CMSC 216 Exercise 5 Fall 2021

shell_jr ("Shellito") **See Piazz for due date**

1 Overview

The objective is to practice fork and exec* system calls by implementing a very simple shell. A shell is a C program that executes commands. There are many shells, including ksh, sh, bash and tcsh (which we have been using). The shell in this exercise is named "shell_jr".

There two types of commands a shell can process: linux commands and shell commands. Linux commands are programs that reside in directories like /usr/bin (eg, /usr/bin/ls). To execute a linux command, the shell forks itself and the child process loads the program using an exec* system call (eg, execvp). Shell commands (eg, cd, exit) do not require fork and exec calls. They are implemented in the shell code using system calls and other resources.

The starter files for this exercise are in the folder `shell_jr` in Grace 216public exercises directory. Copy the folder to your 216 directory. The submit file for this exercise is in that folder.

IMPORTANT: You can discuss this exercise with classmates, but you may not exchange code nor write code together.

2 Grading Criteria

Your assignment grade will be determined as follows:

Results of public tests	28%
Results of release tests	72%

3 Academic integrity statement

Please **carefully read** the academic honesty section of the course syllabus. We take academic integrity matters seriously. Please do not post assignment solutions online (e.g., Chegg, github) where others can see your work. Posting code online can lead to an academic case where you will be reported to the Office of Student Conduct.

4 Shell Jr Functionality

Your shell will have a loop that reads command lines and processes them. The prompt for your shell is "`shell_jr:  `". The commands your shell must handle are:

1. **exit**: When the user enters the **exit** command, the shell prints the message "See you" and terminates by calling exit().
2. **hastalavista**: Has the same functionality as **exit**.
3. **cd *dir***: This command changes the working directory to *dir*. Assume the user always provide a directory as an argument. Use chdir() to change the working directory.
4. A linux command with a maximum of one argument, such as "**pwd**", "**date**", "**wc location.txt**", "**cat location.txt**", etc.

5 Requirements

1. Do NOT use an exec* function to implement the functionality associated with the commands exit, hastalavista, and cd. For other commands, create a child (via fork()) and use execvp() to execute the command.

2. If the user provides an invalid command, print the message "Failed to execute " followed by the command name. In this case the child exits returning the error code EX_OSERR. Use printf to display the message and flush the output buffer (eg, fflush(stdout)). Note that the shell does not terminate when given an invalid command.

3. You don't need to handle the case where the user just types enter. You can assume the user always provides a command.

4. To print the shell prompt, use printf and flush the buffer.

5. It is your responsibility to verify that your program generates the expected results in the submit server.

6. You must use execvp (and no other exec* system call).

7. Your code must be written in the file shell_jr.c.

8. Do not use dup2, read, write, nor pipes.

9. Do not use system() to execute commands.

10. Assume a line of input has at most 1024 characters.

11. Provide a makefile that builds an executable named "shell_jr". The rule that builds the executable must have the target "shell_jr". Also have a target named "clean" that removes the shell_jr executable and any object files.

12. Your program should be written using good programming style as defined at

    http://www.cs.umd.edu/~nelson/classes/resources/cstyleguide/

13. Common error: If you get the submit server message "Execution error, exit code 126" execute "make clean" before submitting your code.

14. Common error: Forgetting to return the correct value (eg, 0) in your code.

15. Your shell_jr C program does not take command line arguments. Hence the main function would have the form (no argc or argv):

```
int main() { }
```

16. Remember that the argv parameter in execvp is an array of strings (so you wold construct an array of strings and pass it to execvp).

17. Your shell should exit when it detects end of file. This explains why public tests do not have exit nor hastalvista as the last command.

18. This exercise relies on Standard I/O, NOT Unix I/O.

19. When a line is entered into shell_jr, it reads at most the first two arguments in the line and ignores any other arguments. For example, if command "wc location.txt" is entered, it reads two arguments ("wc" and "location.txt") and successfully executes the command. If "wc location.txt bla" is entered, it reads two arguments ("wc" and "location.txt") and successfully executes the command.

20. For exit and hastalavista, ignore any values provided after the command (just exit the shell).

21. For cd, ignore any values provided after the directory name. For example, "cd /tmp bla" would change the working directory to /tmp.

22. If an invalid directory is provided to the cd command, your shell should print an error message:

    ```
    "Cannot change to directory INVALID_DIRECTORY_NAME"
    ```

    where INVALID_DIRECTORY_NAME is replaced with the invalid directory name.

23. Do not use signals.

6 How to Start

Start by creating a loop that reads lines and displays them. Then begin to process each command, starting with the exit and cd commands. You are free to process each line any way you want; however, reading a whole line using fgets and then processing the line using sscanf could make things easier. Keep in mind that if sscanf cannot read a value into a string variable, it does not change the variable. This could help you to identify whether a command has an argument.

7 Submitting your assignment (You need to do something extra)

To submit, execute "make clean" and then "submit". Without the "make clean", you may get an error in the submit server.