Chapter II

Curses Libraries Overview

Introduction

For this study, "Development of screen-based file utilities program on UNIX", the program was designed using a menu as the user interface. Thus, the routines and libraries for screen handling were already available.

The goal of this chapter is to investigate some standard tools for controlling the CRT/VDO terminal screens. The screen-handling tools come in the form of a C library called "curses" which is very important to any C programmer writing interactive screen-based programs under UNIX.

1. Curses

This is the highest level of screen control. The library allows the programmer to control screens via a terminal independent data structure called a "window". This protects the programmer from the nitty-gritty of low-level terminal control, and for this reason its use is always preferred to low-level counter parts. When "curses" routines actually update the screen, they attempt to do it as efficiently as possible. Indeed, the library name "curses" is derived loosely from the expression cursor motion optimization. (Goodheart, 1991)

"curses" was first developed at the University of California Berkeley, and has a long history. However, it has only recently been adopted as a part of AT&T System V as a standard part of their version of UNIX known as the Berkeley Software Distribution (BSD). The current version is superset of the library that originated from Berkeley.

The new "curses" package provides even more capabilities, with ever more routines, and support for more terminals. It now includes support for alternate character sets and color, with even better optimization. (Goodheart, 1991)

The "curses" library provides terminal independence and a clean easy-to-work-with interface for the programmer. Moreover, the UNIX "curses" package not only provides the portable standard, but also provides an interface which is both complete and easy to use.

Curses Advantages

Since curses is a cursor optimization utility, it will minimize the amount of cursor movement around the screen when it is updated. The following list shows some advantages of curses : (UNIX System V/386 release 3.2 Programmer's Guide Volume II, 1988)

1. It saves the time describing in a program when the program is updated.
2. It saves a user's time when the screen is updated.
3. It reduces the load on the system's communication lines when the screen is updated.
4. Programmers do not have to worry about the myriad of terminals on which the program might be run.

Overview Feature and Data Structure of Curses Libraries

To master "curses" routines, a programmer needs to understand the "curses" data structure known as a WINDOW. A "window" is an internal data representation of an image of what a particular rectangular section of the terminal display may look like. Its dimensions are defined by its outermost extremities, those being the sides of the physical screen. In contrast, its dimensions can be represented as a one character in length and one character in height. This is the smallest window --the size of one character.

"curses" : The Data structure, which is an internal representation of a curses window, is defined in the "/usr/include/curses.h" include file and is typedefed WINDOW as follows (Goodheart, 1991) :

```
struct_win_st  {
    short    _cury, _curx;
    short    _maxy, _maxx;
    short    _begy, _begx;
    short    _flags;
    chtype   _attrs;
    bool     _clear;
    bool     _leave;
    bool     _scrool;
    bool     _use_idl;
    bool     _use_keypad;
    bool     _use_meta;
    bool     _nodelay;
    chtype   **_y;
    short    *_firstch;
    short    *_lastch;
    short     _tmarg,_bmarg;
};
typedef  struct_win_st  WINDOW;
extern   WINDOW  *stdscr, *curscr;
```

Figure 2.1 The WINDOW Data Structure

According to the window data structure in the standard include file : <curses.h> (as shows in figure 2.1), "curses" provides two WINDOW data structures -- virtual-window pointers, "curscr" and "stdscr". The "curscr" (current screen) window holds a data representation of what is currently displayed on the real terminal screen and it is generally reserved for internal "curses" use. While the window named "stdscr" (standard screen), which is the size of the current terminal's screen, is provided as a default which represents your terminal screen window for program to work with. Furthermore, characters can be written into this window at any position.

To deal with many screens, "curses" allows programmers to create new windows individually within the stdscr --standard screen by using the "curses" "newwin" routine.

All "curses" routines totally depend on this window structure. It is used to solely to hold data and information which describe a window, and is used to build a potential image of a portion of the true terminal screen.

Curses-based Program

General curses program structure is shown below:

```
#include <curses.h>

main()

    initscr();

    /* main body */

    endwin();
    exit(0);
```

Figure 2.2 Program Structure

When writing a curses-based program, the programmer needs to include the header file <curses.h> which, on System V, automatically includes the terminal header files <stdio.h>, <termio.h> and <unctrl.h>. While, if running a BSD or XENIX system, then <sgtty.h> is included also. (Goodheart, 1991)

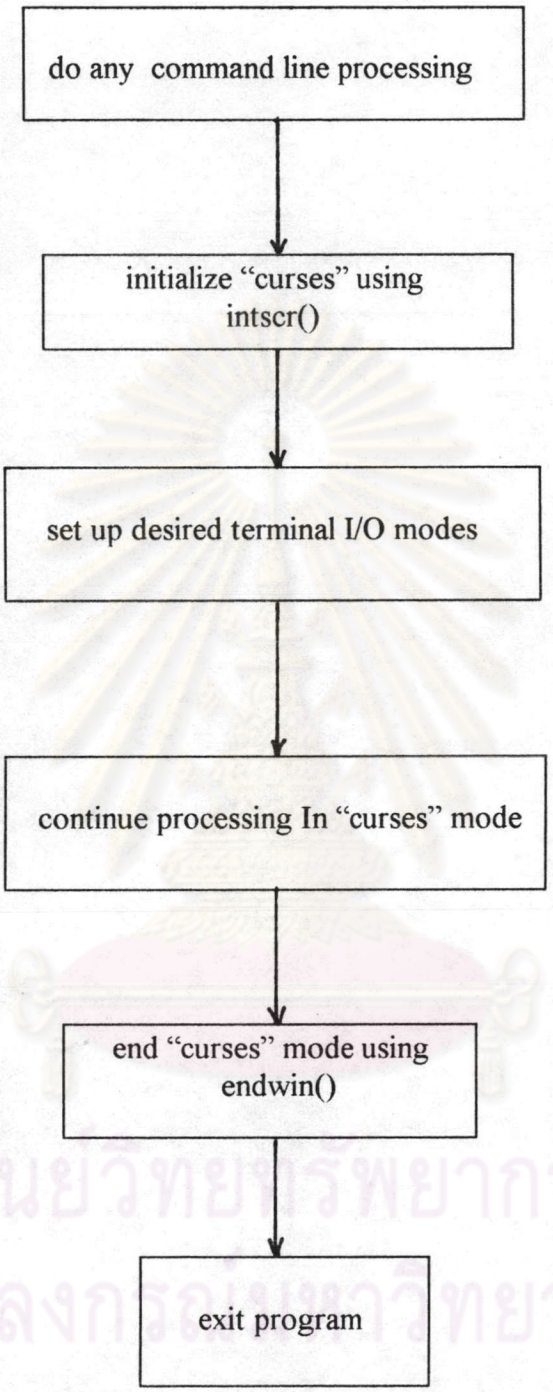

Figure 2.3 The Basic Structure of a Curses Program

As presented in figure 2.2 and 2.3, the initscr() routine must be called prior to any other curses function. It initializes certain curses data structures and also determines the type of terminal from the "TERM" variable in the environment. Similarly, the endwin() function should always be called before the program exits. It will restore the original terminal state, tell curses to set the terminal driver back into and out-of-curses mode and move the terminal's cursor to the lower left-hand corner.

The other functions concerned with curses-based programs are shown below:

refresh()
wrefresh(win)

These two functions are used to update the "stdscr" and the window pointed to by "win" respectively. When refresh() or wrefresh() is called it compares the two screen images (terminal screen and "stdscr" or window) and sends a stream of characters to the terminal that make the physical screen look like stdscr or window

wnoutrefresh(win)

This function updates the curses virtual screen "curscr" with the contents of the window "win". No actual update is done to the physical terminal screen until doupdate() is called.

doupdate()

This function compares the virtual screen (curscr) to the physical screen. It updates the part of the physical screen that has changed since last being updated.

Facilities Provided by "curses"

Curses provides many functions which are specifically set up to deal with the "stdscr" window. Furthermore, to deal with specific window, "curses" provides function names which are consistently prefixed with "w". (Curses Library Manual)

These functions can be classified as follows :

1. Mode Setting

After the call to initscr(), a program that calls curses routines will usually set the terminal mode for input and output. Thus, curses provides a comprehensive set of routines for setting and reading the terminal driver modes.

keypad(win,bf)

This function allows the programmer to toggle the function keypad on or off. If "bf" is TRUE, this function will enable the keypad. By default the keypad is disabled: this means the function getch() will not treat special keys (such as arrow or function keys) any differently from ordinary alphanumeric keys on the keyboard. However, if the keypad is enabled, getch() will return an integer representation of the character sequence received.

cbreak()

Sets the terminal into cbreak mode. This means that characters typed at the terminal are not buffered by the tty driver, they become immediately available to the program as they are entered at the keyboard, without having to wait for a line delimiter. The tty driver continues to process interrupt, quit, start and stop characters, but canonical processing (erase and kill) is turned off.

nocbreak()

Resets the terminal out of cbreak mode. Characters entered at the keyboard are now buffered by the tty driver, and input is processed in units of lines.

noecho()

This mode disables echoing back to the terminal screen. The driver is set into half-duplex mode. Characters are not added to the window by wgetch()

echo()

Enables echoing back to the terminal screen. The driver is set into full-duplex mode. Used in conjunction with wgetch().

If echo is enabled, then wgetch() automatically adds characters typed at the keyboard to the working window. When the window is refreshed the characters read by wgetch() are sent back to the terminal.

nonl(), nl()

Both of these functions control the translation of a new-line. The default mode is nl() which translates a new-line into carriage-return and linefeed on output, and return is translated to new-line on input. The function nonl() disables this translation.

savetty()

This function saves the settings of the current terminal (tty) modes in an internal buffer. It is automatically called by initscr().

resetty()

This function restores the terminal (tty) modes to what was originally saved in the internal buffer, set by savetty().

reset_shell_mode()

Sets current terminal modes into an out-of-curses state. This routine restores the tty driver to the mode previously saved before entering the curses program.

reset_prog_mode()

This resets the tty driver back into an in-curses mode.

2. Creating Windows

To cope with the user-interface design, curses provides a routine to the programmer for creating a new window placed on the terminal screen separate from the default windows supplied by curses (stdscr and curscr) by using the newwin() function displayed as follows :

```
WINDOW  *win;
int     nlines, ncols, begy, begx;

win     =   newwin(nline,ncols,begy,begx);
```

Variables used in newwin() routine are explained below :

nlines : The maximum vertical dimension of the new window, specified in units of lines

ncols : The maximum horizontal dimension of the new window, specified in units of columns.

begy : The line coordinate, specifying where the new window will start in relation to the "stdscr" vertical dimension.

begx : The column coordinate, specifying where the new window will start in relation to the "stdscr" horizontal dimension.

The system will create a new window the same size as the terminal screen if the arguments to newwin() are newwin(0,0,0,0) or newwin(LINES,COLS,0,0).

3. Adding Characters & Strings to the Standard Screen

Curses provides routines for the programmer to put characters and simple strings into both standard screen and a window which can be concluded as follows :

```
int       addch(ch)
chtype    ch;
```

This function puts the character "ch" into the stdscr at the current cursor position of the window and then the cursor position is advanced.

```
int       mvaddch(y, x, ch)
int       y;
int       x;
chtype    ch;
```

Like the addch() function, this function puts the character "ch" into the "stdscr" the programmer can specify the target position on the screen by moving the cursor to the specified y,x (row,col) before putting the character into the screen.

```
int       waddch(win, ch)
WINDOW    *win;
chtype    ch;
```

This functions used for putting the character "ch" within window "win" created by newwin() functions. The following function, is used to put a character into the window also, but it can be specify the position (y,x) on the screen.

```
int       mvwaddch(win, y, x, ch)
WINDOW    *win;
int       y;
int       x;
chtype    ch;

int       addstr(str)
char      *str;
```

This function writes all characters of the null-terminated character string "str" on the standard window --stdscr.

```
int     mvaddstr(y, x, str)
int     y;
int     x;
char    *str;
```

This function provides the coordinates y,x which can be specified on the screen before writing the string "str".

```
int     waddstr(win, str)
WINDOW  *win;
char    *str;
```

This function adds a string pointed to string "str" to the specified window "win" at the current y,x coordinates. Whereas, the next functions provides variables y,x where the string can be placed on the screen.

```
int     mvwaddstr(win, y, x, str)
WINDOW  *win;
int     y;
int     x;
char    *str;
```

4. Printing Formatted Output

```
int     printw(fmt [,arg]...)
char    *fmt;

int     mvwprintw(y, x, fmt, [,arg]...)
int     y;
int     x;
char    *fmt;

int     wprintw(win,fmt [,arg]...)
WINDOW  *win;
char    *fmt;

int     mvwprintw(win,y x,fmt,[,arg]...)
WINDOW  *win;
int     y;
int     x;
char    *fmt;
```

The four functions above are used for printing formatted output. They convert, format and print the arguments "arg", under control of the format argument "fmt", into the standard screen and also specified window pointed to by "win".

5. Moving Around a Window

```
int     move(y,x)
int     y;
int     x;
```

The first function of cursor movement is used to control the movement of the cursor on the stdscr screen by specifying the new coordinates y,x (row,col) relative to the top left-hand corner. Nevertheless, the physical cursor is not actually moved unless the screen is updated or refresh() is called.

```
int     wmove(y,x)
WINDOW  *win;
int     y;
int     x;
```

This function does the same as move(), but it changes the current cursor to the location y,x in the window "win".

```
getyx (win,y,x)
WINDOW  *win;
int     y;
int     x;
```

This function is used to obtain current cursor y,x coordinates of the cursor in the window pointed to by "win". The y,x coordinates are relative to 0,0 of stdscr.

6. Reading from The Keyboard

```
int     getch()

mvgetch(y,x)
int     y;

int wgetch(win)
WINDOW  *win;

mvwgetch(win,y,x)
WINDOW  *win;
int     y;
int     x;
```

The above functions are used to read in a character from the terminal. Moreover, the functions beginning with "mv" can specify the y,x coordinates on the stdscr or window to get character. The first 2 functions deal with stdscr and

the others are associated with the window "win". Normally the program will hang until a character has been entered if in cbreak() mode, or after the first new-line if in nocbreak() mode. However, unless echo has been turned off with noecho(), the character returned will also be echoed into the designated window "win".

Curses provides the functions--getstr(), mvgetstr(), wgetstr() and mvwgetstr()-- which can make a series of calls to getch() and wgetch() until a new-line, carriage return, or the "Enter" key is received. The contents of the input string is placed into the reserved area pointed to by "str" which is assumed to be big enough to contain the resultant input string. If echo is turned on, characters are added to the stdscr or window "win". It can be demonstrated as follows:

```
int     getstr(str)
char    *char;

int     mvgetstr(y, x, str)
int     y;
int     x;
char    *str;

int     wgetstr(win, str)
WINDOW  *win;
char    *str;

int     mvwgetstr(win, y, x, str)
WINDOW  *win;
int     y;
int     x;
char    *str;
```

7. Editing on Existing Screen

Curses has functions which can edit text on the screen, clear whole areas of the screen, delete text, non-destructively insert text on screen and also rearrange the screen as follows :

```
void    erase()

void    werase(win)
WINDOW  *win;
```

To clear the window to blanks, curses provides functions erase() for stdscr and wrease() for specified windows. werase() resets the entire window pointed to by "win" to blanks, without setting the clear flag.

```
int     clear()

int     wclear(win)
WINDOW  *win;
```

Like erase() and werase(), these function are used to clear a window to blanks also. wclear() resets the retire window pointed to by "win" to blanks. It does this by calling the function werase() to erase the contents then sets the current y,x coordinates of the window to 0,0. However, the clear screen will be effective when screen is updated.

The following functions are used for deleting the character under the cursor. Each character after it on the current line is shifted left, and the last character becomes blank. The cursor position remains unchanged. mvdelch() is exactly the same as the others except that the cursor is first moved to the given coordinates.

```
int     delch()

int     mvdelch(y, x)
int     y;
int     x;

int     wdelch(win)
WINDOW  *win;

int     mvwdelch(win, y, x)
WINDOW  *win;
int     y;
int     x;
```

The parameter "win" in the above functions is the specified window pointed to and the parameters y,x represent the cursor position.

The following functions are used for inserting a character "ch" at the current position. All characters to the right of the cursor are moved along by one, making the last character on the current line disappear. The meaning of parameters are the same as the deleting functions.

```
int     insch(ch)
chtype  ch;

int     mvinsch(y, x, ch)
int     y;
int     x;
chtype  ch;
```

```
int     winsch(win,ch)
WINDOW  *win;
chtype  ch;

int     mvwinsch(win, y, x, ch)
WINDOW  *win;
int     y;
int     x;
chtype  ch;
```

8. Video Attribute

Based on the video attributes of the terminal, curses provides functions to display many styles of text. Programmers can exploit the following capabilities to draw a character on the terminal screen in a way that makes it stand out differently from other characters being displayed, styles such as highlight, bold and underline are available.

Curses has a set of functions provided specifically for turning attributes on or off and setting the current attributes for all text displayed on the screen. The current attributes of a window are applied to all characters written into a window either directly or indirectly. The following list presents the functions that manipulate the WINDOW structure variable _attrs for both "stdscr" and "window".

```
int     attron(attrs)
chtype  attrs;

int     attroff(attrs)
chtype  attrs;

int     attrset(attrs)
chtype  attrs;

int     wattron(win, attrs)
WINDOW  *win;
chtype  attrs;
```

The prior four functions set on the current attribute "attrs" in window "win".

```
int     wattroff(win, attrs)
WINDOW  *win;
chtype  attrs;
```

This functions turns off the current attribute "attrs" in window "win".

```
int     wattrset(win,attrs)
WINDOW  *win;
chtype  attrs;
```

This function sets the current attribute in window "win" to that given in "attrs".

In addition, curses provides functions which can turn on and off standout mode displayed on stdscr and specified window pointed to by "win" as follows:

```
int     standout()

int     standend()

int     wstandout(win)
WINDOW  *win;

int     wstandend(win)
WINDOW  *win;
```

The following are the attributes provided by curses.

A_STANDOUT	Curses refers to this attribute as being the terminal's in highlight mode.
A_REVERSE	Characters are displayed inverse (dark on a bright background).
A_UNDERLINE	Characters are displayed on the screen underlined.
A_BOLD	Text will be displayed bold.
A_BLINK	Text will be displayed blinking on and off.

9. Others Functions

```
beep()
```

This function signals the user at the terminal by sounding an audible alarm.

```
flash()
```

This function signals the user at the terminal by making the terminal screen flash.

Compiling Curses-Based Program

A curses program is compiled under UNIX like any other C program. The "cc" compiler is used as usual but the linker must be instructed to link the compiler out put with the curses libraries ("/lib/libcurses.a").

For example, to compile the program june.c you enter at the command line:

cc -o june.c june - lcurses