Assignment 1, P2: Chat Server in C
Design Document

Dhruv Rawat & Anirudh Singh

October 29, 2022

1 Problem description

For this problem, we had to implement a chat server which enables multiple users to send messages to each other. A user should be able to join or leave the chat system, get list of users with the status, send messages to any or all either online or offline.

2 How to run

```
$ gcc chat.c -o server
$ ./server <PORT>
```

In a separate terminal window, run a client by

```
$ telnet localhost <PORT>
```

You can have multiple clients by running telnet in different terminal windows. To access the chatapp from a different machine, use the IP address of the host(the machine where the server is running) instead of localhost.

3 Features

3.1 Enter or leave the chat

```
$ help
```

The `help` command prints a helpful message describing all the allowed commands. These commands are also described in the following sections.

3.2 Enter or leave the chat

```
$ select <m_type>
```

The application prompts every user to enter a unique username when he/she first joins the chat. After that each user can enter a unique chatroom by entering the m_type of the user they want to communicate with and exit that chatroom by typing exit.

Messages persist in a chat, so even if a user is offline, he/she can view the messages received while they were offline when they get online back again.

3.3 Display list of users

```
$ list
```

The list of users currently registered on the server can be viewed by using the `list` command. It also displays the status of each user.

Note: the list of users/groups is not updated automatically on the clients after each change(i.e. when a user joins/leaves or when a group is created). Periodically refresh this data by entering the `list` command.

3.4 Broadcast messages

```
$ broadcast
```

Users can broadcast messages to all other users by entering the broadcast mode. All users present in any chatroom can receive a message sent by all users who are broadcasting.

Users in broadcast mode can send to all users (online/offline), but won't receive any messages from anyone. They can see all the pending messages they received upon entering any normal chatroom.

3.5 Exiting

```
$ exit
```

Use the `exit` command to exit a chat room or to close the client.

If the user is inside a chat room, group chat or is broadcasting, the `exit` command will bring them out of the room and into the main lobby.

If the user is in the lobby, the `exit` command will terminate the connection. When a user exits, their status is updated to `offline`. The user becomes `online` when they re-login with the same username.

Note: Do Not send SIGTERM/SIGKILL etc to close a client. Use the exit command instead to do it gracefully.

3.6 Extra Feature: Message Timestamps

Each chat message contains the timestamp of when it was sent, similar to modern chat apps.

3.7 Extra Feature: Groups

The chatapp also supports multi-user Groups. Groups can be accessed using the following commands:

```
$ create <group_name> <m_type_1> <m_type_2> ...
```

Use the `create` command to create a new group chat or to modify an existing group chat with unique groupnames. If a group chat with the given name doesn't exist, a new one will be created with the given name containing users corresponding to the mtypes specified in the command.

If a group with the given name already exists, the members list will be modified to only include the new list of members provided along with the command.

```
$ group <m_type>
```

Use the `group` command to enter a group chat by specifying the mtype of the group. The server will asynchronously send any messages sent by any member of the group chat. messages will be displayed with the timestamp and the name of the user who sent that message.

4 Implementation Details

There is only one message queue responsible for all intra-process communication allowing the transfer of messages between different clients.

The server has 1 process to listen to new tcp connection requests, 1 process containing the latest information about active users and groups which conveys this information to all other processes using the message queue, and 1 dedicated process for each client. Each dedicated process creates 1 more temporary child process to asynchronously receive all requests sent by any other client via the message queue, and the parent process itself is responsible for putting the messages sent by the connected client on the message queue.