BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
IS F462 : Network Programming
II Semester 2014-15
Assignment-1
Weightage: 12% (36M) Due Date of Submission: 31-MAR-2015

==========

Important to Note:

1. Group of maximum 3 students.
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. Upload instructions are given at the end.
6. For any clarifications please contact me (khari@pilani.bits-pilani.ac.in).

Plagiarism will be thoroughly penalized.

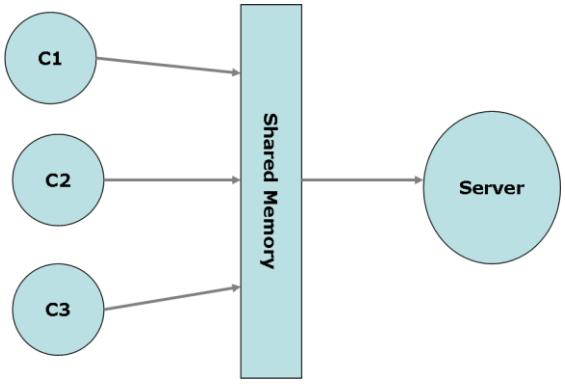
==========

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. **[10M]**

- shell should execute a command using fork()and execve() calls. It should not use temporary files, popen(), or system() calls. It should not use sh or bash shells to execute a command. Once the command is completed it should print its pid and status.
- shell should support <, >, and >> redirection operators.
- shell should support pipelining any number of commands. E.g.: `ls|wc|wc, cat x| grep pat| uniq| sort`
- 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 ",".

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

P2. Write C programs shmclient.c and shmserver.c that do the following. **[10M]**



- Server creates a shared memory of 1 MB. Shared memory is protected by a semaphore *S*. Clients write messages into shared memory. Server sums up *a* and *b* and put the result into *total*.

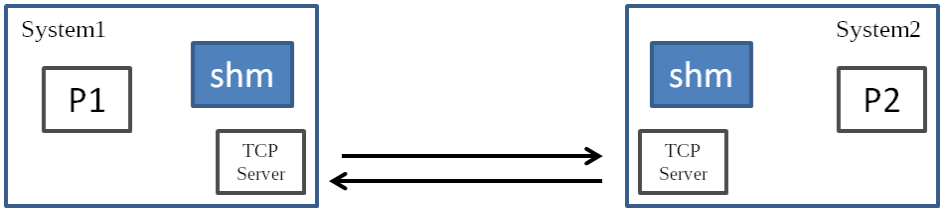
- Message is of the format:

```
struct message{
            int type;
            int pid; //client's pid
            int slno; //incremented for every message
            int a; //any number
            int b; //any number
            int total;//total of a and b, processed by server
            };
```

- A client writes a message *m* of type 1 into shared memory by acquiring sizeof(*m*) bytes from *S*. A client can write several messages into shared memory at one time. But as soon as a client completes writing, control is given to server to read a message from the shared memory.
- Server reads one message at a time, processes and writes the reply with type of client's pid into the shared memory after acquiring necessary bytes from *S*. Once the server writes the reply, control is given to the client, who needs to read that reply. Such a client can't write but only read from shared memory. After the client completes reading, either server (for reading) or any client (for writing) can get chance to access shared memory.

Use semaphores wherever necessary to control access to shared memory. Print client's pid, slno, a and b, shmid, semaphore value for every message the client puts in. Print server's pid (with label "server"), slno, a, b, sum of a and b, shmid, and semaphore value for every message it processes. When server is exited through ctrl-c, it prints pid wise count of messages it processed.

P3. In this problem, goal is to facilitate two processes on two different systems use shared memory for IPC as if they are on the same system as shown in the diagram below. **[16M]**



- P1 and P2 will use shared memory for inter process communication. But they use the following API instead of usual System V shared memory API.

```
int rshmget(key_t key, size_t size);
void rshmat(int rshmid, void* addr);
int rshmdt(int rshmid, void* addr);
int rshmctl(int rshmid, int cmd);
void rshmChanged(int rshmid);
```

API	**TCP Server**	**Remote TCP Servers**
`rshmget`(): it prepares a message with appropriate contents and send to local TCP server.	It checks the key. If it already exists it returns the rshmid otherwise, it creates shared memory locally and stores the shmid. Generates the rshmid, a random number, and sends a message to other nodes about the	Create the shared memory locally and map it with the rshmid in the structure.

	new shared memory creation. Returns the rshmid. This node becomes the owner of the shared memory.	
`rshmat()`:it prepares a message with appropriate contents and send to local TCP server.	It calls shmat() and returns the return address of shmat(). Sends a message to other nodes to increase ref count.	Increase ref count for the corresponding shm segment.
`rshmdt()`:it prepares a message with appropriate contents and send to local TCP server.	It calls the shmdt() and sends message to other nodes to decrease ref count.	Decrease ref count for the corresponding shm segment.
`rshmctl()` : only command applicable is IPC_RMID. it prepares a message with appropriate contents and send to local TCP server.	It calls the shmctl() and informs the same to other nodes using multicast message	Remove the shared memory corresponding to rshmid that is received from the owner.
After completing the write operation, the application calls `rshmChanged()`.it prepares a message with appropriate contents and send to local TCP server.	The server reads the data from shared memory and sends to all other nodes to update their local shared memory segments.	Update the local shared memory corresponding to rshmid received.
	When the server starts up, it sends hello message to the group. It rears the replies and updates its internal state tables.	Reply with the local state table

- TCP server stores the following information for every shared memory segment request it receives.

```
struct rshminfo{
      int rshmid;
                  /*unique id across all systems. created by the
                  first system*/
      key_t key;        /*key used to create shm segment*/
      int shmid;        /*shmid returned by the local system*/
      void *addr;       /*address returned by the local system*/
      int ref_count;    /*no of processes attached to*/
      struct sockaddr_in *remote_addrs; /* list of remote end
      points*/
};
```

- TCP server maintains connections to all remote TCP servers. It uses message queues for communicating with local system processes and TCP connections to communicate with remote systems.

Implement client.c (run as P1 or P2), rshmAPI.c (implementation of API) and rshmServer.c (run as TCP server) for the above specifications. client.c uses rshmAPI.c.

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 your source files into these directories.
- Tar all of them including group.txt into idno1_idno2_idno3_ass1.tar
- Upload on nalanda (http://nalanda).

===End of assignment===