Problems:

1. Implement a simple file transfer protocol (FTP) using connection-oriented and connectionless sockets. The connection-oriented FTP works as follows: At the client side, the file to be transferred is divided into units of 100 bytes (and may be less than 100 bytes for the last unit depending on the size of the file). The client transfers each unit of the file to the server and expects an acknowledgment from the server. Only after receiving an acknowledgment from the server, the client transmits the next unit of the file. If the acknowledgment is not received within a timeout period (choose your own value depending on your network delay), the client retransmits the unit. The above process is repeated until all the contents of the file are transferred. The connectionless FTP works simply as follows: The file is broken down into lines and the client sends one line at a time as a datagram packet to the server. There is no acknowledgment required from the server side.

2. Develop a concurrent file server that spawns several threads, one for each client requesting a specific file. The client program sends the name of the file to be downloaded to the server. The server creates the thread by passing the name of the file as the argument for the thread constructor. From then on, the server thread is responsible for transferring the contents of the requested file. Use connection-oriented sockets (let the transfer size be at most 1000 bytes per flush operation). After a flush operation, the server thread sleeps for 200 milliseconds.

3. Develop a "Remote Calculator" application that works as follows: The client program inputs two integers and an arithmetic operation ('*','/','%','+','-') from the user and sends these three values to the server side. The server does the binary operation on the two integers and sends backs the result of the operation to the client. The client displays the result to the user.

4. Develop a streaming client and server application using connectionless sockets that works as follows: The streaming client contacts the streaming server requesting a multi-media file (could be an audio or video file) to be sent. The server then reads the contents of the requested multi-media file in size randomly distributed between 1000 and 2000 bytes and sends the contents read to the client as a datagram packet. The last datagram packet that will be transmitted could be of size less than 1000 bytes, if required. The client reads the bytes, datagram packets, sent from the server. As soon as a reasonable number of bytes are received at the client side, the user working at the client side should be able to launch a media player and view/hear the portions of the received multi-media file while the downloading is in progress.

5. Develop a simple chatting application using (i) Connection-oriented and (ii) Connectionless sockets. In each case, when the user presses the “Enter” key, whatever characters have been typed by the user until then are transferred to the other end. You can also assume that for every message entered from one end, a reply must come from the other end, before another message could be sent. In other words, more than one message cannot be sent from a side before receiving a response from the other side. For connectionless communication, assume the maximum number of characters that can be transferred in a message to be 1000. The chat will be stopped by pressing Ctrl+C on both sides.

6. Extend the single client – single server chatting application developed in Q5 using connection-oriented sockets to a multiple client – single server chatting application. The single server should be able to chat simultaneously with multiple clients. In order to do this, you will have to implement the server program using threads. Once a client program contacts a server, the server process spawns a thread that will handle the client. The communication between a client and its server thread will be like a single client-single server chatting application.

7. Develop a multicast chatting tool that will be used to communicate among a group of processes. Each process should be able to send and receive any number of messages. The chat tool should have the following functionalities:

1) Get the message from the user and send it to all the other processes belonging to the group. A process can receive a copy of the message.

2) Read the messages sent by any other process and display the message to the user.

8. Develop an election vote casting application as follows: There are two candidates A and B contesting an election. There are five electorates (processes) and each electorate can cast their vote only once and for only one of the two candidates (A or B). The vote cast by an electorate is a character ‘A’ or ‘B’, sent as a multicast message to all the other electorates. The winner is the candidate who gets the maximum number of votes. After casting the vote and also receiving the vote messages from all other electorates, each electorate should be able to independently determine the winner and display it.