Ciclo Formativo DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Módulo 9 Programación de servicios y procesos

Unidad Formativa 3 Sockets y servicios

INICIATIVA Y COORDINACIÓN
IFP Innovación en Formación Profesional

Supervisión editorial y metodológica:
Departamento de Producto de Planeta Formación

Supervisión técnica y pedagógica:
Departamento de Enseñanza de **IFP** Innovación en Formación Profesional

Módulo: Sockets y servicios
UF 3 / Desarrollo de Aplicaciones Multiplataforma

MÓDULO 9

Unidad Formativa 3

Programación de servicios y procesos

Sockets y servicios

Esquema de contenido

1. SOCKETS

   1.1 FUNDAMENTOS

   1.2. PROGRAMACIÓN CON SOCKETS

2. SERVICIOS

   2.1. CONCEPTO DE SERVICIO

   2.2. SERVICIOS EN RED

3. SERVICIOS DE NIVEL DE APLICACIÓN

1. SOCKETS

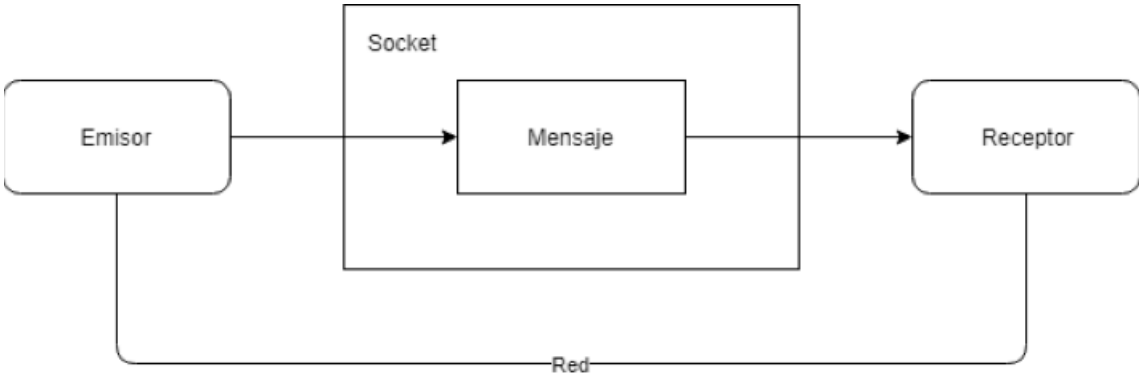
Los sockets son el mecanismo de comunicación básico fundamental que se usa para realizar transferencias de información entre aplicaciones, ya sea a través de redes internas (LAN) o Internet. Proporcionan una abstracción de la pila de protocolos, ofreciendo una interfaz de programación sencilla con la que las diferentes aplicaciones de un sistema distribuido pueden intercambiar mensajes.

¿Sabías que ...?

Los sockets aparecieron por primera vez la versión 4.2 del sistema operativo UNIX BSD, en el año 1981. Desde entonces se han vuelto el mecanismo básico estándar de comunicación entre procesos distribuidos en la mayoría de entornos, incluyendo todos los sistemas operativos UNIX y derivados (Linux, Mac OS X, Android, etc.) y Windows. En la actualidad, la mayoría de lenguajes de alto nivel, como C#, Java o Python, ofrecen interfaces de programación para usar sockets.

1.1 FUNDAMENTOS

Un socket (en inglés, literalmente, un "enchufe") representa el extremo de un canal de comunicación establecido entre un emisor y un receptor. Para establecer una comunicación entre dos aplicaciones, ambas deben crear sus respectivos sockets, y conectarlos entre sí. Una vez conectados, entre ambos sockets se crea una "tubería privada" a través de la red, que permite que las aplicaciones en los extremos envíen y reciban mensajes por ella. El procedimiento concreto por el cual se realizan estas operaciones depende del tipo de socket que se desee utilizar.



Para enviar mensajes por el canal de comunicaciones, las aplicaciones escriben (en inglés urite) en su socket. Para recibir mensajes leen (en inglés read) de su socket.

1.1.1. Direcciones y puertos

Para que las diferentes aplicaciones que forman parte de un sistema distribuido puedan enviarse mensajes, deben poder localizarse dentro de la red de comunicaciones a la que están conectadas. Esto es equivalente a cuando una persona desea enviar una carta a otra por correo ordinario: debe conocer la dirección del destinatario (ciudad, calle, número, código postal, etc.) e indicarlo en el sobre.

En las redes de comunicaciones que usan la pila de protocolos IP, las diferentes máquinas conectadas se distinguen por su dirección IP. Una dirección IP es un número que identifica de forma única a cada máquina de la red, y que sirve para comunicarse con ella. Es algo así como el "número de teléfono" de la má-

quina, dentro de la red. Actualmente existen dos versiones del protocolo IP, denominadas IPv4 (IP versión 4) e IPv6 (IP versión 6). IPv4 es la versión que se usa en Internet y en la mayoría de redes de área local. IPv6 es una versión más moderna y flexible de IP, pensada para sustituir a IPv4 en un futuro, cuando esta quede obsoleta definitivamente.

¿Sabías que ...?

Inicialmente, a cada máquina que se conectaba a Internet se le asignaba una dirección IP fija, que usaba para todas sus comunicaciones. Con la proliferación de los accesos a Internet en casas y oficinas, cada vez resultaba más difícil seguir este modelo, ya que el número de direcciones disponible está limitado. Para evitar este problema se pasó a un modelo de asignación dinámico, en el que la mayoría de máquinas (especialmente las de conexiones domésticas) reciben una dirección IP distinta cada vez que se conectan. Esto permite compartir la misma dirección entre varias máquinas, siempre y cuando no estén conectadas a la vez. Los proveedores de acceso a Internet disponen de una lista de direcciones IP, que asignan a sus clientes cuando se conectan. Cuando un cliente se desconecta, la IP queda libre, y el proveedor puede asignarla a otro usuario. Aun así, este modelo tiene sus limitaciones, ya que el número máximo de direcciones IP existentes sigue siendo fijo. Una de las ventajas que aporta IPv6 es que dispone de un rango de direcciones muchísimo más amplio que IPv4, permitiendo muchas más máquinas conectadas a Internet simultáneamente.

En IPv4 las direcciones IP están formadas por secuencias de 32 bits, llamadas palabras. Cada palabra está, a su vez, dividida en 4 grupos de 8 bits, llamados octetos. Cada octeto consta de 8 dígitos binarios (bits), y con él podemos representar números desde el 0 (en binario 00000000) hasta el 255 (en binario 11111111). Cuando escribimos las direcciones IPv4, representamos cada octeto separado por puntos, y escribimos su valor en decimal, para que sea más cómodo.

¿Sabías que ...?

Un servidor de DNS (Domain Name Service) es una máquina cuya función es traducir nombres simbólicos de máquinas por sus direcciones IP en la red. Una aplicación puede realizar una petición a un servidor DNS para resolver un nombre, por ejemplo www.google.es, y obtener la dirección IP asociada, por ejemplo 179.186.57.221.

Usando la dirección IP, una aplicación puede localizar la máquina en la que reside el receptor de su mensaje, pero en una misma máquina puede haber más de una aplicación funcionando y usando sockets para comunicarse con el exterior. Entonces, ¿cómo distinguir entre todas las aplicaciones que pueden estar ejecutando en la máquina destino?

La solución es utilizar puertos. Un puerto es un número que identifica a un socket dentro de una máquina. Cuando una aplicación crea un socket, esta debe especificar el número de puerto asociado a dicho socket. Podríamos ver la dirección IP .

EJemplo 1

Nuestra máquina tiene la siguiente dirección IPv4:

01100100110100110101110100O11001

Separada en octetos, quedaría de la siguiente forma:

O11OO1OO 11010011 01011101 OOO11OO1

Ahora escribimos Cada Octeto en decimal:

100 211 93 25

El resultado final es la dirección 100.211.93.25

¿Sabías que ...?

Un servidor de DNS (Domain Name Service) es una máquina cuya función es traducir nombres simbólicos de máquinas por sus direcciones IP en la red. Una aplicación puede realizar una petición a un servidor DNS para resolver un nombre, por ejemplo WWW.google.es, y obtener la dirección IP asociada, por ejemplo 173.194.34.223.

Usando la dirección IP, una aplicación puede localizar la máquina en la que reside el receptor de su mensaje, pero en una misma máquina puede haber más de una aplicación funcionando y usando sockets para comunicarse con el exterior. Entonces, ¿cómo distinguir entre todas las aplicaciones que pueden estar ejecutando en la máquina destino?

La solución es utilizar puertos. Un puerto es un número que identifica a un socket dentro de una máquina. Cuando una aplicación crea un socket, esta debe especificar el número de puerto asociado a dicho socket. Podríamos ver la dirección IP como el "nombre de la calle" a la que estamos enviando nuestra carta, y el puerto como el "número de vivienda". Si no especificamos ambos, nuestro mensaje no llegará a su destinatario. Los números de puerto se representan con 16 dígitos binarios (bits), y pueden, por tanto, tomar valores entre 0 (16 ceros en binario) y 65.535 (16 unos en binario).

Nunca puede haber más de un socket asignado a un mismo puerto en una máquina. Cuando una aplicación crea un socket y lo asigna a un puerto determinado, normalmente se dice que ese socket está escuchando (en inglés, listening) por ese puerto.

1.1.2. Tipos de sockets

Existen dos tipos básicos de sockets: sockets stream y sockets datagram, dependiendo de su funcionalidad y del protocolo de nivel de transporte que utilizan.

1.1.2.1 Sockets stream

Los sockets stream son orientados a conexión y, cuando se utilizan sobre la pila IP, hacen uso del protocolo de transporte TCP Son fiables (los mensajes que se envían llegan a su destino) y aseguran el orden de entrega correcto. Un socket stream se utiliza para comunicarse siempre con el mismo receptor, manteniendo el canal de comunicación abierto entre ambas partes hasta que se termina la conexión.

Cuando se establece una conexión usando sockets stream, se debe seguir una secuencia de pasos determinada. En esta secuencia uno de los elementos de la comunicación debe ejercer el papel de proceso servidor y otra, el de proceso cliente. El proceso servidor es aquel que crea el socket en primer lugar y espera a que el cliente se conecte. Cuando el proceso cliente desea iniciar la comunicación, crea su socket y lo conecta al servidor, creando el canal de comunicación. En cualquier momento, cualquiera de los dos procesos (cliente o servidor) puede cerrar su socket, destruyendo el canal y terminando así la comunicación.

Proceso cliente

Para usar sockets stream, un proceso cliente debe seguir los siguientes pasos:

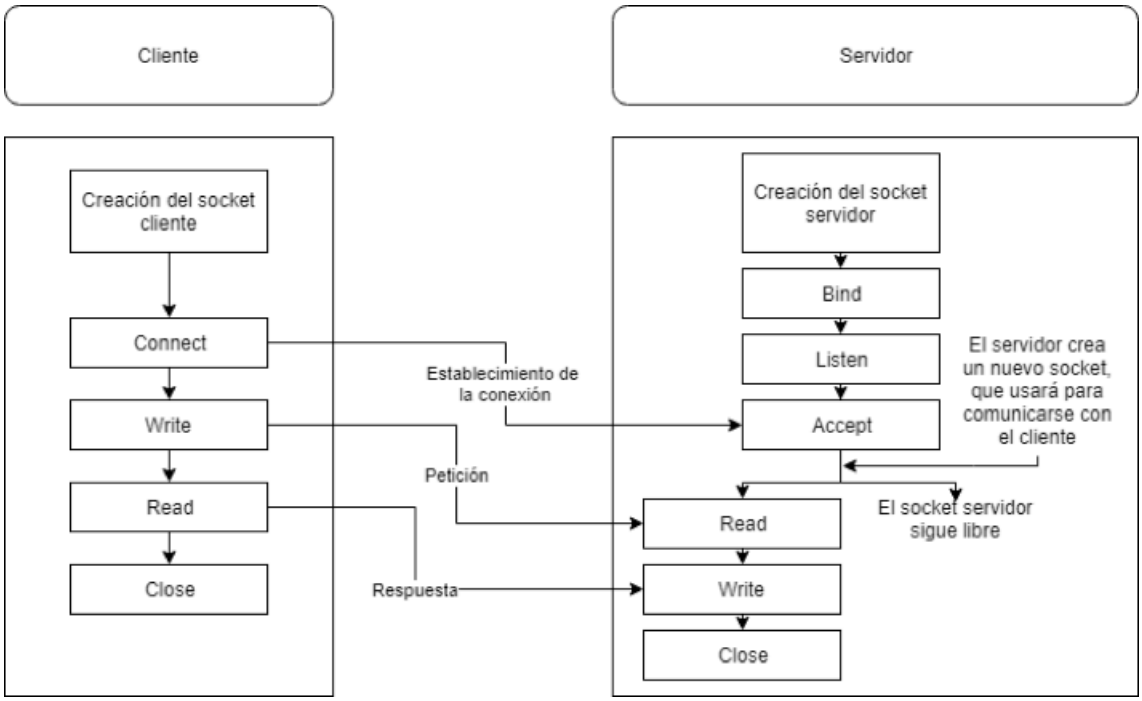
1. **Creación del socket:** Esto crea un socket y le asigna un puerto. Normalmente el número concreto de puerto que usa el socket de un proceso cliente no es importante, por lo que se suele dejar que el sistema operativo lo asigne automáticamente (normalmente le da el primero que esté libre). A este socket se le llama "socket cliente".
2. **Conexión del socket (en inglés connect):** En este paso se localiza el socket del proceso servidor, y se crea el canal de comunicación que une a ambos. Para poder establecer la conexión el proceso cliente debe conocer la dirección IP del proceso servidor y el puerto por el que este está escuchando.
3. **Envío y recepción de mensajes:** Una vez establecida la conexión, el proceso cliente puede enviar y recibir mensajes mediante operaciones de escritura y lectura (write y read) sobre su socket cliente.
4. **Cierra de la conexión (en inglés, close):** Si desea terminar la comunicación, el proceso cliente puede cerrar el socket cliente.

Proceso servidor

En el caso de un proceso servidor, los pasos que se deben seguir para poder comunicarse con el proceso cliente son los siguientes:

1. **Creación del socket:** Esto crea un socket, de forma similar al caso del proceso cliente. A este socket se le llama "socket servidor".
2. **Asignación de dirección y puerto (en inglés bind):** En el caso del proceso servidores importante que la dirección IP 4 y el número de puerto del socket estén claramente especificados. De lo contrario, el proceso cliente no será capaz de localizar al servidor. La operación bind asigna una dirección IP y un número de puerto concreto al socket servidor. Lógicamente, la dirección IP asignada debe ser la de la máquina donde se encuentra el proceso servidor.
3. **Escucha(en inglés listen):** Una vez se ha creado el socket y se le ha asignado un número de puerto, se debe configurar para que escuche por dicho puerto. La operación listen hace que el socket servidor quede preparado para aceptar conexiones por parte del proceso cliente.
4. **Aceptación de conexiones (en inglés accept):** Una vez el socket servidor está listo para aceptar conexiones, el proceso servidor debe esperar hasta que un proceso cliente se conecte (con la operación connect). La operación accept bloquea al proceso servidor esperando por una conexión por parte del proceso cliente. Cuando llega una petición de conexión, se crea un nuevo socket dentro del proceso servidor. Este nuevo socket es que el que queda conectado con el socket del proceso cliente, estableciendo un canal de comunicación estable entre ambos. El nuevo socket se utilizará para enviar y recibir mensajes con el proceso cliente. El socket servidor queda libre, por lo que puede seguir escuchando, a la espera de nuevas conexiones.
5. **Envío y recepción de mensajes:** Una vez establecida la conexión, el proceso servidor puede enviar y recibir mensajes mediante operaciones de escritura y lectura (write y read) sobre su nuevo socket. No se usa el socket servidor para realizar esta tarea, ya que ha quedado fuera de la conexión.

6. **Cierre de la conexión (en inglés close):** Si desea terminar la comunicación, el proceso servidor puede cerrar el nuevo socket. El socket servidor sigue estando disponible para nuevas conexiones.



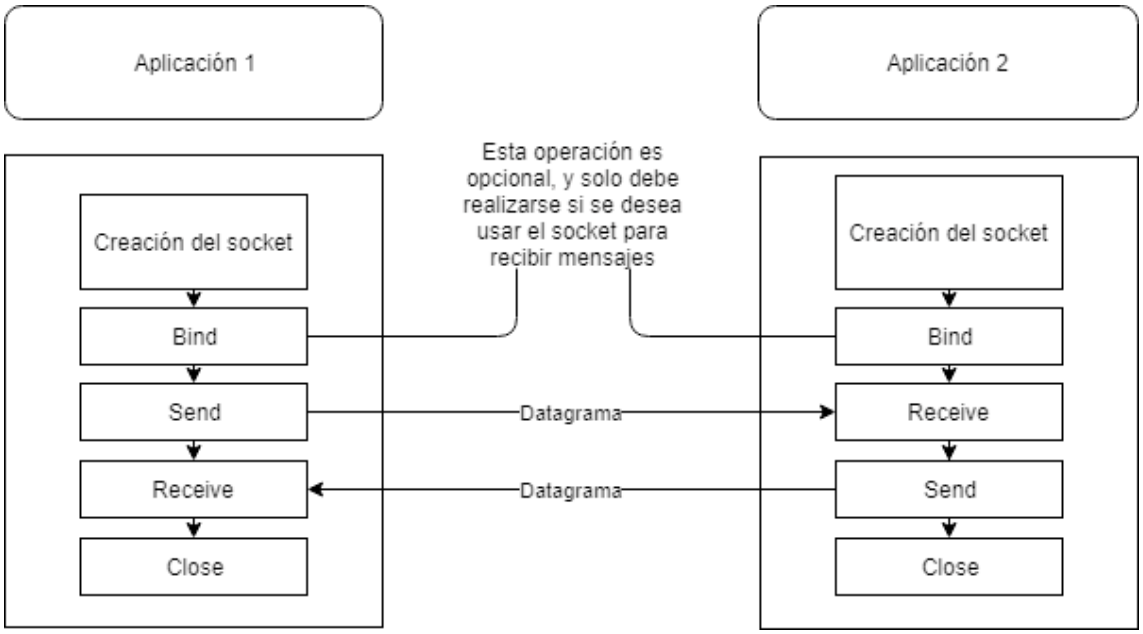
1.1.2.2. Sockets datagram

Los sockets datagram no son orientados a conexión. Pueden usarse para enviar mensajes (llamados "datagramas") a multitud de receptores, ya que usan un canal temporal para cada envío. No son fiables ni aseguran orden de entrega correcto. Cuando se usan sobre la pila IP hacen uso del protocolo de transporte UDP.

Cuando se usan sockets datagram no existe diferencia entre proceso servidor y proceso cliente. Todas las aplicaciones que usan sockets datagram realizan los siguientes pasos para enviar mensajes:

1. **Creación del socket:** Esto crea un socket, de forma similar al caso de los sockets stream.
2. **Asignación de dirección y puerto (en inglés bind):** En el caso de que se desee usar el socket para recibir mensajes, 4 es importante que la dirección IP y el número de puerto del socket estén claramente especificados. De lo contrario los emisores no serán capaces de localizar al receptor. Al igual que en el caso de los sockets stream, la operación bind asigna una dirección IP y un número de puerto concreto al socket. La dirección IP asignada debe ser la de la máquina donde se encuentra la aplicación.
3. **Envío y recepción de mensajes:** Una vez creado el socket, se puede usar para enviar y recibir datagramas. En el 3 caso de los sockets datagram existen dos operaciones especiales, denominadas "enviar" (en inglés send) y "recibir" (en inglés receive). La operación send necesita que se le especifique una dirección IP y un puerto, que usará como datos del destinatario del datagrama.

4. **Cierre de la conexión (en inglés close):** Si no se desea usar más el socket, se puede cerrar usando la operación close.



¿Sabías que ...?

Los sockets datagram no son orientados a conexión, por lo que se puede usar un mismo socket para enviar mensajes a distintos receptores. Simplemente es necesario especificar la dirección y puerto de destino en cada operación send. No es necesario realizar la operación Close para cerrar el canal, como en los sockets stream, porque no hay un canal permanente creado entre emisor y receptor. La operación close se realiza solamente cuando ya no se desea seguir usando el socket.

1.2. PROGRAMACIÓN CON SOCKETS

En la mayoría de lenguajes de programación de alto nivel existen bibliotecas para crear, destruir y operar con sockets sobre la pila de protocolos IP. En Java existen tres clases principales que permiten la comunicación por sockets:

- java.net.Socket, para la creación de sockets stream cliente.
- java.net.ServerSocket, para la creación de sockets stream servidor.
- java.net.DatagramSocket, para la creación de sockets datagram.

En las siguientes secciones veremos cómo se usan en los escenarios típicos de comunicación.

1.2.1. La clase Socket

La clase Socket (java.net.Socket) se utiliza para crear y operar con sockets stream clientes. Sus métodos más importantes se describen en la siguiente tabla:

Método	Tipo de retorno	Descripción
Socket()	Socket	Constructor básico de la clase. Sirve para crear sockets stream clientes.
connect(SocketAddress addr)	void	Establece la conexión con la dirección y puerto destino.
getInputStream()	InputStream	Obtiene un objeto de clase Inputstream que se usa para realizar operaciones de lectura (read).
getOutputStream()	OutputStream	Obtiene un objeto de clase OutputStream que se usa para realizar operaciones de escritura (Write).
close()	void	Cierra el Socket.

En el Ejemplo 2 se muestra un programa en Java sencillo que opera con un socketstream cliente. En él se crea el socket usando la clase Socket, se conecta a un socket stream servidor que se encuentre en la misma máquina (localhost) y escuchando por el puerto 5555 y se le envía un mensaje con el texto "mensaje desde el cliente". Una vez realizadas estas operaciones, se cierra el socket y el programa termina.

Ejemplo 2

Proceso cliente usando sockets stream:

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import java.net.Socket;

public class ClienteSocketStream {
  public static void main (String[] args) {
    try {
      System.out.println("Creando socket cliente");
      Socket clientSocket = new Socket();
      System.out.println("Estableciendo la conexión");
      InetSocketAddress addr = new InetSocketAddress("localhost", 5555);
      clientSocket.connect(addr);
      InputStream is = clientSocket.getInputStream();
      OutputStream os = clientSocket.getOutputStream();
      System.out.println("Mensaje enviado. Cerrando el socket cliente ");
      clientSocket.close();
      System.out.println("Terminado");
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

Como se puede ver en el Ejemplo 2, para indicar la dirección IP y el número de puerto del socketstream servidor al que se desea conectar, el método connect() hace uso de un objeto de clase java.net. InetSocketAddress. Esta clase se utiliza en Java para representar direcciones de sockets, es decir, pares de dirección IP y número de puerto. Tal y como se hace en el ejemplo, la dirección IP se puede sustituir por un nombre de máquina (en el ejemplo, localhost), en cuyo caso se tratará de resolver dicho nombre y obtener su dirección IP asociada.

1.2.2. La clase ServerSocket

La clase ServerSocket (java.net.ServerSocket) se utiliza para crear y operar con sockets stream servidor. Sus métodos más importantes se describen en la siguiente tabla:

Método	Tipo de retorno	Descripción
ServerSocket()	Socket	Constructor básico de la clase. Sirve para crear sockets stream servidores
ServerSocket (String Socket host, int port)	Socket	Constructor alternativo de la clase. Se le pasan como argumentos la dirección IP y el puerto que se desean asignar al socket. Este método realiza las operaciones de creación del socket y bind directamente
bind (SocketAddress bindpoint)	void	Asigna al socket una dirección y número de puerto determinado (operación bind)
accept()	Socket	Escucha por el socket servidor, esperando conexiones por parte de clientes (operación accept). Cuando llega una conexión devuelve un nuevo objeto de clase Socket, Conectado al client
close()	void	Cierra el socket

La clase ServerSocket no tiene un método independiente para realizar la operación listen. Esto no quiere decir que los sockets stream implementados por esta clase no realicen dicha operación. La operación listen se realiza de forma conjunta a accept, cuando se ejecuta el método accept().

El Ejemplo 3 muestra un programa sencillo en Java que opera con un socket stream servidor. En él se crea el socket usando la clase ServerSocket, se le asigna la dirección IP de la misma máquina (localhost) y el puerto 5555 y se realiza la operación accept, a la espera de conexiones por parte de sockets clientes. Cuando llega una conexión, el método accept() establece el canal, creando un nuevo socket (de clase Socket) y devolviéndolo. El proceso servidor utiliza este nuevo socket para recibir un mensaje (de tamaño 25 bytes) y lo imprime por su salida estándar, convertido en un String. Una vez realizadas estas operaciones se cierra el nuevo socket y el socket servidor y el programa termina. Este programa está pensado para operar conjuntamente con el que figura en el Ejemplo 2.

Ejemplo 3

Proceso servidor usado sockets stream:

```
import java.io.IOException;
import java.io.InputStream;
```

```
import java.io.OutputStream;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.net.ServerSocket;


public class ServerSocketStream {
  public static void main (String[] args) {
    try {
      System.out.println("Creando socket Servidor");
      ServerSocket serverSocket = new ServerSocket();
      System.out.println("Estableciendo el bind");
      InetSocketAddress addr = new InetSocketAddress("localhost", 5555);
      serverSocket.bind(addr);
      serverSocket.accept();
      System.out.println("Conexion recibida");
      InputStream is = new Socket.getInputStream();
      OutputStream os = newSocket.getOutputStream();
      Byte[] mensaje = new byte(25);
      is.read(mensaje);
      System.out.println("Mensaje recibido: " + new String(mensaje));
      System.out.println("Cerrando el nuevo socket");
      new Socket.close();
      System.out.println("Cerrando el socket servidor");
      serverSocket.close();
      System.out.println("Terminado");
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

Como se puede ver en los Ejemplos 2 y 3, los sockets stream utilizan métodos estándar read() y write() de objetos de las clases InputStream y OutputStream para enviar y recibir mensajes.

1.2.3. La clase DatagramSocket

La clase DatagramSocket(java.net.DatagramSocket) se utiliza para crear y operar con sockets datagram. Sus métodos más importantes se describen en la siguiente tabla:

Método	Tipo de retorno	Descripción
DatagramSocket()	DatagramSocket	Constructor básico de la clase. Sirve para crear sockets datagram
DatagramSocket (SocketAddress bindaddir)	DatagramSocket	Constructor de la clase con operación bind incluida. Sirve para crear sockets datagrama asociados a una dirección y puerto especificado
send(DatagramPacket p)	void	Envía un datagrama
receive(DatagramPacket p)	void	Recibe un datagrama
close()	void	Cierra el Socket

Ejemplo 4

Proceso que envía un mensaje usando sockets datagram:

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

public class Emisor Datagram {
  public static void main(String[] args) {
    try {
      System.out.println("Creando socket datagram");
      DatagramSocket datagramSocket = new DatagramSocket();
      System.out.println("Enviando mensaje");
      String mensaje = "mensaje desde el emisor";
      InetAddress addr = InetAddress.getByName("localhost");
       DatagramPacket datagrama = new DatagramPacket(mensaje.getBytes(), mensaje.getBytes().
length,
        addr, 5555);
      datagramSocket.send(datagrama);
      System.out.println("Mensaje enviado");
      System.out.println("Cerrando el socket datagrama");
      datagramSocket.close();
      System.out.println("Terminado");
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

El Ejemplo 4 muestra un programa sencillo en Java que opera con un socket datagram para enviar mensajes. En él se crea el socket usando la clase DatagramSocket. Posteriormente se crea un datagrama con el mensaje "mensaje desde el emisor" y se fija su destinatario con la dirección localhost y el puerto 5555. Finalmente, se envía el datagrama usando el método send() y se cierra el socket.

Ejemplo 5

Proceso que recibe un mensaje y luego lo envía usando sockets datagram:

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.InetSocketAddress;

public class Receive Send {
  public static void main(String[] args) {
    try {
      System.out.println("Creando socket datagrama");
      InetSocketAddress addr = new InetSocketAddress("localhost", 5555);
      DatagramSocket datagramSocket = new DatagramSocket(addr);
      System.out.println("Recibiendo mensaje");
      Byte[] mensaje = new byte[25];
      DatagramPacket datagramal = new DatagramPacket(mensaje, 25);
      datagramSocket.receive(datagramal);
      System.out.println("Mensaje recibido: " + new String(mensaje));
      System.out.println("Enviando mensaje");
      InetAddress addr2 = InetAddress.getByName("localhost");
      DatagramPacket datagrama2 = new DatagramPacket(mensaje, mensaje. length, addr2, 5556);
      datagramSocket.send(datagrama2);
      System.out.println("Mensaje enviado");
      System.out.println("Cerrando el socket datagrama");
      datagramSocket.close();
      System.out.println("Terminado");
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

El Ejemplo 5 muestra un programa sencillo en Java que opera con un socket datagram para enviar y recibir mensajes. En él se crea el socket usando la clase DatagramSocket y se le asigna la dirección localhost y el puerto 5555. Posteriormente, se espera la recepción de un mensaje por dicho socket (de hasta 25 bytes) y, una vez recibido, se imprime por la salida estándar en forma de String. A continuación se crea

un nuevo datagrama con el mensaje recibido y se fija su destinatario con la dirección localhost y el puerto 5556. Finalmente, se envía el datagrama usando el método send() y se cierra el socket. Este programa está pensado para operar conjuntamente con el que figura en el Ejemplo 4.

Como se puede ver en los Ejemplos 4 y 5, el manejo del mensaje y la dirección y puerto del destinatario se realiza en los sockets datagram de forma bastante distinta al caso de los sockets stream. En primer lugar, en Java los datagramas se representan utilizando objetos de la clase DatagramPacket (java.net. DatagramPacket). Un objeto de esta clase contiene un datagrama de un tamaño fijo, asociando a un destinatario o emisor (dirección y puerto) concreto. El método send() de la clase DatagramSocket permite enviar datagramas representados por objetos de la clase DatagramPacket, siempre y cuando estos hayan sido correctamente inicializados con su destinatario, tal y como se ve en el Ejemplo 4. A su vez, el método receive() de la clase DatagramSocket permite recibir datagramas y almacenarlos en objetos de la clase DatagramPacket, tal y como se ve en el Ejemplo 5. Esta operación rellena los valores del objeto pasado como parámetro, almacenando en él el datagrama recibido y su remitente (dirección y puerto). Además, el constructor de la clase DatagramPacket utiliza un objeto de la clase InetAddress (java.net. InetAddress) para representar la dirección IP del emisor o destinatario del datagrama. Esta clase representa direcciones IP de forma independiente. Como se puede ver en ambos ejemplos, el número de puerto se debe indicar al constructor de la clase DatagramPacket como un parámetro separado.

Actividad

1. Busca información sobre los tipos de sockets que se utilizan en las aplicaciones de Internet más comunes. ¿Qué tipo de sockets se suelen usar para navegar por la Web? ¿Y para resolver nombres de máquinas (DNS)?
2. El procedimiento de abstracción de las comunicaciones que usan los sockets (simular una "tubería" sobre la que se "lee" y se "escribe") se parece a otros mecanismos que se usan habitualmente en programación ¿A cuáles te recuerda?
3. Busca información sobre cómo se gestionan las direcciones IPv4. ¿Podemos asignar cualquier dirección a una máquina? ¿Qué son las direcciones locales? ¿Qué es una dirección de broadcast?
4. Copia los ejemplos anteriores que usan sockets stream y modifícalos para que el mensaje enviado sea "Mensaje extendido desde el programa cliente" y el puerto por el que escuche el servidor sea el 6666.
5. Copia el ejemplo anterior que usa sockets datagram y modifícalo para que el mensaje enviado sea "Mensaje extendido desde el programa cliente" y el nombre o IP de la máquina destino se pase como argumento del programa.

2. SERVICIOS

En el capítulo anterior se han visto los conceptos fundamentales de la computación distribuida y los mecanismos básicos de comunicación en red. Dentro de estos conceptos fundamentales, se ha desarrollado el modelo de comunicaciones cliente/servidor, el más usado en la actualidad en la computación distribuida. Al definir este modelo, se dice que uno de sus elementos clave es el servidor, una aplicación que proporciona servicios a uno o más clientes. Pero ¿qué es exactamente un servicio? ¿Qué características tiene? A lo largo de este capítulo se profundizará en este y otros conceptos relacionados, y se aprenderá a desarrollar aplicaciones complejas que proporcionen servicios.

2.1. CONCEPTO DE SERVICIO

Desde un punto de vista básico, todo sistema presenta dos partes fundamentales: estructura y función. La estructura está formada por aquellos componentes que conforman el sistema, es decir, las piezas que unidas lo forman. En informática en general, y en los sistemas distribuidos en particular, la estructura del sistema serán aquellos componentes hardware y software que, conectados entre sí, forman el ordenador, sistema distribuido, o lo que estemos analizando. La función, por otro lado, es aquello para lo que está pensado el sistema, es decir, para qué sirve y/o se usa.

Ejemplo 6

La descomposición conceptual en estructura y función se puede aplicara cualquier sistema tecnológico, ya sea en el campo de la informática o fuera de ella. Un electrodoméstico, una lavadora por ejemplo, es un sistema que consta de estructura y función, como cualquier otro. En este ejemplo, la estructura de la lavadora son la piezas mecánicas y electrónicas que lo forman (carcasa, tambor, puerta, controlador electrónico, motor interno...). La función de la lavadora es, obviamente, lavar la ropa.

Ejemplo 7

Determinar la estructura y función de un sistema distribuido es igual de sencillo que determinar. la de cualquier otro sistema. Si pensamos en una aplicación de mensajería instantánea, como WhatSApp, por ejemplo, la estructura estaría formada por las aplicaciones cliente y servidor, es decir, el software que lo compone. La función de este sistema es, sencillamente, permitir que los usuarios puedan comunicarse enviando mensajes de texto, imágenes, vídeos, etc.

Todo sistema tiene, por tanto, una estructura y una función. Como se puede observar, la estructura es algo concreto, relacionado con los componentes hardware y software que forman el sistema. La función, en cambio, suele ser una serie de conceptos abstractos, que explican"lo que hace"el sistema, pero no "cómo lo hace". Para que un sistema realice su función, se deben especificar los mecanismos concretos que este proporciona a sus usuarios. A este conjunto de mecanismos concretos que hacen posible el acceso a la función del sistema se le denomina servicio. Un sistema puede proporcionar uno o más servicios, en función de cuál sea su función.

Además, normalmente, este conjunto de mecanismos, al que se llama "servicio", se especifica de manera concreta e impone restricciones específicas de utilización. Cualquier usuario del sistema ha de seguir una

serie de procedimientos determinados a la hora de acceder al sistema y al usarlo. Esta serie de procedimientos se denomina interfaz del servicio y es el punto de contacto entre el sistema y el usuario.

Ejemplo 8

Continuando con el ejemplo de la lavadora, el servicio proporcionado por esta se define como el conjunto de mecanismos que esta realiza para llevar a cabo su función. En este caso podemos hablar de operaciones como lavado con agua caliente, lavado en frío, programa para tejidos sintéticos, etc. La interfaz del servicio son los componentes con los que interactúa el usuario, como la puerta de carga, el cuadro de mandos, etc. Estos condicionan los procedimientos concretos que el usuario debe seguir para obtener el servicio. Para lavar en frío, por ejemplo, el usuario debe abrir la puerta, cargar la ropa en el tambor, cerrar la puerta, llenar la cubeta del detergente, ajustar el programa adecuado en el panel y pulsar el botón de encendido.

Ejemplo 9

En el ejemplo de la aplicación de mensajería instantánea (Ejemplo 4.2), los servicios son las funciones específicas de la aplicación, como buscar a un amigo, enviarle un mensaje, enviarle una foto, etc. La interfaz del servicio es la propia interfaz de la aplicación cliente, que marca la manera concreta en la que se realizan dichas operaciones. Para enviar una foto, por ejemplo, estas operaciones podrían ser seleccionar al destinatario en la lista de contactos, pulsar el botón de Enviar foto, seleccionar una fotografía de la galería de imágenes y pulsar el botón de Enviar.

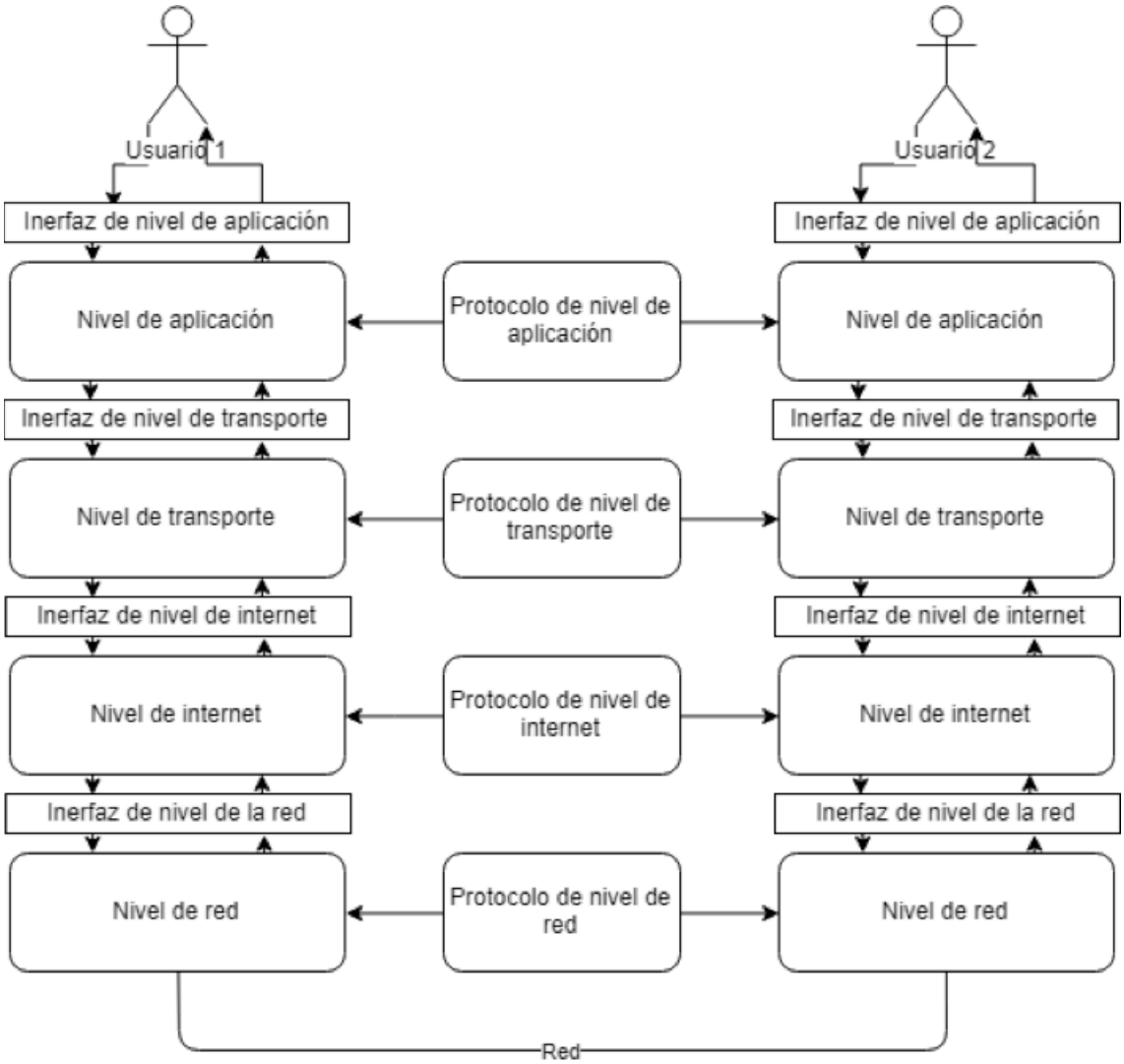
2.2. SERVICIOS EN RED

Durante las comunicaciones entre los diferentes elementos de una aplicación distribuida, se hace uso de multitud de servicios distintos, que cooperan entre sí para conseguir el paso de mensajes entre emisores y receptores. La pila de protocolos IP es, en efecto, un conjunto de sistemas independientes, pero montados unos sobre otros para realizar una tarea compleja. Cada nivel de la jerarquía (red, Internet, transporte y aplicación) proporciona un servicio específico, tal y como se vio en el capítulo anterior, y ofrece una interfaz de servicio al nivel superior, a través de la cual interactúa con este. Además, para que las comunicaciones puedan llevarse a cabo, cada nivel de la pila dispone de su propio protocolo de comunicaciones, que gobierna la interacción a ese nivel con los demás elementos del sistema distribuido.

Se puede considerar, por tanto, un servicio en red como cualquier servicio que se ubique en cualquier nivel de la pila. Las tecnologías de comunicaciones del nivel de red son servicios, los mecanismos de encaminamiento del nivel de Internet son servicios, etc.

¿Sabías que ...?

El sistema de sockets estudiado en el capítulo anterior proporciona servicios del nivel de transporte. Su función es hacer llegar mensajes entre un emisor y un receptor, ya sea por un modelo orientado a conexión (sockets stream) o por medio de datagramas (sockets datagram). La interfaz del servicio está formada, en este caso, por las bibliotecas de programación que permiten hacer uso de los sockets dentro de nuestros programas. Los protocolos TCP y UDP son ejemplos de protocolos de nivel de transporte asociados a este servicio.


Usuario 1
Usuario 2
Inerfaz de nivel de aplicación
Inerfaz de nivel de aplicación
Nivel de aplicación
Protocolo de nivel de aplicación
Nivel de aplicación
Inerfaz de nivel de transporte
Inerfaz de nivel de transporte
Nivel de transporte
Protocolo de nivel de transporte
Nivel de transporte
Inerfaz de nivel de internet
Inerfaz de nivel de internet
Nivel de internet
Protocolo de nivel de internet
Nivel de internet
Inerfaz de nivel de la red
Inerfaz de nivel de la red
Nivel de red
Protocolo de nivel de red
Nivel de red
Red

3. SERVICIOS DE NIVEL DE APLICACIÓN

El nivel más alto de la pila IP lo componen las aplicaciones que forman el sistema distribuido. Estas aplicaciones, al igual que en el resto de niveles, ofrecen una interfaz de servicio para que los usuarios las usen, y disponen de un protocolo de nivel de aplicación que gobierna las comunicaciones entre ellas. La mayoría de aplicaciones distribuidas más comunes se ubican en este nivel, como las páginas web, el correo electrónico o los juegos on-line.

Se define un protocolo de nivel de aplicación como el conjunto de reglas que gobiernan la interacción entre los diferentes elementos de una aplicación distribuida. A la hora de desarrollar un servicio distribuido, definir este protocolo es uno de los pasos fundamentales. En una aplicación cliente/servidor este protocolo especifica cómo se realiza la interacción entre el servidor y los clientes. A su vez, los diferentes elementos de una aplicación cliente/ servidor (los clientes y el servidor) se pueden ver como sistemas independientes, con su estructura y función. En este sentido, el servidor es la pieza clave, ya que es la proporciona el servicio deseado a los clientes. Su estructura estará formada por los componentes software con los que está programado. Su función será aquella que realiza para los clientes, y su servicio el procedimiento mediante el cual la realiza. El protocolo de nivel de aplicación define cómo se interactúa con el cliente y es, por tanto, la interfaz de servicio del servidor.

¿Sabías que ...?

Un servidor web es una aplicación que proporciona páginas web a los clientes que lo solicitan, normalmente navegadores web como Firefox o Internet Explorer. Si se considera el servidor web como un sistema, el servicio proporcionado por este es el acceso a la página o páginas que contiene. La interfaz de servicio del servidor web está gobernada por el protocolo de nivel de aplicación que se usa en Internet para el tráfico web, denominado HTTP (Hipertext Transfer Protocol). Este protocolo se estudiará (entre otros) en este capítulo.

Actividad

1. Identifica estructura, función, servicio e interfaz de servicio en los siguientes ejemplos:
   1. Un procesador de textos (como Word o LibreOffice Writer).
   2. Un autobús urbano.
   3. Una plataforma de juegos online, como Steam o XBOX Live.
   4. Una cámara fotográfica.

Índice

1. SOCKETS 5
1.1 FUNDAMENTOS 5
1.1.1. Direcciones y puertos 5
1.1.2. Tipos de sockets 7
1.2. PROGRAMACIÓN CON SOCKETS 10
1.2.1. La clase Socket 10
1.2.2. La clase ServerSocket 12
1.2.3. La clase DatagramSocket 14
2. SERVICIOS 17
2.1. CONCEPTO DE SERVICIO 17
2.2. SERVICIOS EN RED 18
3. SERVICIOS DE NIVEL DE APLICACIÓN 20
Índice 21