**Exp: no: 7**

Demonstration of Client server-based TCP applications using socket programming

**Aim:**

To implement and demonstrate a client-server application using **TCP (Transmission Control Protocol)** for communication between two systems, showcasing the basics of socket programming and TCP’s reliable, connection-oriented nature and to write the syntax, execute and place the screenshot for all the commands worked on.

1. **Write a client-server program to implement a TCP application.**

**Description:**

In this lab task, we’ll create a simple Client-Server application using TCP. TCP is a connection-oriented protocol that provides reliable communication by establishing a connection before data transfer begins. TCP guarantees that data packets arrive in sequence and without errors, making it suitable for applications where data integrity and reliability are essential.

The client will establish a connection with the server, send a message, and the server will respond back with a confirmation message. This application demonstrates TCP socket programming fundamentals, connection establishment, data transfer, and termination of the connection.

**Procedure:**

1. **Set Up the Environment:**

* Use Java to implement the client-server application. Ensure JDK is installed, and use a text editor or terminal to write and execute the code.

1. **Server Implementation:**

* The server binds to a specific port (5000) using ServerSocket and waits for a client to connect.
* Once a connection is established, it reads messages from the client using DataInputStream and prints them.
* The server continues receiving messages until it gets the termination signal "Over".

1. **Client Implementation:**

* The client connects to the server using its IP (127.0.0.1) and port (5000) via Socket.
* It sends messages to the server using DataOutputStream, taking input from the console.
* The client terminates the connection when "Over" is sent.

1. **Run the Server and Client:**

* Start the Server.java program first to make it ready for incoming connections.
* Then, run the Client.java program to establish the connection and exchange messages.

**Code:**

* Server Code (Server.java):

import java.net.\*;

import java.io.\*;

 public class Server

{

    private Socket          socket   = null;

    private ServerSocket    server   = null;

    private DataInputStream in       =  null;

    public Server(int port)

    {

        try{

            server = new ServerSocket(port);

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

            System.out.println("Waiting for a client ...");

            socket = server.accept();

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

            in = new DataInputStream(

                new BufferedInputStream(socket.getInputStream()));

            String line = "";

            while (!line.equals("Over"))

            {

                try{

                    line = in.readUTF();

                    System.out.println(line);

                }

                catch(IOException i)

                {

                    System.out.println(i);

                }

            }

            System.out.println("Closing connection");

            socket.close();

            in.close();

        }

        catch(IOException i)

        {

            System.out.println(i);

        }

    }

    public static void main(String args[])

    {

        Server server = new Server(5000);

    }

}

* Client Code (Client.java):

import java.io.\*;

import java.net.\*;

 public class Client {

    private Socket socket = null;

    private DataInputStream input = null;

    private DataOutputStream out = null;

    public Client(String address, int port)

    {

        try {

            socket = new Socket(address, port);

            System.out.println("Connected");

            input = new DataInputStream(System.in);

            out = new DataOutputStream(

                socket.getOutputStream());

        }

        catch (UnknownHostException u) {

            System.out.println(u);

            return;

        }

        catch (IOException i) {

            System.out.println(i);

            return;

        }

        String line = "";

        while (!line.equals("Over")) {

            try {

                line = input.readLine();

                out.writeUTF(line);

            } catch (IOException i) {

                System.out.println(i);

            }

        }

        try {

            input.close();

            out.close();

            socket.close();

        } catch (IOException i) {

            System.out.println(i);

        }

    }

    public static void main(String args[])

    {

        Client client = new Client("127.0.0.1", 5000);

    }

}

**Explanation of the Code**

1. **Server Code:**
   * The server creates a ServerSocket on port 5000 to listen for incoming client connections.

* Once a connection is established using accept(), it uses DataInputStream to receive messages from the client.
* Messages are printed on the server console, and the connection is closed after receiving "Over".

1. **Client Code:**

* The client creates a Socket to connect to the server at IP 127.0.0.1 and port 5000.
* It uses DataOutputStream to send messages and System.in to take input from the user.
* The client terminates the connection after sending "Over" and closes all resources.

**Output:**

* Server and Client java files stored in a separate folder

**A screenshot of a computer

Description automatically generated**

* Server.java before starting the Client.java file

A computer screen shot of a computer program

Description automatically generated

* Running Client.java file

A computer screen shot of a program

Description automatically generated

* Server.java after starting the Client.java file

A computer screen with white text

Description automatically generated

**Result:**

Thus, Client server-based TCP applications was demonstrated using socket programming and wrote the syntax, executed and placed the screenshot for all the commands worked on.