Lecture #14 – Terminal I/O (Chapter 18)

- Overview

2 different modes for terminal I/O:

1. Canonical mode - terminal input is processed line at a time
2. Non-canonical mode - input characters are not assembled into lines.

Canonical mode is the default.

Note: see Figure 18.1 on page 632 for overview of terminal input and output queues.

There is an implied link between input and output queues if echo is turned on.
The size of the input queue (MAX_INPUT) may be finite.
There is another limit (MAX_CANON), which is the max canonical line length.
The output queue is also of finite size, but there is no constant telling its length.

Most UNIX systems implement all the canonical processing in a module called the **terminal line discipline**. (Note: Figure 18.2 on page 633)

All characteristics of the terminal are defined in <termios.h> with the termios structure.

```
struct termios {
        tcflag_t        c_iflag;            /*  input flags */
        tcflag_t        c_oflag;            /*  output flags */
        tcflag_t        c_cflag;            /*  control flags */
        tcflag_t        c_iflag;            /*  local flags */
        cc_t            c_cc[NCCS];         /*  control characters */
};
```

Input flags control input of char (strip 8th bit, parity checking, etc).
Output flags control output of char (output processing, map \n to CR/LF, etc.)
Control flags affect the RS-232 serial lines (stop bits, ignore status, etc.)
Local flags affect the interface between the driver and the user (echo, erase, etc.)
The c_cc array contains a mapping for the special control characters (ctrl-C, etc.)

Note: Figure 18.3 on page 635-636 for a listing of the different flags.
Note: Figure 18.4 on page 637 for a listing of functions to affect these flags.

Function	**Description**
Tcgetattr	Fetch attributes (termios structure)
Tcsetattr	Set attributes (termios structure)
Tcdrain	Wait for all output to be transmitted
Tcflush	Flush pending input/output

- Special Input Characters

POSIX.1 defines 11 different characters that are handled specially on input.
SVR4 adds another 6 while 4.3+ BSD add 7.

Note: Figure 18.9 on page 638 for a listing of these characters.

Most of these characters can be changed to whatever we like (except CR & NL)
See discussion for each character on pages 640-642

Example:

```
int main(void)
{
        Int             n;
        struct termios  term;
        long            vdisable;
        char            buf[256];

        if (isatty(0) == 0)
        {
                fprintf(stderr, "stdin is not a terminal\n");
                exit(1);
        }

        if ((vdisable = fpathconf(0, _PC_VDISABLE)) < 0)
        {
                fprintf(stderr, "vdisable not available\n");
                exit(1);
        }

        if (tcgetattr(0, &term) < 0)
        {
                perror("tcgetattr");
                exit(1);
        }

        term.c_cc[VINTR] = vdisable;        /* disable INTR */
        term.c_cc[VEOF] = 2;                /* EOF is now ctrl-B */

        if (tcsetattr(0, TCSAFLUSH, &term) < 0)
        {
                perror("tcsetattr");
                exit(1);
        }
        while ((n = read(0, buf, 128)) > 0)
                write(1, buf, n);


        exit(0);
}
```

- Get / Set Terminal Attributes

```
int     tcgetattr(int filedes, struct termios *termptr);
int     tcsetattr(int filedes, int opt, const struct termios *termptr);
```

If called on non-terminal device, error is returned with errno set to ENOTTY.

The "opt" argument for tcsetattr can be:

TCSANOW	Change occurs immediately
TCSADRAIN	Change occurs after all output has been transmitted
TCSAFLUSH	Change occurs after all output has been transmitted, and all unread input data is discarded.

Note: tcsetattr returns OK if it was able to perform ANY of the requested actions. It is the responsibility of the programmers to check for actual completion.

- Terminal Option Flags

As mentioned before, there are many terminal options (see pages 645-651 for details).

Some notable entries:

Flag	**Description**
ECHO	If set, input char are forward to output
ECHOE	If set and ICANON is set, the ERASE character erases the last character in the current line from the display.
ICANON	If set, canonical mode is in effect which enables EOF, EOL, EOL2, ERASE, KILL, REPRINT, STATUS, WERASE. The input characters are assembled into lines.
ISTRIP	If set, valid input bytes are stripped to 7 bits.
PARENB	If set, parity generation is enabled for output char, and parity checking is performed on incoming char
PARODD	If set, odd parity and even otherwise

- stty command

  The stty command is an interface to examine and modify the terminal options we have been discussing (implemented with functions like tcgetattr, and tcsetattr).

  "stty -a" displays all the terminal options (see output on page 343. Note that options that begin with a "-" are disabled).

- Baud Rate Functions

```
speed_t     cfgetispeed(const struct termios *termptr);
speed_t     cfgetospeed(const struct termios *termptr);

int         cfsetispeed(struct termios *termptr, speed_t speed);
int         cfsetospeed(struct termios *termptr, speed_t speed);
```

  These functions query and control input and output rates for the terminal device.

  speed_t values can be constants like B300, B1200, B2400, B9600, B19200, etc.

- Line Control Functions

```
int    tcdrain(int filedes);
int    tcflow(int filedes, int action);
int    tcflush(int filedes, int queue);
int    tcsendbreak(int filedes, int duration);
```

  tcdrain function waits for all output to be transmitted

  tcflow function gives us flow control over input and output; action argument can be:

TCOOFF	output is suspended
TCOON	output is restarted
TCIOFF	system transmits a STOP character
TCION	system transmits a START character

  tcflush lets us flush ("throw away") either input or output buffers; queue can be:

TCIFLUSH	input queue is flushed
TCOFLUSH	output queue is flushed
TCIOFLUSH	both input and output are flushed

  tcsendbreak transmits a continuous stream of zero bits for a specified duration

- Terminal Identification

```
char    *ctermid(char *ptr);
```

ctermid returns the name of the controlling terminal (usually /dev/tty).

```
int     isatty(int filedes);
char    *ttyname(int filedes);
```

Note: see implementations of these functions on pages 346-348)

Example:

```
int main(void)
{
    char        *ptr;

    if (isatty(0) == 0)
    {
        fprintf(stderr, "stdin is not a terminal\n");
        exit(1);
    }

    ptr = ttyname(0);

    fprintf(stderr, "%s\n", ptr);

    exit(0);
}
```

- Canonical Mode

Canonical mode is pretty straight forward - we issue a read and the terminal drive returns when a line has been entered. Several conditions cause the read to return:

1. When the requested number of bytes has been read.
2. When the line delimiter is encountered (NL, EOL, EOL2, EOF)
3. When a signal is caught, but read is not automatically restarted.

Example (getpass):

```
char *mygetpass(const char *prompt)
{
    static char         buf[MAX_PASS_LEN + 1];
    char                *ptr;
    sigset_t            sig, sigsave;
```

```
        struct termios      term, termsave;
        FILE                *fp;
        int                 c;

        if ((fp = fopen(ctermid(NULL), "r+")) == NULL)
                return NULL;
        setbuf(fp, NULL);

        sigemptyset(&sig);    sigaddset(&sig, SIGINT);    sigaddset(&sig, SIGTSTP);
        sigprocmask(SIG_BLOCK, &sig, &sigsave);

        tcgetattr(fileno(fp), &termsave);     term = termsave;
        term.c_lflag &= ~ (ECHO | ECHOE | ECHOK | ECHONL);
        tcsetattr(fileno(fp), TCSAFLUSH, &term);

        fputs(prompt, fp);

        ptr = buf;
        while ((c = getc(fp)) != EOF && c != '\n')  {
                if (ptr < &buf[MAX_PASS_LEN])
                        *ptr++ = c;
        }

        *ptr = 0;
        putc('\n', fp);

        tcsetattr(fileno(fp), TCSAFLUSH, &termsave);
        sigprocmask(SIG_SETMASK, &sigsave, NULL);
        fclose(fp);

        return (buf);
}

int main(void)
{
     char    *ptr;

     ptr = mygetpass("Enter a password:");
     fprintf(stderr, "Password was %s\n", ptr);

     return(0);
}
```

- Non-canonical Mode

Noncanonical mode is specified by turning off the ICANON flag in "c_lflag" of termios.

Input data under noncanonical mode is not assembled into lines.

In addtion, the characters ERASE, KILL, EOF, NL, EOL, EOL2, CR, REPRINT, STATUS, and WERASE not processed.

Two variables (MIN, TIME) in the "c_cc" array are used to determine what input to return to caller.

Case A: MIN > 0, TIME > 0

> TIME specifies a inter-byte timer that is started only when the first byte is received. If MIN bytes are received before the timer expires, read returns MIN bytes.

Case B: MIN > 0, TIME == 0

> Read does not return until MIN bytes have been received (can block).

Case C: MIN == 0, TIME > 0

> TIME specifies a read timer that starts when "read" is called. Read returns when a single byte is received or when the timer expires.

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

> If some data is available, read returns up to MIN bytes, otherwise returns immediately.

Example:

```
static struct termios     save_termios;
static int                ttysavefd = -1;
static enum { RESET, RAW, CBREAK } ttystate = RESET;

int tty_cbreak(int fd)
{
        struct termios  buf;

        if (tcgetattr(fd, &save_termios) < 0)
                return(-1);

        buf = save_termios;
        buf.c_flag &= ~ (ECHO | ICANON);          /*  turn off ECHO and ICANON */

        buf.c_cc[VMIN] = 1;                       /*  one char at a time, no timer */
        buf.c_cc[VTIME] = 0;

        if (tcsetattr(fd, TCSAFLUSH, &buf) < 0)
                return(-1);

        ttystate = CBREAK;
        ttysavefd = fd;
        return (0);
}

int tty_raw(int fd)
{
        struct termios  buf;

        if (tcgetattr(fd, &save_termios) < 0)
                return(-1);

        buf = save_termios;

        /*  Turn off echo, canonical input, input processing, signal processing */
        buf.c_lflag &= ~ (ECHO | ICANON | IEXTEN | ISIG);

        /*  No SIGINT on BREAK, CR-NL off, parity off, 8-bit, no flow control */
        buf.c_iflag &= ~ (BRKINT | ICRNL | INPCK | ISTRIP | IXON);

        /*  8 bit, no parity */
        buf.c_cflag &= ~(CSIZE | PARENB);
        buf.c_flag |= CS8;

        /*  Output processing off */
        buf.c_oflag &= ~(OPOST);

        buf.c_cc[VMIN] = 1;          buf.c_cc[VTIME] = 0;

        if (tcsetattr(fd, TCSAFLUSH, &buf) < 0)
                return(-1);

        ttystate = RAW;         ttysavefd = fd;
        return(0);
}

int tty_reset(int fd)
```

```
{
        if (ttystate != CBREAK && ttystate != RAW)
                return(0);

        if (tcsetattr(fd, TCSAFLUSH, &save_termios) < 0)
                return(-1);

        ttystate = RESET;
        return(0);
}

void tty_atexit(void)
{
        if (ttysavefd >= 0)
                ttyreset(ttysavefd);
}

int main(void)
{
     char    buf[128];

     fprintf(stderr, "Enter something: ");

     read(0, buf, 1);
     write(1, buf, 1);

     tty_cbreak(0);

     fprintf(stderr, "Enter something else: ");

     read(0, buf, 1);
     write(1, buf, 1);

     tty_reset(0);

     fprintf(stderr, "\n");
}
```

Note: see program 18.21 on page 669 to exercise this routines.

- Terminal Window Size

  Note: It is possible to track changes to the terminal window size (especially helpful if writing an application like an editor…)

```
struct winsize {
        unsigned short      ws_row;
        unsigned short      ws_col;
        unsigned short      ws_xpixel;
        unsigned short      ws_ypixel;
};
```

1. We can fetch the current value of the structure with ioctl.
2. We can store a new value of structure in kernel using ioctl with TIOCSWINSZ (will cause SIGWINCH to foreground process group)
3. Interpretation of the values in the structure is the responsibility of the caller.

Note: See program 18.22 on page 671.