

# EXPERIMENT 8

## Aim

Write a program for interactive application using UDP socket.

## Prerequisite

Nil

## Outcome

To impart knowledge of Computer Networking Technology

## Theory

### 1. Introduction to UDP

UDP (User Datagram Protocol) is a connectionless, lightweight transport layer protocol in computer networking. It is part of the Internet Protocol (IP) suite and is designed for fast, low-overhead data transmission. UDP is known for its simplicity and speed, making it ideal for real-time and time-sensitive applications such as streaming media, online gaming, and VoIP (Voice over Internet Protocol). Unlike TCP, UDP does not guarantee the delivery or order of data packets, making it suitable for scenarios where minor data loss is acceptable in exchange for reduced latency.

### 2. Socket Programming

Socket programming involves the use of APIs and libraries to create, send, receive, and manage data over a network using sockets. Sockets can be created and configured to use either TCP or UDP for communication.

Two primary socket types exist: TCP sockets, ensuring reliable, connection-oriented communication with data sent in a stream, and UDP sockets, enabling fast, connectionless data transfer through discrete packets. TCP is used in applications that require data integrity and order, like web browsing and file transfer, while UDP is ideal for real-time, low-latency tasks such as multimedia streaming and online gaming.

### 3. UDP Server-Client Interaction:

- **Server:** The server program initializes a UDP socket, binds it to a specific port and IP address, and waits for incoming data from clients. Upon receiving data, the server processes the information and may send a response back to the client.
- **Client:** The client program creates a UDP socket, specifies the server's address and port, and sends data to the server. It can also receive responses from the server.

## Procedure

### 1. Server Implementation:

- Create a UDP socket using the appropriate programming language Java.
- Bind the socket to a specific IP address and port to listen for incoming data.
- Receive data from clients using receive().
- Process the received data as needed.
- Send a response back to the client using send().

### 2. Client Implementation:

- Create a UDP socket.
- Specify the server's IP address and port.
- Send data to the server using send().
- Receive responses from the server using receive().
- Process and display the received data.

## Steps

### 1. Server Implementation

```

import java.net.*;
public class UDPServer {
    public static void main(String[] args) throws Exception{
        int port = 9999;
        DatagramSocket ds = new DatagramSocket(port);

        // Getting Data from client
        byte[] b1 = new byte[1024];
        DatagramPacket dp1 = new DatagramPacket(b1, b1.length);
        ds.receive(dp1);

        // Performing square operation on data received
        String str = new String(dp1.getData(), 0, dp1.getLength());
        int num = Integer.parseInt(str.trim());
        int result = num*num;

        // Sending data back to client
        byte[] b2 = String.valueOf(result).getBytes();
        InetAddress ip = InetAddress.getLocalHost();
        DatagramPacket dp2 = new DatagramPacket(b2, b2.length, ip, dp1.getPort());
        ds.send(dp2);
    }
}

```

## 2. Client Implementation

```
import java.net.*;
import java.util.Scanner;
public class UDPClient {
    public static void main(String[] args) throws Exception {
        int port = 9999;
        DatagramSocket ds = new DatagramSocket();
        Scanner sc = new Scanner(System.in); // Create a Scanner object
        System.out.print("Enter number: ");

        int num = sc.nextInt();
        byte[] b1 = String.valueOf(num).getBytes();
        // Sending Data to Server
        InetAddress ip = InetAddress.getLocalHost();
        DatagramPacket dp1 = new DatagramPacket(b1, b1.length, ip, port);
        ds.send(dp1);

        // Receiving Data from Server
        byte[] b2 = new byte[1024];
        DatagramPacket dp2 = new DatagramPacket(b2, b2.length);
        ds.receive(dp2);

        String str = new String(dp2.getData(), 0, dp2.getLength());
        System.out.println("Result: " + str);

    }
}
```

## Output

The screenshot shows two terminal windows side-by-side. The left window is titled 'Run' and shows the output of the UDP Server. It displays the path 'C:\Program Files\Java\jdk-18.0.2\bin' and the message 'Server is running on port 9999'. The right window is also titled 'Run' and shows the output of the UDP Client. It starts with the path 'C:\Program Files\Java\jdk-18.0.2\b'. The user then enters 'Enter number: 6'. The program calculates the square of 6 and prints 'Square of 6 : 36'. Finally, it concludes with 'Process finished with exit code 0'.

```
"C:\Program Files\Java\jdk-18.0.2\bin
Server is running on port 9999

"C:\Program Files\Java\jdk-18.0.2\b
Enter number: 6
Square of 6 : 36
Process finished with exit code 0
```

## Observation & Learning

- UDP sockets provide a lightweight and low-latency communication method suitable for real-time applications.
- The server program creates a UDP socket, binds it to a specific IP address and port, and listens for incoming data from clients.
- The client program establishes a UDP socket, specifies the server's address and port, and sends data to the server.
- UDP does not guarantee the order of delivery or data reliability, but it excels in scenarios where speed and minimal overhead are critical.
- The interactive application can include features like sending and receiving messages, commands, or data between the client and server.

## Conclusion

In this experiment, we successfully implemented an interactive application using UDP sockets. UDP's characteristics, such as low latency and connectionless communication, make it suitable for real-time applications like chat programs, online games, or remote control. The client and server components worked together to send and receive data over the network. UDP socket programming offers flexibility and efficiency for various interactive applications, allowing for faster data exchange and responsiveness.