Centro de Enseñanza Técnica Industrial

Desarrollo de Software

Sockets

Jesús Alberto Aréchiga Carrillo

22310439 5N

Profesor

José Francisco Pérez Reyes

Noviembre 2024

Guadalajara, Jalisco

Introducción

Los sockets son un mecanismo fundamental para la comunicación entre procesos en una red. Representan un punto final en la comunicación bidireccional entre dos dispositivos, ya sea en una misma máquina o en diferentes dispositivos conectados a través de una red.

Los sockets son útiles en una amplia variedad de aplicaciones, como chat en tiempo real, juegos multijugador, transferencia de archivos y sistemas distribuidos. Aunque ofrecen gran flexibilidad, es importante manejar adecuadamente errores como interrupciones de conexión, y asegurar el uso de recursos con un cierre correcto de los sockets al finalizar la comunicación.

Desarrollo

Los códigos para utilizar son:

Server

```
import java.io.BufferedReader;

import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.ServerSocket;
import java.net.Socket;

public class Server {

    protected ServerSocket serverSocket;
    protected Socket clientSocket;
    protected DataOutputStream outputClient;
    protected BufferedReader input;
    protected String menssage;

    public Server() throws IOException {
        serverSocket = new ServerSocket(1234);
        clientSocket = new Socket();
    }

    public void startServer() {
        try {
            System.out.println("Esperando...");
            clientSocket = serverSocket.accept();
            System.out.println("Cliente en línea...");
            //Se obtiene el flujo de salida del cliente para enviarle
mensajes
            outputClient = new
DataOutputStream(clientSocket.getOutputStream());
            //Se le envía un mensaje al cliente usando su flujo de salida
```

```
            outputClient.writeUTF("Petición recibida y aceptada");
            //Se obtiene el flujo entrante desde el cliente
            input = new BufferedReader(new
            InputStreamReader(clientSocket.getInputStream()));
            while((menssage = input.readLine()) != null) {
                System.out.println(menssage);
            }
            System.out.println("Fin de la conexión");
            serverSocket.close();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

    public static void main(String[] args) throws IOException {
        Server server = new Server();
        System.out.println("Iniciando servidor...");
        server.startServer();
    }
}
```

Cliente:

```
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;

public class Client {

    protected Socket serverSocket;
    protected DataOutputStream outputServer;

    public Client() throws IOException{
        serverSocket = new Socket("192.168.1.85", 1234);
    }

    public void startClient() {
        try {
            //Flujo de datos hacia el servidor
            outputServer = new
DataOutputStream(serverSocket.getOutputStream());
            for (int i = 0; i < 10; i++) {
                outputServer.writeUTF("Este es el mensaje número " + (i +
1) + "\n");
                System.out.println("Mensaje " + (i + 1) + " enviado");
            }
            outputServer.flush();
            serverSocket.close();
```

```
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }

    public static void main(String[] args) throws IOException {
        Client client = new Client();
        System.out.println("Iniciando cliente...");
        client.startClient();
    }
}
```

Se van a utilizar dos equipos de cómputo para hacer la ejecución.

En el caso de Windows, es necesario hacer la regla en el firewall para que admita la conexión o desactivar el firewall completamente, en este caso se permitió la conexión en el puerto 1234, como está en el código del servidor y cliente.

Se inicia el servidor ejecutando el código de servidor:

```
Iniciando servidor...
Esperando...
```

Se corre ahora el código de Cliente para que se envíen los mensajes que tiene programados:

```
Iniciando cliente...
Mensaje 1 enviado
Mensaje 2 enviado
Mensaje 3 enviado
Mensaje 4 enviado
Mensaje 5 enviado
Mensaje 6 enviado
Mensaje 7 enviado
Mensaje 8 enviado
Mensaje 9 enviado
Mensaje 10 enviado
```

Se verifica en el servidor que se hayan recibido los mensajes:

```
Cliente en línea...
↔Este es el mensaje n?ºmero 1
↔Este es el mensaje n?ºmero 2
↔Este es el mensaje n?ºmero 3
↔Este es el mensaje n?ºmero 4
↔Este es el mensaje n?ºmero 5
↔Este es el mensaje n?ºmero 6
↔Este es el mensaje n?ºmero 7
↔Este es el mensaje n?ºmero 8
↔Este es el mensaje n?ºmero 9
▲Este es el mensaje n?ºmero 10
Fin de la conexión
```

Los mensajes llegaron correctamente desde el cliente hasta el servidor.

Conclusiones:

Los sockets son una herramienta esencial en la programación de redes, proporcionando un medio eficiente para la comunicación entre dispositivos o procesos. Su versatilidad permite implementar desde aplicaciones simples, como chats, hasta sistemas complejos, como servidores distribuidos.

La elección entre TCP y UDP depende de las necesidades del proyecto: TCP garantiza la entrega de datos de manera ordenada, ideal para aplicaciones donde la fiabilidad es clave, mientras que UDP es más adecuado para transmisiones rápidas donde la pérdida ocasional de paquetes es aceptable.

En Java, el uso de sockets se simplifica gracias al paquete java.net, aunque el programador debe manejar cuidadosamente excepciones, tiempos de espera y el cierre adecuado de conexiones para evitar fugas de recursos. Con una gestión adecuada, los sockets son una base sólida para desarrollar aplicaciones de red robustas y escalables.