ASSIGNMENT 3
SHELL IMPLEMENTATION

Times of Interest:

- Release of Assignment - Wednesday, 21 September, 2016
- Assignment Submission Deadline - 11:55PM, Friday, September 30, 2016 (Hard Deadline)

Educational Objectives:

For many people this assignment will be practice and/or a warm-up. For others, it will be a learning exercise. Regardless of your background, by the end of this assignment, we hope that you will comfortably and confidently be able to do the following:

- Develop clear, readable, well-documented and well-designed programs in the C/C++ Programming Language.
- Understand the working of command line interface in Unix-like environment.
- Develop software in the Andrew/Unix using tools such as gcc, gdb, and make.
- Locate and interpreting "man pages" applicable to application-level system programming.
- Use the POSIX/Solaris API to system functions to manage process and sessions as well as use signals and pipes for interprocess communication.
- Understand the process forking mechanism.

Assignment Overview:

In this assignment you are asked to implement a simple command-interpreter, a.k.a. "shell," for Unix. The shell that you will implement should be similar to linux shell but it is not required to implement as many features as linux shell have.

The shell should have much of the important functionality:

- Execute all the commands (ls, clear, vi etc)
- Shell builtins (cd, pwd, export)
- Print environment variables and text using echo
- Background and foreground functionality : &, fg - Allow the user to execute one or more programs, from executable files on the file-system, as background or foreground jobs.
- Allow for the piping of several tasks : Pipes "|" (multiple)
- Input and Output redirection: (<, >)
- Support for history and '!' operator
- Handle Interrupt Signal

Specifications:

- The look and feel of solution should be similar to that of other UNIX shells e.g. your shell should work in an infinite loop. It should produce a prompt, e.g., xsh>, accept input from the user, and then produce another prompt till an "exit" is entered.
- Your solution should be an application program invoked without command-line parameters or configuration files.
- Parsing the input : For certain characters : &, |, <, and >, your shell should assume that these meta-characters cannot occur inside strings and have to be handled separately.
- Certain combinations of keystrokes will generate signals to your shell instead of appearing within stdin. Your shell should respond appropriately to these signals.
    - Control-Z generates a SIGSTOP. This should not cause your shell to be suspended. Instead, it should cause your shell to suspend the processes in the current foreground job. If there is no foreground job, it should have no effect.
    - Control-C generates a SIGINT. This should not kill your shell. Instead it should cause your shell to kill the processes in the current foreground job. If there is no foreground job, it should have no effect.
- If the user enters something improper, your shell should produce a meaningful error message.
- Empty command lines should be treated as no-ops and yield a new prompt.

Examples:

1. Commands and Shell Builtins:

```
bash prompt:~$ ./a.out
My_Shell:/home/user$ ls
        shell.cpp main.cpp a.out
My_Shell:/home/user$ fdresdsad
My_Shell: fdresdsad: No command found
My_Shell:/home/user$ cd OS
My_Shell:/home/user/OS$
My_Shell:/home/user/OS$ cd ..
My_Shell:/home/user/$ cd /home/user/work/temp
My_Shell:/home/user/work/temp$
My_Shell:/home/user/OS$ echo $PWD
        /home/user/OS
```

2. I/O redirection and pipes:

My_Shell:/home/user/OS$ echo Hello, this is a line > out
My_Shell:/home/user/OS$ cat out
    Hello, this is a line
My_Shell:/home/user/OS$ cat out| wc
    1 5 22
My_Shell:/home/user/OS$ ls l | grep "out" | wc | wc | grep "1" | wc
    1 3 24
My_Shell:/home/user/OS$ ls R my_folder1/ | grep "abc" >out

3. History:

My_Shell:/home/user/OS$ history
. . .
43 man bash
44 man fc
45 man bash
46 fc l 10
47 history
48 ls a
49 vim .bash_history
50 history
51 man history
52 history 10
53 history

My_Shell:/home/user/OS$ history 5
50 history
51 man history
52 history 10
53 history
54 history 5

My_Shell:/home/user/OS$ history | grep cd
33 cd Pictures/
37 cd ..
39 cd Desktop/
61 cd /usr/bin/
68 cd
83 cd /etc/

```
  86  cd resolvconf/
  90  cd resolv.conf.d/

My_Shell:/home/user/OS$ !51                 # displays man page of history for our session
My_Shell:/home/user/OS$ vim file.cpp
My_Shell:/home/user/OS$ echo “hello”
My_Shell:/home/user/OS$  !v                 #Should execute “vim file.cpp”
My_Shell:/home/user/OS$  !!                         #Should execute “vim file.cpp”
My_Shell:/home/user/OS$ ls /usr/share/doc/manpages
My_Shell:/home/user/OS$ echo hello
```

4. Exit:

```
My_Shell:/home/user/OS$ exit
        Bye...
bash prompt:~$
```

Important functions and system calls:

- int chdir(const char *path)
- int execvp(const char *file, char *const argv[])
- void exit(int status)
- pid_t fork(void)
- char *getcwd(char *buf, size_t size)
- char getenv(const char *name)
- void perror(const char *string)
- int setenv(const char *name, const char *value, int overwrite)
- sig_t signal(int sig, sig_t func)
- pid_t wait(int *status)
- pid_t waitpid(pid_t wpid, int *status, int options)
- sighandler_t signal(int signum, sighandler_t handler);
- int dup2(int oldfd, int newfd);

Upload Format:

1. Create a folder named your Rollno_Assignment3.
2. Create a folder “Code” in this folder.
3. Place your '.c' or '.cpp' files and makefile (if used) in this “Code” folder.
4. Create a “Readme.md” file containing the details of functionality implemented.
5. Place “Readme.md” file outside Code folder in the main folder “Rollno_Assignment3”.
6. Create a tar.gz named “Rollno_Assignment3.tar.gz” and upload it.

Example:

2016xxxx/
|- Readme.md
|- Code/
    |- file1.cpp
    |- file2.cpp
    |- file3.cpp
    ....
    |- filen.cpp
    |- makefile (optional, only if you are using one)

Create 2016xxxx_Assignment3.tar.gz and upload it.

NOTE:

1. Use of system() will fetch you ZERO marks.
2. You CANNOT use any data structure or external text file to store the intermediate result in pipes.
3. If you have any confusion regarding upload format kindly clarify on moodle portal.
4. You are allowed to use Standard Template Library of C++ (STL) for this assignment.
5. Don't include any executable file (a.out) or any swap files (prog.cpp~) .
6. It is strictly not allowed to copy the code. Any sort of plagiarism found will fetch you ZERO marks.
7. Submit the code before the deadline. NO CODES will be accepted after the deadline.
8. *Make "man" you best friend.*