# Homework 3: TCP Chat Client/Server Documentation

**1. Purpose of the Task**

The objective of this assignment was to design and implement a multi-user chat system in Linux using the TCP protocol. The system consists of two executables:

- **Server (hw3server)**: multiplexes multiple client connections using select(), manages client state, and routes messages (broadcast and private whispers).
- **Client (hw3client)**: connects to the server, handles user input, and displays incoming messages asynchronously.

**2. Implementation**

**A. Multiplexing and Non-Blocking I/O**

We utilized the select() system call in the main loop of server.c.

The fd_set monitors the listener_fd for new connections and all active client sockets for incoming data.

This ensures the server can accept a new user immediately, even if other users are idle or chatting.

**B. Connection and Identification**

**Handshake:** When a client connects, the server adds them to a client_t array but marks them as "nameless" (State 1).

**Registration:** The client sends its name immediately upon connecting. The server's process_packet function detects the first message as the name, updates the client struct, and prints the required connection message.

**Disconnection:** When recv() returns 0 or -1, the server identifies the disconnected client, prints the disconnection message, and cleans up the file descriptor.

**C. Message Broadcasting**

In process_packet (State 2), any message not starting with @ is treated as a broadcast.

The server formats the string using snprintf to prepend the sender's name and calls broadcast_msg, which iterates through the clients array and sends the data to all active file descriptors.

**D. Private "Whisper" Messages**

The server parses incoming strings to check for the @ prefix.

It extracts the target_name (between @ and the first space).

The function get_client_index_by_name searches the client list. If the target is found, the message is sent *only* to that specific socket descriptor using send_to_fd.

**E. Client "Exit" Handling**

In client.c, before sending input to the server, the code checks strcmp(tmp, "!exit").

If a match is found, the client sends the message so others see "Client exiting", prints its own termination message, and closes the socket cleanly.

**F. Robustness (TCP Stream Handling)**

TCP is a stream protocol, meaning multiple messages can arrive in a single recv call (e.g., "Hello\nBye\n").

Our server implements a **per-client buffer** (pending_buf). Incoming data is appended to this buffer, and a loop extracts complete lines (delimited by \n). This prevents data loss if packets are fused or fragmented by the network.