# BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE PILANI, K. K. BIRLA GOA CAMPUS

## I SEMESTER 2015-2016

## Advanced Operating Systems (CS G623)

## Assignment 2

## Due date 26/08/2015 (11:59 P.M)

---

## Instructions:

(1) Please upload the assignment using **http://photon**
(The file name should be <your id number>.tar.gz Example: **2015H103015G.tar.gz**)

(2) This is an **individual** assignment. Please see section 4b of handout for Malpractice regulations.

(3) The programming assignments will be graded according to the following criteria

- Completeness; does your program implement the whole assignment?
- Correctness; does your program provide the right output?
- Efficiency; have you chosen appropriate algorithms and data structures for the problem?
- Programming style (including documentation and program organization); is the program well designed and easy to understand?
- Viva conducted by me.

DO NOT FORGET to include a README file (text only) in your tar.gz file with the following contents.

**General README instructions**

In the directory you turn in (please upload the assignment as a tar.gz file), you must have a text-only file called README, in which you will cover AT LEAST the following:

1. Your name. If you interacted significantly with others indicate this as well.
2. A list of all files in the directory and a short description of each.
3. HOW TO COMPILE your program.
4. HOW TO USE (execute) your program.
5. A description of the structure of your program.
6. In case you have not completed the assignment, you should mention in significant detail:
   - What you have and have not done
   - Why you did not manage to complete your assignment (greatest difficulties)
     This will allow us to give you partial credit for the things you have completed.
7. Document any bugs of your program that you know of. Run-time errors will cost you fewer points if you document them and you show that you know their cause. Also describe what you would have done to correct them, if you had more time to work on your project.

NB: I will remove the link exactly by 12 Mid night. Those who are taking the lifeline can mail the code (only one lifeline is allowed for the entire set of assignments) to biju@goa.bits-pilani.ac.in within 24 hours of the deadline.

Please refer section 4b of the handout to know more about Malpractice regulations of the course.

## Question:

**This program is to test your understanding in fork, wait/waitpid, exit, exec, dup/dup2, pipe & tee**

In this question your task is to write a **replacement for the shell command line interpreter**. Typical command line shells under Unix, which you may have used, are **bash, sh, csh, tcsh**. The goal of this assignment is to increase your understanding of processes and the role of system software.
Your shell provides a sample interface, prompting the user for commands, and then interpreting the supplied commands. It should have the following features, which constitute an informal specification for your program.

**Input**

The shell reads an input line at a time from the standard input until the end of line is reached. Each input line is treated as a sequence of commands separated by a pipe character (|) or redirection characters (>, >>, <). For example: ls –l | grep alpha | more > file1.txt
(There can be one or more white spaces separating the words – command name, options, pipe and redirection). In this example there are three commands and a redirection. The command ls -l, the command grep alpha, and the command more are the commands and file1.txt is the output redirection. Each command consists of words separated by one or more spaces. No special extra command editing facilities are necessary beyond what is already done by regular terminal input (example see man tty for the default editing features namely backspace, erase line, erase word and end of file).

**Internal commands**

Your program does not need to interpret any internal shell commands except the following ones

**entry**

The command line interpreter will start functioning with this command. It should ask for a pass code. If it matches with the already stored pass code (assume pass code is available in passcode.txt file), the command line interpreter will start functioning. Any external or internal command issued before entering the correct pass code should display "Command line interpreter not started" message. Wrong pass code should show the message "Wrong Pass code!! Try again later!"

**exit**

The command line interpreter exits. Any external or internal command issued after the exit command should display "Command line interpreter not started" message.

**log**

The command line interpreter starts the command and output logging into "command.log" (maximum 20 entries) and "output.log" (unlimited size) files respectively with the log command.

**unlog**

The command line interpreter stops the command and output logging with the unlog command.

**viewcmdlog**

The viewcmdlog command displays the content of command.log file.

**viewoutlog**

The viewoutlog command displays the content of output.log file.

**changedir <argument>**

The changedir command takes the new present working directory path as argument. If the path is valid then the current working directory will be changed to the new directory specified in the argument. **Example:** changedir /home/usr/bin
Directory changed. The present working directory is /home/usr/bin

**External commands**

All commands that are **not internal commands** are **external commands**. An **external command** is a sequence of words separated by white space(s) of which the first is the executable name of the command and the remainder is the arguments to the command. **If no executable file was found at any of the environment paths, the shell must check the same in the current working directory. If the executable still not found then the shell command line interpreter should print the error message: "The command <command name> not found"**. **If the executable is found, the shell should create a new process to run the program**. The child process must use the **exec()** family of system calls to run each program together with the supplied arguments and shell environment variables.

**Processing of commands**

**The tricky thing about this assignment is that you have to arrange for the piping between processes for each command**. For example in the example given before, the **stdout** of **ls** must be piped to the **stdin** of **grep alpha**. The **stdout** of **grep alpha** must be fed to the **stdin** of **more**.
Your program must work for arbitrary numbers of processes on the command line, with arbitrary number of parameters. You would normally use **dup2/pipe/tee** to achieve this.
Your shell should record all input commands by appending to a log file called **"command.log"**. The format of "**command.log**" may be as shown below

| **<Command name> <Date> <Time of execution> <Action taken (success/failure)>** |
|---|

Your program should record the output of all commands, including the intermediate commands, by appending to a log file called **"output.log"**. The format of the **"output.log"** file is such that each line of command output is associated with the command that generated it. For example the command **ls | grep alpha | more** should add the following entries in output.log file. (I would strongly recommend you to follow this format).

**[ls]**
**<output of ls command>**

**[ls] | [grep alpha]**
**<output of the ls | grep alpha commands>**

**[ls] | [grep alpha] | [more]**
**<output of the ls | grep alpha | more commands>**

Make sure no orphan processes are getting created while executing the commands.

## Question - Evaluative

**Implement the shell with the help of dup2 and intermediate files**
**Implement the shell with the help of pipes and tees.**