Chapter I

Introduction

Background and Rationale

Nowadays computers are exploited as tools for managing and processing data which is very important in both the private and public sector. The use of computers is not limited to experts since there are so many application packages and utility programs in the marketplace helping users easily access the power of the computer. These applications act as a medium for quickly and easily commanding, controlling, and managing digital information between the user and the computer. Users need not understand or remember operation system commands. Users just decide what they want the computer to do for them, then use application package or utility program with which they are familiar to perform these tasks through a simple on-screen menu user interface.

The UNIX operating system was designed and developed by Ken Thompson and Dennis Ritchie at AT&T's Bell Laboratories. In 1973, Ritchie and Thompson rewrote the UNIX kernel in C, breaking from the tradition that system software was written in assembly language. With that rewrite, the system became essentially what it is today.

Around 1974 UNIX was licensed to universities for educational purposes and after that, it was ported to many different computers. A few years later it became available for commercial use. Since then, it has spread world-wide, with tens of thousands of systems installed, from microcomputers to the largest mainframes.

However, the designers of UNIX are themselves programmers, so they designed this system for programmers to use as an environment for software development. Thus, in the early day of UNIX, most of the users were experienced programmers who could remember the complex set of commands needed to control the operating system for their purposes. (Haviland and Salama, 1990)

Now, UNIX is installed and runs on a wide range of machine from microcomputers to workstations and even supercomputers. It is also widely used, for Internet network application. These days, everyone uses UNIX, from the most experienced programmers to the every-day end-user. Thus, it would be beneficial to have a simple user interface to the complex UNIX operating system, specifically one that could make file management easier for the end-user who cannot take the time to learn all the complex UNIX commands.

In order to help non-experienced users with managing their files utilities program is an answer. The users would not need to remember the complex commands of the UNIX operating system. They can only navigate arrow keys or select character on menu screen which is a user interface. Thus they can more effectively handle the UNIX operating system to do and to manage files as they want.

Theory used in this study

The design of this thesis consists of 2 parts.

1. User Interface Part

This part uses the Curses Library and the C Programming Language on UNIX.

A curses-based program under UNIX is totally independent of whatever type of terminal you are using. Moreover, a user can determine the size of the screen depending on the characteristics of the physical terminal which curses runs on. Furthermore, user can use default screen of curses packages. By using the curses package, a programmer can manipulate and control the screen via window structure which is an internal representation of an image of what a particular rectangular section of the terminal display may look like. (Goodheart, 1991)

The curses library provides a set of over 200 routines that enable you to manipulate the video attribute of a terminal screen and can also accept input from keyboard. The curses library can independently manage a several windows at the same time on the terminal screen. In addition, each window can contain one or many other windows called "sub-window" within themselves. The only limitation is the amount of memory available to the curses-based program.

1.1 Curses Capabilities

1. Adding a character to a window which can display text in a variety of different ways on the terminal screen such as bold text, underline text, blinking text and so on.
2. Moving a window to a new position within another window.
3. Reading from the keyboard and sending this input to a specific window.
4. Reading from a window, which read a character or string from the current y,x (row,column) coordinates in a window up to its maximum width.
5. Deleting character in a window.
6. Inserting characters into a window.
7. Overlaying windows.
8. Windows within windows creating a new window within a parent window.

9. Deleting a window, this process will delete a window and its contents.

10. Moving a window to a specified location.

2. File Utilities Program Part

This part uses system call capabilities and shell commands on the UNIX operating system for managing file. The program has access to Input/Output facilities and filesystem routines through system calls to the UNIX kernel.

Examples of system calls used for UNIX file access primitive are shown below:

open	opens a file for reading or writing
create	creates an empty file
close	closes a previously opened file
read	extracts information from a file
write	places information into a file
lseek	moves or repositions read/write file offset to a specified byte in a file
link	makes a new name for a file
unlink	removes or deletes a name of the file it refers to
chmod	changes the access permission of files
sync	flushs filesystem buffer

2.1 The UNIX Filesystem

Everything in the UNIX system is a file which is a sequence of bytes. (Kernighan, 1984) As shown in figure 1.1, the UNIX filesystem are attached to a hierarchy of directories. The combination of directories and files makes up a file system.

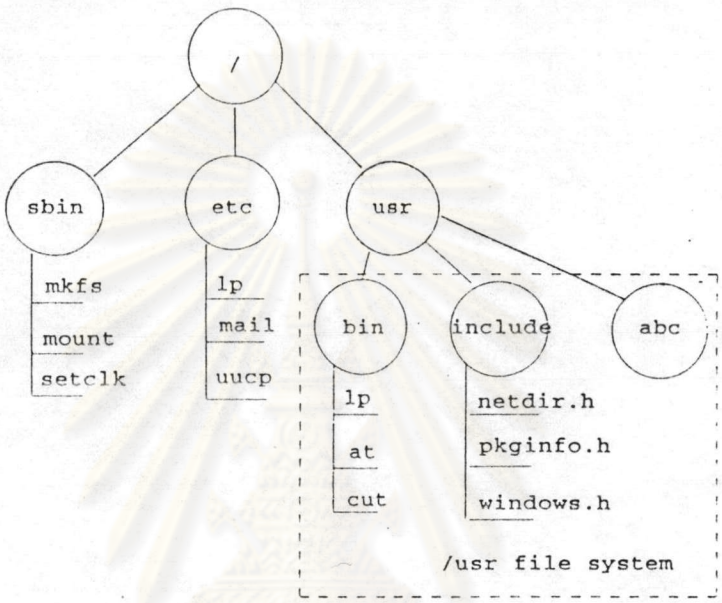


Figure 1.1 A UNIX File System

UNIX extends the file concept to cover the peripheral devices connected to a computer system. Peripheral devices such as printers, disk units and even main memory, are represented by a file name in the file structure. A file which stands for a device in this way is a special file which can be accessed via the file access system calls discussed above.

Concerning the use of UNIX file system, all users will be assigned a user-name and user-id (stored in "/etc/passwd") which identifies the user to the system. The fields in the "/etc/passwd" file are separated by " : " as follows :

june:1acPAkftBKzs:501:100:Priyakorn Pusawiro:/home/june:/bin/sh

The meaning of each field can be explained the following:

field#1 username
field#2 password which is shown in encrypted form
field#3 user-id
field#4 group-id of user
field#5 Optional comment field
field#6 user home directory
field#7 UNIX shell which shows the pathname of the program started after the user has logged in

When a user creates a file the system will determine the owner of file to be that user by reading a non-negative integer called the user-id (abbreviated to uid) which is stored by the system when the file is created. However, the ownership of a file can be later changed, but only by either the file's owner or the system's privileged superuser who has username as root and always has user-id 0.

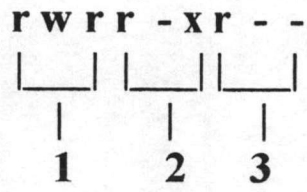
Each file in the UNIX filesystem has a set of permissions associated with itself. Permissions determine how different users can access a file. Three types of users are affected: (Haviland and Salama, 1990)

1. The file's owner.
2. Anyone who belongs to the same group as the one associated with the file. Note that, as far as the file's owner is concerned, the permissions for category (#1) override the permissions for the file's group.
3. Anyone who is not covered by categories (#1) or (#2).

For each category of user, there are three basic types of file permission. These specify whether a member of a particular category can:

1. Read the file.
2. Write to the file.
3. Execute the file. (In this case the file will normally contain a program or a list of shell commands.)

The system stores the permission associated with a file as a bit pattern within an integer called the file mode. For a user, the permissions are represented in a standard symbolic form, as used by the "ls" program. This notation represents permissions as a string of nine characters which can be devided into 3 groups. This can be made simpler by the following example string:



Type of users

1. For file owner
2. For file's group
3. For anyone else or other users

Type of permission

r - read permission
w - write permission
x - execute permission

If a permission bit is not set then the corresponding character is set to a hyphen ("-").

The except case is the superuser who can access all files in the filesystem including bit permissions.

Haviland and Salama (1990) show how the UNIX system uses the "inode" number, which is a 16-byte positive integer, to find each a file structure stored in disk unit. Each inode-number of a file stores file information such as file size, user-id, group-id, permission bit pattern, date of creation, last modified date and the disk addresses of the blocks on disk that hold the file data.

Therefore, when new file is created, the system will determine the inode to of the file by seeking the area to store the inode number from the free inode list (inode equal to 0). After the system find the free area, it will store the inode number and the name of file to that area. In contrast, if there is no area in the inode free list, the system will append the new inode to the old inode number in the directory. This appending will cause the inode number in the directory extended.

The Purpose of This Study

1. To study the work and the management process of the UNIX filesystem.
2. To design a single and effective user interface for users to access and manage files within the UNIX environment.

Scope and Limitations

1. To design and develop a user interface by using the Curses Libraries.
2. To design and develop a file utilities program which can
   2.1 Create a new file or directory.
   2.2 Delete an old file(s) or directory(ies).
   2.3 Rename a file or a directory name.
   2.4 Search and find a file or directory in the system by using some part of the name.
   2.5 Change file or directory permissions.
   2.6 Copy a file within the same directory or to another directory.
   2.7 View a text file.
   2.8 Create and edit a text file using the editor in the system.
   2.9 Compress and uncompress a file.
   2.10 Monitor the summary of disk space used in the UNIX filesystem.
3. This program development was conducted using the C Programming Language and other utilities on the UNIX system.

Procedures and Methods

1. The UNIX file system was investigated and studied.
2. The Curses Libraries was administered and reviewed.
3. The design of user interface was performed.
4. Implementation the file utilities program was set up.
5. Testing and monitoring the action of the file utilities.
6. Conclusions and recommendations.

Expected Advantages

1. Users, especially non-technical users, can easily and quickly manage and access the UNIX file system.
2. This study will be a guideline for further study and development of screen-based file utilities on the UNIX system in the future.