CSE 333 - OPERATING SYSTEMS
Programming Assignment # 2 DUE DATE: 09/12/2016 - 23:00PM

This programming assignment is to write a simple shell. The shell accepts user commands and then executes each command in a separate process. One technique for implementing a shell interface is to have the parent process first read what the user enters on the command line and then create a separate child process that performs the command. Unless otherwise specified, the parent process waits for the child to exit before continuing. However, UNIX shells typically also allow the child process to run in the background – or concurrently – as well by specifying the ampersand (&) at the end of the command. The separate child process is created using the fork() system call and the user's command is executed by using one of the system calls in the exec() family.

The main() function of your program presents the command line prompt "**CSE333sh:** " and then invokes setup() function which waits for the user to enter a command. The *setup function* (given in the attachment of the project) reads the user's next command and parses it into separate tokens that are used to fill the argument vector for the command to be executed. This program is terminated when the user enters ^D (<CONTROL><D>); and setup function then invokes exit. The contents of the command entered by the user are loaded into the *args* array. You have to use "mainSetup.c" file (given in the attachment) to implement the project.

We have already listed the required functions and properties of your shell.

Your shell should support different types of commands:
A. System commands,
B. Built-in commands,
C. Pipe operator.

A. System commands

It will take the command as input and will execute that in a new process. When your program gets the program name, it will create a new process using fork system call, and the new process (child) will execute the program. The child will use one of the *exec* function in the below to execute a new program.

- If the command does not include a pathname, then you can use `execvp()`.

  CSE333sh: ls -l

- If the command includes a pathname, then you can use `execl()` providing full (absolute) pathname.

  CSE333sh: /bin/ls ls -l
  CSE333sh: /bin/ls ls -l /home/user1/Desktop

Important Notes:

1. Using the “system” function for part A is not allowed!

2. In the project, you need to handle foreground and background processes. When a process run in foreground, your shell should wait for the task to complete, then immediately prompt the user for another command.

   333sh: gedit

   A background process is indicated by placing an ampersand (’&’) character at the end of an input line. When a process run in background, your shell should not wait for the task to complete, but immediately prompt the user for another command.

   333sh: gedit &

B. Built-in commands

Your shell must support the following internal (*built-in*) commands. *Note that an internal command is the one for which* ***no new process*** *is created but instead the functionality is built directly into the shell itself.*

- **`cd`** ***`<directory>`*** - change the current directory to ***`<directory>`***.
  - When a user changes the current working directory (e.g. "cd somepath"), you simply call `chdir()` function.
  - If the ***`<directory>`*** argument is not present, your shell should change the working directory to the path stored in the **$HOME** environment variable. You can use `getenv("HOME")` to obtain this.
  - You need to support relative and absolute paths.
  - This command should also change the **`PWD`** environment variable.

- **`dir`** - print the current working directory.
  - Basically, when a user types **`dir`**, you simply call `getcwd()`.

- **`clr`** - clear the screen.
  - You can use `system()` function to execute `clear` command.

- **`wait`** - is described for background processes.
  - By typing a trailing ampersand (**&**), the shell knows you just want to launch the process, but not to wait for it to complete. Thus, you can start more than one job by repeatedly using the trailing ampersand.
  - Sometimes you will want to wait for processes to complete. To do this, in your shell you will simply type `wait`. The built-in command `wait` should not return

until **all** background processes are complete.
In this example below, `wait` will not return until the three processes all are finished.
  - Example:
    - **333sh: gedit &**
    - **333sh: firefox &**
    - **333sh: gnome-calculator &**
    - **333sh: wait**

- **`hist`** – This command is for maintaining a history of commands previously issued.
  - **`hist`** - print up to the 10 most recently entered commands in your shell.
  - **`hist –set num`** – set the size of the history buffer to num. (The default value is 10).
  - **`! number`** - A user should be able to repeat a previously issued command by typing "**`! number`**", where number indicates which command to repeat.
    - Note that "**! 1**" is for repeating the command numbered 1 in the list of commands returned by history, and "**! -1**" is for repeating the last command.
  - **! string** - A user should be able to repeat a previously issued command by typing **! string**, where string indicates the first characters of a most-recent command. If a user has executed "**chmod 777 a.sh**" and wants to repeat this command, "**! chm**" can be used.
  - You are not allowed to use `history` command of Linux for this command. You should keep them by your own data structures.
- **`exit`** - Terminate your shell process.
  - You simply call `exit(0)`; in your C program.
  - If the user chooses to exit while there are background processes, notify the user that there are background processes still running and do not terminate the shell process unless the user terminates all background processes.

C. Pipe operator

The shell must support pipe operator "|". For a pipe in the command line, you need to take care of connecting stdout of the left command to stdin of the command following the "|". For example, if the user types "ls -al | sort", then the "ls" command is run with stdout directed to a Unix pipe, and that the sort command is run with stdin coming from that same pipe.

Bonus: *You will get 10% extra credit if your shell supports multiple pipes!* (e.g., prog1 | prog2 | ... | progN)

<u>Notes:</u>

- You can assume that all command line arguments will be delimited from other command line arguments by white space – one or more spaces and/or tabs.
- For this project, the error messages should be printed to **stderr.**

- Take into account materials and examples covered in the lab sessions. As a starting point, you can consider the example program given in course web site.
- Consider all necessary error checking for the programs.
- No late homework will be accepted!
- In case of any form of **copying** and **cheating** on solutions, all parties/groups will get ZERO grade. You should submit your own work.
- You have to work in groups of two. Group members may get different grades.
- There will be demo section for this assignment. If you cannot answer the questions about your project details in the demo section, even if you have done all the parts completely, you will not get points!

What to submit?

A softcopy of your *source codes* which are extensively commented and appropriately structured should be emailed to **cse333.projects@gmail.com**
Make sure that your file contains your name(s) and student ID(s)!