Java Based TCP Chat Room

Special Assignment Report

Submitted in Partial Fulfillment of the

Requirements for completion of

Course on

2EC702 Computer Networks

By

Soumik Mohapatra (20BEC123)

Aum Dhabalia (21BEC027)

Submitted to

Dr. Sachin Gajjar

Department of Electronics and Communication Engineering,

Institute of Technology,

Nirma University

October 2024

Table of Contents

INDEX

Chapter	Title	Page
1	Introduction	1
	1.1 Introduction to chat applications	1
	1.2 Introduction to TCP	1
	1.3 Introduction to Socket	2
2	Approach	2
	2.1 Methodology	2
	2.2 Tools & Technology used	3
3	Results	3
4	Conclusion	4
5	References	5
6	Appendix	6

List of Figures

Sr. No.	Figure	Page
1	Various Chat Applications	1
2	Working of TCP Protocol	1
3	Socket Programming Flowchart	2
4	Flowchart of Chat-Room	3
5	a Server running at port 4004 and IP 192.168.56.1	3
	b Client running at port 4004 connected to server 192.168.56.1	4
	c Output of chat-room on terminal	4

1. Introduction

1.1. Introduction to chat applications

Chat applications are software programs that provide platform for communication between clients. Chat applications use internet for establishing and managing communication between clients. In chat application, there is a server that listens for clients for connection and manages links of connected clients.

Chat applications either use tcp (*transmission control protocol*) or udp (*user datagram protocol*) based on the requirements. There are several types of chatting applications based on requirements such as instant messaging, group messaging, video conferencing (as of video-chatting), etc.

Key features of any chat application are real-time messaging, multimedia sharing (sharing contents like files, audios, videos, etc.), end-end encryption (encrypting messages for providing security and privacy of users), cross-platform support (message from one application can be sent to another application and vice-versa), special characters usage (allows users to create and send emojis, stickers, etc.).

Fig. 1 Various chat applications

1.2. Introduction to TCP

TCP stands for transmission control protocol, is a connection-oriented protocol, lies between Application layer and Network layer in TCP/IP model suite or in OSI model, a transport layer protocol. TCP is connection-oriented which means, firstly, a secure connection is established and then the packets are exchanged with acknowledgement and after exchange is complete, connection is shutdown.

Key features of tcp are packet transfer in order (packets are numbered and transferred in order), flow control (data transmission rate is maintained for reliable delivery), error control, full-duplex mode (data can be transmitted and received at same time) and congestion control.

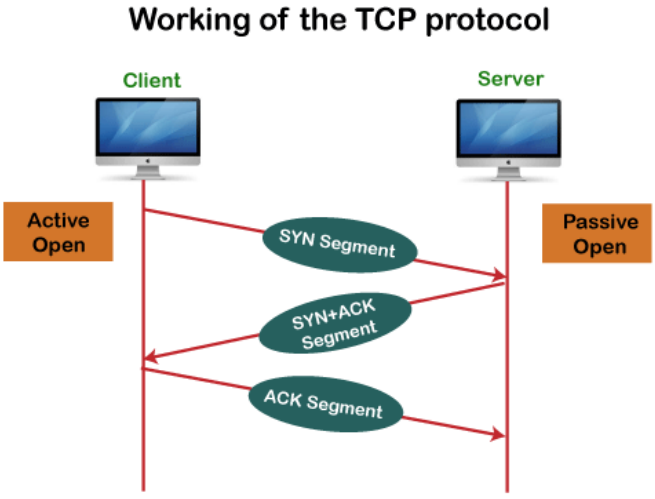

Fig. 2 Working of TCP protocol

1.3. Introduction to Socket programming

A socket is a connection endpoint or a program that connects a device with the network enabling two-way communication between a server and one or more clients. There are two types of sockets namely, stream socket (connection-oriented socket) and datagram socket (connectionless socket).

Stream sockets are connection-oriented which means a secure connection is established and then data stream is sent in sequence and similarly received in sequence with acknowledgements. Whereas datagram sockets are connection-less meaning simply data is sent without establishing connection or any acknowledgements. Data packets are sent out of order but when received they are error free and if packets are lost in between then they are lost forever.

Socket programming is a method for establishing and managing links between two or nodes with each other. A socket is created by providing an IP address (Internet Protocol address) and port and then setting type of socket i.e. stream socket or datagram socket. After this, connections are listened and accepted and communication is established.

In this project, a Java based simple chat-room is made where messages are broadcasted to all clients that are connected to the chat-room.

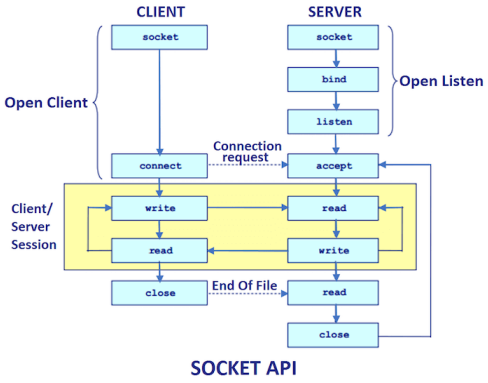

Fig. 3 Socket Programming Flowchart

2. Approach

2.1. Methodology & Flowchart

The chat-room follows client-server model of tcp protocol. The server is first initialized and set to listen for incoming connections. Once server is set, the client is started and it requests the server for connection. The server listens and accepts the connection. Once the connection is established, each message that the client(s) send is broadcasted to rest of the connected clients in the group by the server.

Firstly, the server is set up. A server socket is created by binding IP address with port and providing backlogs (number of connections accepted at a time). Once, server socket is set, multi-threading services are enabled so that multiple client requests can handled simultaneously. The multi-threading service will make thread for each client connected to the server chat-room.

After multi-threading is set, server starts to listen to incoming connections by using *listen()* function. If a client request for connection is listened, then server accepts the request and forms a link. This link is formed as a thread.

From client side, client socket is initialized by providing server IP and port. Once initialized, a connection request is sent to server for establishing a link. Once connection is established, client is

joined inside the chatroom created by the server and any message sent server the client is broadcasted to all other clients present in the server.

From server side, server will always listen to incoming client connection requests till the backlog is full which maximum number of clients connected at once. If backlog is achieved, then new incoming requests will be sent to queue where it will wait for its turn for sending and receiving messages. *Fig. 4* shows the client server model of the chatroom.

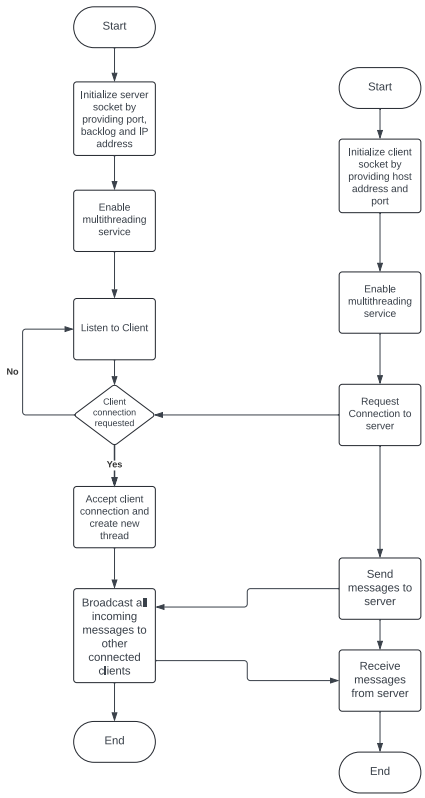

Fig. 4 Flowchart of Chat-Room

2.2. Tools & Technology used

The following chatting application was developed in Java language on Windows operating system. Libraries used for socket programming are *java.net, java.io, java.util* and *java.util.concurrent* on notepad++ editor. The program was executed on Windows terminal using commands *javac program_name.java* and *java program_name.*

3. Results

The given below figures display output of the program. Fig. 5a shows server running, 5b shows client running and 5c shows chatroom where three clients are connected to the server.

```
PS D:\Nirma Files\Computer Networks\Project> javac Server.java
PS D:\Nirma Files\Computer Networks\Project> java Server
Server initialised...
Port : 4004  IP : 192.168.56.1
Maximum no. of connections : 3
```

Fig. 5a Server running at port 4004 and IP 192.168.56.1

```
PS D:\Nirma Files\Computer Networks\Project> java Client
Connecting to host 192.168.56.1
Connection successfull...!!
Please enter your name:
Client 1
Client 1 joined the chat!
testing 123...Hello
Client 1: testing 123...Hello
```

Fig. 5b Client running at port 4004 connected to server 192.168.56.1

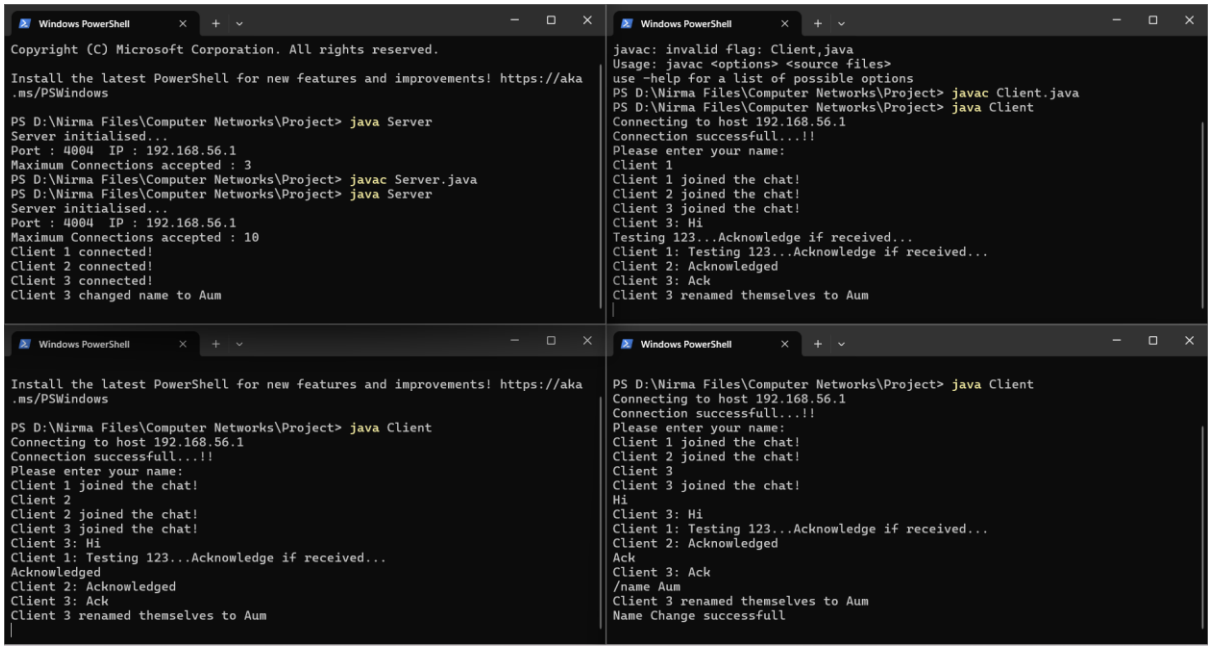

Fig. 5c Output of chatroom on terminal

4. Conclusion

As performing the project, several learning outcomes were achieved like concepts of object-oriented programming, multi-threading, socket programming, etc. Concepts and features of transmission control protocol (TCP) were familiarized. Several features like sequenced flow of packets, error control, flow control, packet numbering, connections management, graceful shutdown, etc. were familiarized.

The concepts of multi-threading were applied for managing multiple clients that were connected to the server. The server created a thread for each of the client that connected to the chatroom. It was observed that new connections were accepted and were held at standby queue so that when some clients are idle, the clients from standby queue are listened and their requests are fulfilled.

5. References

[1] H. Schildt, Java - The Complete Reference Ninth Reference, 2014.

[2] R. M. Reese, Learning Network Programming with Java, 2015.

[3] M. A. Ahmed, “Design and Implement Chat Program Using TCP/IP,” 2019.

[4] P. Balbudhe, T. Buradkar, S. Jha, A. Chaudhary and P. Bajanghate, “Java Chat Application using Jsocket,” 2023.

[5] T. Mattsson, “A Peer-to-Peer based chat system,” 2012.

[6] T. Muldner, “Analysis of Java Client / Server and Web Programming Tools for Development of Educational Systems,” 1998.

6. Appendix

Code of the project

Server-side…

```
import java.io.*;
import java.util.*;
import java.util.concurrent.*;
import java.net.*;
public class Server implements Runnable
{
	private final ArrayList<ConnectionHandler> connections;
	private ServerSocket server;
	private boolean done;
	private ExecutorService pool;

	public Server()
	{
		done = false;
		connections = new ArrayList<>();
	}

	@Override
	public void run()
	{
		try
		{
			int port = 4004;
			int backlogs = 3;
			String IP = "192.168.56.1";
			server = new ServerSocket(port,backlogs,InetAddress.getByName(IP));
			System.out.println("Server initialised...\nPort : "+port+"  IP : "+IP);
			System.out.println("Maximum no. of connections : "+backlogs);
			pool = Executors.newCachedThreadPool();
			while(!done)
			{
				Socket client = server.accept();
				ConnectionHandler handler = new ConnectionHandler(client);
```

```
                                        connections.add(handler);
                                        pool.execute(handler);
                                }
        }
                catch(Exception e){shutdown();}
        }

        public void broadcast(String message)
        {
                for(ConnectionHandler ch : connections)
    if (ch != null)
                                        ch.sendMessage(message);
}

        public void shutdown()
        {
                try
                {
                        done = true;
                        pool.shutdown();
                        if(!server.isClosed())
                                server.close();
    for(ConnectionHandler ch : connections)
                                        ch.shutdown();
        }
                catch(IOException e){e.printStackTrace();}
        }

        class ConnectionHandler implements Runnable
        {
                private final Socket client;
                private BufferedReader in;
                private PrintWriter out;

                public ConnectionHandler(Socket client)
                {
```

```
                this.client = client;
        }

        @Override
        public void run()
        {
                try
                {
                        out = new PrintWriter(client.getOutputStream(), true);
                        in = new BufferedReader(new InputStreamReader(client.getInputStream()));
                        out.println("Please enter your name: ");
                        String name = in.readLine();
                        System.out.println(name + " connected!");
                        broadcast(name + " joined the chat!");
                        String message;
                        while((message = in.readLine()) != null)
                        {
                                if (message.startsWith("/name"))
                                {
                                        String[] messageSplit = message.split(" ",2);
                                        if(messageSplit.length == 2)
                                        {
                                                broadcast(name+" changed name to " +
messageSplit[1]);
                                                System.out.println(name + " changed name to " +
messageSplit[1]);
                                                name = messageSplit[1];
                                                out.println("Name Change successfull");
                        }
                                        else
                                                out.println("No name provided");
                }
                                else if(message.startsWith("/quit"))
                                {
                                        broadcast(name + " left the chat!");
                                        shutdown();
                                }
```

```
                                                    else
                                                                broadcast(name + ": " + message);
                }
            }
                                catch(IOException e) {shutdown();}
        }

// send message method...
        public void sendMessage(String message)
                        {
            out.println(message);
        }
//connection shutdown method...
        public void shutdown()
                        {
                                try
                                {
                                        in.close();
                                        out.close();
                                        if(!client.isClosed())
                                                client.close();
                                }
                                catch(IOException e){e.printStackTrace();}
                        }
                }

//main method...
                public static void main(String[] args)
                {
        Server server = new Server();
        server.run();
    }
}
```

Client-side

```
import java.io.*;
import java.net.*;
public class Client implements Runnable
{
	private Socket client;
	private BufferedReader in;
	private PrintWriter out;
	private boolean done;

	@Override
   public void run()
	{
		try
		{
			client = new Socket("192.168.56.1",4004);
			System.out.println("Connecting to host 192.168.56.1");
			out = new PrintWriter(client.getOutputStream(),true);
			in = new BufferedReader(new InputStreamReader(client.getInputStream()));

			InputHandler inHandler = new InputHandler();
			Thread t = new Thread(inHandler);

			t.start();//Start new thread named inHandler...
			System.out.println("Connection successfull...!!");
			String inMessage;
			while((inMessage = in.readLine()) != null)
				System.out.println(inMessage);
		}
		catch(IOException e){shutdown();}
	}

	public void shutdown()
	{
		done = true;
		try
```

```
                {
                        in.close();
                        out.close();
                        if(!client.isClosed())
                                client.close();
                }
                catch(IOException e){e.printStackTrace();}
    }
        class InputHandler implements Runnable
        {
                @Override
                public void run()
                {
                        try
                        {
                        BufferedReader inReader = new BufferedReader(new InputStreamReader(System.in));
                        while(!done){
                                        String message = inReader.readLine();
                                        if(message.equals("/quit"))
                                        {
                                                out.println(message);
                                                inReader.close();
                                                shutdown();
                                        }
                                        else
                                                out.println(message);
                                }
                        }catch(IOException e){shutdown();}
                }
        }
//main method...
        public static void main(String[] args){
                Client client = new Client();
                client.run();
        }
}
```