Multi-Threaded Chat Server - CS425 Assignment 1

Group Members: Tanmey Agarwal (211098), Kumar Kanishk Singh (210544), Sunny Raja Prasad (218171078)

1 Features

1.1 Implemented Features

- TCP-based server that listens on port 12345
- Handles multiple concurrent client connections
- Authenticating only those users which are in `users.txt` file.
- Supports private messaging (`/msg <username> <message>`)
- Supports broadcasting messages (`/broadcast <message>`)
- Group messaging (`/group_msg <group_name> <message>`)
- Group management:
  - `/create_group <group_name>` to create a group
  - `/join_group <group_name>` to join a group
  - `/leave_group <group_name>` to leave a group

1.2 Not Implemented Features

- No frontend, have to type messages from terminal.
- Persistent storage for messages even if server is closed.

2 Design Decisions

2.1 Multi-threading Approach

- A new thread is created for each client connection to handle multiple clients concurrently.

- `std::mutex` is used to synchronize access to shared data structures such as user lists and group mappings.
- The server maintains four main data structures:
  - `std::unordered_map<int, std::string> clients`: Maps client socket descriptors to usernames.
  - `std::unordered_map<std::string, std::string> user_credentials`: Maps usernames to passwords.
  - `std::unordered_map<std::string, std::unordered_set<int>> groups`: Maps group names to client socket descriptors.
  - `std::unordered_map<std::string, int> sock`: Maps client username to clients socket.

2.2 Synchronization Decisions

- Used `std::lock_guard<std::mutex>` for thread-safe access to shared resources.
- Prevented race conditions by locking before modifying shared data structures.

3 Implementation

3.1 High-Level Overview

- **Server Initialization**
  - Creates a socket and binds it to a port.
  - Listens for incoming client connections.
- **Client Connection Handling**
  - Authenticates user using `users.txt`.
  - Creates a new thread to handle client communication and detach that thread from the main thread.
  - When a new user joins, others receive a message, and the server confirms the connection.
- **Command Processing**
  - Parses incoming messages to determine the type of command (`/msg`, `/broadcast`, etc.).
  - `/msg`-The recipient's client socket is found using the `sock` map, and then the message is sent to that socket.

- `/broadcast`-First, the maps are locked using `std::mutex`, and then the `clients` map is iterated, sending the `/msg` to each individual client.
- `/create group`-First, the maps are locked using `std::mutex`, and then the groups map is updated by adding a new group in groups and making client a part of this group.
- `/leave group`-First, the maps are locked using `std::mutex`, and then the `groups` map is updated by removing the client from the specified group's members.
- `/group msg`-First, the maps are locked using `std::mutex`, then iterating over all group members and sending them direct message.

3.2 Code Flow

1. Server starts and listens for connections.
2. Client connects and enters username/password.
3. Server authenticates the client.
4. Client sends messages (private, broadcast, or group messages).
5. Server routes messages appropriately.
6. Client disconnects, and server removes them from active clients and groups.

4 Testing

4.1 Correctness Testing

- Verified authentication by providing valid and invalid credentials.
- Checked message delivery for private messages, broadcasts, and group messages.

4.2 Stress Testing

- Joined multiple concurrent client connections.

5 Challenges & Solutions

- Handling concurrent client connections → Used multi-threading with `std::mutex`.
- Ensuring thread safety → Used `std::lock_guard<std::mutex>` for shared resources.

- Parsing user commands correctly → Used `std::string::find()` and `substr()`.
- Debugging segmentation faults → Added proper error handling and logging.

6 Restrictions

- Maximum clients: Limited by system resources and thread handling.
- Maximum groups: Theoretically unlimited but constrained by memory.
- Maximum message size: 1024 bytes per message.

7 Individual Contributions

- Tanmey Agarwal (211098): Server architecture, threading, synchronization.
- Kumar Kanishk Singh (210544): Message handling, group management, authentication.
- Sunny Raja Prasad (218171078): Testing, debugging, documentation (README).

8 Sources

- Beej's Guide to Network Programming.
- C++ Reference for `std::mutex`, `std::unordered_map`.
- GeeksforGeeks articles on multi-threading and socket programming.

9 Declaration

- We declare that this assignment was completed independently without plagiarism.

10 Feedback

- The assignment was well-structured and helped in understanding multi-threaded socket programming.
- Some clarifications regarding the expected behavior of group messaging would have been helpful.