COMP 3825 – Network/Information Assurance

Group Members: Nguyen Anh Khoa Tran (ntran5@memphis.edu)

Wesley K. Marizane (wkmrzane@memphis.edu)

COURSE PROJECT – DESIGN OVERVIEW

I. Introduction:

This project concentrates on developing a Client-Server Reliable Chat Application. A client-server structure is used to allow multiple users to exchange messages over a network. Moreover, the application ensures reliability when the messages are exchanged between client and server by preventing message loss and allowing reliable connection.

The project uses socket programming for network communication and threading to handle multiple users connecting to the server at the same time. Additionally, Python is used for this project.

II. System architecture:

a) Overview of the application:

The architecture of the application follows a client-server model where:

- A server listens for connections and manages communication between multiple clients.
- Each client connects to the server to exchange messages.

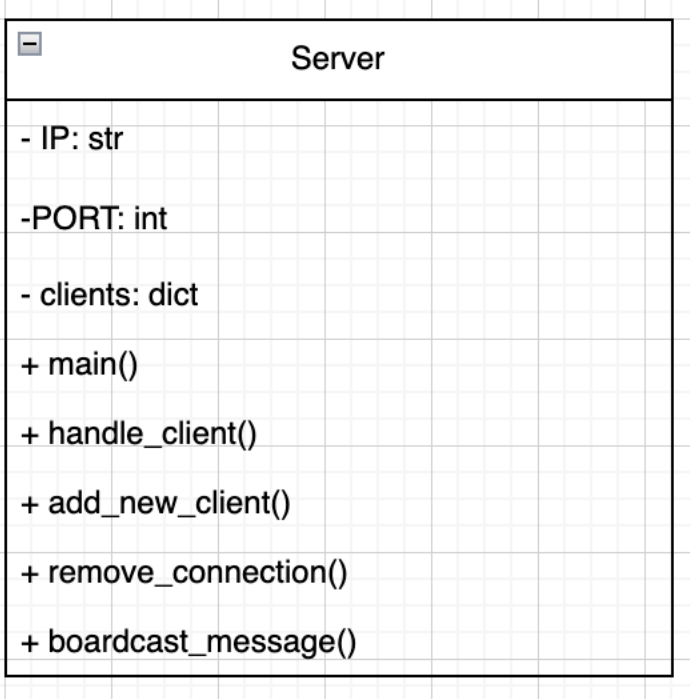
The connections between servers and clients are reliable, ensuring messages are exchanged without loss.

Multithreading is used to handle multiple clients that can exchange messages simultaneously without blocking the connection.

b) Overview of client-server implementation and their UML diagram:

- Server-side:
  1. IP: the IP address of the server is used for listening to the request.
  2. PORT: the port of the server is used for listening to the request.

3. clients: a dictionary is used to store all the available clients' information.
4. handle_client: the server performs some functions of the client such as adding new clients, sending or receiving messages from the clients, etc.
5. add_new_client: get the new client's information and assign a unique ID for them.
6. remove_connection: remove the client from the server's list of clients.
7. broadcast_message: send message from one client to other clients in the chatroom



Server

- IP: str

-PORT: int

- clients: dict

+ main()

+ handle_client()

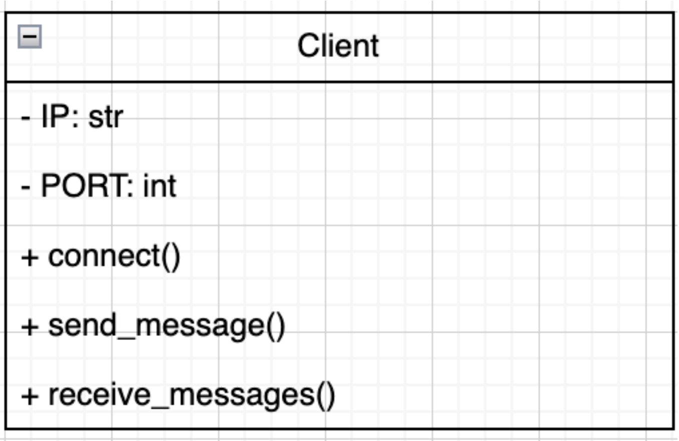
+ add_new_client()

+ remove_connection()

+ boardcast_message()

- Client-side:
  1. IP: the IP address of the server the client wants to connect.
  2. PORT: the PORT of the server the client wants to connect.
  3. connect: the client uses the server's IP and PORT to send a request to the server.
  4. send_message: the client sends messages to the server to broadcast to other clients.

5. receive_message: the client receives messages of other clients from the server.

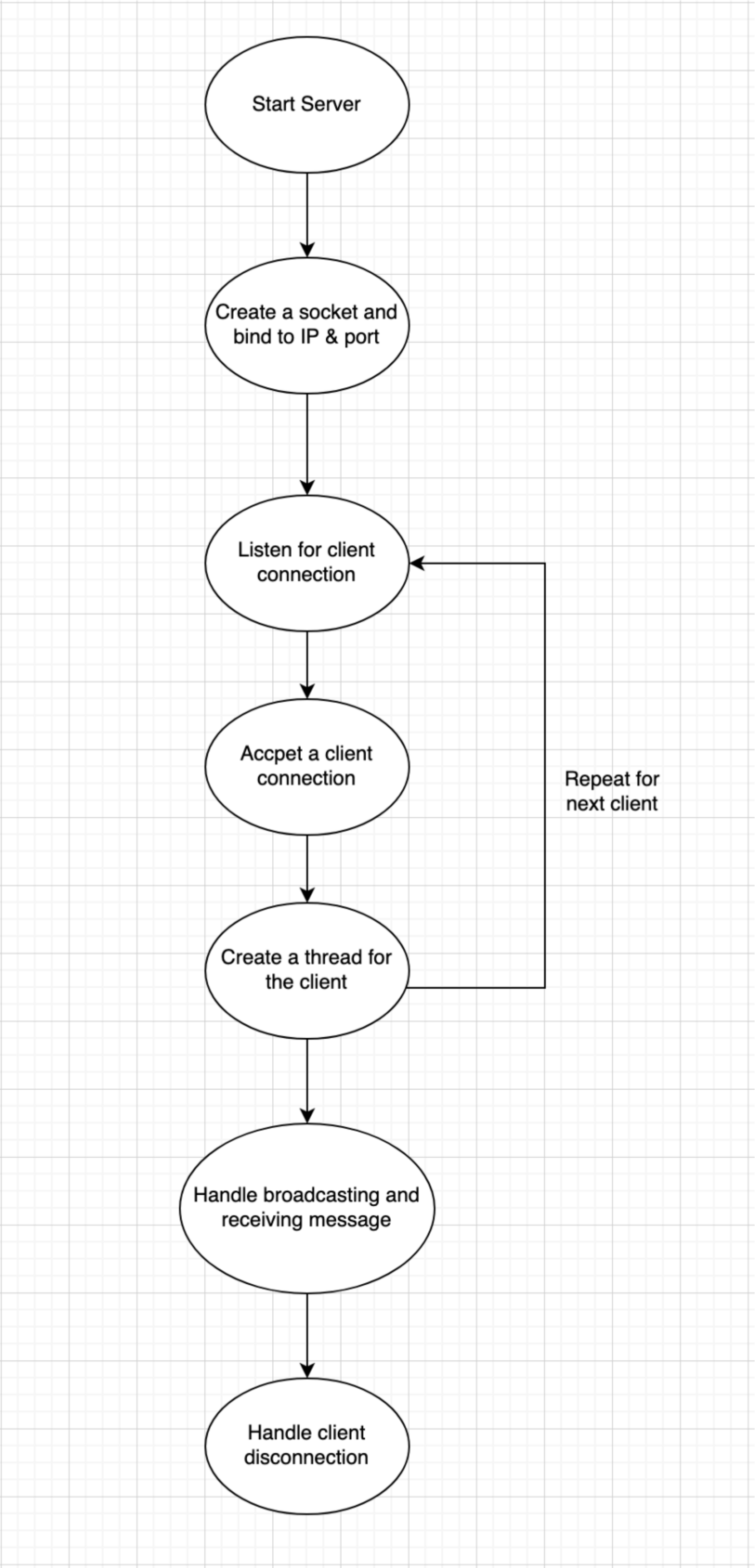


III. Client-server workflow:

a) Server’s workflow:

- Step 1: Run the server program to create a server listening for the client’s request.
- Step 2: Accept any coming client’s request.
- Step 3: Create a new thread for each client.
- Step 4: Handle any communication between the client and the server.
- Step 5: Manage disconnection when the client leaves the chatroom.

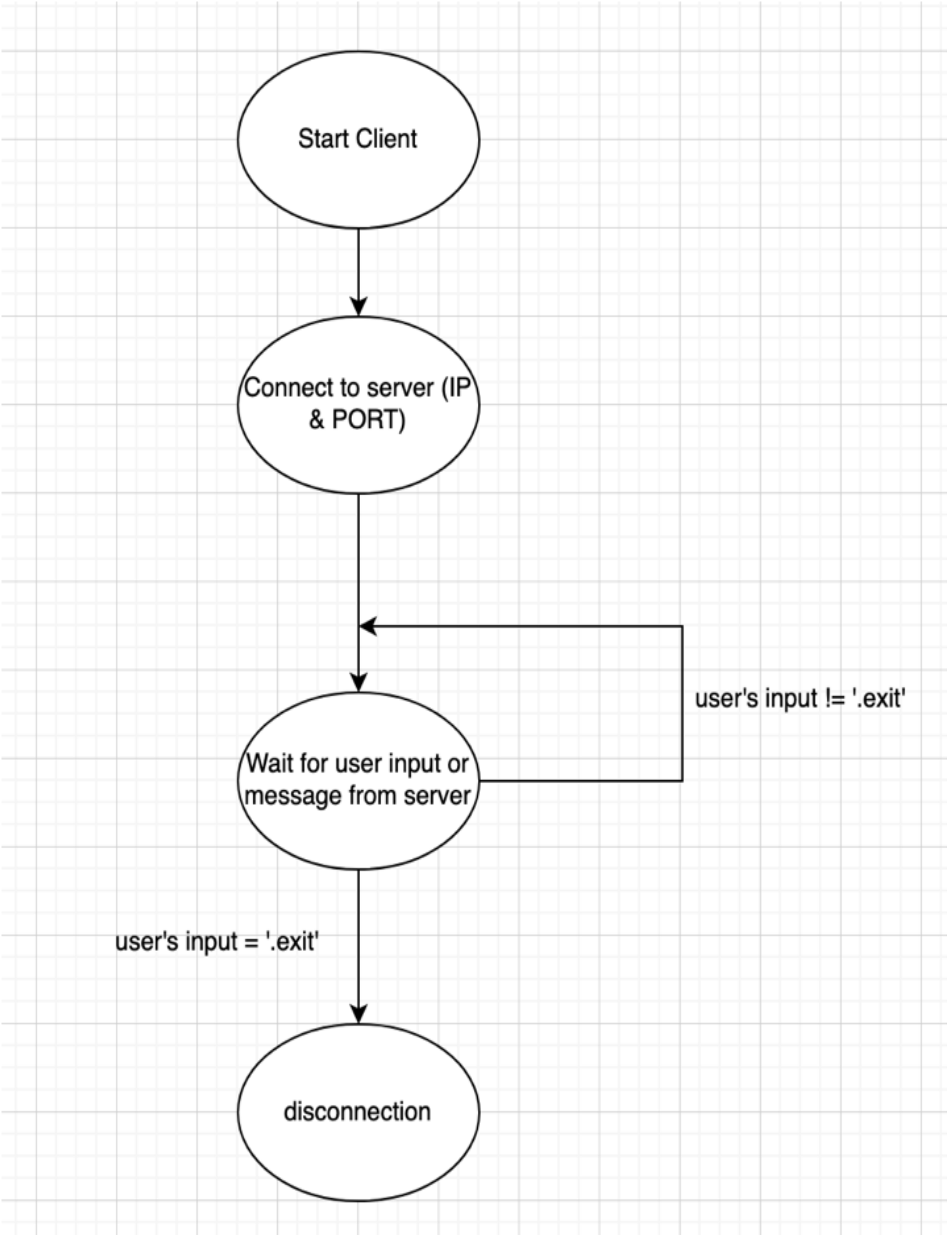
These steps are illustrated in the flowchart below:


Start Server
Create a socket and bind to IP & port
Listen for client connection
Accpet a client connection
Create a thread for the client
Repeat for next client
Handle broadcasting and receiving message
Handle client disconnection

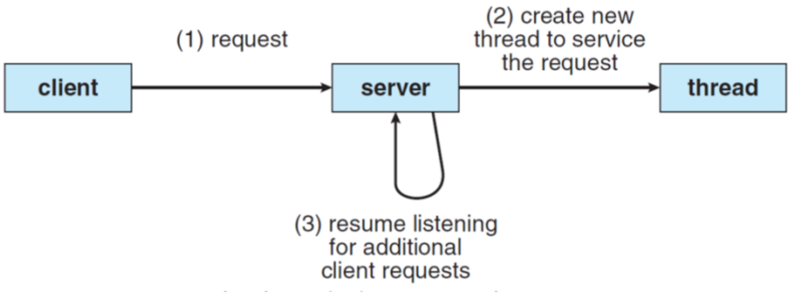
b) Client’s workflow:

- Step 1: Run the client program to initialize a client.
- Step 2: Use the defined IP and PORT to send the connection request to the server.
- Step 3: Handle the communication between the server and client.
- Step 4: Handle disconnection when the user’s input is “.exit”

These steps are illustrated in the flow chart below:


Start Client
Connect to server (IP & PORT)
user's input != '.exit'
Wait for user input or message from server
user's input = '.exit'
disconnection

IV. Client-server interaction flow chart by using multithreading:


(1) request
(2) create new thread to service the request
client
server
thread
(3) resume listening for additional client requests

V. Initially implemented features:

a) Encryption (TLS for message security).

b) Chatting between multiple clients.