

# CS551

## Operation System Implementation & Design

### Project I

### SHELL

**Student Name:**

**Liubin Jiang A20309950**

**Linghan Zhao A20313766**

**Siwei Xiong A20316201**

**Notice: The screen snapshots are included into the text cases file**

#### **I. Introduction:**

In project I, we use the same configuration as in project0, which means that MINIX3 operating system is put into VMWare Workstation and WinSCP is used to modify its content. The purpose of this project is to design a simplified shell which can execute the basic commands of MINIX3 along with few new commands. The programs of this project will be put into MINIX3 and executed by Ash shell.

#### **II. Design:**

##### **1. Shell Invoked From the Ash Shell**

###### Design & Implementation

As it was explained in the introduction, our shell program is located in the directory /root/usr/, our shell folder's name is myshell (Figure 1). We use "cd" command to change directory to and "make" command to compile the new shell. Then we can execute the new shell from the Ash Shell.

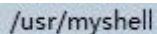


Figure 1

##### **2. HOME Directory and PATH Variables**

###### Function Description

When the shell starts, we should be able to change directory to HOME specified by user and get the PATH variables (/bin & /usr/bin) from PROFILE file.

###### Design & Implementation

The PROFILE file (profile) is located in shell directory which contains our HOME (/usr/myshell) and PATH (/bin & /usr/bin) value. In the beginning of our shell main function, a part of the code will read the content of profile, output the HOME variable and use the C function to change directory to HOME (chdir(home)). Then PATH variables are saved into a data structure (#include "shell.h" struct path\_info{ char\* path1, char\* path2};) and also printed for testing purpose.

###### Exception Handling

As PROFILE defines the initial directory and path variables which are critical to our shell, in the case where PROFILE is not found, we decide to output error to the user and exit directly

(exit(1)).

### 3. Prompt Display

#### Function Description.

When start or change directory of our shell, the prompt will display our shell name, host name and current directory.

#### Design & Implementation

A new function called prompt() in prompt.c is used to print the IP address and current directory in the following format '[shell name] user name @ host name: current path' (Figure 2).

In this function, we use the C function getpwuid(getuid()) to get user name and user's home directory at same time. Host name is obtained by function gethostname(). The C Function getcwd() will get user's current path directory. We can also distinguish root user from normal users, the root user will use '#' and normal users will use '\$', this is decided by function geteuid() to see if its value is 0.

A terminal window showing a shell prompt. The prompt is '[CS551\_shell]root@192.168.183.128:/usr#exit'. The text is white on a black background.

Figure 2

### 4. Use commands provided in /bin and /usr/bin:

#### Design & Implementation

We have already saved the PATH variables into path\_info data structure, their values are /bin and /usr/bin.

There are actually a few kinds of functions that allow us to execute a certain command:

```
int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execlxe(const char *path, const char *arg, ..., char * const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
```

Here we decide to use execvp because its default paths are /usr/bin and /bin, so even if the paths are variables are not specified, it will still use the commands in these 2 directories.

### 5. Indirection Operator:

#### Function Description

This operator allows us to save the output of a command into a file whose name is defined by the user.

#### Design & Implementation

All the commands are converted into parameters by read\_command() function, parameters are separated by space. As we need to use the output of a command instead of putting it directly onto screen, we need to redirect the output in a file (e.g. ls => FILES). In the parsing function, we detect if there is an indirection operator => and mark the flag in the parse\_info data structure as OUT\_DIRECT and save the parameter (e.g. FILES) into out\_file data, which allows the main function to construct a pipe to redirect the output into a file. The construction details of the pipe can be seen in shell.c under the OUT\_REDIRECT condition.

#### Exception Handling

In the case where the user forgets to put anything after this operator, we will output the error

and return to the shell prompt.

## 6. Output redirection & 2<sup>nd</sup> command execution:

### Function Description

This function is implemented with a “reversed” pipe as the example. We will use the second command output as the first command input and make sure that the two commands will both be executed. (e.g. ‘wc \$ fgrep malloc wshell.c’.)

### Design & Implementation

About this problem, when we meet a ‘\$’ as a parameter, we will flag PIPE and separate the command1, parameters1, command2 and parameters2. Then in function proc of wshell.c file, we use close() to close standard output of command2 and dup2() it to first command file descriptor and execute command2 first. We change the standard input of first command, wait and get the result of command2 in the same way. Then we run command1 to get the ideal result.

### Exception Handling

Users will be notified if the parameter after \$ does not exist.

## 7. Calculator and Variables.

### Function Description.

Our calculator is capable of dealing with two operands at the same time. The calculator uses variables which are defined by the user in the shell, for instance, we can calculate also a\*b or ab/cde if all the variables exist.

### Design & Implementation

User input a full expression, the program separate the expression into 3 parts, operate number1, operator and number 2.

Variable names and their actual value are storage in a single text file so that when shell exits, the variables will still persist.

Input variables are implemented by first check if the first value of the expression is symbol ‘\$’. (e.g. \$a=100) If the first character of the expression is ‘\$’, then we enter the writing mode. After writing, the system will automatically exit to reset pointers.

Also, calculate method is classified by operators, there are 4 cases to deal with each calculate method. The calculator can only be terminated by Ctrl+ C.

### Exception Handling

The second number of a division method cannot be 0 and an error will appear if a variable does not exist.