UD3 - 02.- Comunicación entre aplicaciones

Listado de Contenidos:

1. Concepto de comunicación entre aplicaciones.
2. Modelos cliente/servidor:
    - Definición.
    - Características.
    - Ejemplo práctico: Implementar un cliente y un servidor básicos.
3. Modelos P2P (peer-to-peer):
    - Definición.
    - Características.
    - Ejemplo práctico: Simulación de un chat entre nodos P2P.
4. Modelos híbridos:
    - Definición.
    - Características.
    - Ejemplo práctico: Combinación de cliente/servidor y P2P.

Desarrollo de Contenidos

1. Concepto de Comunicación entre Aplicaciones

La comunicación entre aplicaciones es el proceso mediante el cual dos o más programas intercambian información a través de una red. Esta comunicación puede darse de diferentes formas, dependiendo de la arquitectura y los protocolos utilizados. Las aplicaciones pueden ser:

- Clientes: Solicitan servicios o recursos.
- Servidores: Proveen servicios o recursos.
- Nodos iguales (P2P): Ambos roles en una red descentralizada.

Ejemplo sencillo:

- Cuando utilizas un navegador web (cliente) para acceder a una página, este se comunica con un servidor que envía los datos al cliente.

---

2. Modelos Cliente/Servidor

Definición

El modelo cliente/servidor es una arquitectura en la que una aplicación (cliente) solicita recursos o servicios a otra aplicación (servidor). Es la base de muchas aplicaciones modernas como sitios web, bases de datos y servicios de mensajería.

Características:

- Centralización: El servidor gestiona todos los recursos y controla la comunicación.
- Confiabilidad: Los clientes dependen del servidor para obtener los servicios.
- Escalabilidad: Se puede ampliar agregando más servidores.

Ejemplo práctico: Cliente y Servidor TCP

Servidor TCP:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class ServidorTCP {
    public static void main(String[] args) {
        try {
            ServerSocket servidor = new ServerSocket(8080); // Escucha en el puerto 8080
            System.out.println("Servidor esperando conexiones...");

            Socket socket = servidor.accept(); // Acepta una conexión
            System.out.println("Cliente conectado.");

            // Recibir datos del cliente
            BufferedReader entrada = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            String mensaje = entrada.readLine();
            System.out.println("Mensaje recibido: " + mensaje);

            // Enviar respuesta al cliente
            OutputStream output = socket.getOutputStream();
            PrintWriter escritor = new PrintWriter(output, true);
            escritor.println("¡Hola, cliente! Mensaje recibido.");

            // Cerrar conexión
            socket.close();
            servidor.close();
```

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

Cliente TCP:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.Socket;

public class ClienteTCP {
    public static void main(String[] args) {
        try {
            Socket socket = new Socket("127.0.0.1", 8080); // Conecta al servidor en localhost:8080

            // Enviar datos al servidor
            OutputStream output = socket.getOutputStream();
            PrintWriter escritor = new PrintWriter(output, true);
            escritor.println("¡Hola, servidor!");

            // Recibir respuesta del servidor
            BufferedReader entrada = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            String respuesta = entrada.readLine();
            System.out.println("Respuesta del servidor: " + respuesta);

            // Cerrar conexión
            socket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

3. Modelos P2P (peer-to-peer)

Definición

El modelo P2P (peer-to-peer) permite que todas las aplicaciones conectadas a la red actúen como clientes y servidores al mismo tiempo. No hay un servidor central; todos los nodos son iguales.

Características:

- Descentralización: No depende de un servidor central.
- Escalabilidad: Más nodos pueden unirse a la red fácilmente.
- Eficiencia: Los recursos se distribuyen entre los nodos.

Ejemplo práctico: Chat P2P

Este ejemplo simula un chat P2P básico donde un nodo actúa como servidor y otro como cliente.

Nodo P2P:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Scanner;


public class NodoP2P {
    public static void main(String[] args) {
        try {
            // Iniciar servidor en un hilo
            new Thread(() -> {
                try {
                    ServerSocket servidor = new ServerSocket(9090);
                    System.out.println("Nodo P2P esperando mensajes...");
                    Socket socket = servidor.accept();
                    BufferedReader entrada = new BufferedReader(new InputStreamReader(socket.getInputStream()));
                    String mensaje = entrada.readLine();
                    System.out.println("Mensaje recibido: " + mensaje);
                    socket.close();
                    servidor.close();
                } catch (Exception e) {
```

```
                e.printStackTrace();
            }
        }).start();

        // Conectar a otro nodo
        Socket socket = new Socket("127.0.0.1", 9090);
        OutputStream output = socket.getOutputStream();
        PrintWriter escritor = new PrintWriter(output, true);
        Scanner scanner = new Scanner(System.in);
        System.out.println("Escribe un mensaje para otro nodo:");
        String mensaje = scanner.nextLine();
        escritor.println(mensaje);
        socket.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

4. Modelos Híbridos

Definición

El modelo híbrido combina las características del cliente/servidor y P2P. Se utiliza un servidor central para gestionar la red y nodos descentralizados para el intercambio directo de datos.

Características:

- Centralización parcial: El servidor gestiona las conexiones iniciales, pero los nodos pueden comunicarse directamente.
- Escalabilidad: Admite más usuarios que un modelo cliente/servidor puro.
- Eficiencia: Reduce la carga en el servidor central.

Ejemplo práctico: Servidor de Conexión P2P

En este modelo, un servidor central conecta nodos P2P que luego intercambian datos directamente. Este ejemplo muestra solo la conexión inicial.

Servidor Central:

```
import java.net.ServerSocket;
import java.net.Socket;
```

```
public class ServidorCentral {
    public static void main(String[] args) {
        try {
            ServerSocket servidor = new ServerSocket(8080);
            System.out.println("Servidor central esperando nodos...");

            Socket nodo1 = servidor.accept();
            System.out.println("Nodo 1 conectado.");

            Socket nodo2 = servidor.accept();
            System.out.println("Nodo 2 conectado.");

            // Aquí podríamos intercambiar direcciones entre los nodos para que se conecten directamente
            servidor.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

En resumen

Los modelos de comunicación entre aplicaciones (cliente/servidor, P2P e híbrido) son fundamentales para la programación en red. Cada modelo tiene sus ventajas y desventajas dependiendo del caso de uso:

- Cliente/servidor es ideal para servicios centralizados.
- P2P es eficiente para aplicaciones descentralizadas como torrents.
- Modelos híbridos combinan lo mejor de ambos, siendo comunes en sistemas modernos.