**CMPE 382 UNIX SHELL**

**(C Language)**

**Shell execution style:**

1. To execute commands in our shell every command must end with “**;**” OR “**|**”.
2. For example: **$ myShell:> quit** will not execute, to execute the quit command it has to be **$ myShell:> quit; OR $ myShell:> quit|**
3. Our shell can execute nearly all well-known commands except “cd / / /” in our testing.
4. Program will not cease to execute if (“**;;**” “**;|**” “**||**”) OR (“**; ;**” etc.) OR any command input without our dependency characters (“; |”).
5. Command line can be mixed with “**myShell:> ;;; ls ||| ls ;||;”** and execute the commands in between
6. Command arguments can have maximum 48 characters.
7. Command arguments and command and dependency characters can have whitespaces in between them. For example “**myShell:>** **ls ; ls -l | mv char.txt hi.txt ;** ”
8. After empty response **myShell:>** will not be printed but input can still be taken by the shell. If empty input such as enter is pressed again, **myShell:>** can be seen again.
9. Parallel and Sequential execution can be done in the same line. For example, **$ myShell:> ls; ls -l; echo “a” | ls;**
10. The **quit** command is always executed at the end in the sequential and parallel execution.
11. The unknown or inexecutable commands can be mixed with the executable commands and in return our shell is capable of returning inexecutable command names with error massage and proceeds to execute the executable commands without a problem.
12. Our shell is capable of batch mode and will execute lines one by one and echo each line to the user.
13. To be able to see all the lines executed **quit;** or **quit|** needs to be in the last line of batch script.
14. Running batch mode without quit will result in endless loop.

**Design overview:**

**Dependencies:**

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <sys/wait.h>

#include <sys/types.h>

#define  len 512

**Functions:**

char \*\*get\_input(char \*input)

Uses strtok and NULL check to eliminate the unwanted whitespaces between command and argument and limits the maximum argument characters to 48.

void Execute(char input[]){

void Execute handles all of the execution in the function. Takes a char Array as input and in that input, function looks at the characters one by one until it finds a “;” or “|”.

When it reaches the special character creates a string called “substr” with memcpy which is composed of characters previously passed. If the “substr” it gets until that point is nothing the program continues to the next while iteration to hande the “**;;**” etc. CASE1 figureA.

Otherwise, a for loop finds its length and adds it into the integer “substrL” for the previous “memcpy” operation to minus the whole length of the current char index (i) from the total passed character index (substrL).

Next operation is to feed the “substr” into the function char \*\*getinput() to have a parsed string array without extra whitespaces called “str”.

After having str[0] which is command, our program have series of if else’s for cases. Our first if handles the case when only whitespaces are in between special characters, for example when “**; ;**” is entered. This process is designed in a way that, even if the command is empty it creates a whole fork and execvp() procedures, only to have a failed execution of execvp() to maintain program integrity. After that exits the process with exit(1) CASE 2 figureA.

Second if else case detects the “quit” command. So that the “quit” substring does not go into the else case. This if also sets the “flag” integer to 1 to have a exit process later in the line CASE3 figureA.

Finally, the else case handles the substrings with characters in them whether they are genuine commands or not. Child process is created with fork(), and when then child process id is 0 it then proceeds to the execvp() to execute. If the execvp() is successful command result is printed to terminal and the code below the execvp will not run and no error message will show but if the execvp() gets a command it cannot execute, code under it will execute delivering error message and exiting the process with exit(1) CASE4 figureA.

Our body of void Execute has two of these if cases for detecting the special characters, while inside of these ifs are same the only difference between parallel and sequential execution is sequential has an extra else command near the execvp() to wait the child process, while parallel execution does not have wait() in it. This also allows us to switch between the parallel and sequential simultaneously if they are executed in the same lines of command CASE5 figureA.

After while loop is finished, we have an if case to detect whether the flag is 1 which means the quit was in the line of the commands. Exits the whole program with exit(0) CASE6 figureA.

int main(int argc, char\* argv[]){

Our main function detects if the command line is entered with an argument for the batch mode. If argv[1] is NULL then the program goes into interactive mode in a while(true) loop running Execute(input).

Else our program goes into batch mode reading line by executing each line with Execute(line) after each line is echoed with sprintf().

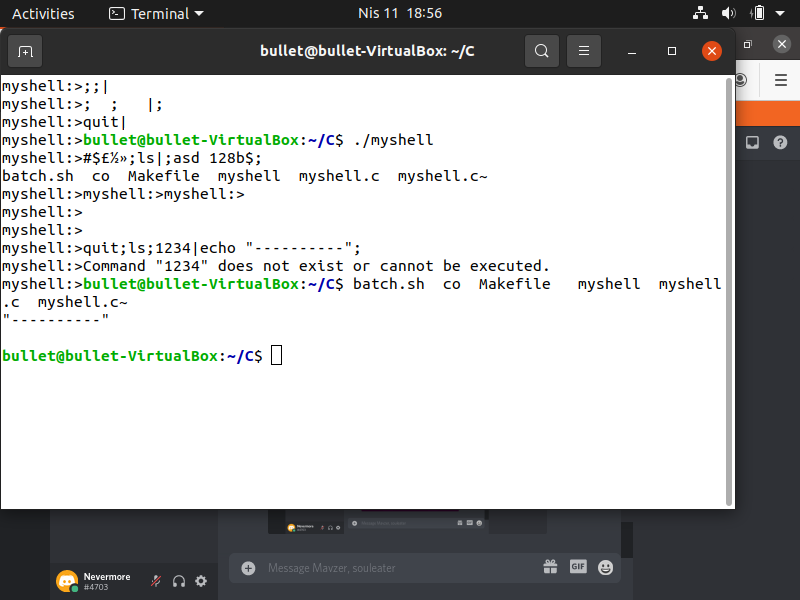
**KNOWN BUGS:**

Our testing resulted with no known bugs.

**PROBLEMS THAT MAY NEED FIXES:**

Batch mode will go into endless loop if no **quit|** or **quit;** is entered in anywhere in the lines.

**IMAGES:**

****

**FIGURE A**