CSC 415 - Building Simple Shell
Project 3

Total Points: 150 Points

Description

For this assignment you will implement your own Shell or Command Line Interpreter (e.g. to replace /bin/bash for simple interactions with the Linux Kernel.). Your shell will be character-oriented, and will fork off processes to execute user entered commands.

Your shell should read lines of user input into a 256-byte buffer, then parse and execute the commands (be sure to clear the buffer between successive commands!) It should be possible for the user to specify the command to execute by giving an absolute path to the file containing the executable (e.g. ./hw1); or to use path expansion to locate the file containing the executable by using the environment PATH variable to construct a series of absolute paths and executing the first file found in this way (note that the execvp() command performs this processing automatically, you do not need to program this yourself!) Your code should parse the input string and separate it into a collection of sub-strings (stored in myargv[]) along with a count of the number of strings encountered (stored in myargc). Note that piped commands will require multiple argc/argv instances!

Shell Requirements

Your shell should support the following functions (Note : this does not mean your shell should implement LS but rather the ability to execute the LS program):

- Must maintain myargc and myargv for commands you will fork. In some cases you may need multiple instances for pipe commands.
- The commands cd and pwd must be implemented in your shell.
- Execute a single command with up to four command line arguments (including command line arguments with associated flags). For example:
  - myshell>> ls –l
  - myshell>> cat myfile
  - myshell>> ls –al /usr/src/linux
- Execute a command in background. For example:
  - myshell>> ls -l &
  - myshell>> ls –al /usr/src/linux &

- Redirect the standard output of a command to a file. For example:
  - myshell>> ls -l > outfile
  - myshell>> ls -l >> outfile
  - myshell>> ls –al /usr/src/linux > outfile2
  - myshell>> ls –al /usr/src/linux >> outfile2
- Redirect the standard input of a command to come from a file. For example:
  - myshell>> grep disk < outfile
  - myshell>> grep linux < outfile2
- Execute multiple commands connected by a single shell pipe. For example:
  - myshell>> ls –al /usr/src/linux | grep linux
  - myshell>> ls -la | wc -l
- Execute the cd and pwd commands
  - myshell>> cd some_path
  - myshell>> pwd

****NOTE**** That in most Linux distros CD and PWD are not a program like ls but rather they are shell built-ins. Built-ins are shell commands that are implemented in the shell and not some external binary. For giving your shell the cd and pwd commands, you need to implement these functions in the shell with your code, even if it is provided by your OS. This can be done with the chdir() and getcwd() functions in unistd.h

Suggested implementation strategy to implement a shell with multiple command line arguments (using iterative refinement):

- Implement your shell to simply initialize your shell, display a prompt, read in user input and print it back to the console.
- Add functionality to your shell to parse user input setting the correct values for myargv and myargc. Once parsed, print myargv and myargc to the console. Note that myargvs need to be null terminated for the exec commands to interpret them correctly.
- Add functionality to your shell to execute simple shell commands. Start with commands like ls, then commands with options like ls -la /home.

- Add functionality to shell to execute input and output redirection. It is required to implement >, >>, and <.

- Add functionality to shell to execute commands in the background. For example commands like : ls -la &

- Add functionality to your shell to execute the cd and pwd commands. Note these need to be implemented in your shell. Use the chdir() and getpwd() functions to implement these shell commands.

- Add functionality to your shell to execute piped commands. These are commands that are connected by a shell pipe, |.

Extra Credit

To get points for EC, your shell needs to work first. Broken shells are not eligible for EC.

- (15 - Points) Implement your shell so that any combination of shell commands from above can be used in a single command line.
  For example, ls -la | wc -l | wc -b OR ls -la | grep *.c | wc -l

- (10 - Points) Implement the shell so the current working directory is shown on the prompt. For example:

  OLD PROMPT:

  myshell >>

  NEW PROMPT:

  * myshell `~/` >>
  * myshell `~/hw3` >>
  * myshell `~/hw3/build` >>
  * myshell `/etc` >>
  * myshell `/etc/apach2` >>

  DO NOT hard code the home path to your computer, you need to detect the home path of the computer your shell is executing on. Make sure `~/` is printed in all correct situations.

- (30 - Points) Add the functionality to your shell to store a history of all commands executed. This includes the ability to scroll through this history as well and rerun previously entered commands. See your basic Linux shell for examples. This also requires the use of the up and down arrow keys to scroll through the list. **Use of readline header files IS NOT ALLOWED**

What to submit

1. source code in myshell.c
2. updated README.md with required fields filled out.

How to submit

- git add .
- git commit -m ” message”
- git push