Assignment 7 (Multi-client chat server system)
Report

Aftab Hussain
22CS60R54

Global Data Structure

I defined the following struct for maintaining the required information.

- `client_table` struct defines a client's socket file descriptor and unique client ID
- `Message` struct defines a message's sender ID, text content, and a pointer to the next message in the linked list
- `message_table` struct defines a LInked list of Message for a specific client ID
- `group_info` struct defines information about a group, including the group ID, an array of admin IDs, whether it is a broadcast-only group, an array of member IDs, the group size, and the admin array size
- `pending_group` struct defines information about a pending group request, including the group ID, the number of pending requests, and arrays of member and admin IDs

There are four arrays of pointers to these structs, including `ctab` for client_table, `mtab` for message_table, `gtab` for group_info, and `pgtab` for pending_group. The size of these arrays is defined by constants `CLIENT_LIMIT` and `GROUP_COUNT_LIMIT`.

Handle "/active"

First It calls the `printRecClient()` function, which returns an integer array containing the IDs of all currently active clients.

The function then retrieves the client ID associated with the provided `sockfd` using the `getClientId()` function and stores it in a variable named `cid`.

The function then loops through all elements of the `active` array and checks if the element is not equal to -1. If the element is not equal to -1, it means that the client with the corresponding ID is currently active and append that client ID to `active_now string`

If the client ID matches the `cid` variable, the function appends the client ID to the `active_now` array along with the string " (You)" .

Finally, the function calls the `sendMsg()` function, which sends the `active_now` string to the client with the provided `sockfd`.

Handle “/send”

If the client with the specified `cid` is currently active (i.e., connected to the server), the function creates a new `Message` struct dynamically using the `malloc()` function and initializes its `sender`, `text`, and `next` fields with the sender's client ID, the message text, and `NULL`, respectively. The `addMsgEntry()` function is then called to add the message to the message table for the specified client ID.

If the client with the specified `cid` is not currently active, the function sends an error message to the client indicating that the recipient might have left the room or that the client ID is invalid.

Handle “/broadcast”

The function then loops through the array of active client IDs and checks if the ID is not equal to `-1`, the `sender` ID. If the ID meets these conditions, the function creates a new `Message` struct dynamically using the `malloc()` function and initializes its `sender`, `text`, and `next` fields with the `sender` ID, the `message` text, and `NULL`, respectively. The `addMsgEntry()` function is then called to add the message to the message table for the current client ID.

Handle “/makegroup”

The code defines a function `handle_makegroup` that is responsible for creating a new group chat. It takes two arguments: the `sockfd` (socket file descriptor) of the client who sent the command to create a group.

The function first creates two integer arrays, `gmembers` and `admins`, to store the group members and the group administrators, respectively. It then initializes these arrays by setting all

elements to `-1`. The first element of `gmembers` and `admins` is set to the client ID of the client who sent the command to create the group.

If the number of members in the group is greater than or equal to 5 (including the client who sent the command), the function sends an error message to the client and returns.

If all the client IDs in `gmembers` are valid, the function generates a new group ID (`gid`) and adds the group information (i.e., `gmembers`, `admins`, `isBroadcastOnly`, and `sender`) to the global `gtab` (group information table) array. It sends a success message to the client and returns. If there are any invalid client IDs in `gmembers`, the function sends an error message to the client and returns.

Handle “/makegroupbroadcast”

This function `handle_makegroupbroadcast` is used to modify a group to be "broadcast only", which means that only group members with admin privileges are allowed to send messages to the group.

If the client executing the command is not an admin of the group, an error message is also sent back.

If the group is valid and the client is an admin, the group's `isBroadcastOnly` flag is set to true, which means that only group members with admin privileges can send messages to the group. A success message is then sent to all group members.

Handle “/makegroupreq”

The function starts by initializing two arrays `gmembers` and `admins`, both of size `GROUP_LIMIT`. The `gmembers` array is used to store the client IDs of all the members of the group. The array `gmembers` is initialized with the ID of the client who are requested to create the group and the `admins` array is initialized with the sender ID.

If all the client IDs provided by the client are valid, the function calls the `addPendingGroupInformation` function, which adds the group information to a pgtab. The function returns the ID of the newly created group.

Handle “/joingroup /declinegroup”

The function takes two arguments, the first being the socket file descriptor for the client making the request, and the second being a boolean value indicating whether the request is a join request (true) or a decline request (false).

If the request is of joining the group, the function won't do any change in the member array but if the request is of not joining the group, the function will change the client's id in member array with -1.

After processing every request, the function will decrements the number of pending requests for the group and check if there are no more pending requests. If there are no more pending requests, it adds the group to the group table by calling the `addGroupInformation()` function and frees the memory allocated for the pending group.

Handle "/sendgroup"

The function takes three parameters: the message to send, the ID of the group to send the message to, and the socket file descriptor of the client who sent the message.

It checks if the sender is a member of the group using the `isMemberofGroup` function, which takes the sender ID and the index of the group in `gtab`. If the sender is not a member of the group, it sends an error message back to the client.

If the group is broadcast-only and the sender is not an admin, it sends an error message back to the client.

If the group is not broadcast-only or the sender is an admin, it calls the `sendGroupMessageHelper` function, passing the sender ID, group index, socket file descriptor, group ID, and message as arguments. The helper function will send the message to all group members.

Handle "/makeadmin"

The function checks whether the client who executed this command is an admin of the group. If not, it sends a message to the client indicating that only admins can execute this command.

If the client is an admin, it checks whether the client who is being made an admin is a member of the group. If not, it sends a message to the client indicating that this member is not part of the group.

If the client is a member of the group, and not already an admin, the function adds the client to the admin list of the group, increments the `asize` (admin count) variable, and sends a message to the client indicating that they are now an admin of the group. It also sends a message to all clients in the group indicating that the specified client is now an admin.

Handle "/addtogroup"

This function `handle_addtogroup` is responsible for adding members to a group. It takes the socket file descriptor of the client who sent the command as input and then extracts the group ID and client IDs from the command message.

First, it checks if the group ID is valid and if the client is an admin of that group. If any of these conditions fails, it sends an appropriate error message to the client and returns.

Then, for each client ID in the command message, it checks if it is a valid and active client and if it is already a member of the group. If any of these conditions fails, it sends an appropriate error message to the client and moves on to the next client ID.

If the client ID is valid and not already a member of the group, it adds it to the group by finding the first empty slot in the members array of that group and storing the client ID in that slot. It also sends a message to the added client informing them that they have been added to the group and sends a message to the client who sent the command informing them that the client has been added to the group.

Handle "/removefromgroup"

This function handles the removal of one or more members from a group.

If the sender is not an admin of the group, the function sends an error message to the sender and returns.

For each member to be removed, the function first checks if the member is the sender. If so, the function sends an error message to the sender and returns.

The function then checks if the member is a valid active client. If not, the function sends an error message and continues to the next member.

The function then checks if the member is actually a member of the group. If not, the function sends an error message and continues to the next member.

If the member is an admin of the group, the function removes the admin status before removing the member from the group. For removing the member the function just replaces -1 with the client id in the member array of the group. The function then sends a message to all group members informing them that the member was removed from the group by the sender.

The function sends a message to the removed member informing them that they have been removed from the group by the sender.The function sends a message to the sender informing them that the member was removed from the group.

Handle “/activegroups”

This function handles the "/activegroup" command, which is used to display the groups that the client is a member of.

A loop iterates over all the groups in the gtab array (which stores information about all the groups in the chat system).For each group that the sender is a member of, retrieves the group ID and the list of members in that group.

It then creates a string that contains the group ID followed by the list of members (with optional tags indicating whether each member is an admin or the sender himself). This string is then sent to the sender's sockfd using the sendMsg() function.

Handle “/quit”

This function is called when a client wants to quit the chat room. It first removes the client from all the groups and then removes the message entry and the client record associated with the client.

Interrupt handler at server

This is a signal handler function that is designed to gracefully handle the server shutdown. It first calls the `printRecClient()` function to get a list of all active client IDs, then loops through the list and sends a message to each client indicating that the server is going offline. Finally, it calls the `handle_quit()` function for each client to properly remove them from all groups, delete their message history, and close their socket connection.

The function ends by printing a message to the server console indicating that the server is exiting and then calls the `exit()` function to terminate the program.

Interrupt handler at client

The `sigCHandler` function is used to handle the SIGINT and SIGTSTP signals and send a quit message to the server using the "/quit" command.

If the server sends the "STOP" message, the program sends a SIGQUIT signal to the child process and terminates.