INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Práctica 2

Sockets TCP Y UDP Java

Desarrollo de sistemas distribuidos

Sánchez Torres Kevin Alexander

4CM12

Tabla de contenido

Introducción

Los sockets son puntos finales de una comunicación bidireccional entre dos dispositivos en una red. En términos más simples, un socket es la combinación de una dirección IP y un puerto que permite a las aplicaciones enviar y recibir datos a través de la red. Java proporciona soporte robusto para trabajar con sockets, permitiendo la implementación de aplicaciones de red utilizando los protocolos TCP (Transmission Control Protocol) y UDP (User Datagram Protocol), los cuales son fundamentales en la comunicación en red.

Sockets TCP (Transmission Control Protocol)

El protocolo TCP es uno de los principales protocolos utilizados en internet para la transmisión de datos. Se caracteriza por ser un protocolo orientado a la conexión, lo que implica que antes de que dos máquinas comiencen a intercambiar datos, deben establecer una conexión formal. TCP garantiza que todos los paquetes de datos lleguen al destino de manera fiable y en el orden correcto.

Algunas características importantes de los sockets TCP son:

- Fiabilidad: TCP garantiza la entrega de los datos sin errores. Si algún paquete se pierde o llega dañado, se retransmite.

- Control de flujo: TCP ajusta la velocidad a la que se envían los datos según las capacidades del receptor.

- Orientado a conexión: Antes de comenzar la transmisión de datos, se establece una conexión entre el servidor y el cliente mediante un proceso de "handshake" (intercambio de señales).

En Java, un servidor TCP generalmente usa un objeto ServerSocket para esperar conexiones de clientes, mientras que un cliente utiliza un objeto Socket para conectarse al servidor y luego intercambiar datos.

Sockets UDP (User Datagram Protocol)

A diferencia de TCP, UDP es un protocolo sin conexión. Esto significa que no se establece una conexión formal entre el cliente y el servidor antes de que comience la transmisión de datos. Los datos se envían en bloques llamados datagramas, que no tienen garantía de ser entregados, ni se asegura que lleguen en el orden correcto.

Las principales características de los sockets UDP son:

- Sin conexión: No se necesita establecer una conexión previa entre el cliente y el servidor.

- Velocidad: Al no tener el control de flujo ni la garantía de entrega que ofrece TCP, UDP es mucho más rápido y eficiente en términos de rendimiento.

- No fiable: Los datagramas pueden perderse, duplicarse o llegar en un orden incorrecto. El manejo de errores y retransmisiones debe ser implementado por la aplicación si se necesita.

En Java, tanto los clientes como los servidores utilizan la clase DatagramSocket para enviar y recibir datagramas, y la clase DatagramPacket para representar cada datagrama.

Comparación entre TCP y UDP

Característica	TCP	UDP
Orientación	Orientado a la conexión	Sin conexión
Fiabilidad	Garantiza entrega y orden correcto	No garantiza entrega ni orden
Velocidad	Más lento debido a su fiabilidad	Más rápido por su simplicidad
Uso de recursos	Mayor uso de recursos	Menor uso de recursos
Casos de uso	Transferencia de archivos, HTTP, correo electrónico	Streaming de video, juegos en línea, DNS

Tabla 1. Tabla comparativa.

Objetivo

Implementar y comprender el funcionamiento de sockets en Java utilizando los protocolos TCP y UDP, explorando las diferencias entre ambos en términos de fiabilidad, velocidad y gestión de la conexión. Además de, aprender a desarrollar aplicaciones cliente-servidor que intercambien datos a través de una red, utilizando sockets TCP para garantizar la entrega de datos y el orden correcto, y sockets UDP para aplicaciones donde la velocidad es prioritaria, aún sin asegurar la entrega de todos los datos.

Problemática

En el contexto de las comunicaciones entre aplicaciones distribuidas, es esencial contar con mecanismos eficientes y fiables para intercambiar información entre dispositivos conectados a una red. Las aplicaciones que requieren realizar cálculos o procesar datos de manera remota deben enfrentar desafíos como la fiabilidad, velocidad, y el manejo adecuado de errores durante la transmisión de datos.

En este escenario, se plantea la necesidad de diseñar un sistema cliente-servidor que permita realizar operaciones aritméticas (suma, resta, multiplicación y división) entre dos números proporcionados por el cliente. Para resolver esta problemática, se debe implementar el sistema

usando dos enfoques diferentes: uno basado en TCP, que garantiza la fiabilidad y el orden de los datos, y otro basado en UDP, que prioriza la velocidad sin asegurarse de que los datos se reciban en el orden correcto o incluso lleguen al destino.

La problemática a resolver implica lo siguiente:

- Comunicación Fiable (TCP): El cliente envía dos números y un operador aritmético al servidor, que realiza la operación correspondiente y devuelve el resultado. Esta solución debe garantizar que los datos lleguen correctamente y en el orden adecuado, lo cual es crucial en aplicaciones donde la integridad de los datos es esencial.

- Comunicación Rápida (UDP): En otra variante, se implementa el mismo sistema, pero utilizando UDP, un protocolo sin conexión que prioriza la velocidad sobre la fiabilidad. Aunque puede haber pérdida de datos, este protocolo es útil en aplicaciones donde el tiempo de respuesta es más importante que la exactitud.

Mediante esta práctica, se busca entender cómo implementar estas soluciones en Java y analizar las diferencias prácticas entre los protocolos TCP y UDP al resolver una misma problemática de comunicación y procesamiento de datos.

Solución

Para crear los sockets servidor-cliente TCP y UDP en java, donde el cliente envía 2 números y un operador al servidor para que este los sume y envié al cliente el resultado, se implementa de la siguiente manera.

Servidor TCP

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

public class Servidor {
    Run | Debug
    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(port:12345)) {
            System.out.println(x:"Servidor escuchando en el puerto 12345...");
            while (true) {
                try (Socket clientSocket = serverSocket.accept();
                        DataInputStream in = new DataInputStream(clientSocket.getInputStream());
                        DataOutputStream out = new DataOutputStream(clientSocket.getOutputStream())) {

                    System.out.println(x:"Cliente conectado.");

                    // Leer los números y el operador del cliente
                    double num1 = in.readDouble();
                    double num2 = in.readDouble();
                    char operador = in.readChar();

                    // Realizar la operación
                    double resultado = 0;
                    switch (operador) {
                        case '+':
                            resultado = num1 + num2;
                            break;
                        case '-':
                            resultado = num1 - num2;
                            break;
                        case '*':
                            resultado = num1 * num2;
                            break;
                        case '/':
                            if (num2 != 0) {
                                resultado = num1 / num2;
                            } else {
                                out.writeUTF(str:"Error: División por cero");
                                continue;
                            }
                            break;
                        default:
                            out.writeUTF(str:"Error: Operador no válido");
                            continue;
                    }

                    // Enviar el resultado al cliente
                    out.writeDouble(resultado);
                } catch (IOException e) {
                    System.out.println("Error en la comunicación con el cliente: " + e.getMessage());
                }
            }
        } catch (IOException e) {
            System.out.println("Error al iniciar el servidor: " + e.getMessage());
        }
    }
}
```

Figura 1. Socket servidor TCP.

Cliente TCP

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

public class Cliente {
    Run | Debug
    public static void main(String[] args) {
        String host = "localhost";
        int puerto = 12345;

        try (Socket socket = new Socket(host, puerto);
                DataOutputStream out = new DataOutputStream(socket.getOutputStream());
                DataInputStream in = new DataInputStream(socket.getInputStream())) {

            Scanner scanner = new Scanner(System.in);

            // Pedir los números y el operador al usuario
            System.out.print(s:"Introduce el primer número: ");
            double num1 = scanner.nextDouble();
            System.out.print(s:"Introduce el segundo número: ");
            double num2 = scanner.nextDouble();
            System.out.print(s:"Introduce el operador (+, -, *, /): ");
            char operador = scanner.next().charAt(index:0);

            // Enviar los números y el operador al servidor
            out.writeDouble(num1);
            out.writeDouble(num2);
            out.writeChar(operador);

            // Leer el resultado del servidor
            double resultado = in.readDouble();
            System.out.println("Resultado: " + resultado);

        } catch (IOException e) {
            System.out.println("Error en la comunicación con el servidor: " + e.getMessage());
        }
    }
}
```

Figura 2. Socket cliente TCP.

Servidor UDP

```java
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class ServidorUDP {
    Run | Debug
    public static void main(String[] args) {
        try (DatagramSocket socket = new DatagramSocket(port:12345)) {
            byte[] buffer = new byte[1024];
            System.out.println(x:"Servidor UDP escuchando en el puerto 12345...");

            while (true) {
                DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
                socket.receive(packet);

                String mensaje = new String(packet.getData(), offset:0, packet.getLength());
                String[] partes = mensaje.split(regex:" ");
                double num1 = Double.parseDouble(partes[0]);
                double num2 = Double.parseDouble(partes[1]);
                char operador = partes[2].charAt(index:0);

                double resultado = 0;
                switch (operador) {
                    case '+':
                        resultado = num1 + num2;
                        break;
                    case '-':
                        resultado = num1 - num2;
                        break;
                    case '*':
                        resultado = num1 * num2;
                        break;
                    case '/':
                        if (num2 != 0) {
                            resultado = num1 / num2;
                        } else {
                            resultado = Double.NaN; // División por cero
                        }
                        break;
                    default:
                        System.out.println(x:"Operador no válido");
                        continue;
                }

                String respuesta = Double.toString(resultado);
                byte[] bufferRespuesta = respuesta.getBytes();
                InetAddress address = packet.getAddress();
                int port = packet.getPort();
                DatagramPacket packetRespuesta = new DatagramPacket(bufferRespuesta, bufferRespuesta.length, address,
                        port);
                socket.send(packetRespuesta);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Figura 3. Socket servidor UDP.

Cliente UDP

```java
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

public class ClienteUDP {
    Run | Debug
    public static void main(String[] args) {
        String host = "localhost";
        int puerto = 12345;

        try (DatagramSocket socket = new DatagramSocket()) {
            Scanner scanner = new Scanner(System.in);

            // Pedir los números y el operador al usuario
            System.out.print(s:"Introduce el primer número: ");
            double num1 = scanner.nextDouble();
            System.out.print(s:"Introduce el segundo número: ");
            double num2 = scanner.nextDouble();
            System.out.print(s:"Introduce el operador (+, -, *, /): ");
            char operador = scanner.next().charAt(index:0);

            // Crear el mensaje a enviar
            String mensaje = num1 + " " + num2 + " " + operador;
            byte[] buffer = mensaje.getBytes();
            InetAddress address = InetAddress.getByName(host);

            DatagramPacket packet = new DatagramPacket(buffer, buffer.length, address, puerto);
            socket.send(packet);

            byte[] bufferRespuesta = new byte[1024];
            DatagramPacket packetRespuesta = new DatagramPacket(bufferRespuesta, bufferRespuesta.length);
            socket.receive(packetRespuesta);

            String respuesta = new String(packetRespuesta.getData(), offset:0, packetRespuesta.getLength());
            System.out.println("Resultado: " + respuesta);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Figura 4. Socket cliente UDP.

Descripción de la solución

El código va realizando las siguientes tareas.

Descripción Servidor TCP

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

Figura 5. Importación de librerías.

Estas librerías son esenciales para manejar la entrada y salida de datos (java.io.*), la comunicación en red (java.net.*), y para interactuar con el usuario a través de la consola (java.util.Scanner).

```
public class Servidor {
    Run | Debug
    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(port:12345)) {
            System.out.println(x:"Servidor escuchando en el puerto 12345...");
```

Figura 6. Inicio del servidor.

Aquí se inicia el servidor creando un objeto ServerSocket, que escucha en el puerto 12345. El servidor espera conexiones entrantes y permanece activo mientras se ejecuta.

```
while (true) {
    try (Socket clientSocket = serverSocket.accept();
            DataInputStream in = new DataInputStream(clientSocket.getInputStream());
            DataOutputStream out = new DataOutputStream(clientSocket.getOutputStream())) {

        System.out.println(x:"Cliente conectado.");
```

Figura 7. Bucle infinito para aceptar clientes.

El servidor entra en un bucle infinito (while(true)) para aceptar múltiples clientes. Cuando un cliente intenta conectarse, se crea un socket cliente (clientSocket).Se crean los flujos de entrada y salida para enviar y recibir datos del cliente.

```java
// Leer los números y el operador del cliente
double num1 = in.readDouble();
double num2 = in.readDouble();
char operador = in.readChar();
```

Figura 8. Recepción de los números y el operador.

El servidor lee los datos enviados por el cliente: dos números de tipo double y un operador aritmético de tipo char.

```java
// Realizar la operación
double resultado = 0;
switch (operador) {
    case '+':
        resultado = num1 + num2;
        break;
    case '-':
        resultado = num1 - num2;
        break;
    case '*':
        resultado = num1 * num2;
        break;
    case '/':
        if (num2 != 0) {
            resultado = num1 / num2;
        } else {
            out.writeUTF(str:"Error: División por cero");
            continue;
        }
        break;
    default:
        out.writeUTF(str:"Error: Operador no válido");
        continue;
}
```

Figura 9. Operación aritmética.

Dependiendo del operador recibido, el servidor realiza la operación correspondiente. Si el operador es / (división), el servidor verifica que no haya división por cero, y si ocurre, envía un mensaje de error al cliente.

```
// Enviar el resultado al cliente
out.writeDouble(resultado);
```

Figura 10. Envío del resultado al cliente.

Una vez realizada la operación, el servidor envía el resultado al cliente usando el flujo de salida.

Descripción Cliente TCP

```
public class Cliente {
    Run | Debug
    public static void main(String[] args) {
        String host = "localhost";
        int puerto = 12345;
```

Figura 11. Inicio del cliente.

Se define el host como localhost (lo que significa que el cliente se conecta al servidor que está en la misma máquina) y el puerto en el que el servidor está escuchando (12345).

```
try (Socket socket = new Socket(host, puerto);
        DataOutputStream out = new DataOutputStream(socket.getOutputStream());
        DataInputStream in = new DataInputStream(socket.getInputStream())) {
```

Figura 12. Conexión al servidor.

El cliente crea un socket para conectarse al servidor. Si el servidor está escuchando, se establece una conexión. Se crean los flujos de entrada y salida para enviar y recibir datos entre el cliente y el servidor.

```
Scanner scanner = new Scanner(System.in);

// Pedir los números y el operador al usuario
System.out.print(s:"Introduce el primer número: ");
double num1 = scanner.nextDouble();
System.out.print(s:"Introduce el segundo número: ");
double num2 = scanner.nextDouble();
System.out.print(s:"Introduce el operador (+, -, *, /): ");
char operador = scanner.next().charAt(index:0);
```

Figura 13. Interacción con el usuario.

El cliente solicita al usuario dos números y un operador aritmético, que se ingresan a través de un Scanner.

```
// Enviar los números y el operador al servidor
out.writeDouble(num1);
out.writeDouble(num2);
out.writeChar(operador);
```

Figura 14. Recepción del resultado.

El cliente envía los dos números y el operador al servidor utilizando el flujo de salida (out).

```
// Leer el resultado del servidor
double resultado = in.readDouble();
System.out.println("Resultado: " + resultado);
```

Figura 15. Interacción con el usuario.

El cliente espera la respuesta del servidor, que contiene el resultado de la operación aritmética, y lo imprime en la consola.

Descripción Servidor UDP

```
public class ServidorUDP {
    Run | Debug
    public static void main(String[] args) {
        try (DatagramSocket socket = new DatagramSocket(port:12345)) {
            byte[] buffer = new byte[1024];
            System.out.println(x:"Servidor UDP escuchando en el puerto 12345...");
```

Figura 16. Inicio del servidor UDP.

Se crea un DatagramSocket que escucha en el puerto 12345. A diferencia de TCP, UDP no requiere una conexión establecida, simplemente espera recibir paquetes. El servidor reserva un buffer de 1024 bytes para recibir datos.

```
while (true) {
    DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
    socket.receive(packet);
```

Figura 17. Recepción del paquete.

El servidor entra en un bucle infinito para recibir paquetes de datos. Cuando un paquete llega, se almacena en el buffer y se lee usando socket.receive(packet).

```
String mensaje = new String(packet.getData(), offset:0, packet.getLength());
String[] partes = mensaje.split(regex:" ");
double num1 = Double.parseDouble(partes[0]);
double num2 = Double.parseDouble(partes[1]);
char operador = partes[2].charAt(index:0);
```

Figura 18. Procesamiento del mensaje recibido.

El mensaje recibido se convierte de bytes a una cadena de texto (String). La cadena se divide en partes: los dos números y el operador, utilizando el método split(" "). Los números se convierten a tipo double y el operador se toma como char.

```
String respuesta = Double.toString(resultado);
byte[] bufferRespuesta = respuesta.getBytes();
InetAddress address = packet.getAddress();
int port = packet.getPort();
DatagramPacket packetRespuesta = new DatagramPacket(bufferRespuesta, bufferRespuesta.length, address,
        port);
socket.send(packetRespuesta);
```

Figura 19. Envío de la respuesta al cliente.

El resultado de la operación se convierte a cadena de texto y se convierte en un array de bytes para ser enviado. Se crea un nuevo DatagramPacket para contener el resultado y enviarlo de vuelta al cliente usando el mismo DatagramSocket.

Descripción Cliente UDP

```
public class ClienteUDP {
    Run | Debug
    public static void main(String[] args) {
        String host = "localhost";
        int puerto = 12345;
```

Figura 20. Inicio del cliente UDP.

El cliente define que se va a conectar al host local (localhost) y al puerto 12345, que es donde el servidor UDP está escuchando.

```
try (DatagramSocket socket = new DatagramSocket()) {
    Scanner scanner = new Scanner(System.in);
```

Figura 21. Configuración del socket.

Se crea un DatagramSocket sin especificar puerto, ya que se utilizará un puerto libre disponible automáticamente.

```java
// Pedir los números y el operador al usuario
System.out.print(s:"Introduce el primer número: ");
double num1 = scanner.nextDouble();
System.out.print(s:"Introduce el segundo número: ");
double num2 = scanner.nextDouble();
System.out.print(s:"Introduce el operador (+, -, *, /): ");
char operador = scanner.next().charAt(index:0);
```

Figura 22. Entrada del usuario.

El cliente solicita al usuario los dos números y el operador aritmético.

```java
// Crear el mensaje a enviar
String mensaje = num1 + " " + num2 + " " + operador;
byte[] buffer = mensaje.getBytes();
InetAddress address = InetAddress.getByName(host);

DatagramPacket packet = new DatagramPacket(buffer, buffer.length, address, puerto);
socket.send(packet);
```

Figura 23. Envío de datos al servidor.

Los datos ingresados por el usuario se combinan en una cadena de texto, separados por espacios, y luego se convierten a un array de bytes. Se envía el paquete a la dirección del servidor y el puerto 12345.

```java
byte[] bufferRespuesta = new byte[1024];
DatagramPacket packetRespuesta = new DatagramPacket(bufferRespuesta, bufferRespuesta.length);
socket.receive(packetRespuesta);

String respuesta = new String(packetRespuesta.getData(), offset:0, packetRespuesta.getLength());
System.out.println("Resultado: " + respuesta);
```

Figura 24. Recepción de la respuesta del servidor.

El cliente prepara un nuevo buffer para recibir la respuesta del servidor. Después de recibir el paquete, convierte los datos de bytes a una cadena y muestra el resultado de la operación en la consola.

Resultado

```
C:\Users\keval>java Servidor
Servidor escuchando en el puerto 12345...
Cliente conectado.
```

Figura 25. Resultado servidor TCP.

```
C:\Users\keval>java Cliente
Introduce el primer número: 5
Introduce el segundo número: 7
Introduce el operador (+, -, *, /): +
Resultado: 12.0
```

Figura 26. Resultado cliente TCP.

```
C:\Users\keval>java ServidorUDP
Servidor UDP escuchando en el puerto 12345...
```

Figura 27. Resultado servidor UDP.

```
C:\Users\keval>java ClienteUDP
Introduce el primer número: 23
Introduce el segundo número: 34.5
Introduce el operador (+, -, *, /): *
Resultado: 793.5

C:\Users\keval>java ClienteUDP
Introduce el primer número: 20
Introduce el segundo número: 3.4
Introduce el operador (+, -, *, /): -
Resultado: 16.6
```

Figura 28. Resultado cliente UDP.

Conclusión

En esta práctica, se implementaron programas de cliente-servidor utilizando el protocolo UDP, lo cual permitió explorar la transmisión de

datos sin conexión y sin garantía de entrega entre ambos extremos. A diferencia de TCP, UDP es más rápido, pero menos confiable, ya que no asegura la llegada o el orden correcto de los paquetes. Esta implementación es adecuada para aplicaciones donde la velocidad es prioritaria y la pérdida ocasional de datos es aceptable.

En el servidor, se procesaron operaciones aritméticas enviadas por el cliente, demostrando cómo se puede utilizar UDP para construir servicios que respondan solicitudes sin necesidad de mantener una conexión persistente. En el cliente, se destacó la interacción con el usuario para ingresar números y un operador aritmético, simulando un escenario de cálculo remoto.