Assignment 1

Chat Server with Groups and Private Messages

Group Member 1: **Prabhat Kumar Yadav (220774)**

Group Member 2: **Atharv Moghe (220250)**

Group Member 3: **Kuldeep Sandip Thakare (220557)**

Course: CS425: Computer Networks

Instructor: Adithya Vadapalli

TAs Incharge: Rohit Kumar and Naman Baranwal

How to Run

- Ensure you have `g++` and `make` installed.
- Run ‘make‘ in terminal to compile both server and client.
- Run the server using ‘./server_grp‘ in one terminal.
- Run multiple clients in separate terminals: ‘./client_grp‘.

Assignment Features

Implemented Features

- Thread-safe handling of client connections and group operations using `mutex` and `lock_guard`.
- All the features mentioned in the assignment are implemented. Apart from that below are some improvisations.
- When a client joins a group, all existing group members of that will be informed, just like when somebody joins the chat.
- If any client leaves the group, all remaining group members would be informed. Same for when anybody leaves chat.
- When a client broadcast, all other chat member would receive in format [Broadcast from `sender_name`]: `message`.
- For any unknown command, error will be printed as ”Error: Unknown Command”.
- No matter in what way client leaves group (e.g. by Ctrl+C or shutting the terminal), left messsage will be delivered to every other chat member.
- Empty messages and group names are rejected. Also for any type of wrong command like joining a non-existing group or making a group with name which is already a group name etc. are met with proper error command.
- Clients leaving the chat would be removed lazily from all groups.
- Group with zero members can exist.

Not Implemented Features

- Persistence of group data across server restarts.

Design Decisions

Threading Model

- A new thread is created for each client connection to handle incoming messages and commands. This allows the server to handle multiple clients concurrently without blocking.
- Threads are detached to run independently in the background, ensuring that the main server thread can continue accepting new connections.

Synchronization

- Mutexes are used to protect shared data structures such as `clients` and `groups` to prevent race conditions.
- `lock_guard` is used to ensure that mutexes are automatically released when the scope ends, reducing the risk of deadlocks.

Lazy Group Cleanup

- When a client disconnects, it is not immediately removed from all groups. Instead, the server performs a "lazy cleanup" by removing disconnected clients from groups only when a group message is sent.
- This design decision was made to optimize performance and avoid unnecessary cleanup operations when no group activity is happening.

Implementation

High-Level Idea of Important Functions

- `int_main`: Handles server, accept new client connections and create new thread to handle client.
- `handle_client`: Handle authentication, and message processing.
- `send_msg`: Sends a message to a specific client.
- `load_users`: Fills the users map with username and password from `users.txt`.
- `broadcast_msg`: Sends a message to all other connected clients.
- `private_msg`: Sends a private message to a specific user.
- `group_msg`: Sends a message to all other members of the group.
- `create_group`: Creates a new group.
- `join_group`: Adds a client to an existing group.
- `leave_group`: Removes a client from a group.

Code Flow

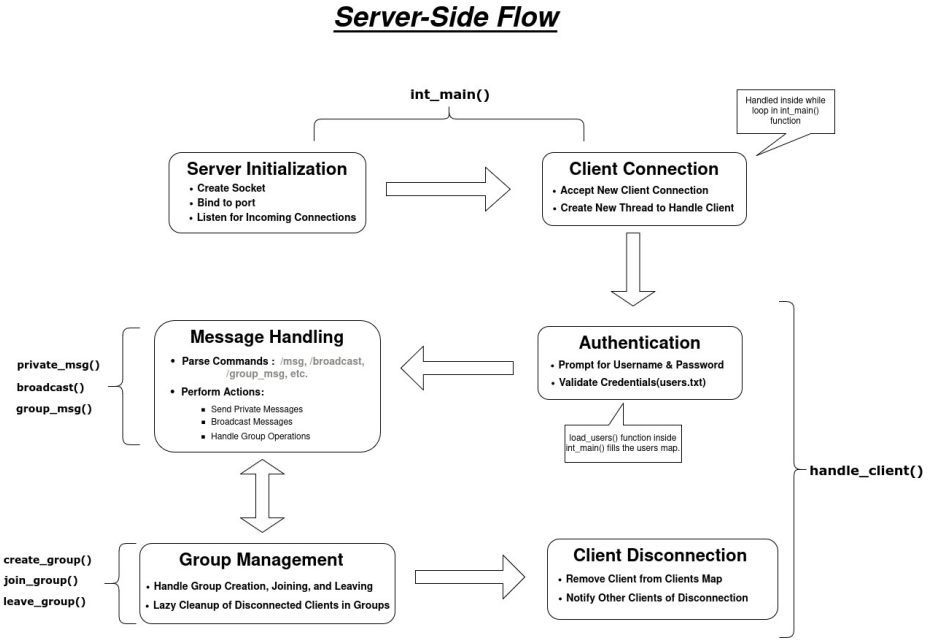

Figure 1: Flow diagram of the server and client interaction.

Testing

Code testing for all commands was done manually with 7 concurrent clients.

Correctness Testing

- Tested user authentication with valid and invalid credentials.
- Verified private messaging between users and ensured broadcast messages are received by all connected clients.
- Tested group creation, joining, and leaving functionalities.
- Tested error messages with all types of wrong commands.

Stress Testing

- Jmetre was used for stress testing.
- Simulated 200 clients connecting to the server simultaneosly.

Restrictions

- **Maximum Clients**: The server can handle up to 1024 concurrent clients, limited by system resources. Backlog limit is set to 30.

- **Maximum Group Size**: The maximum size of a group is 1024 since that is maximum number of concurrent clients.
- **Maximum Groups**: The number of groups is limited by available memory.
- **Message Size**: Messages are limited to 1024 bytes due to the `BUFFER_SIZE` setting.

Challenges

- Initially it was little tough to properly use `mutex` inside code, but by seeing more examples on internet, we understood.
- During initial draft of code, it was very messy and debugging was very tough, but we overcame it by using function for every command and action and it became easier to debug.

Contribution of Each Member

- Implementation was handled by Prabhat.
- Design was handled by Kuldeep.
- Testing was handled by Atharv.
- Everyone contributed in Readme.

Contribution in percentage

- Prabhat Kumar Yadav (220774) - 40.
- Kuldeep Sandip Thakare (220557) - 30.
- Atharv Moghe (220250) - 30.

Sources Referred

- Beej's Guide to Network Programming: `https://beej.us/guide/bgnet/`
- C++ Documentation: `https://en.cppreference.com/`
- Linux Manual Pages: `https://man7.org/linux/man-pages/`

Declaration

We declare that we have not indulged in any form of plagiarism while completing this assignment. All work is original and has been done by the group members.