Robbie Fishel
Birju Patel
Dr. Amy Babay

Project 1: Final Design

Introduction:

Specific commands to run the program are listed in the README. These are Linux commands, which is what we used for development. We have configured the Dockerfile to automatically download all of the dependencies. The readme also contains a guide on how to use the client program. The server is configured to always run on port 18861. To change this, manually modify the port_num variable in server.py.

Our language of choice for this project was Python. We used RPCs to implement the application, specifically the RPyC library (documentation here). This library handles many of the synchronization issues between client and server.

RPyC runs over a TCP connection, so there is no need to worry about messages getting lost in the network. The RPyC logic automatically deals with client server interactions because it formats and sends messages for us. Additionally, it is symmetrical, meaning the client can run code on the server, and the server can run code on the client. This allows for asynchronous event handling, which we used to implement message updates.

Server:

The server contains a ChatServer class, which is an RPyC service. When a new client connects to the server, a new ChatServer instance will be created to serve that client. All instances share access to the group_dict and cv_dict data structures. Access to these data structures is protected by a mutex lock. Methods with the exposed prefix are directly callable by the client.

- __init__()
  - Allows a new client to connect to the server. Initializes local ChatServer variables to initial values.
- exposed_set_username(String)
  - Resets the client's username.
  - If they are in a group, they are removed from that group and signal to all other members of that group that the group membership has changed.
- exposed_join_group(String, callback_function)
  - The client provides the name of the group they want to join and a callback function that the server calls asynchronously to send updates to the client.
  - Adds the client to an existing group.
  - If no such group exists, it creates a new group and creates a condition variable for this group.

    - If the client is already in a different group, they are removed from their current group and all other members are signaled that the group's membership has changed.
    - Spawns a new monitor thread to check when new updates are made to this group. This thread runs the update_client_status function, which waits on a condition variable that other threads signal to each time they modify the group data.
    - After receiving a signal, the thread copies the group's current data from group_dict, as well as the last 10 messages in the log, and sends it back to the client by calling the callback_function.
- exposed_write_message(String)
    - Adds a new message to the group that the client is currently in.
    - Signals to all other members of the group that a new message was written to the log.
- exposed_add_like(int)
    - Given a message id, if that user hasn't already liked the message, add a new like to that message.
    - Signals to all other members of the group that a new like was added.
- exposed_remove_like(int)
    - Given a message id, if the user is listed as having liked that message, removes that like.
    - Signals to all other members of the group that a like was removed.
- exposed_message_history()
    - Returns all messages written in the group the user is currently in, to be displayed to the calling client.
- update_client_status()
    - The monitor thread runs this method, which contains a loop that monitors the group_dict for any changes.
    - Whenever any thread changes a particular group in group_dict, it sends a signal to the group's corresponding condition variable in cv_dict.
    - This method waits on that condition variable. Whenever it is signaled, the thread wakes up and copies the current group's data from group_dict. It only copies the 10 most recent messages in the group.
    - It then sends this data to the client by calling the callback_function provided to the server in exposed_join_group.

The client must first connect to the server, then set their username, then join a group, and then may write messages, like, dislike, or request the message history. If they attempt to do this in an incorrect order, the server will raise an exception.

The server prints debug messages by default. To turn them off, manually change the value of debug_prints to False.

Client:

Each client will run an instance of the ChatClient class. This class will contain a group_obj data structure. Clients will interact with the application through a command line interface built as specified in the project description. Initially, the client will connect to the server. Whenever the user issues a command, the client will then make the corresponding RPC call to the server.

A client-side callback function is created in order to update the client's group_obj data structure whenever an update happens to the group on the server side. The client spawns a thread that listens for any time this callback function, called update_group_obj, returns a value. When it does, the thread updates the client's group_obj data structure, and prints the updated information to the console automatically.

By default, the client clears the console every time we print the most updated data. To prevent this, manually comment out the clear() function calls.

Data Types:

- Group
    - name: str, name of this group
    - users: List[str], all users currently inside this group
    - messages: List[Message], all messages in this group
    - next_msg_id: int, assigns unique message id to each message
- Message
    - id: int, id of message
    - content: str, content of message
    - sender: str, username of the sender
    - likes: List[str], all usernames of users that have liked this message

Data Structures:

- Client
    - group_obj: Most up-to-date version of the client's current group as fetched from the server. An updated copy of this data structure is sent from the server to all clients whenever the group is changed.
- Server
    - group_dict: Stores data for all groups. A dictionary with group names as the keys and Group objects as values.This mapping is present for efficient access to specific groups when necessary.
    - cv_dict: A dictionary with group names as the keys and condition variables as values. Upon the corresponding client joining a group, the ChatServer instance associated with that client will spawn a thread which waits on the condition

variable for the group stored here, so that whenever an update occurs that client can be notified and update its user interface accordingly.