Multiple clients and server

The goal of the assignment is to understand the thread and event models to handle multiple clients. You are provided with an implementation of a single-server single-client application. The server waits for the client to initiate a connection, using socket, bind, listen, and accept system calls. The client connects to the server using the socket and connect system calls.

Source code:

Clone the homework repository from
git clone https://github.com/Systems-IIITD/CS.git

Implementation details

The CS folder contains two files, server.c and client.c, corresponding to server and client.
Both client and server wait for user input on STDIN using the select system call. If data is available on the STDIN of any party, they send it to the other party. Since select can be used to probe multiple descriptors, both client and server also check if data is available on the socket created for the communication. If any party receives the data, they print it on the STDOUT.

Supporting multiple clients

Your goal is to modify this implementation by supporting multiple clients. In this mode, the server accepts connection requests from multiple clients. Multiple clients can send the message to the server that it displays on STDOUT. After reading a message from STDIN, the server sends it to one of the clients of its choice (random is also fine). There is no need to change the client's implementation.

There are two ways to support multiple clients: **threads** and **events**.

In **thread** mode, the server creates multiple threads to accept connection requests from clients using the accept system call. Alternatively, you can do the accept system call in the main thread in a loop, and after every successful accept, you can spawn a thread to handle the communication with the corresponding client. After establishing the connection, you can use the existing event-based approach (i.e., the event_loop) to exchange information between client and server.

For the **events** mode, modify the existing select implementation to also accept connections from clients. Notice that the connect call at the client writes to the socket used in the listen system call at the server.

Your implementation should support at least four clients.
Useful thread APIs: pthread_create and pthread_join

Running the skeleton code:

Compile both client and server using make in the CS folder.
Run the server using ./server
Run the client using ./client localhost from another terminal.
To send a message, type your message on the client/server terminal and press Enter.

Expected Output:

The user should be able to run four clients from four different terminals. Messages entered on any client's terminal should be displayed on the server's terminal. A message entered on the server's terminal should be displayed on one of the client's terminals.

Submission:

Submit two files corresponding to the thread and the event-based implementations of the server, along with a design documentation. Follow the naming convention of homework and assignments for both files.
Add the following points to your design documentation.

1. Brief overview of your implementation.
2. Are messages sent by all four clients displayed on the server's terminal?
3. Name and roll number of the group members.