Linkia FP

Formación Profesional Oficial a Distancia

Linkia FP

DAM – M09 – Clase 06

Programación de servicios y procesos.

UF3 – Sockets y Servicios

Servicios.

M09: Programación de Servicios y Procesos.

99 horas en total.

UF2: Procesos y Hilos (37 horas)
UF3: Sockets y Servicios (37 horas)
UF1: Seguridad y Criptografía (25 horas)

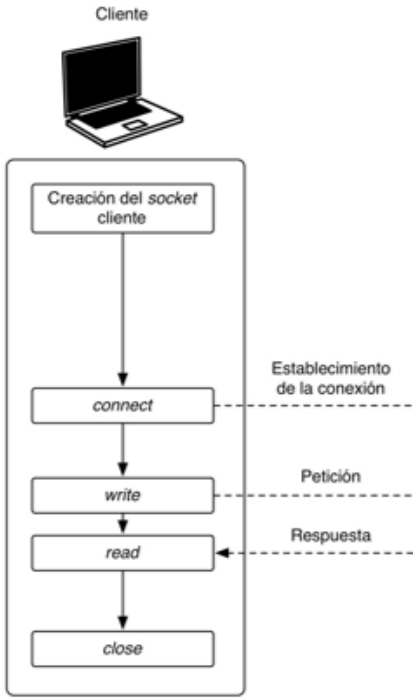
4 Actividades – 3 Tests – 10 Clases Virtuales.

Sockets stream

- Son orientados a conexión.
- Cuando operan sobre IP, emplean TCP.
- 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.
- Una parte ejerce la función de **proceso cliente** y otra de **proceso servidor**.

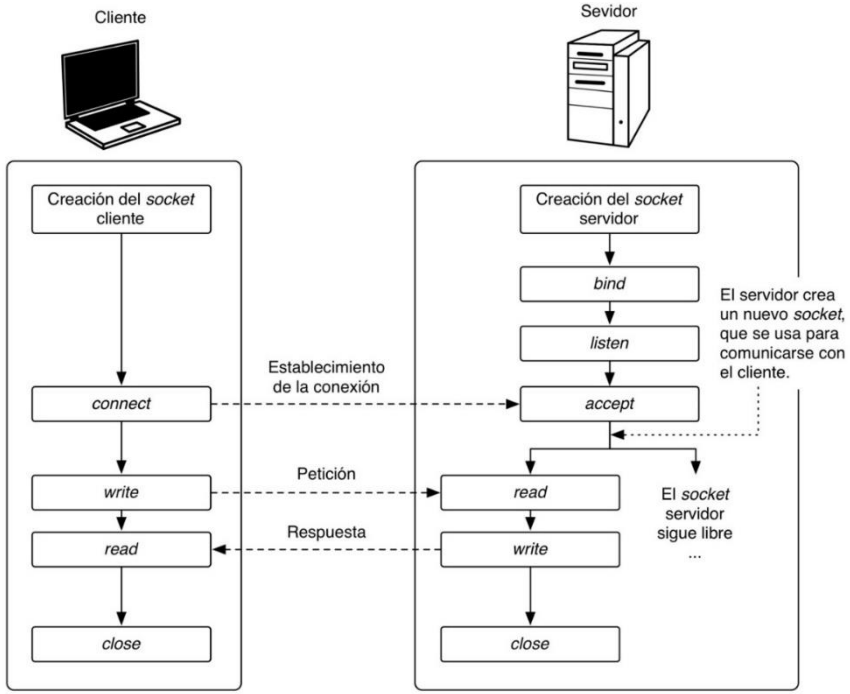
Proceso cliente:

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



***java.net.Socket*, para la creación de *sockets stream* cliente.**
***java.net.ServerSocket*, para la creación de *sockets stream* servidor.**

Sockets stream



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*).

Servidor de cálculo

La lógica de programación suele estar en el servidor, interpreta las órdenes y devuelve los resultados a los clientes.

```
G:\Mi unidad\java\tema4>java -jar ServidorCalculo.jar
```

```
G:\Mi unidad\java\tema4>java -jar ClienteCalculo.jar
Conectado !
suma 5 8
MENSAJE DEL SERVIDOR: La Suma de 5.0 y 8.0 da como resultado: 13.0
resta 10 2
MENSAJE DEL SERVIDOR: La Resta de 10.0 y 2.0 da como resultado: 8.0
multiplica 8 3
MENSAJE DEL SERVIDOR: La Multiplicacion de 8.0 y 3.0 da como resultado: 24.0
sqrt 9
MENSAJE DEL SERVIDOR: La Raiz de 9.0 da como resultado: 3.0
divide 5 2
MENSAJE DEL SERVIDOR: La Division de 5.0 y 2.0 da como resultado: 2.5
final
MENSAJE DEL SERVIDOR: Apagando el servidor...
```

Servidor de ficheros

La lógica de programación suele estar en el servidor, en este caso interpreta envía un fichero que solicita el cliente.

```
G:\Mi unidad\java\tema4>java -jar ServidorFicheros.jar
```

```
G:\Mi unidad\java\tema4>java -jar ClienteFicheros.jar episodio4
Descargando fichero episodio4

Episodio IV: Una nueva esperanza.
---------------------------------

Hace mucho tiempo en una galaxia muy, muy lejana...

Nos encontramos en un periodo de guerra civil. Las naves espaciales rebeldes,
atacando desde una base oculta, han logrado su primera victoria contra el malvado
Imperio Galáctico.

Durante la batalla, los espías rebeldes han conseguido apoderarse de los planos secretos del
arma total y definitiva del Imperio, la ESTRELLA DE LA MUERTE, una estación espacial
acorazada, llevando en sí potencia suficiente para destruir a un planeta entero.

Perseguida por los siniestros agentes del Imperio, la Princesa Leia vuela hacia su patria, a
bordo de su nave espacial, llevando consigo los planos robados, que pueden salvar a su pueblo
y devolver la libertad a la galaxia....

--------------------------------------------------------------------------------
G:\Mi unidad\java\tema4>
```

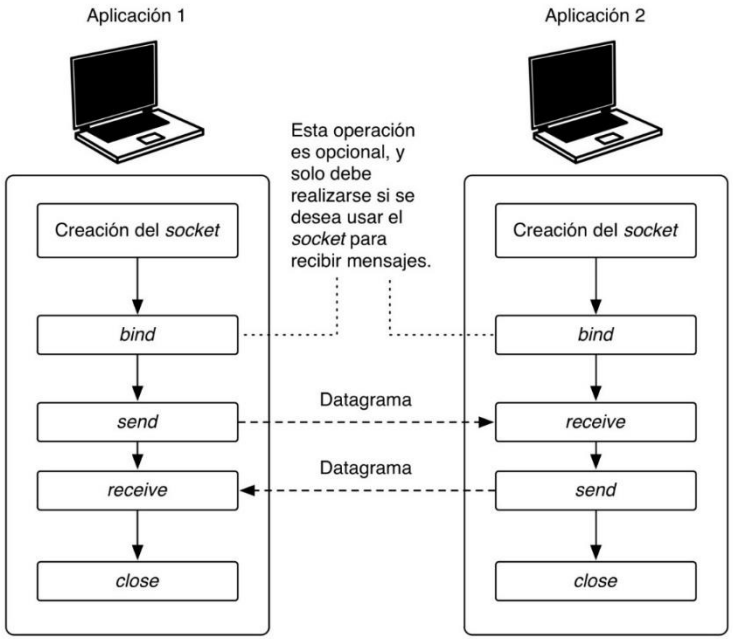
Sockets datagram

- Son no orientados a conexión.
- Cuando operan sobre IP, emplean UDP.
- Cuando se usan *sockets datagram* no existe diferencia entre proceso servidor y proceso cliente.

Pasos para enviar mensajes:

1. Creación del socket.
2. Asignación de dirección y puerto (*bind*).

   Solo necesaria para poder recibir mensajes.
1. Envío y/o recepción de mensajes.
2. Cierre del socket.



***java.net.DatagramSocket*, para la creación de *sockets datagram*.**

Servidor de IP

La lógica de programación suele estar en el servidor, en este caso interpreta envía un fichero que solicita el cliente.

```
G:\Mi unidad\java\tema4>java -jar ServidorIP.jar
```

```java
public class ServidorIP {
    public static void main(String[] args) {
        try {
            InetSocketAddress addr = new InetSocketAddress("localhost", 5556);
            DatagramSocket datagramSocket = new DatagramSocket(addr);
            while (true) {
                byte[] buffer = new byte[25];
                DatagramPacket datagrama1 = new DatagramPacket(buffer, buffer.length);
                datagramSocket.receive(datagrama1);
                String respuesta = datagrama1.getAddress().toString();
                DatagramPacket datagrama2 = new DatagramPacket(respuesta.getBytes(),
                    respuesta.getBytes().length, datagrama1.getAddress(), datagrama1.getPort());
                datagramSocket.send(datagrama2);
                }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
G:\Mi unidad\java\tema4>java -jar ClienteIP.jar
Mi dirección es /127.0.0.1
```

Servicios

Todo sistema está formado por dos partes fundamentales:

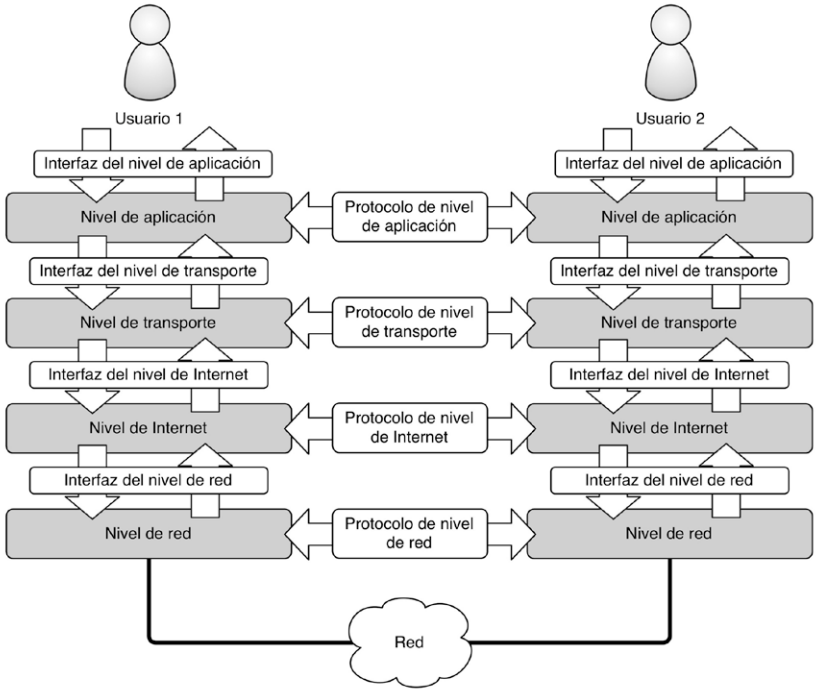
- **Estructura**: Componentes hardware y software que lo forman.
- **Función**: Aquello para lo que está pensado el sistema, es decir, para lo qué sirve y/o se usa.

El **servicio** de un sistema es el conjunto de mecanismos concretos que hacen posible su 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 especifico 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.



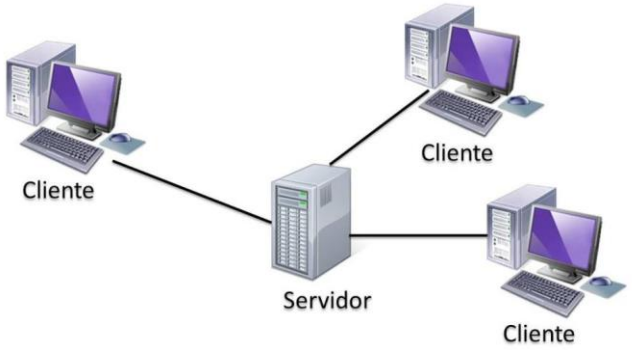
Servicios de nivel de aplicación

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.

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.

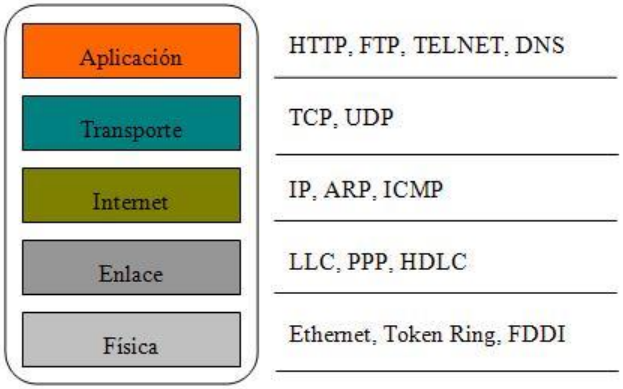
- Los *sockets stream* son más fiables y orientados a conexión, por lo que se deben usar en aplicaciones complejas en las que clientes y servidores intercambian muchos mensajes.
- Los *sockets datagram* son menos fiables, pero más eficientes, por lo que es preferible usarlos cuando las aplicaciones son 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.



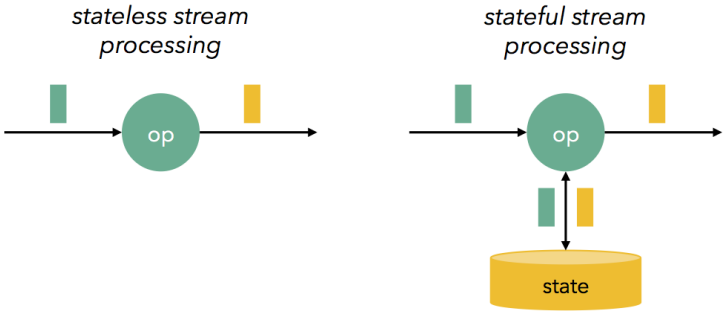
Protocolos sin estado y con estado.

- **Protocolos sin estado (*stateless*):**

Son aquellos en los que la secuencia concreta en la que se reciben los mensajes no es importante, ya que no afecta al resultado de estos. El servidor se limita a recibir las peticiones del cliente y a responderlas una por una, de forma independiente.

- **Protocolos con estado (*stateful*):**

Son aquellos en los que la secuencia concreta en la que se reciben los mensajes es importante, ya que afecta al resultado final de las peticiones. En estos casos, el servidor debe almacenar información intermedia durante la sesión, denominada el *estado de la sesión*. Conocer este estado es imprescindible para poder resolver las peticiones de manera correcta.

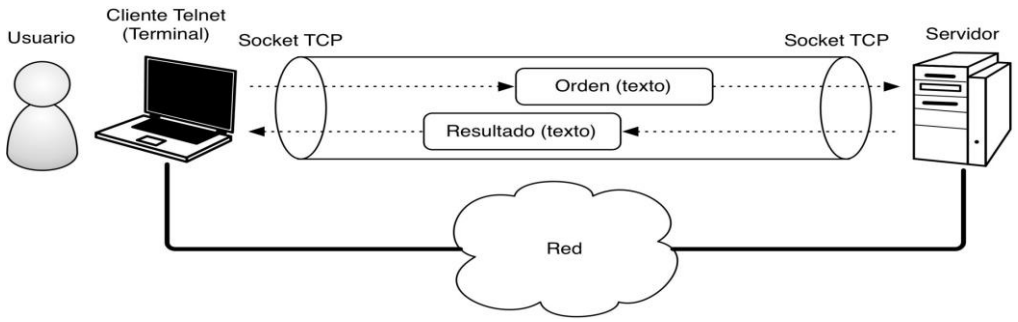


Protocolos estándar a nivel de aplicación.

- 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.

Telnet y SSH.

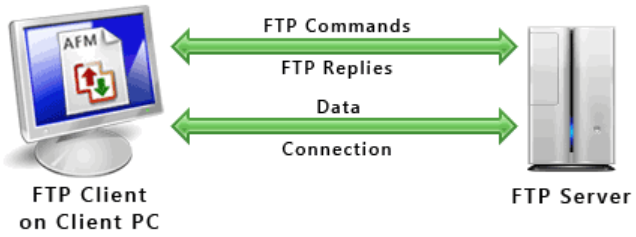
- Telnet: Protocolo de nivel de aplicación diseñado para proporcionar comunicación bidireccional basada en texto plano (caracteres ASCII) entre dos elementos de una red.
- Telnet se ha usado para conectarse remotamente a maquinas, creando una sesión de línea de mandatos como las *Shell* de los sistemas operativos UNIX o el "símbolo de sistema" de Windows (C:\>).
- Es un protocolo con estado (*stateful*). Usa por defecto el puerto 23.



- Un protocolo de nivel de aplicación muy similar a Telnet. Se usa par establecer sesiones de línea de mandatos con servidores remotos. Incorpora sistemas para hacer que la comunicación sea segura, por lo que es un alternativa muy recomendable a Telnet.
- Es un protocolo con estado (*stateful*). Usa por defecto el puerto 22.

FTP (File Transfer Protocol)

- Protocolo de nivel de aplicación diseñado para la transferencia de archivos a través de una red de comunicaciones.
- FTP establece un par de conexiones simultáneas entre cliente y servidor, que usa de forma distinta.
- FTP permite tanto la descarga de archivos (del servidor al cliente) como la subida de estos (del cliente al servidor).
  - Conexión de control: Se usa para enviar órdenes al servidor y obtener información.
  - Conexión de datos: Se utiliza para transferir los archivos.
- Es un protocolo con estado (*stateful*). Usa por defecto el puerto 21.



FTP (File Transfer Protocol)

Linkia FP

Ejemplo de código para implementar un cliente FTP utilizando la librería Apache MINA FTP server: Conexión y descarga del fichero.

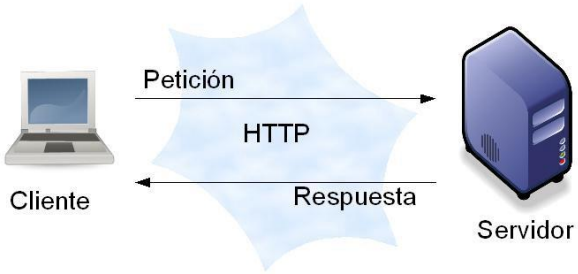
```
FTPClient clienteFtp = new FTPClient();
clienteFtp.connect(IP, PUERTO);
clienteFtp.login(USUARIO, CONTRASENA);
//Modo pasivo es siempre el cliente quien abre las conexiones
//Da menos problemas con los firewalls
clienteFtp.enterLocalPassiveMode();
clienteFtp.setFileType(FTPClient.BINARY_FILE_TYPE);
```

```
. . .
// Lista los ficheros del servidor
FTPFile[] ficheros = clienteFtp.listFiles();
for (int i = 0; i < ficheros.length; i++) {
  System.out.println(ficheros[i].getName());
}
// Seleccionamos el fichero remoto y el local
String ficheroRemoto = "/ejemplo.txt";
File ficheroLocal = new File("ejemplo.txt");
System.out.println("Descargando fichero '" + ficheroRemoto + "' del servidor . . .");
// Descarga un fichero del servidor FTP
OutputStream os = new BufferedOutputStream(new FileOutputStream(ficheroLocal));
if (clienteFtp.retrieveFile(ficheroRemoto, os))
  System.out.println("El fichero se ha recibido correctamente");

os.close();
. . .
```

HTTP (Hypertext Transfer Protocol)

- **HTTP** es, probablemente, el protocolo de nivel de aplicación más importante de la actualidad.
- La mayoría del tráfico que se realiza en la *World Wide Web*, la parte más visible de la red Internet, utiliza este protocolo para controlar la transferencia de información.
- Es un protocolo sin estado (*stateless*).
- Usa por defecto el puerto 80.



Un servidor HTTP organiza la información que contiene (fundamentalmente documentos de *hipertexto*, es decir, páginas web) usando *recursos*.

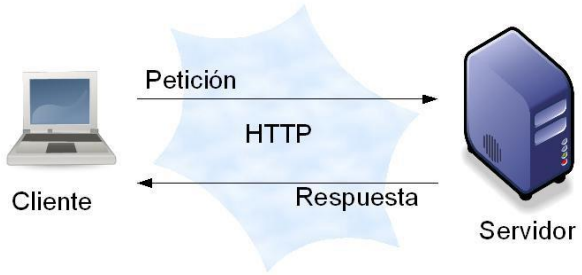
- GET: Obtener un recurso.
- POST: Modificar un recurso.
- PUT: Crear un recurso.
- DELETE: Eliminar un recurso.
- etc.

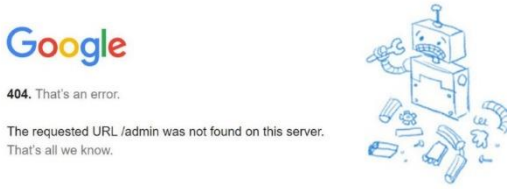
HTTP (Hypertext Transfer Protocol)

Los clientes pueden realizar diferentes peticiones al servidor.

Las respuestas de un servidor HTTP contienen una cabecera y un cuerpo de mensaje.

- Cabecera: Información sobre el servidor y estado de la operación solicitada.
- Cuerpo: Contenido de la respuesta, normalmente un documento de *hipertexto*.





La cabecera de una respuesta siempre contiene un código (número) de estado:

- Información (códigos que empiezan por 1)
- Estado de éxito (códigos que empiezan por 2)
- Redirección (códigos que empiezan por 3)
- Error del cliente (códigos que empiezan por 4)
- Error del servidor (códigos que empiezan por 5)

HTTP (Hypertext Transfer Protocol)

Para implementar un cliente HTTP podemos utilizar un componente JEditorPane en el que fijamos una URL . Este componente es capaz de renderizar paginas web de forma muy básica.

```
. . .
JEditorPane paginaWeb = new JEditorPane();
String url = "http://www.google.es";
try {
  paginaWeb.setPage(url);
  } catch (IOException ioe) {
    // Error al intentar cargar la URL
  }
}
. . .
```

HTTP (Hypertext Transfer Protocol)

Podemos leer un pagina web y obtener su código html.

```java
public class Ejemplo2URL {

	public static void main(String[] args) {

		URL url=null;

		try {
			url = new URL("https://www.linkiafp.es");
		} catch (MalformedURLException e) {
			e.printStackTrace();
		}

		BufferedReader in;
		try {
			InputStream inputstream = url.openStream();
			in = new BufferedReader(new InputStreamReader(inputstream));
			String inputLine;
			while ((inputLine = in.readLine()) != null)
				System.out.println(inputLine);
			in.close();
		} catch (IOException e) {
			e.printStackTrace();
		}

	}// Fin de main

}//Fin de Ejemplo2URL
```

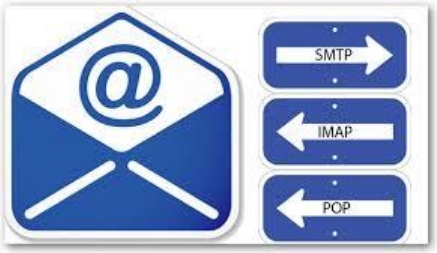
```
<!doctype html>
<html lang="es-ES" prefix="og: http://ogp.me/ns#">
<head>
<meta charset="utf-8">
<meta http-equiv="x-ua-compatible" content="ie=edge">
<meta name="viewport" content="width=device-width, init
<title>Linkia FP - FormaciÃ³n Profesional a Distancia C
<link rel="alternate" hreflang="es" href="https://linki
<link rel="alternate" hreflang="ca" href="https://linki
<!-- This site is optimized with the Yoast SEO plugin v
<meta name="description" content="FP FormaciÃ³n Profesi
<link rel="canonical" href="https://linkiafp.es/" />
```

POP3 y SMTP

POP3 (Post Office Protocol ver.3)

- Lo usan las aplicaciones de correo electrónico para acceder a los mensajes alojados en servidores.
- Es un protocolo sin estado (*stateless*).
- Usa por defecto el puerto 110.

SMTP (Simple Mail Transfer Protocol)



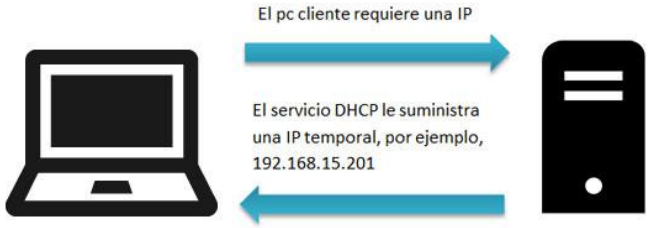
- Es el protocolo estándar para la transferencia de correo electrónico.
- Lo usan todos los servidores de e-mail de Internet.
- Se usa también para enviar mensajes desde cliente de correo locales, como Thunderbird o Outlook.
- Es un protocolo sin estado (*stateless*).
- Usa por defecto el puerto 587.

Cliente SMTP

Cliente Java que envía un mensaje de correo a un servidor SMTP utilizando la librería JavaMail

```
try {
            // Inicializa una sesión
            Properties props = System.getProperties();
            props.put("JavaMailSMTP", SERVIDOR);
            Session sesion = Session.getDefaultInstance(props, null);
            // Creamos el mensaje
            Message mensaje = new MimeMessage(sesion);
            mensaje.setFrom(new InternetAddress(SERVIDOR));
            mensaje.setRecipients(Message.RecipientType.TO, InternetAddress.parse(TO, false));
            mensaje.setSubject(SUBJECT);
            mensaje.setText(BODY);
            // Se establece la fecha de envío
            mensaje.setSentDate(new Date());
            // Envía el mensaje
            System.out.println("Enviando mensaje . . .");
            Transport.send(mensaje);
            System.out.println("Mensaje enviado.");
            } catch (Exception e) {
                        e.printStackTrace();
}
```

Otros protocolos de nivel de aplicación importantes.



- **DHCP (*Dynamic Host Configuration Protocol*):**

Para configurar máquinas dentro de una red.

- **DNS (*Nomain Name Service*):**

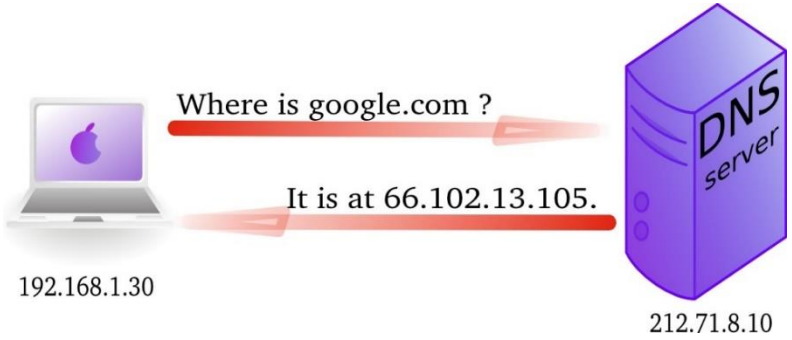
Para localizar direcciones IP a partir de nombre simbólicos.

- **NTP (*Network Time Protocol*):**

Para sincronizar relojes entre máquinas.

- **TLS (*Transport Layer Security*) y SSL *(Secure Socket Layer*):**

Para proporcionar comunicaciones seguras.



Linkia FP

Pregunta de la 6a clase virtual.

¿Qué diferencia existe entre un protocolo con estado y sin estado?.

CLASE

Linkia FP

Formación Profesional Oficial a Distancia