| Name of the Student: Neha Gupta |
| --- |
| Class and Div and Roll No: TE-1 / 49 |
| Experiment No.6 |
| Socket programming using TCP or UDP. |
| Date of Performance: 12/08/2025 |
| Date of Submission: 9/09/2025 |

**Experiment No 6**

Aim: Socket programming using TCP or UDP.

Theory:

Sockets:

A socket is one endpoint of a two way communication link between two programs running on the network. The socket mechanism provides a means of inter-process communication (IPC) by establishing named contact points between which the communication takes place.

Socket are generally employed in client server applications. The server creates a socket, attaches it to a network port addresses then waits for the client to contact it. The client creates a socket and then attempts to connect to the server socket.

When the connection is established, transfer of data takes place.

Types of Sockets :

There are two primary and common types of Sockets: the datagram socket and the stream socket.

1.Datagram Socket :

This is a type of network which has connection less point for sending and receiving packets. It is similar to mailbox. The letters (data) posted into the box are collected and delivered (transmitted) to a letterbox (receiving socket).

2.Stream Socket:

In Computer operating system, a stream socket is type of interprocess communications socket or network socket which provides a connection-oriented, sequenced, and unique flow of data without record boundaries with well defined mechanisms for creating and destroying connections and for detecting errors. It is similar to phone.

A connection is established between the phones (two ends) and a conversation (transfer of data) takes place.

**Program for TCP:**

|  |  |
| --- | --- |
| **Client Side Program:**  // TCPClient.js  const net = require("net");  const readline = require("readline");  const rl = readline.createInterface({  input: process.stdin,  output: process.stdout  });  // Connect to server  const client = net.createConnection({ port: 1234, host: "localhost" }, () => {  console.log("Connected to Server");  // Ask user to enter message  rl.question("Enter Message: ", (msg) => {  client.write(msg);  });  });  // Receive response from server  client.on("data", (data) => {  console.log("Server Msg:", data.toString());  client.end(); // close after receiving one response (like your Java code)  });  // Handle disconnect  client.on("end", () => {  console.log("Disconnected from Server");  }); | **Server Side Program:**  // TCPServer.js  const net = require("net");  const readline = require("readline");  const rl = readline.createInterface({  input: process.stdin,  output: process.stdout  });  const server = net.createServer((socket) => {  console.log("Client Connected");  // Receiving message from client  socket.on("data", (data) => {  console.log("Client Msg:", data.toString());  // Asking user for response  rl.question("Enter Response: ", (answer) => {  socket.write(answer);  });  });  socket.on("end", () => {  console.log("Client Disconnected");  });  });  // Listen on port 1234  server.listen(1234, () => {  console.log("Server is waiting for client on port 1234...");  }); |

**Output:**

|  |
| --- |
| **Server Side:** |
| **Client Side:** |

**Program for UDP:**

|  |  |
| --- | --- |
| **Client Side Program:**  // UDPClient.js  const dgram = require("dgram");  const readline = require("readline");  const client = dgram.createSocket("udp4");  const rl = readline.createInterface({  input: process.stdin,  output: process.stdout  });  const SERVER\_PORT = 1234;  const SERVER\_HOST = "localhost";  rl.question("Enter Message: ", (msg) => {  client.send(msg, SERVER\_PORT, SERVER\_HOST, (err) => {  if (err) {  console.error(err);  client.close();  }  });  });  client.on("message", (msg) => {  console.log("Server Msg:", msg.toString());  client.close(); // close after one exchange  }); | **Server Side Program:**  // UDPServer.js  const dgram = require("dgram");  const readline = require("readline");  const server = dgram.createSocket("udp4");  const rl = readline.createInterface({  input: process.stdin,  output: process.stdout  });  server.on("listening", () => {  const address = server.address();  console.log(`UDP Server listening on ${address.address}:${address.port}`);  });  server.on("message", (msg, rinfo) => {  console.log(`Client Msg: ${msg.toString()}`);  rl.question("Enter Response: ", (answer) => {  server.send(answer, rinfo.port, rinfo.address, (err) => {  if (err) console.error(err);  });  });  });  server.bind(1234); // listen on port 1234 |

**Output:**

|  |
| --- |
| **Server Side:** |
| **Client Side:** |

**Conclusion:**

In this experiment we have seen how to create sockets, establish connection between Server and Client and enable two-way communication between them.

**Questions:**

1. What is socket programming?

Ans- Socket programming is a way to enable communication between two programs over a network. It uses sockets, which are endpoints identified by an IP address and a port number, to establish a connection between processes running on the same or different machines. This programming allows data exchange using protocols like TCP or UDP and is fundamental to client-server applications and network communication.

2. What is the difference between TCP and UDP sockets?

Ans- The main differences between TCP and UDP sockets are:

1. TCP sockets are connection-oriented, requiring a connection to be established before data transfer, while UDP sockets are connectionless and send data without setting up a connection.
2. TCP ensures reliable data delivery with error checking, retransmission of lost packets, and data ordering. UDP provides no guarantee of delivery, ordering, or error recovery, offering faster but less reliable communication.
3. TCP sockets transmit data as a stream of bytes, whereas UDP sockets send data in individual packets (datagrams).
4. TCP is slower due to its overhead, while UDP is faster and more efficient for real-time applications like video streaming or gaming where speed is critical.

3. How do TCP and UDP handle packet loss?

Ans- TCP handles packet loss by detecting missing packets through sequence numbers and acknowledgments. When a packet is lost, TCP requests retransmission, ensuring reliable and ordered delivery. UDP, on the other hand, does not handle packet loss; it simply sends packets without acknowledgments or retransmissions, prioritizing speed over reliability.

4. How is port number important in socket programming?

Ans- Port numbers are important in socket programming because they identify specific processes or services on a device. They allow multiple applications to use the network simultaneously without interference by directing incoming network traffic to the correct program. A socket combines an IP address and a port number to create a unique communication endpoint, enabling precise data exchange between devices. This helps manage multiple connections efficiently and supports various network services on the same machine.

5. Why might you choose UDP over TCP in socket programming?

Ans- I might choose UDP over TCP in socket programming because UDP is faster and has less overhead. It doesn’t require establishing a connection or guaranteeing delivery, which helps reduce latency. This makes UDP ideal for real-time applications like video streaming, online gaming, or voice calls where speed is more important than perfect reliability. I also use UDP when I need to broadcast or multicast data efficiently to multiple recipients.