UNIX System Programming
Lecture 8: Terminals

• Outline
  • Introduction to Terminals
  • The Terminal Driver
  • The termios Routines

• Reference
  • BLP: Chapter 5
Lecture 8: Terminals
Introduction to Terminals

- UNIX systems transparently support full-screen applications (*emacs*, *top*, etc) on hundreds of terminals, both real (DEC, Tektronix) and emulated (xterm, putty, etc)

- Such programs require more than just a stdio interface to the screen. We need to position the cursor at any point on the screen. We also need to be able to read a single character at a time.
Lecture 8: Terminals
Introduction to Terminals

- There is a terminal driver between the program and terminal. The driver processes input and output. You must change the driver mode to read a character at a time.

- The `terminfo` database and routines provide a standard interface to terminals. We can use this interface to clear the screen, position the cursor, etc regardless of the type of terminal being used.
Lecture 8: Terminals
Is stdout a terminal?

- Use the `isatty(fd)` routine to determine if file descriptor `fd` is a terminal (`tty` device).
- **open** and **read/write** the `/dev/tty` device to communicate directly with the terminal even if stdin or stdout have been redirected. See `check_tty.c` for an example.
- In a shell script redirect **read/echo** from/to `/dev/tty` to communicate with the terminal.
Lecture 8: Terminals

Canonical Mode

- By default, the terminal driver is in standard or *canonical* mode and `read` will block until ENTER is pressed. Backspaces, etc are handled by the driver and do not reach the program.

- In *non-canonical* or *raw* mode we have much greater control over input processing and can read a single character at a time.
The termios Routines

- The `termios` routines (`tcgetattr()` and `tcsetattr()`) are used to get and set `terminal driver` modes (or attributes).
- You can change input, output, control, and local attributes. (See `man termios`)
- Your program should always restore terminal settings to the values they had before the program was run.
Use `tcgetattr()` to get current attributes:
```
tcgetattr(int fd, struct termios *t);
```

Use `tcsetattr()` to set attributes:
```
tcsetattr(int fd, int act, struct termios *t);
```

`act` controls when the changes are applied. It may be `TCSANOW` (apply now), `TCSADRAIN` (apply when output is complete), or `TCSAFLUSH` (apply when output is complete, flush input).
• The **termios** routines use a **termios** struct to get and set **modes**:

```c
#include <termios.h>
struct termios {
    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 */
};
```
• Input and output modes control how input and output are processed. Control modes specify hardware characteristics (parity, stop bits, etc). Local modes can turn on/off raw mode, echoing, and signal processing.

• Attributes are set or cleared by logically ORing constants defined in termios.h. See the passwd.c, menu4.c, and whichchar.c programs for examples of mode changes.
The control characters array of the `termios` struct allows us to control which characters cause the terminal driver to generate INT and QUIT signals and which characters are used for ERASE, KILL, EOF, etc. These are typically `^C, ^\, ^H, ^K, ^D`.

The TIME and MIN elements (see pg 190) are used in RAW mode and determine when characters are returned by a `read()`.
Lecture 8: Terminals
The stty Utility

- Use `stty` to control the terminal driver from the command line. Use "stty -a" to see the current `termios` settings.

- Using "stty -icanon min 1 time 0" in a shell script will cause a shell `read` to return a character at a time.

- Try typing 'reset' or 'stty sane' if your terminal starts acting strangely or locks up.
Lecture 8: Terminals
Using the terminfo Database

- Older programs used the `termcap` database (`/etc/termcap`) and library.
- Newer programs use the `terminfo` terminal capability database and libraries. Every terminal is described in a separate binary file under the `/lib/terminfo` directory. The `tic` program can generate database files for new terminal types.
Lecture 8: Terminals

Introduction to Terminals

- The **TERM** environment variable is used to indicate the type of terminal you have.
- Use **infocmp** to display the terminfo description of a particular terminal. (See **man 5 terminfo**)

```bash
> infocmp xterm
> infocmp vt100
```
Lecture 8: Terminals
Using the terminfo Database

- The **ncurses** library (install the libncurses5-dev package) provides routines for low-level access to the **terminfo** database. You must first call **setupterm**.

  ```
  setupterm(0, 1, (int *)0);
  ```

- The first arg is the terminal name (NULL to use TERM value), the second is the output file descriptor and the third is optional error return location.
Lecture 8: Terminals
Using the terminfo Database

- `tigetflag()`, `tigetnum()` and `tigetstr()` return bool, numeric and string capabilities.

- Refer to the `sizeterm.c` program to see how `tigetnum()` is used to get the terminal size. Link with the ncurses library (`-ln curses`).

- Use the `putp()` or `tputs()` routines to send control (escape) sequences to the terminal. (They will add any extra delay required by the terminal. `printf()` will not.)
Several of the string terminfo capabilities contain parameters (row or column number for example) that must be replaced before sending the escape sequence to the terminal. Use `tparm()` to do this.

Refer to the `menu5.c` program to see an example.

Other routines in the ncurses library provide a much high-level interface to the terminal.
Lecture 8: Terminals
Fun terminal stuff

- `/usr/games/*` (bsdgames package)
- `/usr/lib/ncurses/examples/*` (ncurses-examples package)
- `aview` (aview package)
- `bb` (bb package)