BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI, K. K. BIRLA
Goa Campus, I SEMESTER 2014 – 2015
Advance Operating Systems (CS G623)
Assignment 2
Due date 31/08/2014 (12:00 Midnight)

Instructions: Please mail your completed assignment to biju@goa.bits-pilani.ac.in
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?*
- *Individual Viva.*

Submit your assignment as a tar.gz file.
DO NOT FORGET to submit a README file (text only) with following contents.
General README instructions

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 (You need to submit MAKEFILE for each question).*
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:*
    a. *What you have and have not done,*
    b. *Why you did not manage to complete your assignment (e.g., greatest difficulties)*
7. *This will allow us to give you partial credit for the things you have completed.*
8. *Document any bugs in your program that you know of. Run-time errors will cost you less penalty 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: Late submission will not be entertained.

Question #1. **2014H103022G, NIKHIL VYAS**

Write a program to create a process tree as shown in Figure #1 (if the command line argument is 4,2) and print the process identifiers in post-order traversal form. The number of levels and maximum number of children per process are command line arguments. The total number of processes in each level is increasing with Fibonacci series (1, 1, 2, 3, 5, 8 …). The program should create simultaneously executing processes in the same level as quickly as possible before child's execution starts. Each process should print its process identifier and its parent process identifier by using the function getpid( ) and getppid( ) respectively. Your program should not create an orphan process while execution. The parent process should also print the exit status of its child processes.

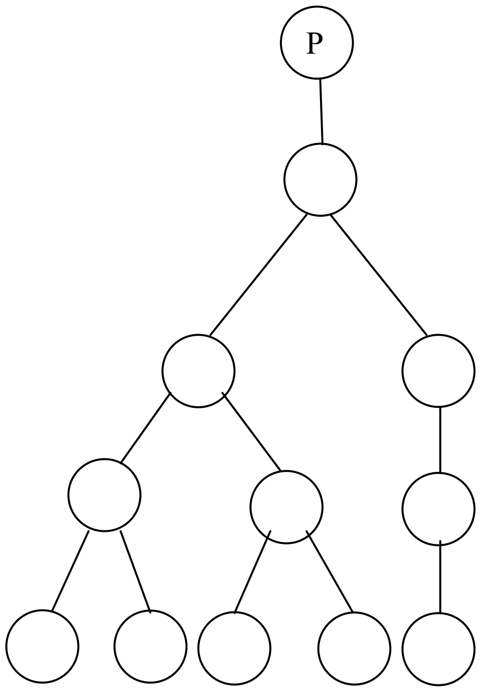

Figure #1

Question #2

In this programming exercise, you are asked to use **fork()** and **execv()** programming constructs. The program uses N input files each of them containing UNIX commands inside. Each line in a file will be a UNIX command with or without options. The command **./Que2 sampledata1.txt sampledata2.txt ……. sampledataN.txt** should read the **sampledata#. txt** files from the user.

Your task is to create **'N' simultaneously executing processes as quickly as possible before child's execution starts**. The number **'N' must be equal to the number of input files** specified in the command line argument. Each of the newly created process should open an input file and execute the commands using **execv(….)**.

In the program the sequence of execution should be first command of the first file followed by first command in the fourth file, first command of the seventh file…. till all the commands in these files finishes. After all these files finish their execution, start the first command of the second file, followed by first command in the fifth file …. till all commands in these files finishes. After all these files finishes its execution, start first command of the third file followed by the first command in the sixth file …. till all the

command finishes. For each command in the file the program fork a new process, which should then use **execv(…)** to overlay the forked process with the command specified in the file.
In this exercise we can have any number of parameter for a command (up to 100 parameters each with a maximum size of 100 characters).
Another important point to note is that if a file begins with **setpath**, then you must interpret this line directly, reading a supplied path string, and prefixing further **exec** command paths with this string. For example if you have just had a **“setpath /usr/local/bin”** line, and next line is **“detox a b”** then you should execute **execv “/usr/local/bin/detox a b”.**
The parent process must wait for all its children to complete before termination. You are asked to handle all exception and validation conditions in this program.