UF 3.2
Programación De Comunicaciones En Red

SOCKETS

Universidad Europea de Madrid

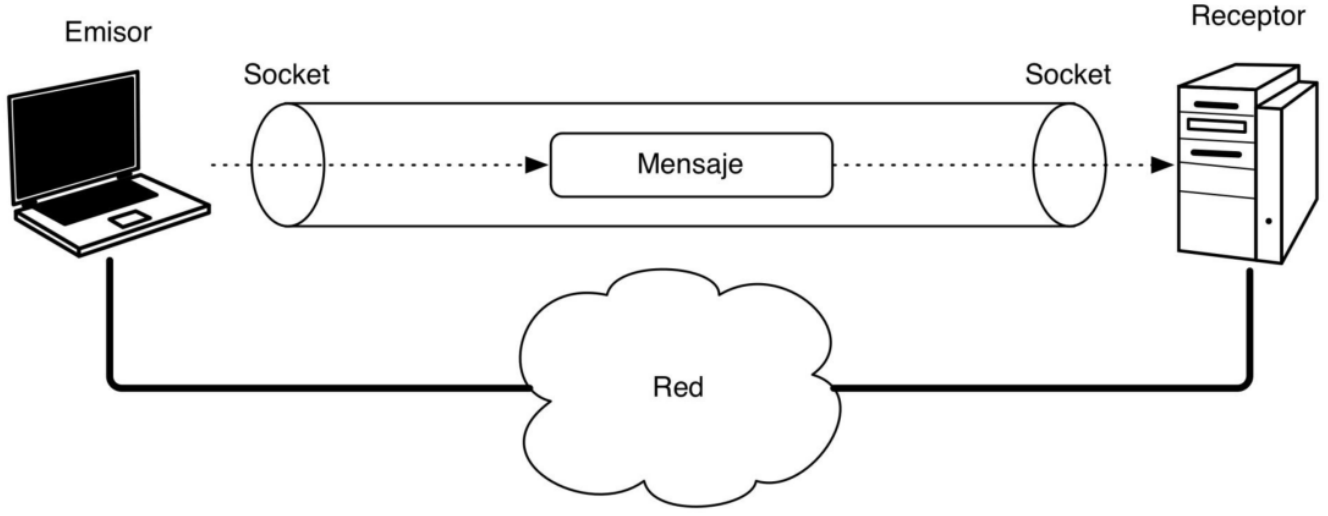
LAUREATE INTERNATIONAL UNIVERSITIES

CONTENIDOS

1. Fundamentos
2. Mensajes, Direcciones y Puertos
3. Tipos de Sockets
   - Sockets Stream
   - Sockets Datagram
4. Programación con Sockets
   - Clase Socket
   - Clase ServerSocket
   - Clase DatagramSocket

FUNDAMENTOS

- Los sockets son el mecanismo de comunicación básico fundamental que se usa para realizar transferencias de información entre aplicaciones.
- Proporcionan una abstracción de la pila de protocolos, ofreciendo una interfaz de programación sencilla con la que diferentes aplicaciones de un sistema distribuido pueden intercambiar mensajes.
- 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 hecho, se crea una “tubería privada”



MENSAJES, DIRECCIONES Y PUERTOS

Mensajes

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

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

Puertos

- 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, ésta debe especificar el número de puerto al que está asociado (16 bits: [0 – 65.535])

TIPOS DE SOCKETS

Sockets Stream

- Son **orientados a conexión** y cuando operan sobre IP, emplean TCP.
- Son fiables, es decir, los mensajes se envían y llegan a su destino y además, 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.
- Secuencia de pasos a realizar:

Proceso Cliente

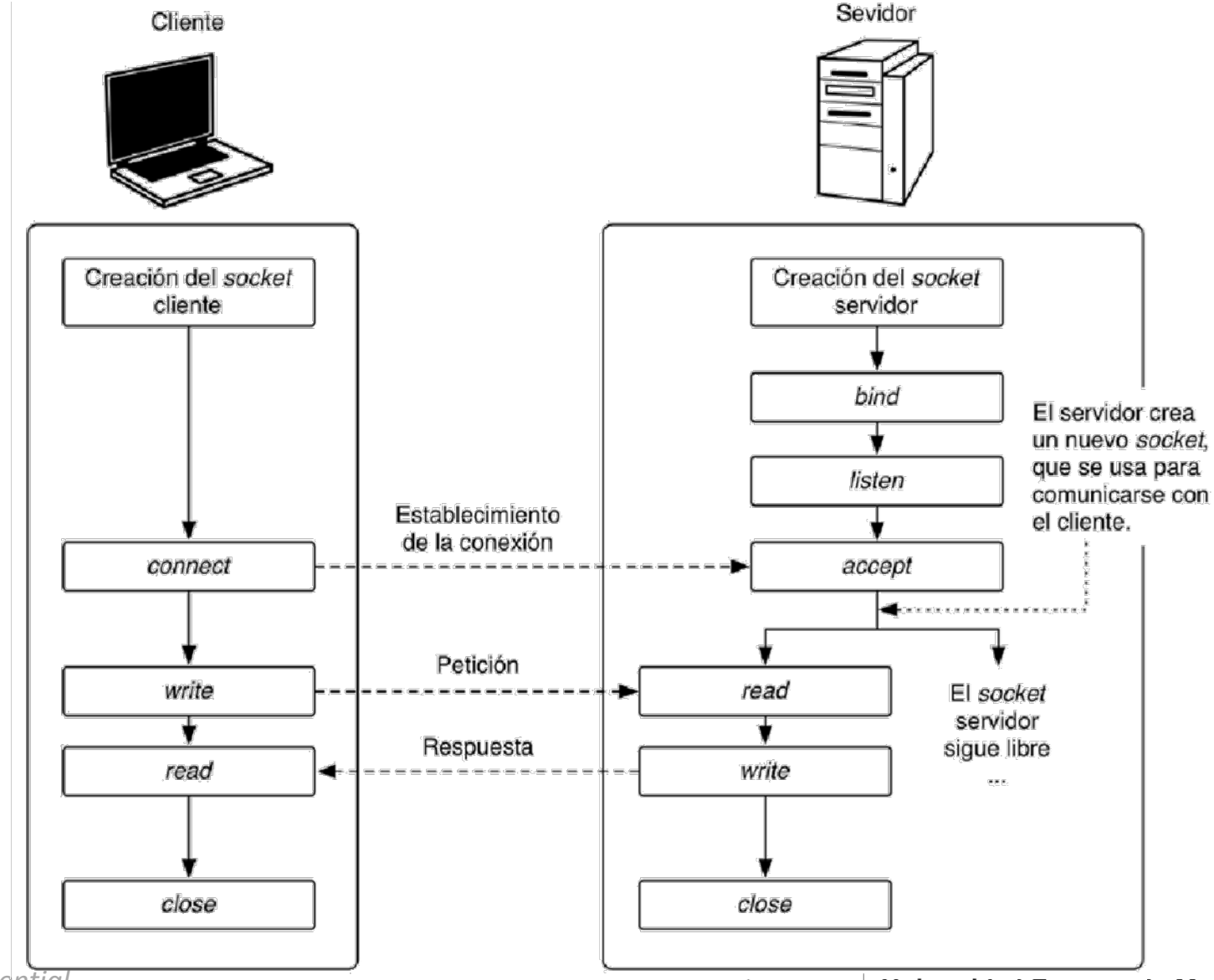
1. Creación del socket.
2. Conexión del socket (connect).
3. Envío y recepción de mensajes.
4. Cierre de la conexión (close).

Proceso Servidor

1. Creación del socket.
2. Asignación de dirección y puerto (bind).
3. Escucha (listen).
4. Aceptación de conexiones (accept). Esta operación implica la creación de un nuevo socket, que se usa para comunicarse con el cliente que se ha conectado.
5. Envío y recepción de mensajes.
6. Cierre de la conexión (close).

TIPOS DE SOCKETS

Sockets Stream



Utilización de hilos en aplicaciones cliente servidor

- La mayor parte de servidores están pensados para atender a múltiples clientes simultáneamente. Esto quiere decir, que si al servidor le llega una nueva petición mientras está atendiendo a un cliente, se deben cumplir las siguientes condiciones:
  - La nueva petición no debe interferir con la que está en curso.
  - La nueva petición debe ser atendida lo antes posible.
- Para cumplir dichas condiciones, la mayor parte de los servidores optan por implementar un enfoque **multihilo**.
- Los **sockets stream** son especialmente adecuados para implementar servidores multihilo.
  - Cada vez que el socket servidor recibe un nuevo intento de conexión, la operación *accept* crea un nuevo socket.
  - El procedimiento típico en este caso es arrancar entonces un nuevo hilo.
  - El hilo principal y el socket servidor permanecen libres, realizando la operación *accept* a la espera de nuevas conexiones.

TIPOS DE SOCKETS

Sockets Datagram

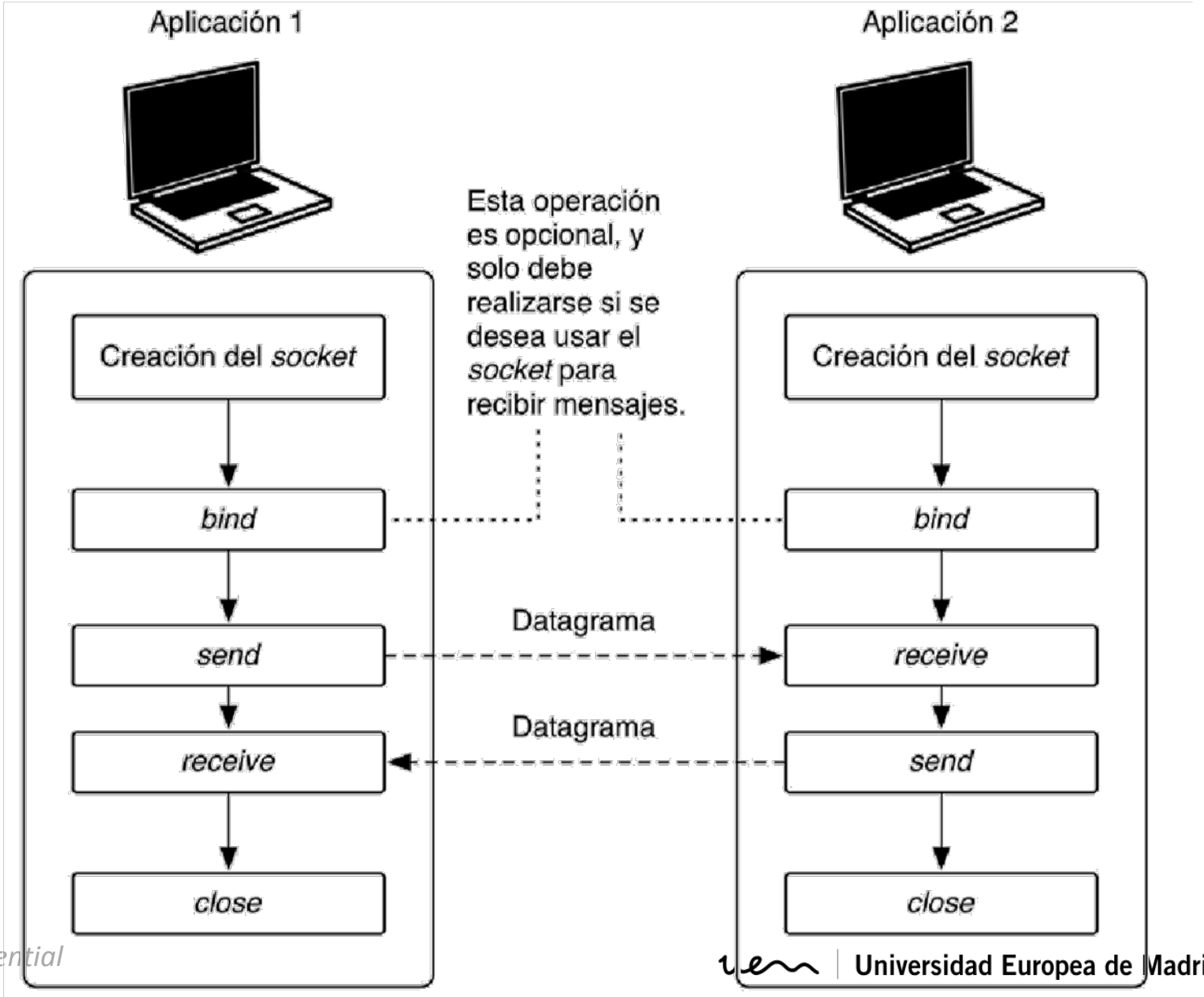
- Son **no orientados a conexión** y cuando operan sobre IP, emplean UDP.
- Pueden utilizarse para enviar mensajes a multitud de receptores, ya que usan un canal temporal para cada envío.
- No son fiables ni aseguran el orden de entrega correcto.
- Cuando se usan *sockets datagram* no existe diferencia entre proceso servidor y proceso cliente.

Pasos para enviar mensajes:

1. Creación del socket (similar al socket stream)
2. Asignación de dirección y puerto (*bind*). Solo necesaria para poder recibir mensajes.
3. Envío y/o recepción de mensajes.
4. Cierre del socket.

TIPOS DE SOCKETS

Sockets Datagram



PROGRAMACIÓN CON SOCKETS

En la mayoría de los lenguajes de programación de alto nivel existen bibliotecas para crear, destruir y operar con sockets.

En Java existen tres clases principales:

- *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*.

La clase Socket

- Se utiliza para crear y operar con **sockets streams clientes**. Sus métodos más importantes se pueden ver en el siguiente enlace.
- En el siguiente ejemplo se muestra un programa sencillo en el que se crea un socket que se conecta a un socket stream servidor que se encuentra en localhost escuchando por el puerto 5555 y se envía el mensaje "mensaje desde cliente". Por último, se cierra el socket y el programa termina.

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

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 ("Enviando mensaje");
                  String mensaje = "mensaje desde el cliente";
                  os.write(mensaje.getBytes());
                  System.out.println ("Mensaje enviado");
                  System.out.println ("Cerrando el socket cliente");
                  clientSocket.close();
                  System.out.println ("Terminado");
            } catch (IOException e) {
                  e.printStackTrace();
            }
      }
}
```

La clase ServerSocket

- Se utiliza para crear y operar con **sockets streams servidor**. Sus métodos más importantes se pueden ver en el siguiente enlace.
- Esta clase no tiene un método independiente para la operación de *listen*, ya que ésta se realiza conjuntamente con la operación *accept*.
- El comienzo es similar al anterior, cuando llega una conexión el método accept() establece el canal, creando un nuevo socket y devolviéndolo. El proceso servidor utiliza este nuevo socket para recibir un mensaje de 25 bytes de tamaño y lo imprime por su salida estándar convirtiéndolo en un String.

```
import java.io.*;
import java.net.*;
public class ServeridorSocketStream {
	public static void main(String[] args) {
		try {
			System.out.println ("Creando el socket servidor");
			ServerSocket serverSocket = new ServerSocket();
			System.out.println ("Realizando el bind");
			InetSocketAddress addr = new InetSocketAddress("localhost", 5555);
			serverSocket.bind(addr);
			System.out.println ("Aceptando conexiones");
			Socket newSocket = serverSocket.accept();
			System.out.println ("Conexión recibida");
			InputStream is = newSocket.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");
			newSocket.close();
			System.out.println ("Cerrando el socket servidor");
			serverSocket.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
```

La clase DatagramSocket

- Se utiliza para crear y operar con **sockets datagram**. Sus métodos más importantes se pueden ver en el siguiente enlace.

```
import java.io.*;
import java.net.*;
public class EmisorDatagram {
      public static void main(String[] args){
            try {
                  System.out.println ("Creando el 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();
      }
}
```

Universidad Europea

LAUREATE INTERNATIONAL UNIVERSITIES

Madrid Valencia Canarias