Chat Room

Tan Karageldi
Comp 4911 – Networks

This document describes a Chat Room Client & Server as a final project for this course. The project was motivated by my interest on chat rooms, especially growing up playing video games with my friends. Starting with TeamSpeak, Skype, Discord, my interest on advanced chat rooms, and importance of real time communication between people especially playing video games, motivated me to pick this topic for my project.

Introduction

Chat rooms are online systems, that allow multiple users to communicate over a network, with real time communication. Chat rooms are hosted on a server, enabling users, that have the client software, around the world to communicate with each other. This project aims to develop a robust chat room server and client system to facilitate communication. By using client-server architecture, chat room will have a centralized server, which is the host of the program, will provide data for multiple clients that are using the chat room client. Chat rooms can be used for various topics of communication and can be used for different purposes. It's mainly used for sharing text-based information among a group of users simultaneously. Unlike instant messaging tools, which are primarily intended for one-on-one conversations, chat rooms allow users to engage in discussions with multiple participants within the same conversation. There are many advanced chat rooms that are currently used by millions, which has extended usages other than just communication. (Skype, Discord, Microsoft Teams.)

Socket

First, we need to know about Sockets. A network socket is a structure of a computer network, that serves as an interface, for sending and receiving data across the network. Socket, in networking terminologies, serves as a connector, the application layer to the transport layer in the TCP/IP protocol. Sockets are used for establishing communication between a server and client/s. Socket programming involves establishing communication between two nodes on a network. One node act as a listener, listening on a specific port at an IP address, while the other node initiates the connection. The node acting as the listener creates a socket to accept incoming connections, while the other node connects to it.

Python

This project, I will be using python language, for server development, which has a library on socket programming. (References, No 3). Python is a great programming language for this project, because of the clear and easy to understand syntax, which makes the development faster and efficient. It allows the developers to focus more on the logic rather than getting stuck down in the language complexity. Another reason to use python, is it comes with a rich library that includes handling sockets, http requests, and other networking tasks, which is what are we looking for in this project. Also, if we need to use, there are more advanced networking libraries, that makes it easier to develop a server-client model.

Server

Chat room server is the host of our system. The server will run a specific environment, which will stay open and running as long as we want the users to communicate and use the chat room. Server accepts connections from the clients, retrieve messages from the clients and distribute that message to other online clients which are currently in the chat room. The server will establish a socket, which has specified IP address and a port number assigned by the server user. The server should stay open and running to receive connection requests. Server needs to use multi-threading to handle with multiple user connections. Multi-threading, as its defined below, is handling multiple thread executions from simultaneous messages coming from different clients.

Client

The chat room server is practically useless without clients. Client program is like the user interface, which our user is going to be using to establish connection and send messages to other users throughout the server. Chat room client will try to connect to the server socket, which has an assigned IP address and port number client should know. Client program will check if there is any input coming from either sides, client, and server, after connection. And will do respective actions, such as sending the message to the server for server to process and display the message for the other users online or displaying the message from server on client’s terminal.

Multi-Threading

A thread refers to a sequence of code the computer must execute. In computer architecture, multithreading is the ability of a central processing unit (CPU) to provide multiple threads of execution concurrently, supported by the operating system. (Ref. 4, lines 1-4). In this context of our online chat room, we will use multi-threading to handle concurrent connections and communication between clients and the server. By using this approach, we will ensure that server will handle multiple client connections simultaneously without blocking or enabling real-time connection.

Responsibilities

Server:

- Accepting Connections
- Managing Client Connections
- Processing Messages
- Printing Messages
- Keeping the State of the Chat Room

Client:

- Establishing Connection
- Sending Messages
- Receiving Messages
- Displaying Active Users
- Establish Disconnection

Server Script

Starting with server-side script. To start with the python script, we need to import both of the libraries I have mentioned above.

```
import socket
import threading
import sys
```

The Python `socket` module is a library that provides a way for networking using Python. It allows for the creation of socket objects and offers various socket methods to carry out a wide range of tasks such as socket creation, binding to a specific address and port, listening for connections, accepting connections, and sending and receiving data over the sockets. It provides the building blocks for creating TCP/IP or UDP client and server applications.

The Python `threading` module is used for creating and managing threads in Python. Threads are light-weight processes within a process that can run concurrently. They can be used to improve the performance of an application through parallelism, as they allow multiple operations to occur at the same time.

The Python `sys` module is used for providing various functions and variables that are used to manipulate different parts of the Python runtime environment. Which in this case we will be using it to get command line arguments.

In the context of this project, threading is used to handle multiple client connections simultaneously. This is done by creating a new thread for each client connection, ensuring that the server can handle multiple requests at the same time without blocking. This is crucial for maintaining real-time communication between the server and its connected clients.

Now, that we imported and explained our modules that are used in this program, we need to construct the server, using socket. (*AF_INET*) indicates that we are using an internet socket rather than an unix socket. The second parameter stands for the protocol we want to use. *SOCK_STREAM* indicates that we are using TCP and not UDP.

```
server = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
```

After we construct our server with TCP, we need to bind the host, by passing IP address and port number. We get the ip address and the port address from command line arguments, and bind it to the server.

```
ip = str(sys.argv[1])
port = int(sys.argv[2])
```

```
server.bind((ip,port))
```

This three lines of code bind the server to a specific IP address and port number. The `sys.argv[1]` and `sys.argv[2]` are command line arguments. The `sys.argv[1]` is used as the IP address and `sys.argv[2]` is used as the port number. The `server.bind((ip,port))` function binds the server with this IP address and port number. Clients can then use this IP address and port number to connect to the server.

Before assigning the ip and port values from command line arguments, the server goes through an conditional statement to check if the correct arguments are passed. Program prints out a error message if the expected usage is not satisfied.

Now we call the `server.listen()` method for setting up the server to listen to accept incoming client connections.

```
server.listen()
```

Since there is going to be multiple clients using the server, the server needs a way to store all the clients, we achieve this by creating an array, for clients. Also, for each client connecting to the server, I added another array, which will hold the names of the clients.

```
clients=[]
client_names=[]
```

Now we're done with constructing the server, lets start with the functions the server is going to use. First off, we will create a broadcast function. This function is going to iterate through the clients list, to send the message to all the clients currently in the server.

```
def broadcast(message):
        for client in clients:
                client.send(message)
```

broadcast method is going to be used in the next method, to send messages through all clients.

This next part , is the most important aspect of this project. `thread()` method.

```
def thread(connection,address): # Function to handle the connection with the clien
    while True: # Infinite loop to keep the connection alive
            chat = connection.recv(2048) # Receive a message from the client
            if chat:
                print("< "+ address[0] +" > " + chat.decode()) # Print the message
                broadcast(chat) # Broadcast the message to all clients
            else:
                i = clients.index(connection) # Get the index of the client
                clients.remove(connection) # Remove the client from the list of cl
                connection.close() # Close the connection with the client
                name = client_names[i] # Get the name of the client
                client_names.remove(name) # Remove the name of the client from the
```

This `thread(connection, address)` function is used to manage each client that connects to the server.

It runs in an infinite loop, continuously receiving messages from the client (`chat = client.recv(1024)`).

If a message is received (`if chat:`), it prints the message to the server console, attributing it to the client that sent it. The message is then sent to all other connected clients using the previously defined `broadcast()` function.

If the client sends no message (`else:`), it means the client has disconnected. The server then removes this client from the `clients` list and also removes the client's name from the `client_names` list.

In essence, this function handles receiving messages from clients, sending received messages to all clients, and managing client disconnections.

Now that we handled broadcasting messages through the clients, and constructed thread function for receiving and sending messages throughout clients, our server is ready to run.

To finish up with the server-side script, we need the main loop for the server to receive messages and call functions that was described above.

```
def start():
    print("Server is running on IP: ", ip, " and port: ", port) # Print the IP add
    print("Waiting for connections...") # Print that the server is waiting for con
    while True:
        client,address = server.accept() # Accept a connection from a client
        print("Connection established with: ", address) # Print the address of the

        client.send('Please enter your name:'.encode()) # Send a message to the cl
        name = client.recv(2048).decode() # Receive the name of the client

        client_names.append(name) # Append the name of the client to the list of n
        clients.append(client) # Append the client to the list of clients

        print("Name of the client is: ", name) # Print the name of the client
        broadcast("{} joined the chat room!".format(name).encode()) # Broadcast th

        # Start Handling Thread For Client
        th = threading.Thread(target=thread, args=(client,address))
        th.start()
```

When our server script is run, by using `python server.py <ip> <port>` command through the terminal, this part of the program will run.

The `start()` function in the server-side script initiates the server and prepares it for accepting connections from clients. What function does:

1. `print("Server is running on IP: ", ip, " and port: ", port)`: This line prints the IP address and port number of the server.
2. `print("Waiting for connections...")`: This line indicates that the server is waiting for clients to connect.
3. `while True:`: This infinite loop keeps the server running and ready to accept connections at any time.
4. `client, address = server.accept()`: This line accepts a connection from a client. The function `server.accept()` returns a socket object representing the client and the address of the client. These are stored in `client` and `address` respectively.
5. `print("Connection established with: ", address)`: This line prints the address of the client that has just connected.
6. `client.send('Please enter your name:'.encode())` and `name = client.recv(2048).decode()`: The server sends a message to the client asking for their name and then receives the client's response.
7. `client_names.append(name)` and `clients.append(client)`: The server then adds the client's name to the `client_names` list and the client object to the `clients` list.
8. `print("Name of the client is: ", name)`: This line prints the name of the client.
9. `broadcast("{} joined the chat room!".format(name).encode())`: The server announces to all connected clients that a new client has joined the chat room.
10. `th = threading.Thread(target=thread, args=(client,address))` and `th.start()`: This creates and starts a new thread for each client that connects. The `thread` function is called for each client to handle their messages and keep communication with them running concurrently with other clients.

```
start() # start the server.
```

And with that loop, our server-side script is done. Here is the output we get when we run our server:

```
tankarageldi$ python server.py 127.0.01 3131
Server is running...
Waiting for connection...
```

The server here is in a infinite loop, waiting for a client.

Client Script

The client script will be responsible of :

- Establishing a Connection: The client script initiates a connection with the chat room server using its assigned IP address and port number.
- Sending Messages: sending messages from the user to the server. This is done through user input on the client-side interface.
- Receiving Messages: receives messages from the server and displays them to the user. These messages could either be from other users in the chat room or administrative messages from the server itself (e.g., notifications of users joining or leaving the chat room).

As we did with the server script, lets explain every line neatly. First off, we start by importing the same modules.

```
import socket
import select
import sys
```

To start up with the client, we will ask for the user's name.

```
name = input("Enter your name: ")
```

Now we will initialize the sockets, the same way it was done in the server script. Im not going to give detailed explanation of duplicate code, that we used in server-side script as well.

```
client = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
```

Now that the program initializes the client's socket. Next step is checking if the command line arguments are in correct format. For this, client script uses the same conditional statement to check and print an error message.

```
ip = str(sys.argv[1]) # Get the IP address from the command line
port = int(sys.argv[2]) # Get the port number from the command line
client.connect((ip, port)) # Connect to the server
```

are used to establish a connection to the server. Here's a detailed explanation:

1. `ip = str(sys.argv[1])`: This line retrieves the IP address
2. `port = int(sys.argv[2])`: This line retrieves the port number
3. `client.connect((ip, port))`: This line attempts to establish a connection to the server using the retrieved IP address and port number.

Now that we looked through the parts that establishes connection with the server, lets look at the functions that sends and gets messages to/from server.

```
# Function to send messages to the server
def send():
    while True:
        message = '{} : {}'.format(name, input(''))
        client.send(message.encode()) # Send the message to the server
```

This `send()` function is used to send messages from the client to the server. Here's a breakdown:

1. `message = '{} : {}'.format(name, input(''))`: This formats the message we want to send to the server.
2. `client.send(message.encode())`: This line sends the formatted message to the server.

```
# Function to receive messages from the server
def get():
    while True:
        try: # Try to receive messages from the server
            message = client.recv(1024).decode()
            if message == 'Please enter your name:': # If the server asks for the
                client.send(name.encode())
            else:
                print(message)
        except: # If an error occurs, close the connection with the server
            client.close()
            break
```

The `get()` function is used by the client to receive messages from the server. Here's what each line does in this function :

1. `message = client.recv(2048).decode()`: This line receives a message from the server.
2. `if message == 'Please enter your name:':`: This line checks if the received message is a prompt from the server asking for the client's name.
3. `client.send(name.encode())`: If the received message is indeed a prompt for the client's name, this line sends the client's name to the server.
4. `else: print(message)`: If the received message is not a prompt for the client's name, it is printed out to the client's console.
5. `except:`: If any error (an exception) occurs during the execution of the `try:` block, the commands in this block are executed.
6. `client.close()`: This line closes the connection with the server.
7. `break`: This line breaks the infinite loop, ending the `get()` function.

And lastly, now that our client script is ready to connect to a server, we need to handle threading on clients as well. These following lines, are using the threading module, to handle concurrent connections between clients and the server.

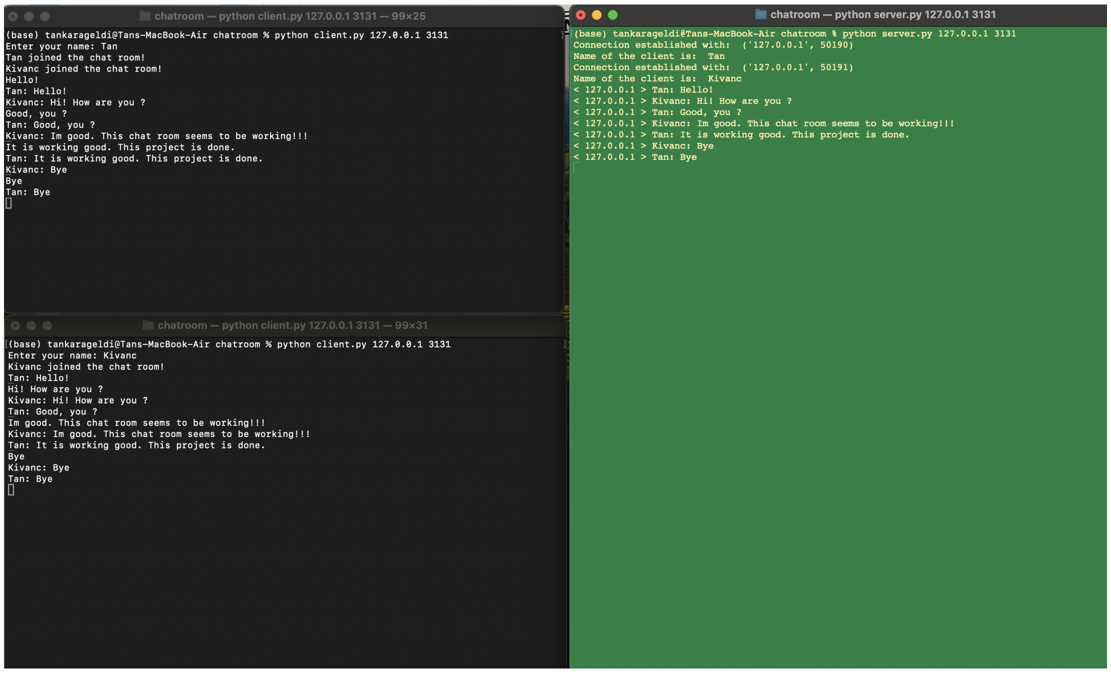
```
thread_get = threading.Thread(target=get) # Create a thread to receive messages fr
thread_get.start() # Start the thread

thread_send = threading.Thread(target=send) # Create a thread to send messages to
thread_send.start() # Start the thread
```

With this design, more than one clients can connect to the same server, to share information.

Below, I have included an example usage of the server and client scripts.

Example Usage


chatroom — python client.py 127.0.0.1 3131 — 99×25

```
(base) tankarageldi@Tans-MacBook-Air chatroom % python client.py 127.0.0.1 3131
Enter your name: Tan
Tan joined the chat room!
Kivanc joined the chat room!
Hello!
Tan: Hello!
Kivanc: Hi! How are you ?
Good, you ?
Tan: Good, you ?
Kivanc: Im good. This chat room seems to be working!!!
It is working good. This project is done.
Tan: It is working good. This project is done.
Kivanc: Bye
Bye
Tan: Bye
```

chatroom — python client.py 127.0.0.1 3131 — 99×31

```
(base) tankarageldi@Tans-MacBook-Air chatroom % python client.py 127.0.0.1 3131
Enter your name: Kivanc
Kivanc joined the chat room!
Tan: Hello!
Hi! How are you ?
Kivanc: Hi! How are you ?
Tan: Good, you ?
Im good. This chat room seems to be working!!!
Kivanc: Im good. This chat room seems to be working!!!
Tan: It is working good. This project is done.
Bye
Kivanc: Bye
Tan: Bye
```

```
(base) tankarageldi@Tans-MacBook-Air chatroom % python server.py 127.0.0.1 3131
Connection established with:  ('127.0.0.1', 50190)
Name of the client is:  Tan
Connection established with:  ('127.0.0.1', 50191)
Name of the client is:  Kivanc
< 127.0.0.1 > Tan: Hello!
< 127.0.0.1 > Kivanc: Hi! How are you ?
< 127.0.0.1 > Tan: Good, you ?
< 127.0.0.1 > Kivanc: Im good. This chat room seems to be working!!!
< 127.0.0.1 > Tan: It is working good. This project is done.
< 127.0.0.1 > Kivanc: Bye
< 127.0.0.1 > Tan: Bye
```

Conclusion

Simple chat room project has been completed. In this paper, i've went through every parts of the server and client scripts. More information about threading, sockets, python and chat rooms in general can be found above, in respective sections.

Included with the final submission of this project, there will be a server-side script, which will implement the ideas I have talked about above, to offer a real time connection with client users. Also, the submission will include client-side script, which will also implement the ideas above, to connect to the respective server using the IP address client is connecting to, send and receive messages, and print the messages on command line.

References:

- *What is a chat room?*

https://www.techtarget.com/whatis/definition/chat-room#:~:text=A%20chat%20room%20is%20an,hold%20conversations%20about%20various%20topics.

https://en.wikipedia.org/wiki/Chat_room

- *Python socket library.*

https://docs.python.org/3/library/socket.html

https://docs.python.org/3/library/socketserver.html

- *Multi-Threading*

https://en.wikipedia.org/wiki/Multithreading_(computer_architecture)