## **Introduction**

The project is a multi-client chat server that supports direct communication between clients, each of whom receives a unique identifier upon connecting. It allows clients to send messages to each other directly, retrieve conversation histories, and gracefully handles disconnections. The server keeps track of all active clients and their interactions, enabling efficient communication and maintaining a log of conversations.

## **Server Design and Client Interactions**

The server is designed to handle multiple client connections simultaneously. It utilizes threading to manage each client in a separate thread, ensuring that the server can listen to and communicate with multiple clients without blocking.

### **1. Client Communication**

* Each client connects to the server over a TCP connection.
* Clients can send messages directly to other clients by specifying their target client ID.
* The server keeps a record of all active clients and manages their communication, including forwarding messages and providing conversation history.

### **2. Storage of Conversation History**

* All conversations between clients are stored in a history dictionary.
* This dictionary is keyed by a pair of client IDs, with the value being a list of exchanged messages between those clients.
* The server updates this history whenever a message is forwarded from one client to another.

### **3. Client Identification and Management**

* Each client receives a unique ID upon connection, which allows the server to identify and manage each client independently.
* These IDs are used by clients to communicate directly with each other.
* The server maintains a dictionary called client\_list where each client ID maps to its respective connection and address information.

## **Commands Supported by the Server**

The server supports several commands that clients can use to interact with the server and each other. Below are the key commands:

### **1. Normal Input**

* Any text input that doesn't match a specific command is treated as a normal message.
* The message is logged by the server, and an acknowledgment is sent back to the client confirming receipt.

### **2. List Command**

* **Command**: list
* **Description**: The client can request a list of active client IDs from the server.
* **Functionality**: The server compiles a list of currently connected client IDs and sends it back to the requesting client.
* **Client Disconnection Handling**: When a client disconnects, either by sending an "exit" command or through an unexpected disconnect (e.g., Ctrl-C), their ID is removed from the client\_list.
* **Data Structure**: client\_list is a dictionary where each key is a client ID, and each value is a tuple containing the client’s connection object and address.

### **3. Forward Command**

* **Command**: Forward <ID> <message>
* **Description**: Allows a client to send a message directly to another client identified by their ID.
* **Functionality**:
  + The server checks if the target client exists in the client\_list.
  + If the target client exists, the message is forwarded to them, and an acknowledgment is sent to the sender.
  + The message is stored in the history dictionary under a key representing the pair of client IDs involved.
  + **Data Structure**: history is a dictionary where each key is a tuple of client IDs (e.g., (1, 2)) and the value is a list of messages exchanged between those clients.
* **Error Handling**: If the target client does not exist, the server sends an error message back to the sender, informing them that the target client ID is not found.

### **4. History Command**

* **Command**: history <ID>
* **Description**: A client can retrieve their conversation history with another client by specifying the target client ID.
* **Functionality**:
  + The server looks up the conversation history between the requesting client and the specified target client.
  + If a history exists, it is sent back to the requesting client, message by message.
  + If no history is found, the client is informed that there is no history available with the specified ID.

### **5. Exit Command**

* **Command**: exit
* **Description**: Allows the client to disconnect from the server.
* **Functionality**:
  + Upon receiving this command, the server sends a "Goodbye" message to the client, closes the connection, and removes the client from the client\_list.
  + After a client disconnects, if there are no more clients connected, the server automatically shuts down by setting the shutdown\_event.

## **Error Handling**

The server is designed to handle errors and unexpected disconnections gracefully, ensuring a robust communication system. Key aspects of error handling include:

### **1. Client Disconnection Handling**

* **Client-initiated Disconnection**: When a client sends an "exit" command or disconnects unexpectedly (e.g., using Ctrl-C), the server:
  + Removes the client from the client\_list.
  + Closes the client's socket to free up resources.
  + Notifies other clients if necessary, such as during list updates.
* **Graceful Handling**: This ensures that the server’s record of active clients remains accurate and avoids attempts to communicate with disconnected clients.

### **2. Server Shutdown Handling**

* **Server-initiated Disconnection**: If the server is shut down using Ctrl-C, it gracefully closes all client connections before terminating:
  + The shutdown\_server function sends a shutdown message to each connected client, allowing them to properly close their connections.
  + After notifying clients, the server sets the shutdown\_event, causing the main server loop to exit.
  + The server then closes its listening socket, ensuring that the port is freed for future use.
* **Client Notifications**: This ensures that clients are aware of the server shutdown and can handle it accordingly, avoiding unexpected connection drops on the client side.

## **Conclusion**

The multi-client chat server is a robust and flexible solution for managing client-to-client communication over a TCP network. It supports direct messaging, conversation history, and handles both client and server-initiated disconnections gracefully. The use of threading allows it to manage multiple simultaneous connections effectively, making it suitable for a variety of chat-based applications.