BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
IS F462 : Network Programming
I Semester 2014-15
Assignment-1
Weightage: 10% (15M) Due Date of Submission: 30-SEP-2014

==========

Important to Note:

1. Groups of 3 students utmost. Inform group info latest by 10th Sep by sending mail to isf462@gmail.com with subject Assignment1-group-info.
2. **Don't use temporary files and system() function.**
3. **Only working programs will be evaluated. If there are compilation errors, it will not be evaluated.**
4. *Provide makefile for each problem.*
5. For any clarifications please contact me (khari@pilani.bits-pilani.ac.in).

Plagiarism will be thoroughly penalized.

==========

P1. In this problem you will implement a command line interpreter or shell**.** The shell takes in a command from user at its prompt and executes it and then prompts for more input from user. [5M]

- shell should execute any command using fork()and execve() calls. Once the command is completed it should print its pid and status.
- shell should support background jobs.
    - Handles Ctrl-C and Ctrl-z as discussed in Lecture 5 slides.
    - When a command is typed ending with '&', it is run as background job. Assume that you can take only one command at a time for this problem.
    - Each job is identified with a job id (JID), a positive integer
    - Support the following built-in commands
        - *jobs*: lists all background jobs with JID and process id and status
        - *bg <jid>*: restarts job by sending it SIGCONT signal and runs it in the background
        - *fg <jid>*: restarts job by sending it SIGCONT signal and runs it in the foreground
        - *kill <jid>*: terminates the job

Write a program called *shell.c* that implements the above requirements.

P2. Write programs namely parent.c and game.c for the following requirements. Use System V Message Queues for IPC. [5M]

a. Parent takes N, K as command-line argument and creates N children. Note: N<K. Every child knows only one of the other children as shown in the figure below.

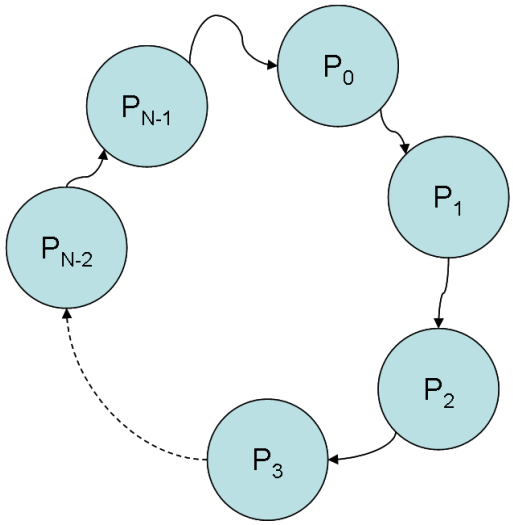
b. Every child is loaded with a game executable compiled from game.c.

c. Parent sends a SIGUSR1 signal to one of the children. That child sends the integer K to the next child it knows. The next child reduces K by 1 and sends to the next child. Like this it goes on.

d. Whenever a child receives it prints *"process <pid> received <number>"*.

e. In the process of decrementing and sending, whichever child receives 0, it sends K to the next and it prints *"i am a foolish process <pid>, defeated".* Next time when it receives an integer, it will simply forward it to the next child without decrementing. That means it will not participate in the game. This way, one after another, the children will be defeated and become passive observers of the game.

f. If a child receives the same number K it has sent, that means it is the winner. That child will print *"i am a truthful process <pid>, only the truth that always wins>".*



P3. Develop a TCP based client-server application for the following. Do not use temporary file to store results. [5M]

- Server takes port number on command line. Server can handle multiple clients concurrently.
  - Server should use process-per-client method for handling concurrent clients.
  - Assume that the searchable files reside in the same directory where the server is.
- Client takes server domain name and a port number. Client connects to the server.
- Client asks the user to enter
  - *query string*: string to search for
  - *in the preceding results*: Y or N. Y means search will be performed in the results of the previous query. N means search will be performed on the file specified.
  - *file*: If second argument is N, then filename has to be specified.

- Client sends the query in the following format (assuming that $ will not occur in the arguments:
  - `Query string$Y/N$filename`
- Server searches the file using the *grep* command and returns the results back to the client.
  - There can be errors on server side namely FILE_NOT_FOUND (if the file not found), RECORD_NOT_FOUND (if the query doesn't match any line the file).
  - *grep* command can be executed as grep <search query> <file>
  - In case, the search has to be done on the results of previous query, those results need to be sent to grep using pipe and dup calls.
  - Do not use temporary file to store results.

Implement *client.c* and *server.c* as per the above requirements.

How to upload?

- Make a directory for each problem like P1, P2 etc and copy your source files into these directories.
- Tar all of them into idno1_idno2_idno3_ass1.tar
- Send to isf462@gmail.com with subject *NP141-assignment1-submission* within due date.

===End of assignment===