BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJASTHAN)
II SEMESTER 2018-2019
ASSIGNMENT-2

Course No.:	**IS F462**	**Course Title:**	**Network Prog.**
Deadline:	**14-April-2019**	**Maximum Marks:**	**60M (15%)**

Important to Note:
1. Group of maximum 2 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. In this problem, two approaches will be taken to implement a chat server. First one is using thread-based and the second one is using event-driven approach.

Approach1: In approach1, new thread is created for every new client.

Approach2: It is a event-driven concurrent server that receives IO notifications through epoll API. All sockets are set to be non-blocking. Message queues are used as FIFO queue to queue the events. Every request goes through at most 3 events: reading, processing, writing.

(a) Implement the above two approaches for the following protocol:

- *JOIN <name>\r\n*: This is the first message sent by the client providing its own name.
- *LIST*: This message will fetch all the connected online user names.
- *UMSG <tname>\r\n <msg>\r\n*: This is the message used for sending message to particular person named *tname*. If such a person is online then message will be delivered, otherwise server sends ERROR <not online>.
- *BMSG <msg>\r\n*: This message will deliver *msg* to all the online users.
- *LEAV\r\n*: This message will make the server remove the client entry in its data structures and close the connection.

(b) Implement a client which can generate traffic to test the above implementation. It should take N, M, T as parameter. N is the number of concurrent connections to server, M is the number of messages within a connection and T is the total number of connections to make.

(c) Choose suitable parameters to compare the performance of the two approaches. Generate necessary data from the executions of server and client ad compare the performance.

Deliverables:

- Design document
- multithread.c, eventdriven.c, client.c
- Comparison

[30M]

P2. TFTP (Trivial File Transfer Protocol) is used to transfer files between two machines. It doesn't need any authentication like FTP. Use UDP socket programming to implement a TFTP server that can handle Read Requests (RRQs). Write requests (WRQs) need not be implemented. TFTP protocol is defined in RFC 1350 [http://www.ietf.org/rfc/rfc1350.txt]. Use any TFTP client [command-line program 'tftp' installed on Linux machines] to test your TFTP server. The server needs to achieve the following requirements:

TFTP server should accept a port number as command line argument over which it waits for incoming connections. <u>It is a concurrent server using IO multiplexing i.e. using select () system cal</u>l.

[shell$ ./tftp 8069]

(a) Your server should print out information about every message received and every message sent with client identity. It uses RRQ, DATA, ACK, and ERROR messages wherever they are required. The sequence of steps required for downloading a file

- The client C sends an RRQ (read request) to Server S (at the specified port), containing the filename and transfer mode.
- S replies with a DATA message to C. Message is sent from a port other than the one it used for receiving RRQ and client needs to send further messages to S at this port. In each DATA, 512 bytes of data is sent.
- All DATA messages are sequentially numbered. Once S sends DATA message, it will not send another DATA message unless it receives ACK for this DATA message. C responds to each DATA with corresponding ACK. Once S receives ACK for DATA number n, it sends DATA number n+1. If the ACK is not received within a specified time, S resends DATA.
- C detects termination of data transfer if it receives DATA with <512 bytes size. If the total size of the file is exactly a multiple of 512, then a zero-size DATA is sent to indicate the termination of transfer.

Your server should support binary mode (octet) of transferring data. Time-out and retransmission mechanisms should be used to ensure delivery of messages. You can use a timeout of 5 seconds for receiving acknowledgements. No of retries for retransmitting can be utmost 3. Server should respond with appropriate ERROR message wherever it is required either when it receives RRQ or while transferring the file. See RFC for list of error options.

(b) Server in (a) can be extended to implement a dynamically calculated timeout value. Use Jacobson's algorithm to compute timeout and update it using the Karns algorithm. Initially start with a reasonable value. Demonstrate it by printing the RTT values, RTO values to console. Refer to Chapter 22 in Textbook.

[30M]

Deliverables:

- Design document
- udp_select.c, udp_select_timeout.c

===End of assignment2===