds values of characters that have special meaning to terminal device drivers
- i.e., end of input or program break characters

POSIX Special Control Characters

canonical mode	Noncanonical mode	description	Usual default
VEOF		EOF character	Ctrl-D
VEOL		EOL character	none
VERASE		ERASE character	Backspace or delete
VINTR	VINTR	INTR character	Ctrl-C
VKILL		KILL character	Ctrl-U
	VMIN	MIN value	1
VQUIT	VQUIT	QUIT character	Ctrl-\
VSUSP	VSUSP	SUSP character	Ctrl-Z
	VTIME	TIME value	0
VSTART	VSTART	START character	Ctrl-Q
VSTOP	VSTOP	STOP character	Ctrl-S

c_iflag

Controls the way a terminal handles input

field	flag	description
c_iflag	BRKINT	signal interrupt on break
	ICRNL	map CR to NL on input
	IGNBRK	ignore break condition
	IGNCR	ignore CR
	IGNPAR	Ignore characters with parity errors
	INLCR	map NL to CR on input
	INPCK	enable input parity check
	ISTRIP	strip character
	IXOFF	enable start/stop input control
	IXON	enable start/stop output control
	PARMRK	mark parity errors

c_oflag

Controls the way a terminal handles output

c_oflag	OPOST	postprocess output
	OCRNL	map CR to NL on output (POSIX:XSI Extension)
	ONOCR	no CR output at column 0 (POSIX:XSI Extension)
	ONLRET	NL performs CR function (POSIX:XSI Extension)

c_cflag

Specifies hardware control information for the terminal

c_cflag	CSIZE	character size (CS5-CS8) for 5 to 8 bits, respectively)
	CSTOPB	send two stop bits, else one
	CREAD	enable receiver
	PARENB	enable parity
	PARODD	odd parity, else even
	HUPCL	hang up on last close
	CLOCAL	ignore modem status lines

c_lflag

Controls the editing functions of the terminal

c_lflag	ECHO	enable echo
	ECHOE	echo ERASE as an error-correcting backspace
	ECHOK	enable KILL
	ECHONL	echo a newline
	ICANON	canonical input (erase and kill processing)
	IEXTEN	enable extended (implementation-defined) functions
	ISIG	enable signals
	NOFLSH	disable flush after interrupt, quit, or suspend
	TOSTOP	send SIGTTOU for background output

Setting Flags

- Set an action by performing a bitwise OR of the appropriate flag
- Clear an action by performing a bitwise AND with the complement of the flag
- The following code clears the ECHO flag:

Struct termio term:

```
Term.c_flag &= ~ECHO
```

ttysetchar.c

```
#include <termios.h>
#include ...
int ttysetchar (int fd, int flagname, char c) {
    int error;
    struct termios term;
    if (tcgetattr(fd, Yterm) == -1)
        return -1;
    term.c_cc[flagname] = (cc_t)c;
    while (((error = tcsetattr(fd, TCSAFLUSH, &term)) = -1)    && (errno ==
        EINTR));
    return error; }
...

if (ttysetchar(STDIN_FILENO, VEOF, 0x07) == -1)
    perror("Failed to change end-of-file character");
```

setecho.c

```
#include <termios.h>
#include ...
#define ECHOFLAGS (ECHO | ECHOE | ECHOK | ECHONL)
int setecho (int fd, int onflag) {
    int error;
    struct termios term;
    if (tcgetattr9fd, &term) == -1)
        return -1;
    if(onflag)
        term.c_lflag |= ECHOFLAGS;        /* turn echo on */
    else
        term.c_lflag &= ~ECHOFLAGS;      /* turn echo off */
    while (((error=tcsetattr(fd,TCSAFLUSH,&term))== -1)&&(errno==EINTR));
    return error; }
...
setecho(STDIN_FILENO, 0);
```

Input Processing

- Keyboard and Screen are not directly connected
- They are separate devices
- Device drivers receive bytes from keyboard
- Bytes are buffered and edited as specified by settings for these devices

Canonical Input

- The usual method of handling terminal input
- Process input one line at a time
- POSIX special control characters (table 6.1) are used to terminate input or perform simple editing
- A line is a sequence of bytes delimited by newline, end-of-file, or end-of-line
- Read requests do not return until user enters line delimiter (or process receives a signal)
- Erase and kill characters work
- If defined, lines cannot be longer than `MAX_CANON`
- Intended for character special files, not block special files

Noncanonical Input

- Input is not assembled into lines
- Device driver does not respond to ERASE and KILL characters
- Has two controlling parameters
 - MIN – Controls the smallest number of bytes to be gathered before read returns
 - TIME – Refers to a time with a 0.1 second granularity used for timing out bursty transmissions

MIN/TIME Definitions

case	meaning
MIN > 0, TIME > 0	If TIME expires or MIN bytes are received, read is satisfied
MIN > 0, TIME = 0	read blocks until at least MIN bytes received
MIN = 0, TIME > 0	read is satisfied when a single byte arrives or TIME expires
MIN = 0, TIME = 0	minimum of bytes requested or bytes available are returned

MIN/TIME Examples

Assume 5 characters have been typed and you try to read 10 bytes from standard input

- a. MIN=5, TIME=0 – receive 5 bytes immediately
- b. MIN=0, TIME=100 – receive 5 bytes immediately
- c. MIN=20, TIME=100 – receive 5 bytes after 10 seconds
- d. MIN=3, TIME=100 – receive 5 bytes immediately
- e. MIN=20, TIME=0 – block until at least 5 more characters are entered
- f. MIN=0, TIME=0 – receive 5 bytes immediately