Project 4: User-Level Shell CS 416 OS Design
Deadline: 10th December 2020 [23:55]

You do not trust anything other than the OS; and so there is no place for typical shell programs like Bash or zsh. You and your partner in crime set out to write your own shell.

In this project, you shall build a shell written in C.

For evaluation, we shall test typical shell commands like ls, wc; with piping (|) and output redirection (>, >>).

You can do this project in groups of <=2.

Part 1: Implementing a C-shell (100 pts)

The goal of this project is to implement a shell in C using different system calls provided by the OS.

Following are the capabilities we shall be testing in your shell

1) **Prompt**: A prompt indicates that the shell is available to take a user command. The prompt should show the present working directory (getcwd). Example prompt:

   /home/baobun $

2) **Command input**: Given the prompt, a user can now type in commands to run using exec* (execvp)command in C . Command **cd** needs to be implemented using chdir.

   Unless the output is redirected, your shell spits the output to STDOUT.

   A typical shell has different special characters that mean different functionalities:
   a) **;** : A semicolon in between two commands mean that the commands run in succession.
      Eg: $ ls ; echo "hello world" ## This will first run ls and then the echo command
   b) **>** / **>>** : Output redirection is one where output of the first command can be stored in a file. > truncates the file to size 0 before writing to it whereas >> appends to the file. Use dup2 for implementation in C.
      Eg: $ ls > out.txt
   c) **|** : Piping enables the first command's output to be the second command's input. Use pipes for implementation.
      Eg. $ ls | wc -l

Take a look at this https://ryanstutorials.net/linuxtutorial/piping.php to understand piping and redirection.

3) Signal Handler: When a user inputs ctrl-c, it should not exit the shell itself; rather it should only cancel a running command/program if any. Else it would print a new prompt on the next line. To quit the shell, one must type "exit".
Take a look at https://linuxhint.com/signal_handlers_c_programming_language/ for signal handler programming help.
Note: In order to kill that particular user command, make sure your commands run as a child process to your shall program. Specifically, you would need **fork** for executing commands.

Suggested Steps:

1. Since you have not been given a boiler plate code, you are free to implement this project in your way.
2. Once the prompt works, given the input commands from the user, tokenize them using strtok. This will divide different parts of the input command based on your provided delimiters.
3. After you are able to isolate a command with its options (ls -la is one command); try executing it using execvp. Now you have a very basic shell.
4. Now take care of piping and redirection using your tokens to identify them.
5. Add signal handler functionality by killing the child (forked) process.
6. Check for bugs, and Voila, you have your shell.

Compiling and Benchmark Instructions:

Please use a Makefile for compiling. Include it in your submission with the code.

Some commands we intend to test your shell with is ls, wc, cd, cat, echo with combinations of piping and output redirection. You do not have to implement other bash specials like <, & etc. to make this project simple.

For this assignment, you can use either ilab machines or one of the following:

- kill.cs.rutgers.edu
- cp.cs.rutgers.edu
- less.cs.rutgers.edu
- ls.cs.rutgers.edu

Submission Format:

You need to submit your project code and report in a compressed tar file on sakai. ONLY ONE SUBMISSION PER GROUP IS REQUIRED.

You MUST use the following command to archive your project directory.
`$ tar zcvf <netid1>_<netid2>.tar.gz project4` ##netid1 of the person submitting on sakai

NOTE: Your grade may be reduced if your submission does not follow the above instruction. For students working alone, netid2 == netid1.

Files to be included is cshell.c, Makefile, Readme.

For further questions, please visit the TAs in their office hours and/or recitations.