|  |  |
| --- | --- |
| Semester | T.E. Semester V – Computer Engineering |
| Subject | Computer Network |
| Subject Professor In-charge | Prof. Amit K. Nerurkar |
| Assisting Teachers | Prof. Amit K. Nerurkar |
| Laboratory | Lab number |

|  |  |
| --- | --- |
| Student Name | Deep Salunkhe |
| Roll Number | 21102A00014 |
| TE Division | A |

**Title: Two-Way Chat Application with TCP and UDP**

**Explanation:**

**Server-side:**

* The server offers the user a choice between TCP and UDP.
* For TCP, it establishes a server socket, accepts client connections, and handles two-way communication.
* For UDP, it creates a datagram socket, receives messages from clients, and sends responses back.

**Client-side:**

* The client prompts the user to choose between TCP and UDP.
* For TCP, it connects to the server, spawns a thread to continuously receive messages, and allows the user to send messages.
* For UDP, it creates a datagram socket, spawns a thread to continuously receive messages, and allows the user to send messages.

**The main differences between TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) chats in the provided code lie in the characteristics of these protocols:**

* **Connection-oriented vs. Connectionless:**
* **TCP:** It is a connection-oriented protocol. The server and client establish a connection before exchanging data. This ensures reliable, ordered, and error-checked delivery of information. The **ServerSocket** and **Socket** classes are used in Java for TCP communication.
* **UDP:** It is a connectionless protocol. Communication is achieved by sending independent packets, known as datagrams, to each other. UDP is faster but doesn't guarantee delivery, order, or error checking. The **DatagramSocket** and **DatagramPacket** classes are used in Java for UDP communication.
* **Reliability:**
* **TCP:** Reliable and ensures that data is received in the order it was sent. It also handles retransmission of lost packets and error detection.
* **UDP:** Unreliable, as it doesn't guarantee delivery, order, or error checking. It's often used in scenarios where a small amount of data loss is acceptable, such as real-time applications.
* **Overhead:**
* **TCP:** Higher overhead due to its reliability features and the need to establish and maintain a connection.
* **UDP:** Lower overhead since it's connectionless and doesn't include mechanisms for reliability.
* **Usage:**
* **TCP:** Suitable for applications where accurate and ordered delivery of data is crucial, such as file transfers, email, and web browsing.
* **UDP:** Used in scenarios where low latency and high-speed data transmission are more critical, such as video streaming, online gaming, and real-time communication.

**Implementation:**

**Server-side:-**

import java.io.\*;

import java.net.\*;

public class ChatServer {

private static final int TCP\_PORT = 12345;

private static final int UDP\_PORT = 12346;

public static void main(String[] args) {

System.out.println("Chat Server");

try {

BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

System.out.println("Choose the server type:");

System.out.println("1. TCP Server");

System.out.println("2. UDP Server");

System.out.print("Enter your choice: ");

int choice = Integer.parseInt(reader.readLine());

switch (choice) {

case 1:

startTCPServer();

break;

case 2:

startUDPServer();

break;

default:

System.out.println("Invalid choice. Please enter 1 or 2.");

}

} catch (IOException e) {

e.printStackTrace();

}

}

private static void startTCPServer() {

try {

ServerSocket serverSocket = new ServerSocket(TCP\_PORT);

System.out.println("TCP Server listening on port " + TCP\_PORT);

while (true) {

Socket clientSocket = serverSocket.accept();

System.out.println("TCP Client connected: " + clientSocket.getInetAddress());

Thread clientThread = new Thread(() -> handleTCPClient(clientSocket));

clientThread.start();

}

} catch (IOException e) {

e.printStackTrace();

}

}

private static void handleTCPClient(Socket clientSocket) {

try {

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

PrintWriter writer = new PrintWriter(clientSocket.getOutputStream(), true);

BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in));

while (true) {

String message = reader.readLine();

if (message == null || message.equals("exit")) {

System.out.println("TCP Client disconnected: " + clientSocket.getInetAddress());

break;

}

System.out.println("TCP Received from " + clientSocket.getInetAddress() + ": " + message);

System.out.print("Enter your response: ");

String response = consoleReader.readLine();

writer.println("Server: " + response);

}

clientSocket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

private static void startUDPServer() {

try {

DatagramSocket serverSocket = new DatagramSocket(UDP\_PORT);

System.out.println("UDP Server listening on port " + UDP\_PORT);

while (true) {

byte[] receiveData = new byte[1024];

DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

serverSocket.receive(receivePacket);

InetAddress clientAddress = receivePacket.getAddress();

int clientPort = receivePacket.getPort();

String message = new String(receivePacket.getData(), 0, receivePacket.getLength());

System.out.println("UDP Received from " + clientAddress + ":" + clientPort + ": " + message);

if (message.equals("exit")) {

System.out.println("UDP Client disconnected: " + clientAddress + ":" + clientPort);

continue;

}

String replyMessage = "Server: " + message;

byte[] sendData = replyMessage.getBytes();

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, clientAddress, clientPort);

serverSocket.send(sendPacket);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Client-side:-**

import java.io.\*;

import java.net.\*;

public class ChatClient {

private static final int TCP\_PORT = 12345;

private static final int UDP\_PORT = 12346;

public static void main(String[] args) {

System.out.println("Chat Client");

try {

BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

System.out.println("Choose the client type:");

System.out.println("1. TCP Client");

System.out.println("2. UDP Client");

System.out.print("Enter your choice: ");

int choice = Integer.parseInt(reader.readLine());

switch (choice) {

case 1:

startTCPClient();

break;

case 2:

startUDPClient();

break;

default:

System.out.println("Invalid choice. Please enter 1 or 2.");

}

} catch (IOException e) {

e.printStackTrace();

}

}

private static void startTCPClient() {

try {

Socket socket = new Socket("localhost", TCP\_PORT);

System.out.println("TCP Client connected to server");

BufferedReader serverReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);

BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in));

new Thread(() -> {

try {

while (true) {

String response = serverReader.readLine();

System.out.println("Server: " + response);

}

} catch (IOException e) {

e.printStackTrace();

}

}).start();

while (true) {

System.out.print("Enter your message (type 'exit' to quit): ");

String message = consoleReader.readLine();

writer.println(message);

if (message.equals("exit")) {

break;

}

}

socket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

private static void startUDPClient() {

try {

DatagramSocket socket = new DatagramSocket();

InetAddress serverAddress = InetAddress.getByName("localhost");

BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in));

new Thread(() -> {

try {

while (true) {

byte[] receiveData = new byte[1024];

DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

socket.receive(receivePacket);

String response = new String(receivePacket.getData(), 0, receivePacket.getLength());

System.out.println("Server: " + response);

}

} catch (IOException e) {

e.printStackTrace();

}

}).start();

while (true) {

System.out.print("Enter your message (type 'exit' to quit): ");

String message = consoleReader.readLine();

byte[] sendData = message.getBytes();

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, serverAddress, UDP\_PORT);

socket.send(sendPacket);

if (message.equals("exit")) {

break;

}

}

socket.close();

} catch (IOException e) {

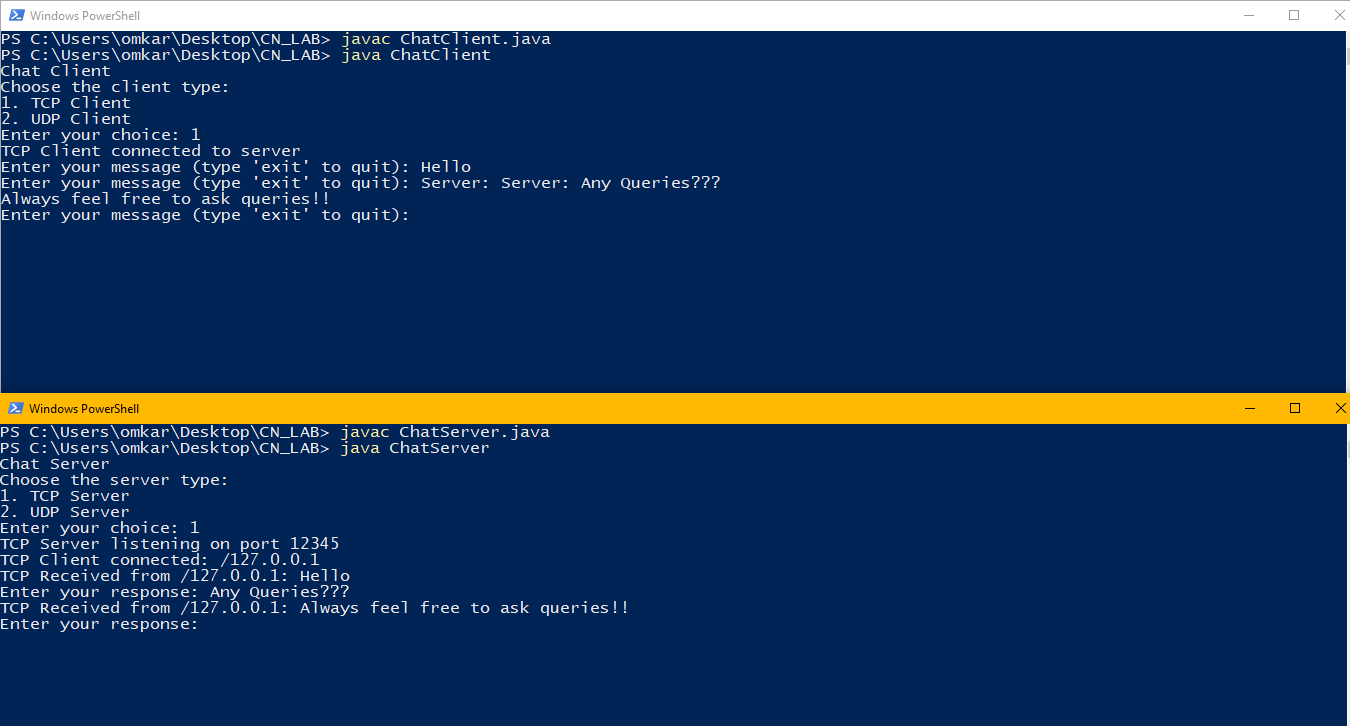
e.printStackTrace();

}

}

}

**End Result:**



**Conclusion:**

The TCP chat implementation in the provided code showcases a reliable and connection-oriented communication model, ensuring ordered and error-checked message exchange. This makes it suitable for applications prioritizing data integrity, such as file transfers or text-based communication. In contrast, the UDP chat leverages a connectionless, low-overhead approach, offering faster data transmission but without guarantees of reliability or ordered delivery. The choice between TCP and UDP in a chat application depends on the specific requirements, balancing factors like message integrity and real-time responsiveness. The code provides a practical illustration of these fundamental differences in socket-based communication.