Matthew Stridiron
mas2545

Data Structures on Client and Server Side

Client_table - dictionary that stores all information about the clients. Key name is the clientName. Value is a tuple. The client table is used on the client side to send private messages. It will know whether clients are active without going to the server for that information. Unless the client is unable to private message another user, however, any updated table information should come from the server.
Example: client_table[clientName] = (clientName, clientIP, client_port, "Yes")

clientName_table - set that stores all of the clientNames. This structure exists to make indexing the dictionary become O(1) time.

group_chats - dictionary that stores all information about the clients in group chats. Key name is the groupName. Value is a set of tuples.
Example: group_chats[groupName] = set((client1, clientIP, client1_port), (client2, clientIP, client2_port))

Additional Data Structures on Client Side

Buffered_private_messages - structure that stores all of the private messages sent to a client while it is in group chat mode. As soon as the client leaves the group chat mode, the structure will print out all of the buffered messages and empty out.

userGroup - a string that identifies whether a user is in a group. If it is not in a group, the string remains “”. This string is particularly useful because the user can only execute certain commands while in group/non-group chat modes. Its value serves as a limiter to what commands the user can type within the clientMode function.

usersPrivateMessaged - a dictionary that stores all of the private message sessions that a client has with other users. The key is a clientName, and the value is a true/false statement. You use this structure in the “send” command of clientMode. Initially, the value for a given clientName is set to false, but if it hears back from the receiving end, it is set to true. When the value is set to true, the server won’t remove the receiving client from the session because the connectivity to the client was verified. If the value remained false, no connection would have been made. It would have been removed from the session, and all other client tables among different users would have been updated.

serverResponding - a global variable that confirms whether a server sent an ACK back to a client. Many of the functions send 5 packets to the server for it to parse, in the event that one of them were to be lost in transit, though the functions send fewer packets if the server happens to respond to one of them sooner. Initially, serverResponding's value is set to false. Should the server never send an ACK back to the client, this variable's false value will cause the client to disconnect from the session because the server is down. On the other hand, if the server is up, the client will update this variable to true and remain in session.

FILE STRUCTURE, IN-BUILT FUNCTIONS, AND DESIGN CHOICES

ChatApp.py

All input to the terminal is parsed using argparse.
Expected server input is **python3 ChatApp.py -s <server_port>**. Input is passed into serverMode.
Expected client input is **python3 ChatApp.py -c <client_name> <client_ip> <server_port> <client_port>**. Input is passed into clientMode.

All testing was done with a clientIP = localhost or 127.0.0.1.

UDPClient.py

clientMode() - Everything within the while True statement is for the client to input text within group mode and non-group mode. A socket is dedicated to sending out messages. Here are the commands:

Non-Group Mode:
send <client_name> <message> - The clients should communicate to each other directly and must not use the server to forward chat messages. This command should make the client look up the IP address and port number of the recipient client from its local table and send the message to the appropriate client. The client which sends the message has to wait for an ack and likewise, the client which receives the message has to send an ack once it receives the message. If there is a timeout (500 msecs) for a message sent to another client, it means the client at the receiving end is offline, and so the message has not been delivered. The client should notify the server that the recipient client is offline, and both the server and client should update their tables. The client should also keep track of whether it is in a group chat room. If the client is in a group chat room, it should not print any private messages while it is in the room and instead store the messages in the **buffered_private_messages** structure. Once the client leaves the group chat, those messages should be displayed.

dereg <client_name> - a book-keeping function to keep track of active clients. This functionality involves both client and server parts.

On the server's end:

When the server receives a de-registration request from a client, it has to change the respective client's status to offline in the table. This offline status prevents other users from private messaging the selected client. The server then has to broadcast the updated table to all the active (online) clients, as well as send an ack to the client which requested de-registration.

On the client's end:
You can only deregister yourself, not other users. When a client is about to go offline, it has to send a de-registration request to the server to announce that it is going offline. The client has to wait for an ack from the server within 500 msecs. If it does not receive an ack, the client should retry for 5 times. All the other active clients, when they receive the table from the server, should update their respective local tables.

Once this is completed, the client disconnects from the session entirely.

create_group <group_name> - creates a group name that does not yet exist. If the groupname does exist, then the client will receive a response from the server saying so.

list_groups - a request made by the client to the server requesting a list of all available groups to join. If there are no groups, the server returns an empty string.

join_group <group_name> - puts a user into a group. If the group does not exist, the server will tell the client that it needs to be created. Once a user joins a group, they will not have access to some of these commands unless they leave the group.

Group Mode:

send_group <message> - sends a message to the other users in the group. Once the server receives the message, it broadcasts it to the other clients. As the server also sends a separate ack request to the sending client, it awaits ACKs from the receiving clients. If ACKs are not received by the clients, they are removed from the group.

list_members - returns a list of all of the members in the group

dereg - Similar to dereg in non_group mode, this command removes a client from the group chat, updates the status of the client in the table to offline (a "No" value), and removes it from the session entirely.

leave_group - returns a client to non_group mode.

clientListen() - Everything inside the while true statement involves a listening socket receiving messages coming from the server or other users. Each message that is sent has a "header" value. Depending on what that "header" value is, the client will process the information accordingly.

For example: When the program enters the whileLoop statement for the first time, it immediately requests the client table information from the server using the “REQUEST_TABLE” header. This statement always runs exactly once because it marks the first time that the client session is enabled. If the client table ever needs to be updated, the listen_socket will parse an “UPDATEDTABLE” header and refer to the “updateTable” function elsewhere.

UDPServer.py

serverMode() - given how multiple users are interacting in a session, there is a need for multithreading. Upon reaching the serverMode function, a new thread is created so the incoming requests can be processed instantaneously.

ack_handle() - Each message that the users send to the server has a “header” value. Depending on what that “header” value is, the client will process the information accordingly.

For example, in the “registration” header, it parses out the client_port, client_ip, and clientName so that the user is appended to the server’s client table. A general ACK is sent back to the client, which confirms that it was recognized by the server.