OPERATING SYSTEMS

PROJECT2 REPORT

**Group Members :** Sevda DURDU – 150115043

Emre Barkın BOZDAĞ – 150115030

Aim of this project is that write a simple shell. The main() function of our program presents the command line prompt “myshell: ” and then invokes setup() function which waits for the user to enter a command. Given to us setup() function reads the user’s next command and parses it into separate tokens that are used to fill the argument vector for the command to be executed. It can also detect background processes. The contents of the command entered by the user is loaded into the args array.

For making 3 parts (and extra pipe part) we used these methods.

Firstly, we create 2 linked list which handles foreground and background processes. Actually for foreground linked list is unnecessary because there is only one foreground processes running. Reason of create that list is we need the use foreground list for making pipes. Because in pipes we need to wait processes after all processes run.

**callCommand() :**

We take argument array, background variable, input file, output file, file\_cond variable, previous pipe fd, next pipe fd.

In this method, the child will use the execl() function which means that you will have to read the PATH environment variable, then search each directory in the PATH for the command file name that appears on the command line in the below to execute a new program. For that search we use our checkPaths() method. When your program gets the program path(if exists), it will create a new process using fork() system call, and the new process will execute the program.

Before the calling execl we check input and output file. If these are exist we will redirect to input and output into these files. We did this with set\_io\_file() method. This method also check file\_cond variable to control output file flags(append, overwrite, error redirection).

Before the calling execl we also check prev and next pipe. If these are exist we will redirect the process input and output into that pipes. We did this with set\_pipe\_default() method.

Before the calling execl, we call set\_background() method. It checks the last argument of argument array, if it is equal to ‘&’ this function will make background variable null and erase that ‘&’ sign from argument array.

After calling execl , parent will wait childrocess. If background is 1 then parent won’t wait the child (using WNOHANG). It also add the process ids to background and foreground linked lists. We control this wait\_childpid() method. Therefore if we redirect input and outputs to pipes we won’t call wait\_chidpid() method because we don’t need to wait these processes. We will wait them after all of them start to run.

**alias\_list() :**

Firstly, we create alias\_list linked list. This list holds alias\_name and command. We implement the push() , deleteCommand(alias\_name),search() (by alias name) and print\_alias\_list() method. In push method we firstly call deleteCommand method for prevent duplication.

We take only input string variable which is all input line. And we take command name which is inside 2 double quotes. After that we take alias name which is located at after the last double quotes. We put ‘\n’ as a last character for command because when user want to run this command we will send this command into setup() function for get argument array. And lastly we push alias name and command into our alias linked list.

**my\_pipe() :**

Firstly we count total pipe signs in setup() function. With pipe\_check() function we check that count(if count is different it means user use “||” or “|||” etc. instead of single pipe sign). We separate each command and for each command firstly we call search for checking alias linked list. After that we call redirection() for any case of input or output sign existence. If there is no file sign, redirection still call callCommand() with given command. And inside callCommand() ,with set\_io\_file() method we redirect the input output and set\_pipe\_default() method will redirect the input and output to given pipes.

So our program can take command which includes pipes, input output sign and alias name.

**redirection() :**

We take argument array, background variable, previous pipe and next pipe. For determine input and output file existence we call check\_file\_signs() method. After that we separate command and input and output file names. In this separation we also check type of input output sign (like “<”,”>>”,”2>”). We determine the file\_cond from that sign(Ex for append file\_cond will be 1). After separation, we search command in alias linked list with search command. And finally we call callCommand() function. Inside callCommand() , with set\_io\_file() method we redirect the input output.

**PART A**

First question wants to take the command as input and will execute that in new process. For this part we call callcomand() method after setup function.

**PART B**

This part must support the following internal commands.

**alias/unalias :**

If user enters alias, we call alias\_list() method. And we sent the all input line which is given by user.

If user enters unalias, we firstly call search() method. If given alias name exist, we call deleteCommand() method to delete that record from alias linked list.

If user enters alias –l, we call print\_alias\_list() method.

For calling commands which pushed in alias\_list. We call search() method after setup() function. If search finds record it will edit the command variable and send that into setup function for getting argument array of that command. After that it will call callCommand() method the execute new argument array.

**^z :**

We create catchCtrlZ() method and it will run when SIGTSTP signal occurs. In that method we stop the running foreground process with kill function.

**clr :**

We call system(“clear”).

**fg :**

We wait all processes which inside in background processes list.

**exit :**

Firstly, we check background process list. If list is empty, we exit the program. If list is not empty, you cannot exit the program.

**PART C**

The part must support I/O redirection on either or both stdin and/or stdout and it can include arguments as well.

For this part, we call redirection() function.

**EXTRA PART**

Firstly we check pipe operator in setup function. If pipe operator exist in user input we call my\_pipe() function.

**OUTPUT SAMPLES**

