**CS551 Project 1 Minix Shell Implementation**

**Design Document**

**Team Members**

Balaji A R (A20347964)

Paulo Vitor Becker Ayrosa Monterio de Andrade (A20341532)

Sai Ravali Nannuru (A20354346)

**Flow Chart for our Shell**

Run the executable to launch a new shell

New shell will read .profile file to set HOME and PATH environment variables. Additionally TIMER value also read.

Display the shell prompt to get user input

If input has ‘\t’

Read history file to suggest command to user

Yes

No

If input has ‘=>’

Parse input command parameters to execute cmd and save o/p to specified file.

Yes

No

Split input to command arguments

If execution blocks

Initialize Block Timer

Save command to history file and Execute the Command

Discontinue…?

Block Execution

Continue Execution

No No

Yes

**New Shell Program breakdown:**

1. **.profile**

When our new shell starts to execute. Our program will read profile file to set HOME and PATH environment variables. Additionally it will read TIMER value. Once read successfully. It will display values in the command prompt.

1. **myshell.c**

This file contains the source code for the shell. It consists of the following functions :

* 1. get\_var():

This function reads. profile file and finds the value of the “TIMER”, “PATH”, “HOME” variable and returns it. This functionality is based on the parameters it receives.

* 1. clear\_line() :

This function clears the current prompt displayed to the user.

* 1. suggest() :

This function based on the user input, it will read cmd\_history file to suggest commands to user. In case user types second tab it will suggest next best match.

* 1. parse\_cmdline2():

Method to parse input line, change all possible separators into terminators (\0)

and subsequently.

* 1. read\_cmdline():

Read user input character by character and stores in the array. Once either tab or newline is encountered. The functions reads the array of characters, before return it will save the command to the history file.

* 1. parseArguments():

This function based on the arguments passed either it will invoke execCommands or exec\_io\_redirection.

* 1. sigchild\_handler(),child\_sigquit\_handler():

These functions are used to handle signals received from child processes.

* 1. execCommands():

Based on the input from parse\_cmdline2, this function creates a child process to execute the commands.

* 1. exec\_io\_redirection():

Based on the input from parse\_cmdline2, this function creates a child process to perform input redirection operation

* 1. main():

main functions allows users to input commands to our shell and provide necessary output after execution of the commands.

1. **To test timer functionality:**

Execute execshell and provide “cat” command as the input. The cat command will wait for the input parameters. Based on the timer value, our shell will prompt for user input to terminate the command or continue execution of the command.

**4. Exception handling during command execution:**

1. During fork operation child process creation failed: Collect return value of the fork function, display the error message accordingly.

2. After the execution of commands in the child process, get the return value to see whether the execution of commands are done successfully, else prompt proper error message.

3. If command execution takes more time, the timer will get expired and it will prompt user the option to terminate the command, based on user input it will either terminate the command or continue execution of the command.

**5. Test Plan and Test Cases:**

The following areas have been tested.

1. .profile file : The file should be read and set environment variables accordingly:
   1. TIMER : Change the TIMER values in the .profile file, see if it the timer value read is same as in .profile file
      1. By default timer value is set to 5.
      2. After the timer expires, it will ask user to choose option for program to continue execution or terminate the execution.
   2. PATH, HOME: Change the PATH and HOME key value in .profile file , then execute our shell to see the values read properly.

2. To test command suggestion: follow execution of below commands

#./execshell

$ mo

after typing mo, press tab to see the suggestion from history file. it will display “mount val”

3. To test input redirection:

# ./execshellio

-> ls => test

after the command execution the test file will be created in the current directory and cat the test file to see the contents matches the files in the current directory.

1. Shell functionality testing:

# ./execshellio

-> ls

after the command execution our shell will shows the list of files present in the current directory.

1. Type exit command to exit the shell.