Project 3: Simple Unix Shell

CMSC 257 – Computer Systems

Due date: 11:59 pm EST, 12/07/2019

Introduction:

The shell is a program that interacts with the user through a terminal or takes the input from a file and executes a sequence of commands that are passed to the Operating System.

In this project you will be creating your own simple shell, with basic functionality.

This project is expected to be less than 200 lines, including code and comments.

Interactive vs Batch Shell

- Interactive
  - User types commands in, hits return to invoke them
- Batch
  - shell reads from an input file
- What is the difference?
  - where the commands come from
- You need to implement the Interactive shell model only.

Input/Output

- C has 3 standard files to be used for input and output.
  - stdin = input
  - stdout = output
  - stderr = error output
- printf("foo") == fprintf(stdout,"foo")
- scanf("%s",str) == fscanf(stdin,"%s", str)
- fprintf(stderr,"Panic!") prints an error message separately
- For signal safety, in some cases, you may want to use **Sio_puts** or similar functions defined in csapp.c

Process Control

- Your shell should execute the next command line **after** the previous one terminates – you must wait for any programs that you launch to finish
- You don't have to provide the functionality of launching multiple simultaneous commands with ";" separating them

Hints

- Starter code
  - You can use the code supplied by the textbook. Or you can start from scratch.
  - Goto : http://csapp.cs.cmu.edu/3e/code.html
    - Download shellex.c
    - Also download csapp.c and csapp.h files.
    - Compile as gcc -pthread csapp.c shellex.c -o shellex

- A shell is a loop
  - read input
  - execute program
  - wait program
  - repeat

- Useful routines
  - fgets() for string input
  - strtok() for parsing
  - exit() for exiting the shell
  - raise() for sending signal to self
  - getpid() for finding the current process ID
  - getppid() for finding the parent process ID
  - getcwd() for getting the current working directory
  - getenv()/setenv()
  - chdir() for changing directories

- Executing commands
  - fork() creates a new process
  - execvp() or other exec family commands runs a new program and does path processing
  - wait(), waitpid() waits for a child process to terminate

Requirements:

- <executable> -p <prompt> should allow the user to select an user-defined prompt. Otherwise, the default should be “my257sh> ”.
  - Shell functions to be implemented separately as built in commands : exit, pid, ppid, cd, help.
  - exit: Exits the shell. For implementing “exit” from the shell, use the raise() system call.
  - pid: prints the process id of the shell

- ppid: prints the parent process id of the shell
- help: prints developer name for shell,
  - usage (how to change shell prompt, maybe list of built-in commands)
  - refer user to use man to get help if they are looking for non-built-in commands.
- cd prints the current working directory; whereas "cd <path>" will change the current working directory.
- All other shell commands will need a child process using fork() and then calling execvp() (or another exec family function).

- Inputs are guaranteed to be less than 100 characters.
- Only the interactive system needs to be implemented (batch system is not needed)
- Background process execution (using &) is NOT required.
- Each time a child process is created, evaluate its exit status and print it out.
- ^C should not take us out of the shell; use a signal handler. Hint: you can use the same signal handler code from the slides.

What to turn in

When you're ready to turn in your assignment, do the following:

1. Put your files into a folder named pr3
2. Prepare a make file, which creates executable shell file when typed make, and cleans everything except source and header files when typed make clean.
3. Executable name generated for shell must be my257sh
4. In the pr3 directory:

   $ make clean

   $ cd ..

   $ tar czf pr3_*<username>*.tar.gz pr3

   make sure the tar file has no compiler output files in it, but does have all your source. Following command will show them.

   $ tar tzf pr3_*<username>*.tar.gz

5. Turn in pr3_<username>.tar.gz and a screenshot or screenshots of sample run showing

- ls -l
- ps w
- pid
- ppid
- cd ..
- ls -l
- cd p3
- ls -l
- help
- bogus (to show how an unknown command is handled)
- CTRL C (To Show SIGTERM entered from keyboard does not terminate the shell)

Grading

We will be basing your grade on following elements:

- The degree to which your code satisfies the requirements (90 pts)
- **If code does not compile, if there is no make file or make file doesn't work, you will get 0 points. We will not check the reason.**
- The readability of your code. While we don't have detailed, formal coding style guidelines that you must follow, you attempt to mimic the style of code that we've provided you in project 2. Aspects you should mimic are conventions you see for capitalization and naming of variables, functions, and arguments, the use of comments to document aspects of the code, and how code is indented. Please also add your name on top of the files you modified as author. (10 pts)