# Introduction

- Most inter-process communication uses the *client-server* model.
- The client process connects to the server process typically to make a request for information.
- Sockets provide the communication mechanism between two computers using TCP/UDP.
- Stream sockets use TCP which is a reliable, stream-oriented protocol.
- Datagram sockets use UDP, which is unreliable and message oriented.

**Creating Socket on Server Side:**

- Create a socket with the socket() system call.
- Bind the socket to an address using the bind() system call.
- For a server socket on the Internet, an address consists of a port number on the host machine.
- Listen for connections with the listen() system call.
- Accept a connection with the accept() system call. This call typically blocks until a client connects with the server.
- Send and receive data using read() and write() system calls.

**Creating Socket on Client Side:**

- Create a socket with the socket() system call.
- Connect the socket to the address of the server using the connect() system call.
- Send and receive data using read() and write() system calls.

# Problem Statement for Assignment 1

Write two C programs, one for a TCP server (which handles requests for a single client) and another for a TCP client.

**Server Side:**

The server creates a socket and listens on some specific port for client connection. After connecting with the client, it waits for client requests. The server contains a default file, and it should be able to perform READX and INSERTX operations on the file as described below.

- The server should parse the string received from the client, which will contain either of NLINEX, READX or INSERTX requests.
- Requests must be strictly in the following format:
    - **NLINEX:** Returns the no. of lines in the file to the client. This is a helper command for the client to know the no. of lines in the file.
    - **READX <*k*>:** Reads the *k*-th line from the file, containing, say, *N* lines and returns it to the client (you might need to find *N* first based on the situation). You must implement both forward and reverse indexing, i.e., for a file containing *N* lines,
        - In forward indexing, lines are indexed as 0, 1, ..., (*N*-2), (*N*-1)
        - In reverse indexing, lines are indexed as -*N*, -(*N*-1), …, -2, -1
        - Thus, the permissible range of *k* is [-*N, N*)
        - If *k* is not specified, read the first line of the file.

      Return the line (if successful) else a message of failure (invalid index, etc.) to the client.
    - **INSERTX <*k*> <*message*>:** Inserts the string specified in *message* in the *k*-th line to the file (*k* can be either forward or reverse index, as discussed above).
        - It should be ensured that <*message*> does not contain newline characters.
        - All the contents of the file from the *k*-th line to EOF should be shifted down by one line.
        - If *k* is not specified, append the message to the end of the file.
        - For security purposes, you should not write to the same file while reading. Use a temporary file to store the new contents, and after

completing all operations replace the original file with the new temporary file.

Return a message of success or failure to the client.

- The server should reject requests that do not adhere to the above format by sending an error message back to the client. Please note that the user should type in these exact commands at the terminal; you cannot use menu-driven control, etc.
- *You are requested to implement the NLINEX, READX and INSERTX commands in such a way that no extra space is used. You should not use additional files (for INSERTX you need to create a temporary file, which is fine). You should also not use arrays or other data structures to store the contents of the file. All these operations should be performed by strictly using file operations such as ftell(), fseek() and rewind().*

**Client Side:**

The client establishes a connection with the server socket on the specified port. Then, it passes NLINEX, READX and INSERTX requests to the server.

- The client takes NLINEX, READX or INSERTX commands from the user, along with the required parameters.
- It constructs the request and passes it to the server
- Fetches the output message from the server and displays it to the user.

**Submission Instructions:**

- You should have two files for server and client, named as **"server.c"** and **"client.c"** respectively.
- Save this in a folder named in the format: **<Roll No.>_CL2_A1**. Compress this folder to tar.gz format, creating a compressed file **<Roll No.>_CL2_A1.tar.gz**. Upload this compressed file to moodle. *Example: If your roll no. is 21CS60R05, the folder should be* 20CS60R05_CL2_A1*, and the compressed file should be* 21CS60R05_CL2_A1.tar.gz.
- Not adhering to these instructions can incur a penalty.

**Marking Scheme:**

| | |
|---|---|
| Establishing connection between server and client, able to exchange messages | **5** |
| NLINEX | **15** |
| READX | **30** |
| INSERTX | **35** |
| Error Handling | **15** |