BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
IS F462 Network Programming
II Semester 2016-17
Assignment-1
Weight: 11% (66M)   **Due Date of Submission:** 02-Mar-2017

===

Important to Note:

1. Group of maximum 3 students.
2. There are two programming problems.
3. For any clarifications please contact me (khari@pilani.bits-pilani.ac.in).

Plagiarism will be thoroughly penalized.

===

P1. [Unix shell implementation]

You are required to build a bash-like shell for the following requirements. Your program should not use temporary files, popen(), system() library calls. It should only use system-call wrappers from the library. It should not use *sh* or *bash* shells to execute a command.

a) Shell should wait for the user to enter a command. User can enter a command with multiple arguments. Program should parse these arguments and pass them to execv() call. For every command, shell should search for the file in PATH and print any error. Shell should also print the pid, status of the process before asking for another command.

b) shell should support <, >, and >> redirection operators. Print details such as fd of the file, remapped fd.

c) shell should support any number of commands in the pipeline. e.g. `ls|wc|wc|wc`. Print details such as pipe fds, process pids and the steps. Redirection operators can be used in combination with pipes.

d) shell should support two new pipeline operators "||" and "|||". E.g.: `ls -l || grep ^-, grep ^d`. It means that output of ls -l command is passed as input to two other commands. Similarly "|||" means, output of one command is passed as input to three other commands separated by ",".

e) Shell should mask all signals except SIGQUIT and SIGINT. When SIGINT is received, it should print last 10 commands executed by the user, along with status of each. When SIGQUIT is pressed, it should ask user "Do you really want to exit?". If yes, it should exit.

Deliverables:

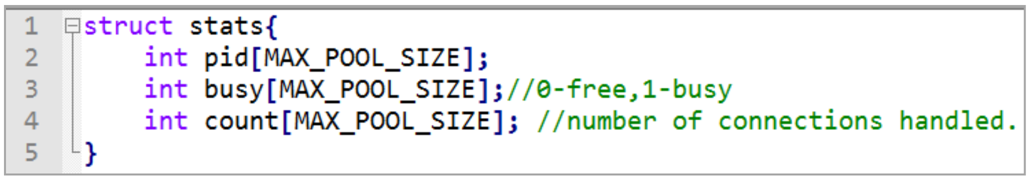
- Brief Design Document (.pdf)
- Code in folder named Code
- Test cases in testcases.txt

[20 M]

P2. [Web Server implementation]

A web server needs to be designed using pre-forking (process pool) model. Write a program named webserver.c and fastCGI.c for the following requirements. Server can accept two kinds of requests: request for .cgi files (dynamic content) and request for any "other" file type (static content).

a) Initial process pool size(I) and port number is taken as the command-line argument. Server adjusts the pool size by taking note of the free processes in the pool. The adjustment is done using the following



```
struct stats{
    int pid[MAX_POOL_SIZE];
    int busy[MAX_POOL_SIZE];//0-free,1-busy
    int count[MAX_POOL_SIZE]; //number of connections handled.
}
```

formula. If number of free processes in the pool exceed 25% of I, kill the excess processes which have handled highest number of connections. If number of free processes go below 25% of I, add new processes to the pool to bring to 25% of I. The number of free processes is computed using a structure in *shared memory*. Structure is updated by processes in the pool and the parent process. Structure is given below.

b) When web server receives a connection for either .cgi or other file types, such as .html, .pdf, .doc, or image files, this connection is taken up by one of the process in the pool. This process checks the file type. If the file type is .cgi, it will process according to steps given later. For other type of file, it will check whether that file exists/accessible and send that file to the user. Process closes the connection when client closes it or observes that no request received for last 5mins on that connection.

c) Web server also keeps a log file (log.csv) with columns as timestamp, user ip address, user port number, user agent, request size, query string, process time, HTTP error codes for each request received. This is for all files .cgi or other. Logging is done by the parent process which accesses these records using a message queue and writes them to log file.

d) When a request for .cgi is received, process sends this request to a FAST CGI process using full duplex pipe or TCP socket. This option is taken on CLA (-p or -t). A fast CGI process is created by each server process when .cgi request is received for the first time. A fast CGI process is a persistent process i.e. once created it stays as long as server process exists. A fast CGI process waits for the request, processes it and sends back the reply on the same connection. A server process after sending .cgi request to fast CGI process, waits for the reply. It receives the reply and sends back to the client. How to pass request to a fastCGI process is explained here.

   Suppose a request such as `localhost/getStudentName.cgi?idno=2013A1PS999P` is received. This request is sent in the following format to fastCGI process. It will execute a file named getStudentName.cgi which is a compiled C file, get the output and pass onto the server process. The protocol used in fastCGI has multiple features. Here we will go for a simple format. Server process will just forward the HTTP request received from the client to fastcgi process. fastCGI process will reply using the HTTP response format.

Program should print details of process pool creation, stats when read by parent, log entries to output.

[46M]

Deliverables:

- Brief Design Document (.pdf)
- Code in folder named Code
- Test cases in testcases.txt

How to upload?

- Create group.txt file and put idno, name of members into this file.
- Make a directory for each problem like P1, P2 etc and copy deliverables into these directories.
- Tar all of them including group.txt into idno1_idno2_assignment1.tar
- Upload on http:/nalanda.

===End of assignment===