Assignment 3
Multi Client Chat server

In this assignment you need to implement a client-server chat application using TCP sockets. There will be a single server and multiple clients communicating with the server. The server process can handle at most 5 concurrent connections. Each client process will open a new connection with the server and add the client socket id to it's fdset(). Use select to handle multiple client requests.

Part 1 : Simple Chat with Group Functionality.

1. Connection Establishment : The client should send the connection request to server. The server should reply client with appropriate messages if connection can be established or not.

Successful :
If the connection can be established successfully, then generate appropriate identifiers for the client and store them at server. Identifiers includes - Unique Id (5 Digit Random Number). After connection is established, send client the above details with a welcome message.

Unsuccessful :
If the number of clients connected are already 5 then no further client is allowed to connect and server should inform client that "Connection Limit Exceeded !!".

Client details table - Keep the details in memory at server side (Use proper data structure for simplicity)

2. Data Transfer Phase : Client should send a query message to server asking the details of online clients. Server should send the details of all the online clients[**each with unique key**]. After receiving details, client can transfer messages to any other client of choice by using its unique id. (Note that this is a one to one communication).

Message info table - Create another table which stores message details.

NOTE: There can be a situation when a client A gets the list of online clients and before it can send any message to client B, client B goes offline. The sender in this case should be notified that client is now disconnected and that message should be discarded.

3. Connection Termination : In order to disconnect, the client should send an EXIT message to the server. The server should **notify all other clients** with the details of client which is going to disconnect. Then terminate the client process.

4. Broadcast : A client should be able to send a broadcast message by typing "/broadcast ". The message should be delivered to all clients connected to the server.

5. Group Formation : In this part, a client should be able to make a group having unique id with available active members (max 10 in a group). Two basic paradigms that have to be implemented are:

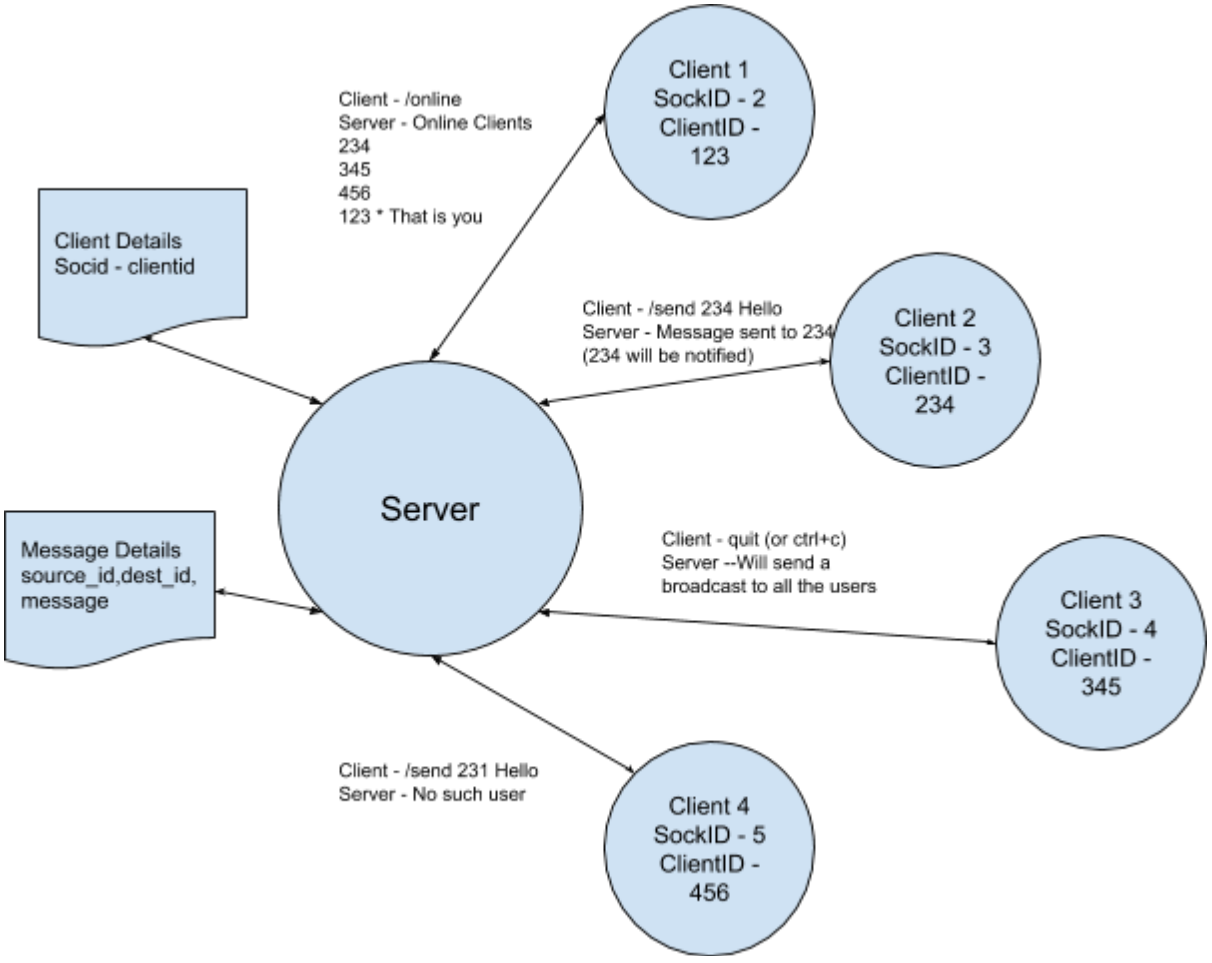
a. **Without permission of other members:** The client should be able to make a simple group by typing "/makegroup" followed by the client ids of all the members that the admin want to include and the members will be automatically joined (making a group in WhatsApp).
b. **With permission of other members:** The client should be able to make a request of forming a simple group by typing "/makegroupreq" followed by the client ids of all the members that the admin want to include. A notification should be sent to that specific clients asking for permission to join the group. If a particular client wants to join, then it may send "/joingroup" followed by group id as the response.
NOTE: Group will be created only after receiving replies from all the clients to whom join request has been sent before by the admin(or creator) of the group.

Functionalities to be included:

1. **/active** : To display all the available active clients that are connected to the server.
2. **/send <dest client id> <Message>** : To send message to the client corresponding to its unique id.
3. **/broadcast <Message>** : Message should be broadcasted to all the active clients.
4. **/makegroup <client id1> <client id2> … <client idn>** : A group with unique id will be made including all the mentioned clients along with the admin client.
5. **/sendgroup <group id> <Message>**: The sender should be in the group to transfer the message to all his peers of that group. The message should be send to all the peers along with group info.
6. **/activegroups** : To display all the groups that are currently active on server and the sender is a part of.
7. **/makegroupreq <client id1> <client id2> … <client idn>** : A group having unique id should be made with currently only the admin client. The request message for joining the group should be notified to all the specified clients. Clients can respond to join that group.
8. **/joingroup <group id>** : If this message is sent by a client having the request for joining the group, then he will be added to group immediately.
9. **/declinegroup <group id>** : If this message is sent by a client having the request for joining the group, then client will not be added to the group.
10. **/quit** : The client will be removed from the server. This client will be removed from all the active groups.

11. **/activeallgroups** : To display all the groups which are active on the server.
12. **/joingroup <group id>:** If this message is sent by a client having the request for joining the group, then he will be added to group immediately. Otherwise a request should be passed to the admin of that group and if admin responds to the request positively then he should be joined to that group.

The 11 & 12 functions are of bonus as discussed in the class.



Simple scenario

Things to remember:

Server Side

1. Use select() for the implementation.
2. Use send() and recv() system call. It will make your life easier.
3. If client B wants to send message to client A, B won't be able to send to A's socket directly. Instead use **message details table** at server side. Client B will send message to server and then server will pass on the message to client A.

4. Server will check in message details table. If it founds an entry of a particular client with dest_id as it's sock_id, then send the message to the client. Remove that entry.
5. Log messages on server (server's terminal).
6. Handle cases when a client is no longer a part of a group, and similar cases.
7. If a client who initiate the group quits from the server then every group which he owns should be automatically deleted.

Client Side

1. Reading from the standard input and writing to the server (send() system call) and reading from the server (recv() system call) will be handled by different processes (**use fork here, we believe you know the reason**).

Things to remember:

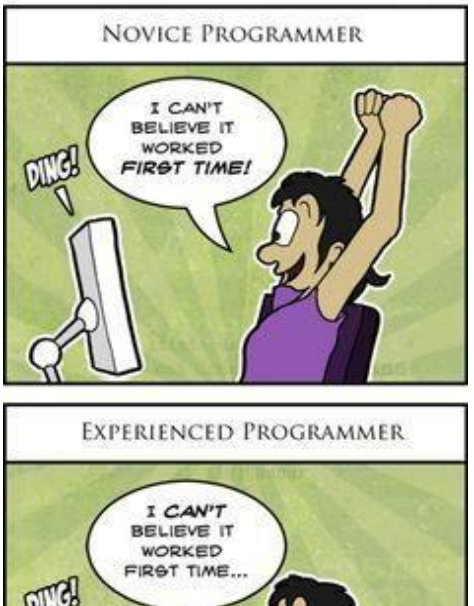
1. Include **Readme.txt** file, write 2-3 lines on what functionalites have been implemented and how to execute your code.
2. Include **makefile.**
3. Follow proper naming conventions.

You can make your own architecture. But try to handle all the corner cases.

<<Please do not copy code from internet or from your classmates. >>

If found copied then straight away zero will be given and can lead to your suspension.

Come up with new ideas and spend time designing and implementing problem.



Act like an experienced programmer and you'll never face any issues in corner cases ;)

Happy Coding !!

Peace ✌