# CSE 333 - OPERATING SYSTEMS
# Programming Assignment # 2 DUE DATE: 11/12/2015 - 23:00

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.

A C program that provides the basic operation of a command line shell is given in the attached. This program is composed of two functions: *main()* and *setup()*. The *main()* function of your program presents the command line prompt "**333sh: "** and then invokes *setup()* function which waits for the user to enter a command. The *setup()* function 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. The contents of the command entered by the user are loaded into the *args* array. For example, if the user enters "**333sh:** ls –a" prompt, args[0] becomes equal to the string ls and args[1] is set to string to -a. This program is terminated when the user enters "exit" and setup() then invokes exit().

Your shell should support different types of commands:
**A.** System commands,
B. Built-in commands,
**C.** I/O Redirection,
**D.** Pipe operator.

Necessary functionalities and components of your shell are listed below:

## 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 *execv* function in the below to execute a new program.

- Use **execv**() which means that you will have to read the PATH environment variable, then search each directory in the PATH for the command file name that appears on the command line.
    - int **execv** (const char *path, char *const argv[]);

**333sh: ls -l**

**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 &**

   With background processes, you will need to modify your use of the wait() system call so that you check the process id that it returns.

**B. Built-in commands**

It 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 build directly into the shell itself.*

- **`ps_all` -** display the status of each background job (process). Your shell must keep track of background job information using an appropriate data structure. Each background job should be assigned a job number to be used as an index when displaying information about the job. The job numbers should be incremented for each new background job, and when no background job exists, the job number should reset to 1. In addition to this, the PID values of the processes are also displayed. It should be noted that the status information of background jobs are displayed in two lists: list of background jobs still running, and the list of background jobs that have finished execution since the last time the status is displayed with the **`ps_all`** command. Consider the following example:

```
MYSHELL:  ps_all
Running:
     [1]  xterm (Pid=1000)
     [2]  emacs myfile1 (Pid=1001)
     [4]  netscape (Pid=1002)
Finished:
     [3]  myprog1 > temp_file
```

  Note that job [3] will not be listed the second time. You should clear out the background job information from your jobs data structure alter a job displayed in the finished list.

- **kill –** Two versions of this command is considered:
  - **kill %num –** terminate the process numbered *num* in the list of background processes returned by **ps_all** command. If there is no such process, an appropriate error should be reported.
  - **kill num –** terminate the process whose process id (pid) is equal to *num*. If there is no such process, an appropriate error should be reported.

- **fg %num**- Move a process numbered num (in the list of background processes returned by **ps_all** command) to foreground. Note that it will not be included in the data structure that keeps the background process.

- **exit –** Terminate your shell process. 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. I/O Redirection**

The shell must support I/O-redirection on either or both *stdin* and/or *stdout* and it can include arguments as well. For example, if you have the following commands at the command line:

- **333sh: myprog [args] > file.out**
  Writes the standard output of **myprog** to the file **file.out.**
  **file.out** is created if it does not exist and truncated if it does.

- **333sh: myprog [args] >> file.out**
  Appends the standard output of **myprog** to the file **file.out.**
  **file.out** is created if it does not exist and appended to if it does.

- **333sh: myprog [args] < file.in**
  Uses the contents of the file **file.in** as the standard input to program **myprog**.

- **333sh: myprog [args] >& file.out**
  Writes the standard error of **myprog** to the file **file.out**.

- **333sh: myprog [args] < file.in > file.out**
  Executes the command **myprog** which will read input from **file.in** and stdout of the command is directed to the file **file.out**

**D. 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 pipe with I/O redirections.* e.g., prog1 | prog2 > out. *To receive the full extra credit, you must be able to deal with multiple pipes!* (e.g., prog1 | prog2 | ... | progN)

**Notes:**

- You can assume that all command line arguments (including the redirection symbols, <, >, &, >> ) will be delimited from other command line arguments by white space – one or more spaces and/or tabs.
- 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 and a *project report* (minimum 3-page) that contains the detailed information about your implementation should be emailed to **cse333.projects@gmail.com**
Make sure that your zip file contains your name(s) and student ID(s)!