Chapter IV

The Design and Development Procedure

The development of file utilities in this thesis employed the capabilities of curses and UNIX shell command as previously explained for the following reasons.

1. All UNIX systems provide standard shell utilities for manipulating files and directories.
2. This program was designed to be portable to any UNIX like system.
3. The User-Interface program was design to be a terminal independent program

This design consists of two parts

1. User-Interface module
2. File Utilities module

1. The Design of User-Interface Module

1.1 Menu Interface Screen

This was designed to generate a user's menu choice by constructing curses-based program. This chapter will be restated "<curses.h>" libraries as follows:

1. Define portions of the terminal screen as windows.
2. Read input from a terminal screen into a program.
3. Write output from a program to the terminal screen.
4. Manipulate the information in a window in a virtual screen area and then send it to the physical screen.

Users can select the desired operation for mastering this program. They can manipulate and manage their files and directories following the suggestions on the screen. The goals of this design are "ease-of-use" and standardization for all menu screens. In other word, this design would like users to be familiar with all screens which have the same concept.

In brief, this menu design was composed of three parts (see screen illustration in Appendix A.)

1.1.1 Title topic shows the menu description.

1.1.2 Menu choice displays the operations choice for file(s) and directory(ies). The highlight attribute is set when the cursor is moved to the specified choice. Hence, it makes the choice stand out on the screen. When running this program, users see the Main Menu first. After that, if users would like to go further, they would see a different Sub-Menu screen. Although, users work with a Sub-Menu screen, they can quit from the program immediately and need not to go back to the previous menu.

1.1.3 Menu instructions on the bottom of the screen presents some directions for using program.

1.2 Operation Interface Screen

This was designed to wait for file and directory names entered by the user. The input from this screen will be passed to the program module which invokes the shell command. Moreover, the program shows error and warning messages on the screen if the process fails. Conversely, if the users enter the correct input the program shows a prompt message to confirm the success of the operation.

1.3 Output Screen

This shows output from the shell operation screen.

2. The Design of File Utilities Module

This was designed to accept a command, the directory name and the file name passed from the calling function. After that, the program will invoke a shell command according to the passed values and report the output after the process finishes. The subject of this file utilities program encompasses the following example:

1. Creating and sometime removing file(s).
2. Moving and renaming the files.
3. Opening and closing files used by the operation .
4. Transferring information from file(s) to file(s).
5. Finding or searching the target file or pathname for user.
6. Changing the permissions bit of the file.
7. Creating and editing the text file.
8. Viewing the text file.
9. Copying or deleting the file(s).
10. Compressing and uncompressing the file.
11. Canceling the program process.
12. Exiting to shell prompt (shell escape).
13. Monitoring disk used in the file system.

3. Program Procedure

This section describes the design process in the curses-based program development. The following pseudocode, in conclusion, demonstrates a step-by-step description of how a program was conducted and will show some of the functions used in this development.

1. main() function

This is the first function used for declaring and initializing variables used in this program. It calls "initscr()" function which set program to curses mode, and "main_menu()" function.

pseudocode

```
/* Main Program for calling Main Menu */

/* include all libraries used in this program
   and define constant value for used variable */

main()
{
     initscr();

     /* initialize all value use in this program */
     initialize();

     /* display main_menu() until quit program */
     do {
          main_menu();
     } while (loop);

     clear screen before exit;
     endwin();
     exit(0);
}
```

2. main_menu() function

This function displays the Main Menu screen and calls the following functions; the "highlight()" function for standout the selected choice, the "win_dir_menu()" function, and the "win_fle_menu" function to display Sub-Menu of directory and file operation respectively.

<u>pseudocode</u>

```
/* Display Main Menu :called form main()  */

main_menu()
{
    /* item number in each screen        */
    int  choice;

    display menu in this window;
    highlight the selected choice;

    /* Do menu selected*/
    switch (choice)  {

            case   1:
                /* call: Directory Operation Menu */
                win_dir_menu();
                break;

            case   2:
                /* call: File Operation Menu*/
                win_fle_menu();
                break;

            case   3:
                summarizes disk space on filesystem;
                break;

            case   4:
                shell escape;
                break;

            case   5:
                quit from program;
                break;

            default :
                show warning message;
    }
}
```

3. win_dir_menu() function

Because the "win_dir_menu()" function is similar to the "win_fle_menu()" function, the following example will describe only "win_dir_menu()" function to present the Sub-Menu procedure.

The "win_dir_menu()" function depicts the directory operation menu which shows the menu choice before calling the Operation screen. According to the menu choice, the user can use the arrow key or enter menu number to select the desired choice. Then, the highlight will apply to the selected choice. To accept a choice, press ENTER to continue.

<u>pseudocode</u>

```
/* Display & Action with Directory Menu
   : called from main_menu() */
win_dir_menu()
{
/* item number in each screen      */
int  choice;

do  {

        display menu in this window;
        highlight the selected choice;

        display "[Q]uit" message;

        /* Do menu selected*/
        switch(choice)  {
            case  1:
                /* Create new Directory*/
                win_mkdir_dmenu();
                break;

            case 2:
                /* Delete/Remove Old Directory */
                win_rmdir_dmenu();
                break;

            case 3:
                /* Rename Directory */
                win_renme_dmenu();
                break;
                case 4:
                     /* Search/Find Directory */
                     win_find_dmenu();
                     break;

                case 5:
                     /* Change permission of File */
                     win_chmod_dmenu();
                     break;

                case 6:
                     /* latest Access Time */
                     win_time_dmenu();
                     break;

                case 'Esc':
                     return to main_menu();
                     break;

                case 'q':
                     quit from program;
                     break;

                default:
                     show warning message;

                }
        } while (loop)
}
```

4. win_mkdir_dmenu() function

Like "win_dir_menu()", this function will be a representative example of all functions which are called from a sub-menu function. The "win_mkdir_dmenu()" will display an operation screen prompt for entering the target directory and file name before all values are passed to shell program. If a called function needs to display output on screen, this development performs a function to create a new window for displaying output from the shell program.

pseudocode

```
/* Create New directory module */
win_mkdir_dmenu()
{
    display the body of window;
    display text for entering directory name;

    wait for specified working directory;
    get working directory;

    if (no input or blank)
    default display the list of directories contained in working directory
    prompt for user to select the directory name;

    wait for the being created directory name;
    get the being created directory;

    if (no input or blank)  {
            show warning message;
    return;
    }

    show confirming message;

    /* invoke shell command "mkdir" */
    create new directory by invoke shell program;

    if (failure)
        show some suggestion message;
    show "Successful" message ;
    return to win_dir_menu();
}
```