CSCE **3402** Spring 2021
Exercise **2** – Implementing a Shell

Simple Shell

Assigned: Tuesday, February 9th in Lab

Due: ***Tuesday***, *February 23rd at* ***Lab time.***

Delayed submission with penalty until Thursday, February 15th at 11:55pm.

Purpose

In this lab exercise you are required to build a simple shell that runs on Linux. This lab exercise is an individual one that you should work on it on your own and is designed to enrich your systems programming skills and get you ready to the next more advanced lab exercises in which you will be asked to amend and change the kernel and the way it functions; through either writing kernel modules or modifying the kernel source tree. This assignment is designed to get familiar with the system call interface and get acquainted with the systems programming tools provided in the Linux environment.

The most legitimate question that you might ask now is "What is a Shell?". But based on our first lab lecture and exercise you should confidently know now that that a shell is a program that allows a user to run other programs:)

Basics

You have the code presented in the lab lecture slides as a minimal shell seed start that you can use freely. Your shell should always print a shell prompt with the current directory name as part of the prompt text. You need to extend the code provided in the lab lecture slides to implement more shell features like redirection and pipes. Your shell should also support a default current directory that you can change using the "cd" command. Your shell should exit upon entering the "exit" command on the input prompt or pressing CRTL+D. You should be able to utilize all needed system calls like fork, wait, pipe, chdir, … etc. to achieve the most basic functionalities of a UNIX shell. Essentially, your shell should support the shell built-in command "echo" and supports simple environment variables.

All your code should be in a single c file with the name myshell.c and you should use the gcc compiler to compile your shell with ANSI C switches enabled.

Details

Your shell must support the following as a minimum:

1. Your shell should exit upon entering the shell “exit” command or pressing CRTL+D. You should utilize the ***exit()*** system call.
2. Your sell should start with a default directory as the home directory of the current user and can change directory using shell “cd” which should invoke the ***chdir()*** system call. To know the current user home directory you might need to use the ***getenv()*** library function.
3. Your shell should be able to execute command with no arguments such as “ls”, “ps”, “pwd”, … etc. To do that you should utilize system calls like ***fork()***, ***wait()***, ***execvp()***, … etc. The shell should block after forking a new child to execute the command and should wait for the command to finish, The shell should then resume execution by reading the next command from the user.
4. Your shell should be able to run commands with arguments. This should work with the same mechanism as in point # 3, in addition to parsing the command ans extracting the arguments and passing it a command-line arguments to the child process. Examples:
    - ls -l *.txt
    - ps -ef
    - cat /*etc*/hosts

   and more ...
5. Your shell should implement both input and output redirection and should be able to account for combined input/output redirection in the same command. You should use functions like ***dup()***, ***freopen()***, ***close()***, … etc. Examples:
    - ls -l > file
    - cat < /*etc*/hosts
    - cat < /*etc*/hosts > dump.txt
6. Your shell should be able to support pipes and be able to fork multiple child processes for multiple commands which are organized in a pipeline where each process and pipe its output into the input of the next process. You should use the ***pipe()*** system call, and you should build the right data structures to support a predefined maximum number of pipes per command. Examples:
    - ls -l | grep “*.txt”
    - ps -ef | grep bash
7. Your shell should support any combination of the above. Examples:
    - cat < /*etc*/hosts | grep “127” > dump.txt
8. Your shell should allow the user to define variables, assign them with a constant, and assign them to the output of other commands using the “`” shell qualifier. Your shell should also be able to print any defined variables. Example:
    - X=100
    - X=`cat /etc/hosts`
    - echo $X
    - Y = 200

    - X=$Y

You should be able to handle all parsing errors and report them correctly. Also you should be able to handle all errors returned from any library function or system call that you decide to use and your shell should act accordingly. Essentially you should read all the man pages thoroughly and understand all the function before using them.

Again, you have to use ANSI C through the gcc compiler switch -ansi; if you do not you will lose marks.

Deliverables

Everything should be submitted on Blackboard. Include a Readme file to explain anything unusual to the TA. Your code and other associated files must be in a single directory that build properly. You must provide a make file that can build and clean your shell. So, you need to provide in your Makefile a target for **myshell** and another for **clean** to delete the generated files and executables.

Your design document should be called design.pdf and it should be in PDF format and should be in the same directory. Formats other than PDF are not acceptable; please convert other formats (Word, LaTeX, HTML, …) to PDF. Your design document should describe the design of your assignment in enough detail that a knowledgeable programmer could duplicate your work. This includes descriptions of the data structures you use, all non-trivial algorithms and formulas, and a description of each function including its purpose, inputs, outputs, and assumptions it makes about the inputs or outputs.

Deliverables Summary:

1. Design Document (.pdf)
2. README.txt
3. Source code: myshell.c and Makefile. You can break your code into different .c and .h file is you want (optional), but you need to have your main program in myshell.c.

Grade

This lab exercise is worth 10% of the overall lab grade. The lab exercise will be graded on a 100% grade scale, and then will be scaled down to the 10% its worth. The grading of the assignment will be broken down as follows:

1. 10 % for just submitting a meaningful assignment before or on the due date. This 10% does not account for the full correctness of your assignment but submitting an empty assignment without code will definitely results in loosing this 10% and consequently the whole grade of this assignment.
2. 70 % for the correctness of the functionality and the quality of your code.
3. 10 % for the quality of your inline documentation and the readme file.
4. 10 % for the design document.

Delays

You have up to 2 working days of delay, after which the assignment will not be accepted and your grade in that case will be ZERO. For every day (of the 2 allowed days), a penalty of 10% will be deducted from the grade. And of course, you will lose the 10% mentioned in point 1 above under the "Grade" section.