Operating Systems Laboratory (CS39002)
Spring Semester 2017-2018

<u>Assignment 2</u>

Implement the functionalities of a Unix/Linux Shell

Assignment given on:	January 15, 2018
Assignment deadline:	January 29, 2018
	(Interim submission deadline: January 22, 2018)

Write a menu-driven program that will carry out several common functionalities of a normal shell. The program will display a menu like the following:

A. Run an internal command
B. Run an external command
C. Run an external command by redirecting standard input from a file
D. Run an external command by redirecting standard output to a file
E. Run an external command in the background
F. Run several external commands in the pipe mode
G. Quit the shell

The internal commands are those that have to be executed by directly calling some system call, like **`mkdir()`**, **`chdir()`**, **`rmdir()`**. The external commands refer to executables that are stored as files in some folder. When one of the options in the menu is selected, the program will ask for the command. Some example commands are shown below:

```
• mkdir  abc                // Option A
• chdir  ..                 // Option A
• ./a.out                   // Option B
• cc –o runfile test.c      // Option B
• ./a.out < inputfile       // Option C
• ls > outfile              // Option D
• ./a.out &                 // Option E
• grep xuz.txt | more       // Option F
• cat abc.c | sort | more   // Option F
```

<u>Implementation Guideline:</u>

The generic structure of the shell will be similar to Assignment 1a. After reading a command to be executed, the parent will fork one or more child processes, which will take the responsibility of executing the command.

For redirecting the standard input or output, you can refer to the book: "*Design of the Unix Operating System*" by Maurice Bach. Actually, the kernel maintains a file descriptor table or FDT (one per process), where the first three entries (index 0, 1 and 2) correspond to standard input (**`stdin`**), standard output (**`stdout`**), and standard error (**`stderr`**). When files are opened, new entries are created in the table. When a file is closed, the corresponding entry is logically deleted. There is a system call **`dup(xyz)`**, which takes a file descriptor **`xyz`** as its input and copies it to the first empty location in FDT. So if we write the two statements: **`close(stdin); dup(xyz);`** the file descriptor **`xyz`** will be copied into the FDT entry corresponding to **`stdin`**. If the program wants to read something from the keyboard, it will actually get read from the file corresponding to **`xyz`**. Similarly, to redirect the standard output, we have to use the two statements: **`close(stdout); dup(xyz);`**

Normally, when the parent forks a child that executes a command using **`execlp()`** or **`execvp()`**, the parent calls the function **`wait()`**, thereby waiting for the child to terminate. Only after that, it will ask the user for the next command. However, if we want to run a program in the background, we do not give the **`wait()`**, and so the parent asks for the next command even while the child is in execution.

A pipe between two processes can be created using the **`pipe()`** system call, followed by input and output redirection. Consider the command: **`ls | more`**. The parent process finds out there is a pipe between two programs, creates a pipe, and forks two child processes (say, X and Y). X will redirect its standard output to the output end of the pipe (using **`dup()`**), and then call **`execlp()`** or **`execvp()`** to execute **`ls`**. Similarly, Y will redirect its standard input to the input end of the pipe (again using **`dup()`**), and then call **`execlp()`** or **`execvp()`** to execute **`more`**. If there is a pipe command involving N commands in general, then you need to create N-1 pipes, create N child processes, and connect each pair of consecutive child processes by a pipe.

Submission Guideline:

- Create a single program for the assignment, and name it **Ass2_<groupno>.c** (replace <groupno> by your group number), and upload it.
- Submit an interim version within January 22, 2018 for intermediate evaluation.
- You must show the running version of the program(s) to your assigned TA during the lab hours.