Socket Project: TCP Server and Client

1. Project Definition

In this project, you will create a simple TCP server and client. The server will listen on a specific port and accept incoming connections. The client will send a message to the server, and the server will echo the message back.

2. Socket Programming Steps

A. Socket Creation

- Every socket connection begins by creating a socket. This means obtaining a file descriptor that represents the connection over the network.

B. Bind and Listen for Server

- On the server side, the socket is bound to a specific IP address and port using bind().
- The server then listens for incoming connections using listen().

C. Accepting the Connection

- When the server is listening, it accepts an incoming connection using the accept() function.
- This call returns a new file descriptor that represents the connection with the client.

D. Handling the Connection

- The server reads the message sent by the client using recv() and sends it back using send().
- The client sends a message and waits for the server's response.

E. Closing the Connection

- Both the client and server close their sockets after the message has been exchanged and processed.

3. Steps in the Project

Step 1: Basic Setup and Build Environment

- Create a directory for your project, and start by creating separate C++ programs for the server and client logic.
- Create a Makefile to compile the project easily.

Step 2: Socket Creation and Connection Opening

- Start by learning how to create a socket on the server.
- Learn to use socket(), bind(), and listen() functions to make the server listen on a specific port.

Step 3: Client Connection Creation

- Create a simple client program that uses socket() and connect() to establish a connection with the server.

Step 4: Sending and Receiving Messages

- The server must use recv() and send() functions to receive and send messages.
- The client will use the same functions to send a message and receive the server's response.

Step 5: Testing

- First, run the server program, then run the client program in a different terminal.
- Verify that the client sends a message, and the server echoes it back.

4. Important Considerations:

- IP address and port: Use localhost (127.0.0.1) and an available port (e.g., 8080) for local connections.

- Error handling: Always check if socket creation and connection opening are successful.

- Always close the sockets: After the connection is handled, remember to close both the client and server sockets.

5. Extensions:

- Once the basic version works, you can expand the project by adding multi-client handling or using the poll() function to manage multiple connections simultaneously.