# **Name: Abdurrahman Qureshi**

# **Roll No: 242466**

Practical No: 7

Date Of Performance: 19/03/2025

Aim: To Study and Implement Socket Programming Using TCP

Lab Objectives:

The objective of this lab is to study and implement **TCP (Transmission Control Protocol)** socket programming using Python. Students will create a client-server communication model to understand how TCP, a connection-oriented protocol, facilitates reliable data transmission. The lab focuses on creating sockets, establishing connections, and sending and receiving data streams between the client and server. This experiment will help students grasp the fundamental concepts of network communication and the differences between UDP and TCP, emphasizing TCP's reliability, error-checking, and connection management features.

Lab Outcomes:

By the end of this lab, students will be able to create and configure **TCP sockets** for communication between two endpoints. They will gain hands-on experience in establishing connections, sending, and receiving data streams using various Java methods. Additionally, students will understand the key characteristics of TCP, such as its connection-oriented nature, reliability, error-checking, and flow control mechanisms, and compare it with UDP. This lab will also enhance their problem-solving skills in network programming, connection management, and error handling.

Theory:

Introduction to TCP:

* TCP (Transmission Control Protocol) is a transport layer protocol in the Internet Protocol (IP) suite.
* Unlike UDP (User Datagram Protocol), TCP is connection-oriented, meaning it establishes a dedicated end-to-end connection before transmitting data.
* TCP ensures reliable, ordered, and error-checked delivery of data between applications. It is widely used for applications where data integrity and reliability are critical, such as web browsing, email, and file transfers.

Characteristics of TCP:

* Connection-Oriented: A connection is established using a three-way handshake (SYN, SYN-ACK, ACK) before data transmission begins.
* Reliable: TCP guarantees the delivery of data packets. Lost or corrupted packets are retransmitted.
* Ordered: Data packets are delivered to the application in the same order they were sent.
* Error Checking: TCP uses checksums to detect and correct errors in transmitted data.
* Flow Control: TCP manages data flow to prevent overwhelming the receiver by using windowing mechanisms.
* Congestion Control: TCP adjusts the rate of data transmission to avoid network congestion.
* Stream-Based: Data is treated as a continuous stream of bytes, not as discrete packets.

Advantages of TCP:

* Reliability: Ensures data is delivered accurately and in the correct order.
* Error Correction: Automatically detects and retransmits lost or corrupted packets.
* Flow and Congestion Control: Prevents network congestion and ensures efficient data transfer.
* Wide Compatibility: Supported by virtually all network devices and applications.

Disadvantages of TCP:

* Higher Overhead: The connection setup, reliability features, and flow control mechanisms increase latency and bandwidth usage.
* Slower than UDP: The additional features make TCP slower compared to UDP for real-time applications.
* Complexity: TCP's mechanisms (e.g., handshaking, retransmission) add complexity to implementation and management.

Applications of TCP:

* Web Browsing: HTTP/HTTPS protocols rely on TCP for reliable web page loading.
* Email: SMTP, IMAP, and POP3 protocols use TCP for sending and receiving emails.
* File Transfers: FTP and SFTP protocols use TCP for reliable file transfers.
* Remote Access: SSH and Telnet use TCP for secure and reliable remote connections.
* Database Communication: TCP is used for database connections and queries (e.g., MySQL, PostgreSQL).

Comparison with UDP:

* Connection: TCP is connection-oriented, while UDP is connectionless.
* Reliability: TCP ensures reliable delivery, whereas UDP does not.
* Speed: UDP is faster due to lower overhead, while TCP is slower but more reliable.
* Use Cases: TCP is ideal for applications requiring data integrity (e.g., web, email), while UDP is better for real-time applications (e.g., gaming, streaming).

CODE (Server):

import java.net.\*;

import java.io.\*;

public class EXP9Server\_A extends Thread {

    private ServerSocket serverSocket;

    public EXP9Server\_A(int port) throws IOException {

        serverSocket = new ServerSocket(port);

        serverSocket.setSoTimeout(10000);

    }

    public void run() {

        while (true) {

            try {

                System.out.println("Waiting for client on port " + serverSocket.getLocalPort() + "...");

                Socket server = serverSocket.accept(); *//* Accept client connection

                System.out.println("Connected to " + server.getRemoteSocketAddress());

                DataInputStream in = new DataInputStream(server.getInputStream());

                System.out.println("Client says: " + in.readUTF());

                DataOutputStream out = new DataOutputStream(server.getOutputStream());

                out.writeUTF("Thank you for connecting to " + server.getLocalSocketAddress() + "\nGoodbye!");

                server.close(); *//* Close connection

            } catch (SocketTimeoutException s) {

                System.out.println("Socket timed out!");

                break;

            } catch (IOException e) {

                e.printStackTrace();

                break;

            }

        }

    }

    public static void main(String[] args) {

        int port = 6066; *//* Default port

        try {

            Thread t = new EXP9Server\_A(port);

            t.start();

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

}

Q) Write a Java Program to create a client that connects to a server on localhost at port 6066, sends a message, receives a response, and then closes the connection.

CODE (Client):

import java.net.\*;

import java.io.\*;

public class EXP9Client\_A {

    public static void main(String[] args) {

        String serverName = "localhost"; *//* Server address

        int port = 6066; *//* Port number

        try {

            System.out.println("Connecting to " + serverName + " on port " + port);

            Socket client = new Socket(serverName, port);

            System.out.println("Connected to " + client.getRemoteSocketAddress());

            DataOutputStream out = new DataOutputStream(client.getOutputStream());

            out.writeUTF("Hello from " + client.getLocalSocketAddress());

            DataInputStream in = new DataInputStream(client.getInputStream());

            System.out.println("Server says: " + in.readUTF());

            client.close(); *//* Close connection

        } catch (IOException e) {

            e.printStackTrace();

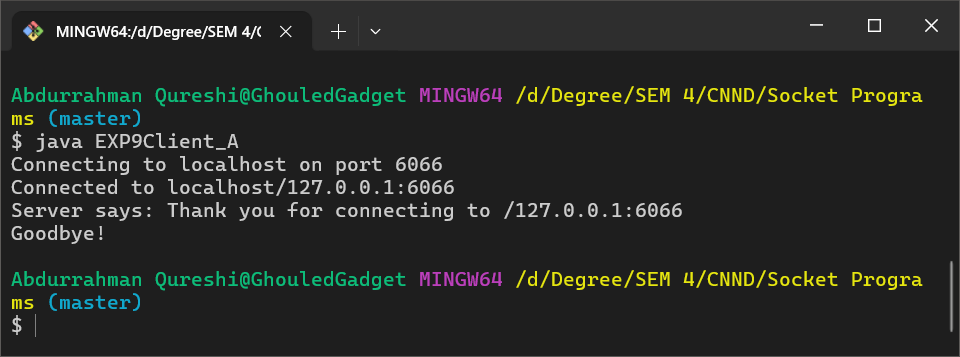
        }

    }

}

OUPTUT:





Q) Write a Java Program to create a client that connects to the time.nist.gov server on port 13 (Daytime Protocol), retrieves the current date and time from the server, and prints it to the console.

CODE (Client):

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.net.Socket;

import java.net.UnknownHostException;

public class EXP9Client\_B {

    public static void main(String[] args) {

        String hostname = "time.nist.gov";

        int port = 13;

        try (Socket socket = new Socket(hostname, port)) {

            InputStream input = socket.getInputStream();

            InputStreamReader reader = new InputStreamReader(input);

            int character;

            StringBuilder data = new StringBuilder();

            while ((character = reader.read()) != -1) {

                data.append((char) character);

            }

            System.out.println(data);

        } catch (UnknownHostException ex) {

            System.out.println("Server not found: " + ex.getMessage());

        } catch (IOException ex) {

            System.out.println("I/O error: " + ex.getMessage());

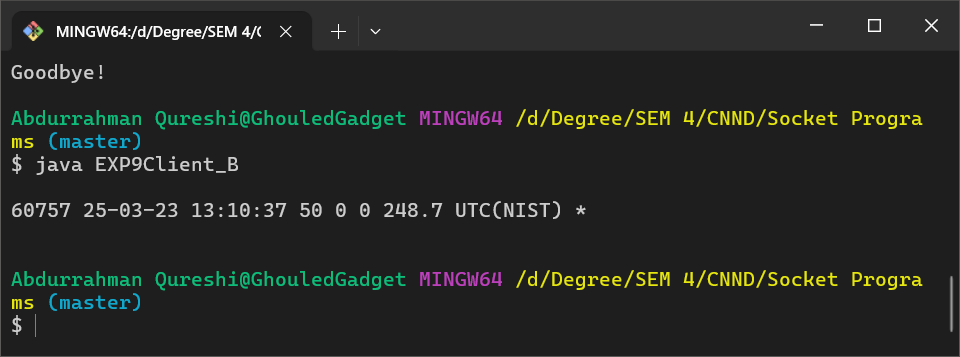
        }

    }

}

OUPTUT:





Conclusion:

This experiment provided key insights into **TCP socket programming**, highlighting its reliability, connection-oriented nature, and error-handling mechanisms. Students gained hands-on experience in establishing connections, managing data streams, and ensuring reliable client-server communication. By comparing TCP with UDP, they understood the trade-offs between reliability and speed. The experiment reinforced TCP's importance in applications requiring data integrity and error-free transmission, enhancing their grasp of modern networking concepts.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Performance  (7M) | Journal  (3M) | Lab Ethics  (2M) | Attendance  (3M) | Total  (15M) | Faculty Signature |
|  |  |  |  |  |  |