1. The client-server communication is studied in this experiment using socket programming to understand the TCP protocol. The server program and client program are executed separately. Initially, the TCP socket is created on the server and client sides. The binding is carried with the server address. Ensure that the client initiates the communication. Check the response of the client from the server side. Process the packet and send a reply to the client. Observer and ensure that the data transfers between Client and Server are reliable.

Aim:

To write a socket programming to establish a connection between client and server using TCP protocol.

Algorithm

Server-Side Algorithm (Java):

1. Initialize the server-side:
    - Create a ServerSocket to listen for incoming connections on a specific port.
    - Wait for a client to connect.
2. Accept the client connection:
    - When a client connection is accepted, obtain a Socket object to communicate with the client.
    - Display a message indicating that the client has connected.
3. Set up input and output streams:
    - Create BufferedReader and PrintWriter objects to read from and write to the client's socket.
4. Receive data from the client:
    - Read data from the client using the BufferedReader.
    - Process the received data as needed (e.g., modify or analyze it).
5. Send a response to the client:
    - Prepare a response message.
    - Send the response message to the client using the PrintWriter.
6. Close resources:
    - Close the input and output streams.
    - Close the client's socket.
    - Close the ServerSocket.

Client-Side Algorithm (Java):

1. Initialize the client-side:
    - Create a Socket object to connect to the server's IP address and port.
2. Set up input and output streams:
    - Create PrintWriter and BufferedReader objects to write to and read from the server's socket.
3. Send data to the server:
    - Prepare a message to be sent to the server.

- Send the message to the server using the PrintWriter.

4. Receive a response from the server:
    - Read the server's response using the BufferedReader.
    - Process the received response as needed.
5. Close resources:
    - Close the input and output streams.
    - Close the client's socket.

Program

TCP Server

```
import java.io.*;
import java.net.*;
public class TCPServer {
   public static void main(String[] args) {
      int port = 8080; // Port number to listen on

      try {
         // Create a server socket
         ServerSocket serverSocket = new ServerSocket(port);
         System.out.println("Server is listening on port " + port);

         // Wait for a client to connect
         Socket clientSocket = serverSocket.accept();
         System.out.println("Client connected from " +
clientSocket.getInetAddress().getHostAddress());

         // Set up input and output streams for communication
         BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
         PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

         // Read data from the client
         String clientMessage = in.readLine();
         System.out.println("Received from client: " + clientMessage);

         // Process the received data (You can implement your own logic here)
         // For example, you can modify the message or perform some calculations
         String responseMessage = "Hello, Client!"; // Change this message as needed

         // Send a reply back to the client
         out.println(responseMessage);
         System.out.println("Sent to client: " + responseMessage);

         // Close the sockets and streams
         in.close();
         out.close();
         clientSocket.close();
         serverSocket.close();
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

TCP Client

```
import java.io.*;
import java.net.*;

public class TCPClient {
    public static void main(String[] args) {
        String serverAddress = "127.0.0.1"; // Server's IP address
        int serverPort = 8080; // Server's port number

        try {
            // Create a socket and connect to the server
            Socket socket = new Socket(serverAddress, serverPort);
            System.out.println("Connected to the server.");

            // Set up input and output streams for communication
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

            // Send a message to the server
            String message = "Hello, Server!"; // Message to send to the server
            out.println(message);
            System.out.println("Sent to server: " + message);

            // Read the server's response
            String serverResponse = in.readLine();
            System.out.println("Received from server: " + serverResponse);

            // Close the socket and streams
            out.close();
            in.close();
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Result

Thus a socket programming to establish a connection between client and server using TCP protocol was executed successfully.