

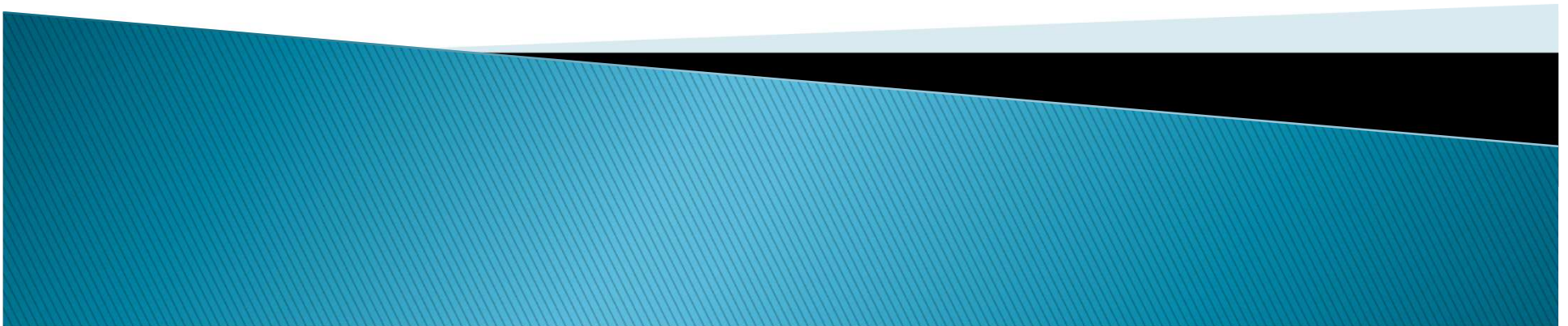
TEMA 4:

GENERACION DE

SERVICIOS DE RED

Programación de Servicios y Procesos

José Manuel García Sánchez



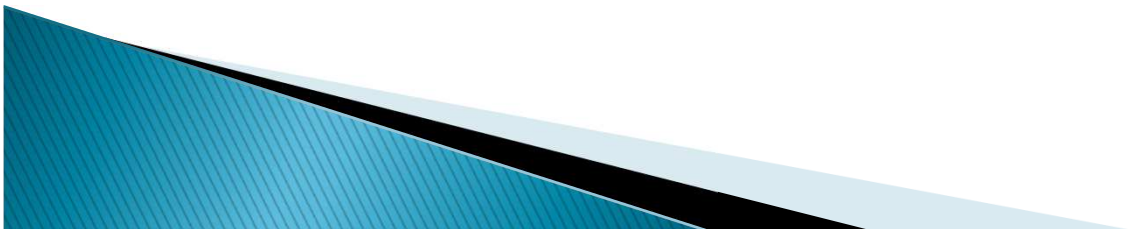
ÍNDICE

- ▶ Servicios
- ▶ Programación de aplicaciones cliente y servidor
- ▶ Protocolos estándar de nivel de aplicación
- ▶ Técnicas avanzadas de programación de aplicaciones distribuidas



Servicios

- ▶ Todo sistema está formado por dos partes fundamentales:
 - Estructura: Componentes hardware y software que lo forman.
 - Función: Aquello para lo que está pensado el sistema, es decir, para lo qué sirve y/o se usa.



Concepto de servicio

- ▶ El **servicio** de un sistema es el conjunto de mecanismos concretos que hacen posible una función.
- ▶ Los usuarios interactúan con el sistema y hacen uso de sus servicios a través de la **interfaz del servicio**.

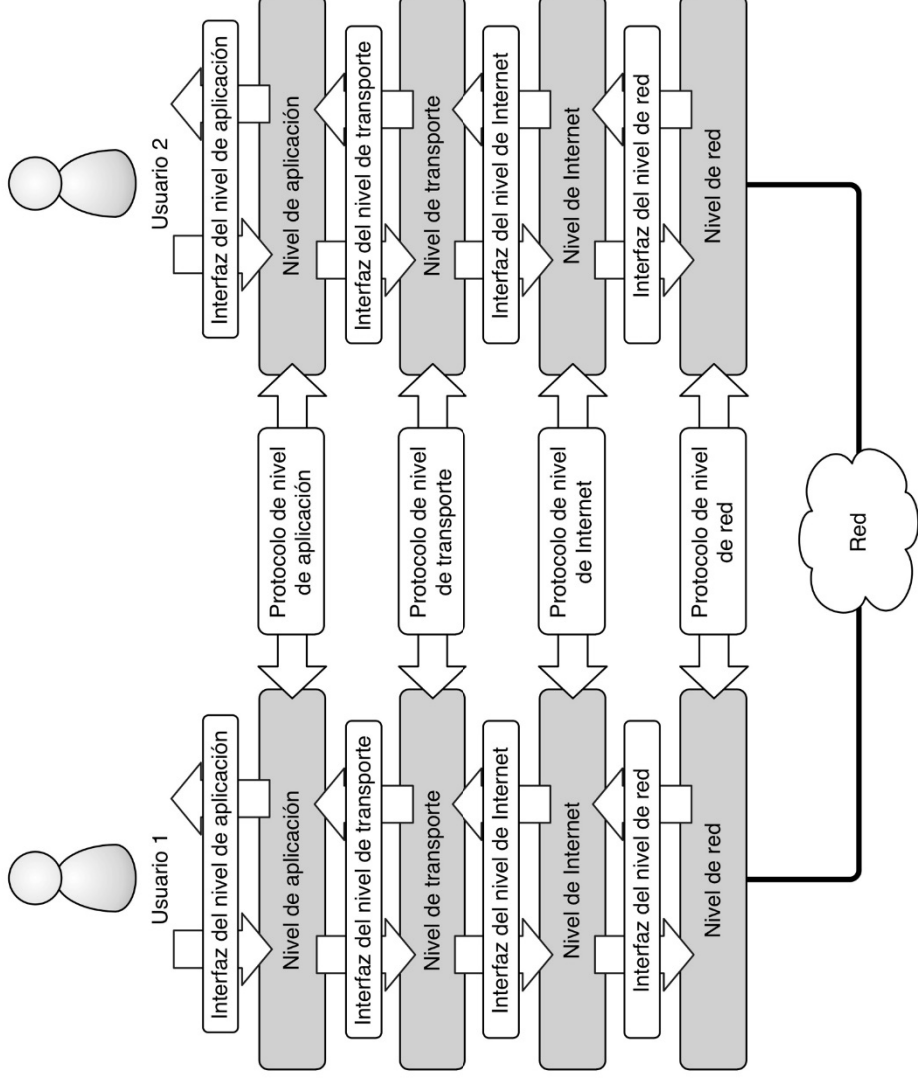


Servicios en red

- ▶ La pila de protocolos IP es un conjunto de sistemas independientes, montados unos sobre otros para realizar una tarea compleja.
- ▶ Cada nivel de la jerarquía (red, Internet, transporte y aplicación) proporciona un servicio específico y ofrece una interfaz de servicio al nivel superior.
- ▶ 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.

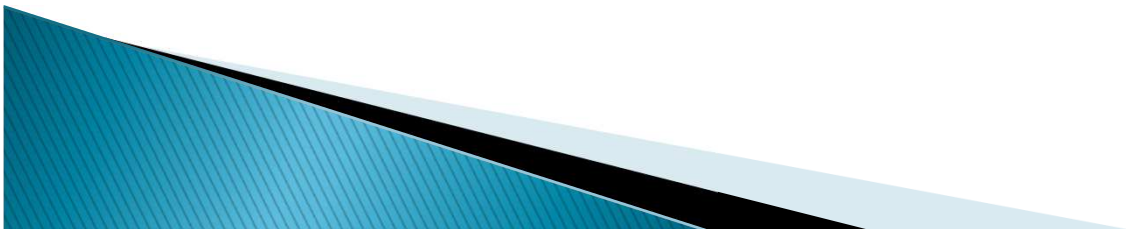


Servicios en red



Servicios de nivel de aplicación

- ▶ El nivel más alto de la pila IP lo componen las aplicaciones que forman el sistema distribuido.
- ▶ Un **protocolo de nivel de aplicación** es el conjunto de reglas que gobiernan la interacción entre los diferentes elementos de una aplicación distribuida.



Programación de aplicaciones cliente y servidor

- ▶ A la hora de programar una aplicación siguiendo el modelo cliente/servidor, se deben definir de forma precisa los siguientes aspectos:
 - Funciones del servidor.
 - Tecnología de comunicaciones.
 - Protocolo de nivel de aplicación.



Funciones del servidor

- ▶ Definir de forma clara qué hace el *servidor*.
 - ¿Cuál es la función básica de nuestro servidor?
 - El servicio que proporciona nuestro servidor, ¿es rápido o lento?
 - El servicio que proporciona nuestro servidor, ¿puede resolverse con una simple petición y respuesta o requiere del intercambio de múltiples mensajes entre este y el cliente?
 - ¿Debe ser capaz nuestro servidor de atender a varios clientes simultáneamente?
 - Etc.



Tecnología de comunicaciones

- ▶ Se debe escoger la tecnología de comunicaciones que es necesario utilizar.
- ▶ Los dos mecanismos básicos de comunicación que se han estudiado son los *sockets stream* y los *sockets datagram*.
- ▶ Cada tipo de *socket* presenta una serie de ventajas e inconvenientes.
- ▶ Es necesario escoger el tipo de *socket* adecuado, teniendo en cuenta las características del servicio que proporciona nuestro servidor.
 - Los *sockets stream* son más fiables y orientados a conexión, por lo que se deben usar en aplicaciones complejas en las que clientes y servidores intercambian muchos mensajes.
 - Los *sockets datagram* son menos fiables, pero más eficientes, por lo que es preferible usarlos cuando las aplicaciones son sencillas, y no es problema que se pierda algún mensaje.



Definición del protocolo de nivel de aplicación

- ▶ Un **protocolo de nivel de aplicación** es el conjunto de reglas que gobiernan la interacción entre los diferentes elementos de una aplicación distribuida.
- ▶ Se debe definir explícitamente el formato de los mensajes que se intercambian entre cliente y servidor, así como las posibles secuencias en las que estos pueden ser enviados y recibidos.
- ▶ La definición del protocolo de nivel de aplicación debe contener todos los posibles tipos de mensajes (tanto peticiones como respuestas) que pueden ser enviados y recibidos, indicando cuándo se puede enviar cada uno y cuándo no.



Protocolos estándar de nivel de aplicación

- ▶ En muchos casos el cliente y el servidor de sistemas distribuidos modernos son aplicaciones independientes, desarrolladas por diferentes personas/compañías y muchas veces ni siquiera implementadas usando el mismo lenguaje de programación.
- ▶ La definición exhaustiva de un protocolo de nivel de aplicación es fundamental en estos casos para garantizar que clientes y servidores pueden comunicarse de manera efectiva.
- ▶ Para la mayoría de aplicaciones distribuidas comunes se han definido, a lo largo de los años, protocolos de nivel de aplicación estándar, que facilitan la tarea de desarrollar servidores y clientes para ellas.



Protocolos Estándar de nivel de aplicación

- ▶ Telnet
- ▶ FTP
- ▶ POP3
- ▶ SMTP
- ▶ HTTP
- ▶ DHCP
- ▶ DNS
- ▶ ...



Envío de Objetos a través de Sockets.

- ▶ Hasta ahora se han intercambiado cadenas de caracteres entre programas.
- ▶ Los Stream soportan diversos tipos de datos como son:
 - los bytes
 - los tipos de datos primitivos
 - los objetos.



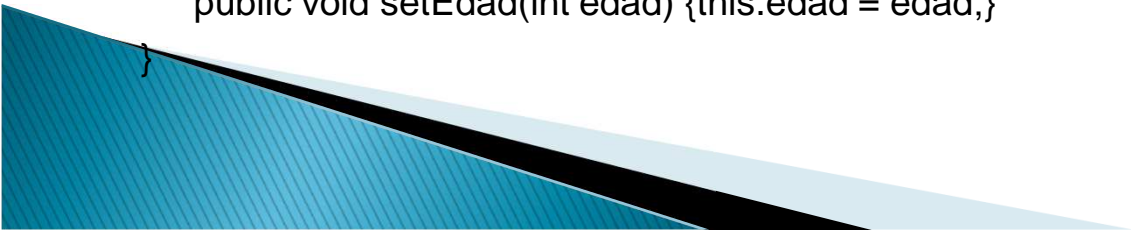
Objetos en Sockets TCP

- ▶ Las clases **ObjectInputStream** y **ObjectOutputStream** permiten enviar objetos a través de sockets TCP.
- ▶ Se usan los métodos:
 - **readObject()** para leer el objeto del stream.
 - **writeObject()** para escribir el objeto al stream.
- ▶ Se usa el constructor que admite un **InputStream** y un **OutputStream**.
- ▶ Para preparar el flujo de salida para escribir objetos
 - `ObjectOutputStream outObjeto = new ObjectOutputStream(socket.getOutputStream());`
- ▶ Para preparar el flujo de entrada para leer objetos:
 - `ObjectInputStream inObjeto = new ObjectInputStream(socket.getInputStream());`
- ▶ Las clases de estos objetos deben implementar la Interfaz **Serializable**. (Proceso de convertir un objeto a bytes)
 - http://chuwiki.chuidiang.org/index.php?title=Serializaci%C3%B3n_de_objetos_en_java

Objetos en Sockets TCP. Ejemplo

Clase Persona con 2 atributos, nombre y edad, 2 constructores y los métodos get y set correspondientes:

```
import java.io.Serializable;
@SuppressWarnings("serial")
//Esta anotación se utiliza para evitar un error en tiempo de compilación al implementar la interfaz java.io.Serializable
public class Persona implements Serializable {
    String nombre;
    int edad;
    public Persona(String nombre, int edad){
        super();
        this.nombre=nombre;
        this.edad=edad;
    }
    public Persona() {super();}
    public String getNombre() {return nombre;}
    public void setNombre( String nombre) {this.nombre = nombre;}
    public int getEdad() {return edad;}
    public void setEdad(int edad) {this.edad = edad;}
}
```



Aplicación Cliente-Servidor. Objeto Persona

- ▶ Intercambio de objetos Persona mediante TCP.
- ▶ El servidor crea un objeto Persona le da valores y se lo envía al programa cliente, el cliente realiza cambios en el objeto y se lo devuelve modificado al servidor.




Programa Servidor Objects

```
import java.io.*;
import java.net.*;
public class ServidorObjeto {
    public static void main(String[] arg) throws IOException, ClassNotFoundException {
        int numeroPuerto = 6000; //Puerto
        ServerSocket servidor = new ServerSocket(numeroPuerto);
        System.out.println("Esperando al cliente ....");
        Socket cliente = servidor.accept();
        // preparar el flujo de salida para objetos
        ObjectOutputStream outObjeto = new ObjectOutputStream( cliente.getOutputStream());
        //Preparar un objeto y enviarlo
        Persona per = new Persona("Juan",20);
        outObjeto.writeObject(per); //enviando objeto
        System.out.println("Envio: " + per.getNombre() + "*" + per.getEdad());
        // obtener un stream para leer objetos
        ObjectInputStream inObjeto = new ObjectInputStream(cliente.getInputStream());
        Persona dato = (Persona) inObjeto.readObject();
        System.out.println("Recibo: " + dato.getNombre()+"*" +dato.getEdad());
        //Cerrar Streams y Sockets
        outObjeto.close();
        inObjeto.close();
        cliente.close();
        servidor.close();
    }
}
```



Programa cliente Objects

```
import java.io.*;
import java.net.*;
public class ClienteObjeto {
    public static void main(String[] arg) throws IOException, ClassNotFoundException {
        String Host= "localhost";
        int Puerto = 6000; //Puerto remoto
        System.out.println("Programa Cliente Iniciado...");
        Socket cliente = new Socket(Host,Puerto);
        // Flujo de entrada para objetos
        ObjectInputStream perEnt = new ObjectInputStream(cliente.getInputStream());
        //Se recibe un objeto
        Persona dato = (Persona) perEnt.readObject(); // recibir objeto
        System.out.println("Recibo; " + dato.getNombre() + "*" + dato.getEdad());
        //Modificación del objeto
        dato.setNombre("Juanito Maravilla");
        dato.setEdad(7);
        //Flujo de salida para objetos
        ObjectOutputStream perSal = new ObjectOutputStream(cliente.getOutputStream());
        // envio del objeto
        perSal.writeObject(dato);
        System.out.println("Envio: " + dato.getNombre()+"*" +dato.getEdad());
        // Cerrar Streams y Sockets
        perEnt.close();
        perSal.close();
        cliente.close();
    }
}
```



Ejercicio Servicio de Calculadora

- ▶ Implementación Cliente/Servidor de un servicio de Calculadora. El servidor debe ser capaz de admitir varios clientes a la vez a los cuales atenderá en hilos diferentes.
- ▶ Crear una clase Datos con tres propiedades para almacenar tres números y una operación.
- ▶ Enviar el objeto al servidor que devuelve el objeto con el resultado calculado.

