Tema 4. Servicios en red y protocolos

Comunicación cliente-servidor

- Tiene que existir un acuerdo entre dos o más equipos que se comunican entre sí.
- Ese acuerdo da sentido a los mensajes que se envían.
- A ese acuerdo se denomina PROTOCOLO.
- Un protocolo define todos los aspectos que rigen la comnicación entre los equipos: puertos, tamaño de los mensajes, tipos de mensaje, etc.

Protocolos con y sin estado

- Que un protocolo sea sin estado significa que cada petición enviada por el cliente es independiente de las peticiones anteriores. El servidor, pues, no cambia de estado según recibe peticiones.
- En un protocolo con estado, las peticiones del cliente pueden depender o estar relacionadas con peticiones anteriores.
- Un protocolo sin estado no puede manejar información relacionada con sesiones, usuarios, etc.

Protocolos TCP y UDP

- Los protocolos estándar que veremos a continuación son protocolos de nivel de aplicación.
- Aunque en este tema solamente veremos protocolos TCP, existen otros UDP como el protocolo DNS.
- Por tanto, en el caso de querer implementar un cliente o servidor sobre alguno de estos protocolos, generalmente se utilizará la clase Socket o SSLSocket (Secure Socket Layer).

Protocolo DHCP

- DHCP se utiliza para que los equipos de una red puedan adquirir de forma automática una configuración válida que les permita interactuar con el resto de la red.
- Cliente y servidor intercambian los siguientes mensajes:

  <C>DHCPDISCOVER → <S>DHCPOFFER → <C>DHCPREQUEST → <S>DHCPACK / DHCPNAK
- Los puertos utilizados son el 67 para el servidor y 68 para el cliente (ambos UDP).

Protocolos de control remoto

- Permiten controlar una máquina de forma remota como si estuviésemos manejando su consola de comandos.
- Telnet. Hoy en día obsoleto debido a su carencia de seguridad. Todas las comunicaciones son no cifradas. Por defecto, su puerto es el 23.
- SSH (Secure Shell). Ha sustituido a Telnet, pero proporciona la misma funcionalidad. Encripta las comunicaciones mediante un algoritmo de clave pública. Su puerto por defecto es el 22.
- PuTTY es un cliente muy común de SSH.

Transferencia de archivos

- FTP (File Transfer Protocol). Permite acceder al sistema de archivos del servidor, consultar su contenido, navegar por él, realizar cambios (copiar archivos, renombrarlos, etc), y transferir archivos bidireccionalmente entre el cliente y el servidor.
- Originalmente, carece de métodos de seguridad. Existen protocolos más modernos relacionados con FTP (por ejemplo, FTPS, FTP over SSH) que implementan seguridad criptográfica en las comunicaciones.
- Filezilla es un cliente muy común. Los navegadores también tienen implementado el protocolo FTP para descargar archivos.

Transferencia de archivos (II)

- El protocolo FTP maneja dos conexiones simultáneas entre cliente y servidor: una de control para el envío de comandos y respuestas (por el puerto 20), y otra de datos para transferir archivos (por el puerto 21).
- Por cuestiones de configuración, el servidor tiene dos modos: el modo activo y el modo pasivo.
- El modo activo, más antiguo, supone cierta configuración extra (que puede implicar vulnerabilidades de seguridad) en el cliente. El modo activo traspasa esa responsabilidad en el servidor.

Protocolos de correo electrónico: POP3

- POP3 (Post Office Protocol version 3). Es un protocolo pensado para que el cliente descargue los mensajes de correo electrónico del servidor para manejarlos en local.
- Era especialmente popular cuando las conexiones eran lentas o intermitentes y de buzones de correo con limitaciones de tamaño, pues permitía descargar todos los mensajes de correo en local y el borrado de su copia remota en una sola conexión.

Ejemplo de uso de POP3 sin seguridad

```
//El cliente conecta con el servidor por el puerto 110 TCP
InetAddress dir=InetAddress.getByName("pop.educa.madrid.org");
int port=110;
Socket socket=new Socket(dir,port);
BufferedReader br = new BufferedReader(new InputStreamReader(socket.getInputStream()));
BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
//El servidor manda un mensaje de bienvenida
System.out.println(br.readLine()); //+OK POP3 perditon ready on …
//El cliente se identifica con los comandos USER y PASS (el servidor responde a cada uno)
bw.write("USER usuario\n");  //el usuario de la cuenta es “usuario”
bw.flush();
System.out.println(br.readLine()); //+OK USER usuario set, mate
bw.write("PASS password\n"); //la contraseña es “password”
bw.flush();
System.out.println(br.readLine()); //+OK
```

```
//El cliente lanza el comando STAT. El servidor responde con el número de
//mensajes y el total que ocupan en bytes.
bw.write("STAT\n");
bw.flush();
System.out.println(br.readLine()); //+OK 200 43542740
//El cliente lanza el comando RETR con el número del mensaje que quiere
//consultar. El servidor manda la respuesta: el mensaje de correo
//electrónico con las cabeceras, en texto plano y en HTML, terminando el
//mensaje con una línea que contiene un punto “.”.
bw.write("RETR 1\n");
bw.flush();
String linea="";
while(!linea.equals(".")) {
linea=br.readLine();
System.out.println(linea);
}
```

Protocolos de correo electrónico: SMTP

- Protocolo SMTP (Simple Mail Transfer Protocol). Sirve para enviar correos. Un cliente SMTP indica al servidor la información precisa para que éste envíe correos electrónicos a otro servidor.
- El envío de correos entre distintos servidores (por ejemplo, entre gmail y yahoo) también se realiza mediante SMTP.
- Muchos servidores de correo electrónico no permiten el acceso a clientes de origen desconocido. Generalmente se puede configurar la cuenta para que acepte estas conexiones, pero podría marcar nuestros correos como spam.

Ejemplo de uso de SMTP por SSL

- Para realizar conexiones por sockets seguros se utiliza la clase SSLSocket.

```
SSLSocketFactory factory = (SSLSocketFactory) SSLSocketFactory.getDefault();
SSLSocket ssocket = (SSLSocket) factory.createSocket(direccion, puerto);
```

- Para enviar las credenciales, se utiliza la case Base64.Encoder y su método encodeToString.
- Una vez realizada la conexión, el resto de procedimientos es igual que con Sockets: se crean BufferedReader y BufferedWriter para efectuar la comunicación entre cliente y servidor.

Ejemplo de uso de SMTP por SSL (II)

```
//El cliente crea los canales de comunicación

BufferedReader sbr = new BufferedReader(...);

BufferedWriter sbw = new BufferedWriter(...);

//El servidor manda un mensaje de bienvenida

System.out.println(sbr.readLine());

//220 smtp.educa.madrid.org ESMTP

//El cliente saluda con el mensaje EHLO seguido de un nombre de
dominio. El servidor responde con una serie de líneas indicando,
entre otras cosas, el sistema de login. Cada mensaje comienza
con un código de tres cifras seguido de un guion y a
continuación la información. La última línea carece de guion.

sbw.write("EHLO apple.com\n");

sbw.flush();

String mensajeServidor="";

do {

   mensajeServidor=sbr.readLine();

   System.out.println(mensajeServidor);

}while (mensajeServidor.charAt(3)=='-');
```

```
//El cliente manda el mensaje AUTH PLAIN, el servidor responde
con un mensaje 334 y a continuación el cliente envía sus
credenciales codificadas mediante Base64.Encoder

sbw.write("AUTH PLAIN\n");

sbw.flush();

mensajeServidor = sbr.readLine();

System.out.println(mensajeServidor); //334

Base64.Encoder encoder=Base64.getEncoder();

sbw.write(

  encoder.encodeToString("\0nombre_usuario\0password".getBytes()

  +"\n");

sbw.flush();

System.out.println(sbr.readLine()); //235 nice to meet you
```

Ejemplo de uso de SMTP por SSL (III)

```
//Para mandar un correo, se envían los mensajes MAIL FROM (remitente), MAIL TO (correo de respuesta), RCPT TO (destinatario) y DATA

sbw.write("MAIL FROM:<"+mailfrom+">\n");

sbw.flush();

System.out.println(sbr.readLine()); //250 ok

sbw.write("MAIL TO:<"+mailfrom+">\n");

sbw.flush();

System.out.println(sbr.readLine()); //250 ok

sbw.write("RCPT TO:<"+mailto+">\n");

sbw.flush();

System.out.println(sbr.readLine()); //250 ok

sbw.write("DATA\n");

sbw.flush();

System.out.println(sbr.readLine()); //334 go ahead
```

```
//Una vez enviado el mensaje DATA, el cliente manda el mensaje. Debe ir terminado por una línea con un solo punto. El servidor responde con un mensaje de ok

String mensaje="From: \"Yomismo\" <"+mailfrom+">\n";

mensaje+="To: \"Perico\" <"+mailto+">\n";

mensaje+="Subject: Prueba\n\n";

mensaje+="Mensaje mandado automáticamente\n";

mensaje+="\n.\n";

sbw.write(mensaje);

sbw.flush();

System.out.println(sbr.readLine()); //250 ok <datos>

//El cliente finaliza la sesión con el mensaje QUIT

sbw.write("QUIT\n");

sbw.flush();

System.out.println(sbr.readLine()); //221 Goodbye
```

Navegación por internet

- HTTP (HyperText Transfer Protocol). Es el protocolo de transferencia de páginas web, que hoy en día ha diversificado su uso para permitir comunicaciones entre cliente y servidor a través de aplicaciones web.
- Originalmente, al igual que Telnet o FTP, la transferencia se realizaba en texto plano. Hoy en día se utiliza frecuentemente HTTPS, que es HTTP mediante TLS (Transport Layer Security). De este modo, la transferencia de datos es cifrada e invulnerable a sniffers.
- HTTP es un protocolo, en principio, sin estado. Por defecto, utiliza el puerto 80. HTTPS utiliza el 443.

Estructura del mensaje HTTP

- Un mensaje HTTP tiene tres partes:
  - Línea inicial. Contiene la petición del cliente o el código de respuesta del servidor. Debe ir terminada por /r/n. Ejemplos:

    `GET /index.html HTTP/1.1` (Petición del cliente)

    `HTTP/1.1 200 OK` (Respuesta del servidor)
  - Cabeceras, también conocidas como metadatos. Cada cabecera va en una línea, con el formato <nombre de cabecera>:<valor>. Al término de las cabeceras tiene que haber una línea vacía (/r/n) Ejemplos:

    ```
    Content-Type: text/html
    Date: Wed, 8 Jun 2016 14:35:44 GMT
    ```
  - Cuerpo del mensaje. Opcional. Generalmente contiene, en la respuesta del servidor, código HTML o el contenido de un archivo referenciado en éste.

Mensajes más importantes de HTTP

- GET. Solicita al servidor que envíe el recurso solicitado (normalmente, pero no exclusivamente, una página web).
- HEAD. Igual que GET, pero solamente se envía la cabecera.
- POST. Realiza una petición al servidor con información para ser procesada por él. Utilizado por ejemplo para enviar unos campos de formulario al servidor para procesar un login.

Ejemplo de diálogo HTTP (I)

El siguiente ejemplo está extraído de Wikipedia

- Petición del cliente:

```
GET /index.html HTTP/1.1
Host: www.example.com
Referer: www.google.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
Connection: keep-alive
[Línea en blanco]
```

Ejemplo de diálogo HTTP (II)

- Respuesta del servidor:

```
HTTP/1.1 200 OK
Date: Fri, 31 Dec 2003 23:59:59 GMT
Content-Type: text/html
Content-Length: 1221

<html lang="eo">
<head>
<meta charset="utf-8">
<title>Título del sitio</title>
</head>
<body>
<h1>Página principal de tuHost</h1>
  .
  .
  .
</body>
</html>
```