UD3: PROGRAMACIÓN DE COMUNICACIONES EN RED

Tercera parte: protocolos estándar de comunicaciones en el nivel de aplicación.

1. Introducción

Los servicios son programas auxiliares utilizados en un sistema de computadoras para gestionar una colección de recursos y prestar su funcionalidad a los usuarios y aplicaciones. Por ejemplo, cuando enviamos un documento a una impresora que está formando parte de una red estamos usando un servicio de impresión, este servicio permite gestionar y compartir la impresora de la red.

El único acceso que tenemos al servicio está formado por el conjunto de operaciones que ofrece, por ejemplo, un servicio de ficheros ofrece operaciones de lectura, escritura o borrado de ficheros.

Recordar que el modelo TCP/IP está compuesto por cuatro capas o niveles. En la capa de aplicación definimos un protocolo de nivel de aplicación como el conjunto de reglas que gobiernan la interacción entre los diferentes elementos de una aplicación distribuida. En una aplicación cliente/servidor, este protocolo especifica cómo se realiza la interacción entre el servidor y los clientes.

Todos los servicios de Internet implementan una relación cliente-servidor, por lo que estudiaremos estos servicios y usaremos java para programar clientes de los servicios de Internet que se usan más frecuentemente.

Por tanto, la capa de aplicación maneja protocolos de alto nivel que implementan servicios como:

- conexión remota: Telnet
- correo electrónico: SMTP
- acceso a ficheros remotos: FTP,NFS,TFTP
- resolución de nombres de ordenadores: DNS,WINS
- World Wide Web: HTTP

Capas TCP/IP
Aplicación
Transporte
Internet
Red

Capas y protocolos TCP/IP	
SMTP, Telnet, FTP, HTTP	NFS,SNMP,DNS
TCP	UDP
IP	
Protocolos de subred	

Todas las aplicaciones que implementan TCP/IP se basan en el modelo cliente/servidor. Veamos algunas de ellas:

- TELNET (Telecomunication Network). Emulación de terminal; permite a un usuario acceder a una máquina remota y manejarla como si estuviese sentado delante de ella. Es el sistema empleado para arreglar fallos de máquinas remotas o para realizar consultas a distancia como por ejemplo para consultar los fondos de

una biblioteca. Su principal problema es la seguridad ya que los nombres de usuario y contraseñas viajan por la red como texto plano.

- SMTP (Simple Mail Transfer Protocol): protocolo simple de transferencia de correo electrónico; es probablemente el servicio más popular entre los usuarios de la red. Este estándar especifica el formato exacto de los mensajes que un cliente en una máquina debe enviar al servidor en otra. Administra la transmisión de correo electrónico a través de las redes informáticas.
- FTP (File Transfer Protocol): protocolo de transferencia de ficheros; es un servicio confiable orientado a conexión que se utiliza para transferir ficheros de una máquina a otra a través de Internet. Los sitios FTP son lugares desde los que podemos descargar o enviar ficheros.
- TFTP (Trivial File Transfer Protocol): protocolo trivial de transferencia de Hipertexto, utilizado por los navegadores web para realizar peticiones a los servidores web y para recibir las respuestas de ellos. Es un protocolo que especifica los mensajes involucrados en un intercambio petición-respuesta, los métodos, argumentos, resultados y las reglas para representar todo ello en los mensajes.
- NFS (Network File System). Sistema de ficheros de red, ha sido desarrollado por Sun Microsystems y permite a los usuarios el accesos en línea a ficheros que se encuentran es sistemas remotos, de esta forma el usuario accede a un fichero como si este fuera un fichero local.
- SNMP (Simple Network Managemente Protocol): protocolo simple de administración de red, es un protocolo utilizado para intercambiar información de gestión entre los dispositivos de una red. Permite a los administradores monitorear, controlar y supervisa el funcionamiento de la red.
- DNS (Domain Name System): sistema de nombre de dominio, es un sistema que usa servidores distribuidos a lo largo de la red para resolver el nombre de un host IP en una dirección IP, de esta manera no hay que recordar y usar su dirección IP.

2. FTP

FTP es un protocolo de nivel de aplicación diseñador para la transferencia de archivos a través de una red de comunicaciones. De hecho es la forma habitual de publicación en Internet.

Para usar FTP para transferir ficheros entre dos ordenadores, cada uno debe tener un papel, es decir uno debe ser el cliente FTP y otro el servidor FTP. El cliente envía comandos al servidor (subir, bajar o borrar ficheros, crear un directorio) y el servidor los lleva a cabo. Podemos imaginarnos al servidor como un gran contenedor en el que podemos encontrar gran cantidad de ficheros y directorios.

Hay dos tipos fundamentales de acceso a través de FTP:

- Acceso anónimo: cuando la conexión con la máquina servidora la realiza un usuario sin autentificar y sin ningún tipo de privilegio en el servidor. En este caso el usuario es recluido a un directorio público dónde solo se le permite descargar ficheros.
- Acceso autorizado: el usuario que realiza la conexión con la máquina servidora está registrado y tiene ciertos privilegios en el servidor. En este caso, y una vez autenticado, el usuario es recluido a su directorio personal dónde puede subir y bajar ficheros; normalmente se le asigna una cuota de espacio.

FTP utiliza dos conexiones TCP distintas, una conexión de control y otra de transferencia de datos. La primera se encarga de iniciar y mantener la comunicación entre el cliente y el servidor, la segunda se encarga de enviar datos entre cliente y servidor y existe únicamente cuando hay datos a transmitir.

Cuando un cliente se conecta a un servidor FTP, el cliente emplea un puerto aleatorio pero el servidor se conecta en el puerto 21. Para la transferencia de datos no se utilizan los mismos puertos, el cliente obtiene un nuevo puerto y el servidor en el proceso de transferencia de datos usa el puerto 20.

2.1. Java para comunicar con un servidor FTP

Existen librerías Java que nos permiten crear programas cliente para comunicar con un servidor FTP. Apache Commons Net™ proporciona una librería de componentes que nos permite implementar el lado cliente de muchos protocolos básicos de Internet. La librería incluye soporte para protocolos como FTP, SMTP, Telnet, TFTP, etc,..

A continuación vamos a ver cómo acceder desde un programa cliente Java, a un servidor FTP, podremos conectarnos, listar los ficheros y directorios, subir ficheros, eliminarlos, etc..

Necesitaremos la librería commons-net-3.4-bin.zip y de ahí el commons-net-3.4.jar que se pueden descargar desde la URL de Apache commons.

La clase FTPClient encapsula toda la funcionalidad necesaria para almacenar y recuperar ficheros de un servidor FTP. Esta clase se encarga de todos los detalles de bajo nivel de la interacción con un servidor FTP. Para utilizarla primero es necesario realizar la conexión al servidor con el método connect(), comprobar el código de respuesta para ver si ha ocurrido algún error, realizar las operaciones de transferencia y cuando finalice el proceso, cerrar la conexión usando el método disconnect().

En el siguiente ejemplo realizamos una conexión a un servidor FTP(ftp.adobe.com) comprobamos si se ha realizado correctamente o no y cerramos la conexión:

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

```
import java.net.SocketException;
import org.apache.commons.net.ftp.*;
public class ClienteFTP1{
     public static void main(Strinng [] args) throws SocketException, IOException{
      FTPClient cliente = new FTPClient();
      String servFTP = "ftp.adobe.com"; //servidor FTP
      System.out.println("Nos conectamos a : "+servFTP);
      cliente.connect(servFTP);
      //respuesta del servidor FTP
      System.out.print(cliente.getReplyString());
      //código de respuesta
      int respuesta = cliente.getReplyCode();
      //comprobación del código de respuesta
      If(!FTPReply.isPositiveCompletion(respuesta)){
            cliente.disconnect();
            System.out.println("conexión rechazada: "+ respuesta);
            System.exit(0);
            }
            //desconexión del servidor FTP
            cliente.disconnect();
            System.out.println("conexión finalizada.");
            }
      }
```

La clase FTPReply almacena un conjunto de constantes para códigos de respuesta FTP. Para interpretar el significado de los códigos se puede consultar la RFC 959 (http://www.ietf.org/rfc/rfc959.txt). Los nombres nemónicos usados para las constates son transcripciones de las descripciones de los códigos de la RFC 959.

El método *isPositiveCompletion(int respuesta*) devuelve *true* si un código de respuesta ha terminado positivamente. El código 220 significa que el servicio está preparado.

Algunos métodos de la clase FTP son:

Métodos	Descripción
void connect (String host)	Abre la conexión con el servidor FTP indicado en host
int getReplyCode()	Devuelve el valor entero del código de respuesta de la última respuesta FTP
String get ReplyString()	Devuelve el texto completo de la respuesta del servidor FTP

Algunos de los métodos de la clase FTPClient (derivada de FTP) son:

Métodos	Descripción
void disconnect()	Cierra la conexión con el servidor FTP y restaura los parámetros de conexión a los valores predeterminados.
boolean login(string user, String passwd)	Inicia sesión en el servidor FTP usando el nombre d usuario y la contraseña proporcionados. Devuelve true si se inicia la sesión con éxito, si no devuelve false
boolean logout()	Sale del servidor FTP
String printWorkingDirectory()	Devuelve el nombre de ruta del directorio de trabajo actual
FTPFile[] listFiles(String path)	Obtiene una lista de ficheros como un array de objetos FTPFile del directorio indicado en path
String[] listNames()	Obtiene una lista de ficheros del directorio actual como un array de cadenas
FTPFile [] listDirectories (String parent)	Obtinee la lista de los directorios que se encuentran en el directorio especificado en parent
boolean changeWorkingDirectory(string pathname)	Cambia el directorio de trabajo actual de la sesión FTP al indicado en pathname
boolean storeFile(String nombre, InputStream local)	Almacena un fichero en el servidor con le nombre indicado tomando como entrada el inputStream.
boolean retrieveFile (String nombre, OutputStriam local)	Recupera un fichero del servidor y los escribe en el OutputStream dado.
boolean deleteFile (String pathname)	Elimina un fichero en el servidor FTP
boolean rename (String antiguo, String njuevo)	Cambia el nombre de un fichero del servidor FTP
Boolean removeDirectory(String pathname)	Elimina un directorio en el servidor FTP(si está vacio)
Boolean makeDirectory (String pathname)	Crea un nuevo subdirectorio en el servidor FTP en el directorio actual

Todos los métodos de comunicación con el servidor pueden lanzar IOException. El servidor FTP puede optar por cerrar antes de tiempo una conexión si el cliente ha estado inactivo durante más de un periodo de tiempo determinado (generalmente 900 segundos). La clase FTPClient detectará un cierre prematuro de la conexión con el servidor FTP y se puede producir la excepción FTPConnectionClosedEXception.

Lo más normal es conectar a un servidor FTP con un nombre de usuario y su clave. Para identificarnos usaremos el método login() que devuelve true si la conexión se realiza correctamente; para desconectarnos usamos le método logout().

La clase FTPFile se utilizar para representar información acerca de los ficheros almacenados en un servidor FTP. Algunos métodos importantes son:

Métodos	Misión
String getName()	Devuelve el nombre del fichero
long getSize()	Devuelve el tamaño del fichero en bytes
int getType()	Devuelve el tipo del fichero, 0 si es un fichero (FILE_TYPE),1 un directorio (DIRECTORY_TYPE) y 2 un elnace sumbólico (SYMBOLIC_LINK_TYPE).
String getUser()	Devuelve el nombre del usuario propietario del fichero
boolean isDirectory()	Devuelve true si el fichero es un directorio.
boolean isFile	Devuelve true si es un fichero
boolean isSymbolicLink()	Devuelve true si es un enlace simbólico

En el siguiente ejemplo nos conectamos al servidor FTP ftp.rederis.es utilizando un acceso anónimo. Nos conectamos como usuario *anonymous* y clave igual para mostrar la lista de ficheros del directorio actual. Usamos el método listFiles() que devuelve un array de la clase FTPFile con información de los ficheros y directorios encontrados;

recorremos el array visualizando el nombre del fichero o directorio y el tipo que puede ser fichero o directorio o enlace simbólico:

```
import java.io.*;
import org.apache.commons.net.ftp.*;
public class ClienteFTP2 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		FTPClient cliente = new FTPClient();
		String servFTP= "ftp.rediris.es";
		System.out.println("Nos conectamos a: "+ servFTP);
		String usuario ="anonymous";
		String password = "anonymous";
		try{
			cliente.connect(servFTP);
			boolean login = cliente.login(usuario, password);
			if (login)
				System.out.println("Login correcto...");
			else{
				System.out.println("Login incorrecto..");
				cliente.disconnect();
				System.exit(1);
			}
			System.out.println("Directorio actual: " +
cliente.printWorkingDirectory());
			FTPFile [] files = cliente.listFiles();
			System.out.println("Ficheros en el directorio actual: "+ files.length);
			//array para visualizar el tipo de fichero
			String tipos [] = {"Fichero ", "Directorio", "Enlace simb."};
			for(int i =0; i<files.length; i++){
				System.out.println("\t"+files[i].getName() + " => " +
tipos[files[i].getType()]);
			}
			boolean logout = cliente.logout();
			if (logout)
				System.out.println("Logout del servidor FTP...");
			else
				System.out.println("Error al hacer el logout...");
			cliente.disconnect();
			System.out.println("Desconectando...");
		}catch (IOException ioe){ioe.printStackTrace();}
	}

}
```

La ejecución muestra la siguiente salida:

```
Nos conectamos a: ftp.rediris.es
Login correcto...
Directorio actual: /
Ficheros en el directorio actual: 12
	. => Directorio
	.. => Directorio
	debian => Fichero
	incoming => Directorio
	ls-lR => Fichero
	ls-lR.Z => Fichero
	ls-lR.gz => Fichero
	mirror => Directorio
	outgoing => Directorio
	pub => Directorio
```

```
            sites => Directorio
            welcome.msg => Fichero
      Logout del servidor FTP...
      Desconectando...
```

2.2. Subir ficheros al servidor.

Para los siguientes ejemplos necesitamos tener acceso a un servidor FTP. Podemos crearnos un hosting web gratuito que ofrezca servicio de FTP(http://www.x90x.net/, http:/byethost.com/, http://lk6.com.ar/, etc). O podemos instalar el servidor FTP Filezilla Server en nuestra máquina local.

Para subir ficheros al servidor necesitamos un usuario, su clave y un espacio en el servidor y tener privilegios para ello. En primer lugar necesitamos situarnos en le directorio dónde vamos a subir los ficheros; por ejemplo, supongamos que es un subdirectorio que cuelga del directorio raíz(/) y se llama NUEVODIREC:

```
String direcc = "/NUEVODIREC"
Cliente.changeWorkingDirectory(direc);
```

A continuación con el método setFileType() se indica el tipo de fichero a subir. Este tipo es una constante entera definida en la clase FTP. Se suele poner BINARY_FILE_TYPE que permite enviar ficheros de cualquier tipo:

```
Cliente.setFileType(FTP:BYNARY_FILE_TYPE);
```

Creamos un steam de entrada con los datos del fichero que vamos a subir y se los pasamos al método storeFile(), en el primer parámetro indicaremos el nombre que tendrá el fichero en el directorio FTP y en el segundo el InputStream:

```
BufferedInputStream in = new BufferedInputStream(new
FileInputStream("D:\\CAPIT4\\TEXTO!.txt"));
Cliente.storeFile("TEXTO1.txt", in);
```

Por último será necesario cerrar el flujo de entrada. En el ejemplo completo que se muestra a continuación, se suben 2 ficheros al directorio NUEVODIREC, uno de texto y una imagen.

```
import java.io.*;
import org.apache.commons.net.ftp.*;
public class SubirFichero {public static void main (String[] args){
      FTPClient cliente = new FTPClient();//cliente
      String servidor = "Servidor.ftp.es"; /servidor
      String user = "usuario";
      String psw = "psw";

      try{
        System.out.println ("conectandose a " + servidor);
        cliente.connect (servidor);
        boolean login = cliente.login(user, pasw);
        String direc ="/NUEVODIREC";
        If (login){
              cliente.changeWorkingDirectory(direc);
              cliente.setFileType(FTP.BINARY_FILE_TYPE);
```

```
        //stream de entrada con el fichero a subir
        BufferedInputStream          in          =          new          BufferedInputStream(new
FileInputStream("D:\\CAPIT4\\TEXTO1.txt"));
        Cliente.storeFile("TEXTO!.txt", in);
        in= new BufferedInputStream( new FileInputStream ("D:\\CAPIT\\Homi.jpg"));
        cliente.storeFile("Homi.jpg",in);

        in.close(); //cerrar flujo
         cliente.logout(); //logout del usuario
        cliente.disconnect(); //desconexión del servidor
        }catch (IOException ioe){ ioe.printStackTrace();}
        }
}
```

Para renombrar un fichero se usa el método *rename (antiguo, nuevo).* Devuelve true si renombra el fichero con éxito, en caso contrario devuelve false.

2.3. Descargar ficheros del servidor.

Para descargar un fichero del servidor en nuestro disco duro, usamos el método retrieveFile(String remote, OutputStream local). Necesitamos saber el directorio desde el que descargaremos el fichero. El método devuelve true si el proceso se realiza satisfactoriamente, en caso contrario devuelve false. Necesitaremos crear un stream de salida para escribir el fichero en nuestro disco duro.

Por ejemplo para descargar el fichero de nombre TEXTO".TXT, que se ubica en la carpeta del servidor FTP/htdocs/NUEVODIREC/Nuevo en nuestro disco duro en la carpeta D/CAPIT$ y con nombre TEXTO2nuevo.txt, escribo lo siguiente:

```
//descargar fichero
String direcc= "/htdocs/NUEVODIREC/NUEVO";
cliente.changeWorkingDirectory(direcc);
//stream de salida para recibir el fichero descargado
BufferedOutputStream     out     =     new     BufferedOutputStream     (new     FileOutputStream
("D:\\CAPIT4\\TEXTO2nuevo.txt"));
If (cliente.retrieveFile("Texto2.txt",out))
        System.out.println("Recuperado correctamente…");
else
System.out.println("no se ha podido descargar…")
```

2.4. Creación de un cliente FTP

A continuación vamos a crear un cliente sencillo desde el que podremos subir, descargar y eliminar ficheros; y crear y eliminar directorios o carpetas en nuestro sitio FTP. Como servidor FTP usaremos Filezilla Server instalado en nuestra máquina local.

Para el ejemplo se ha creado un usuario con nombre “usuario1” y clave “usu1”; se ha creado una carpeta en D:\xampp\htdocs de nombre usuario1 y se han almacenado datos en ella. Desde la pantalla Filezilla Server Interface se asigna al usuario la carpeta creada dándole todos los permisos.

Los datos que necesitaremos para la conexión al servidor FTP son el nombre del servidor, el nombre del usuario y su clave. En el programa se han usado las siguientes variables para almacenar estos datos y se han asignado los siguientes valores: servidor= “127.0.0.0.1”, user = “usuario1” y pasw = “usu1”.

Utilizaremos la variable direcInicial para definir el directorio inicial del usuario a partir del cual realizamos la navegación, diresSelec para saber en todo momento cuál es el directorio de trabajo actual y ficheroSelec para saber el último fichero seleccionado. Se inicializa con los siguientes valores:

```
Static String direcInicial = “/”,
Static String direcSelec = direcInicial;
```

El ejemplo completo se puede ver en el anexo I.

3. Telnet

Telnet es un protocolo de red que pertenece a la familia de protocolos de Internet. Permite a los usuarios acceder a un ordenador remoto y realizar tareas como si estuviesen trabajando directamente delante de él. El acceso al ordenador remoto se realiza en modo terminal, es decir, no se muestra una pantalla gráfica (como el escritorio de Windows) para realizar las tareas; se trabaja desde la línea de comandos. Es útil para arreglar fallos de forma remota y para consultar información. Se utiliza bastante en sistemas UXIS-LINUX y en equipos de comunicaciones para la configuración de routers. Utiliza el puerto TCP 23.

El principal problema de Telnet es la seguridad, ya que los datos necesarios para conectarse a máquinas remotas (nombre de usuario y clave), no son cifrados y viajan por la red como texto plano, facilitando a cualquier que espíe la red mediante un programa sniffer obtener estos datos.

Telnet sigue un modelo cliente-servidor, para poder utilizarlo necesitamos tener instalado en una máquina un servidor Telnet y en otra máquina el cliente Telnet para poder acceder a ella.

3.1. Instalación y uso de un servidor Telnet

Para instalar un servidor Telnet en Windows pulsamos en inicio->panel de control-> Programas y características-> Activar y desactivar las características de Windows; se abre una ventana desde la que podremos activar o desactivar características de Windows. Marcamos las casillas Servidor Telnet para instalar el servidor y Cliente Telnet para instalar el cliente. Después se pulsa aceptar y se reinicia el equipo.

El servidor Telnet se instala como un servicio de Windows, será necesario iniciarlo para poder trabajar con él. Accediendo a los servicios de Windows (Inicio->panel de control->Herramientas administrativas ->Servicios) se puede indicar el tipo de inicio: manual, automático, etc…

Para hacer que los usuarios se puedan conectar a la máquina que tiene instalado el servidor Telnet hay que crear usuarios. Esto se puede hacer desde Inicio->Panel de control-> Cuentas de usuario; ser crea por ejemplo un usuario estándar de nombre usuario2 y clave usu2.

Para permitir que los usuarios obtengan acceso al servidor Telnet, es necesario agregarlos al grupo TelnetClientes. Seguiremos estos pasos:

- Ejecutamos el comando cmd.exe en el modo ejecutar como administrador. Se abre una ventana DOS.
- Escribimos el siguiente comando para añadir el usuario creado al grupo TelnetClients.

  net localgroup TelnetClients /add usuario2.

  Se debe visualizar el mensaje: “Se ha completado el comando correctamente”.
- Para comprobar si se ha añadido el usuario, ejecutamos esta orden. Net localgroput TelnetClients.
- Para probar la conexión (en modo local) con el servidor Telnet ejecutamos desde la línea de comandos la orden telnet localhost (tiene que estar el cliente Telnet instalado). Se muestra la pantalla del cliente Telnet que muestra un mensaje que nos pregunta si deseamos enviar la contraseña aunque no sea seguro (ya que no viajará encriptada por la red), escribirnos s para aceptar. Después muestra una nueva pantalla con el mensaje de bienvenida y nos pide el nombre de usuario y la contraseña, los escribimos; una vez comprobados se muestra la consola desde la que podemos escribir comandos e interactuar con el equipo remoto (que en este caso es el mismo equipo dónde está el servidor).

3.2. Java para comunicar con un servidor Telnet

La librería Apache Commons Net™ proporciona la clase TelnetClient (extiende SocketClient) que implementa el sencillo terminal virtual de red (NVT) para el protocolo Telnet según RFC854. Presenta dos tipos de constructores:

Constructor	Descrpición
TelnetClient()	Constructor pro defecto, establece como tipo de terminal VT100
TelnetCliente(String termtype)	Se establece como tipo de terminal el especificado en el String termtype(XTerm, VT100,VT200, etc)

La clase utiliza el método connect() de la clase SocketClient para conectarse al servidor. Una vez conectado se obtienen un InputStream y un OutpuStream para leer y enviar datos a través de la conexión que se pueden obtener mediante los métodos getInputStream() y getOutputStream().

Algunos métodos de esta clase son:

Métodos	Descripción
void disconnect()	Desconecta la sesión Telnet, cierra los input y output strea así como el socket.
InputStream getInputStream()	Devuelve el stream de entrada de la conexión Telnet, se usa para leer las respuestas que envía el servidor Telnet
OutputStream getOutputStream()	Devuelve el stream de salida de la conexión Telnet, se usa para enviar los comandos al servidor Telnet.

En el siguiente ejemplo nos conectamos al servidor Telnet local con nombre de usuario usuario2 y clave usu2 y le enviamos el comando DIR para que nos muestre el contenido del directorio. Vamos a ver paso a paso las operaciones de lectura mostrando lo que nos envía el servidor y de escritura enviando lo que nos pide (login, password..).

En primer lugar creamos el cliente Telnet mediante el segundo constructor estableciendo como tipo de terminal XTerm (emulador de terminal par el sistema de ventanas X Window System). A continuación se realiza la conexión con el servidor mediante el método connect(). Una vez conectados se crean el OutputStream o flujo de salida para enviar datos al servidor y el InputStream o flujo de entrada para leer los datos que el servidor envía.

```
import java.io.*;
import org.apahe.commons.net.telnet.*;
public class ClienteTelnet1 {
static InputStream in;
static PrintStream out;
public static void main (String [] args){
TelnetClient telnet = new TelnetClient ("xterm");
String server= "localhost";
String login= "usuario 2";
String password= "usu2";
try {
```

```
telnet.connect(server); //conexión con el servidor
//Se definen los flujos para enviar y recibir datos
out= new PrintStream(telnet.getOutputStream();
in = telnet.getInputStream();
```

Recordemos que lo primero que hace el servidor Telnet es mostrarnos en pantalla el mensaje de bienvenida y pedirnos el login.

```
//Recibe las primeras líneas y pide login:
Lectura datos();
// Se envía el nombre de usuario
EnvioDatos (login)
//Ahora pide pide password:
Lectura datos();
Se envía el password
EnvioDatos (password);
```

Una vez identificados se hace una nueva lectura de lo que envía el servidor, en este caso unas líneas de presentación y el prompt desde el que se pueden introducir los comandos

```
//Se muestra la presentación y el prompt
Lectura datos();
//se manda el comando DIR
EnvioDatos ("DIR");
//lectura visualizacion de la salida del comando DIR
Lectura datos();
```

Por último nos desconectamos del servidor Telnet.

```
//desconectar
telnet.disconnect();
} catch (exception e) {
e.printStackTrace ();
}
}//fin main
```

El método para leer datos que el servidor nos envía es el siguiente:

```
//Lectura y visualización de lo que envia el servidor Telnet
static void LecturaDatos() throws IOException  {
byte []buff= new byte [1024];
int nbytes;
nbytes= in.read(buff);//lee los bytes
System.out.print(new String(buff,0,,nbytes));//los visualiza
}//lectura
```

Por último, el método para que el usuario envíe datos al servidor es:

```
//Envío de datos al servidor Telnet
```

```
private static void EnvioDatos(String cad) {
out.println(cad);
out.flush();
}//escritura
}//.. ClienteTelnetl
```

Podemos hacer que este proceso de lectura se realice en un hilo. Una vez conectados al servidor Telnet se inicia el hilo y se leen los datos o respuestas que el servidor envía desde el hilo.

```
//Lectura de las respuestas que envía el servidor
class LecturaRespuestas extends Thread{
TelnetClient tc =  null;
//constructor
public LecturaRespuestas (TelnetClient tcli) {tc = tcli;}
public void run(){
        InputStream in = tc.getInputStream();
        byte[] buff = new byte[1024];
        int nbytes = 0;
        try{
                nbytes = in.read(buff); //lectura de bytes
                while(nbytes>0){
                System.out.print(new String (buff, 0, nbytes));
                nbytes = in.read(buff);
                }
                }catch(IOException e){
                System.err.println("Error al leer el imputStream…"+e.getMessage());
                }
                }//run
                }//fin hilo
```

Lo más normal es que le usuario escriba por teclado las órdenes a enviar al servidor. Una vez conectados e iniciado el hilo de lectura de respuestas del servidor hacemos un bucle desde el que se introducen datos por el teclado y se envían al servidor. El proceso termina cuando el usuario escribe *exit.*

```
Public class ClienteTelnet2{
static TelnetClent telnet = new TelnetClient (“xtrem”);
public static void main (String[] args) {
        String server = “localhost”;
        byte[] buff = new byte[1024];
        int nbytes=0;
        boolean repetir = true;
        OutputStream out;
        try{
                telnet.connect(server);//conexión al servidor
                System.out.println(“Cliente Telnet conectado…”);
                }catch (IOException e){
        System.err.println(“Error al conectar el cliente…”+e.getMessage());
        }
        //lanzamos el hilo
        LecturaRespuestas hilo = new LecturaRespuestas(telnet);
        hilo.start();

        //stream dónde escribo las órdenes que mando al servidor
        out = telnet.getOutputStream();
        while (repetir == true){
                        try{
                                //lectura de los comandos por teclado en bytes
                                nbytes = System.in.read(buff);
                                }catch(IOException e){
                                System.err.println(“Error al leer por teclado..”
+e.getMessage());
                                }
                                if(nbytes>0){
                                //convierte bytes a cadena
                                String cadena = new String (buff, 0, nbytes);
                                try{
                                        //enviar los comandos al servidor
                                        out.write(buff,0,nbytes);
                                        out.flush();
                                }catch (IOException e ){ System.err.println(“Error al
escribir en el OutputStream…”+e.getMessage());
                                }
                                //el proceso termina cuando se escribe exit
                                if(cadena.trim().equalsIgnoreCase(“exit”))
                                        repetir== flase;
                                        }
                        }//fin while
                        try{
```

```
                        telnet.disconnect();
                        System.out.println("Fin de la conexión…");
                    }catch(IOException e){
                        System.err.println("Error al cerrar el cliente
Telnet…"+e.getMessage());
                    }
                    System.exit(0); //salir
                }//main
                } //fin cliente Telnet2
```

Al ejecutarlo introducimos por teclado el login del usuario y luego su password, nos mostrará las líneas de presentación y a continuación el prompt desde el que podemos introducir comandos DOS.

4. SMTP

SMTP (*Simple Mail Transfer Protocol)* es el protocolo estándar de Internet para el intercambio de correo electrónico. Funciona con comandos de texto que se envían al servidor SMTP (por defecto al puerto 25). A cada comando que envía el cliente le sigue una respuesta del servidor compuesta por un número y un mensaje descriptivo. Las especificaciones de este protocolo se definen en la RFC 2821.

Normalmente, cuando creamos una cuenta de correo en un proveedor de servicios de internet, el proveedor nos proporciona los datos del servidor POP3 (o IMAP) y del servidor SMTP. Estos son necesarios para configurar clientes de correo como Microsoft Outlook, Eudora… El primero se utiliza para recibir los mensajes(es decir, para configurar el correo entrante) y el segundo para enviar nuestros mensajes (configurar el correo saliente).

4.1. Instalación de un servidor de correo electrónico.

Un servidor SMTP es un programa que permite enviar correo electrónico a otros servidores SMTP. Para instalar un simple servidor SMTP en nuestro equipo local vamos a la página http://www.argosoft.com/rootpages/Download.aspx y ahí tendríamos nuestro servidor SMTP.

NOTA: La versión gratuita solo dura 3º días.

OBS: a la hora de inicializarlo, hay un ayudante que nos guía para crear la conexión. Se puede usar como dirección DNS 80.58.0.33

4.2. Uso de Telnet para comunicar con el servidor SMTP

Para enviar un correo de forma manual usando Telnet al puerto 25 debemos ir a la línea de comandos y escribir *telnet* y a continuación *open localhost 25.* El servidor normalmente responde con un número de tres dígitos dónde cada uno tiene un significado especial (como en FTP). Por ejemplo, los números que empiezan en dos (220,250,..) indican que la acción se ha completado con éxito.

Los comandos principales los recoge la siguiente tabla.

Comando	Descripción
HELO o EHLO	Se utiliza para abrir una sesión con el servidor
MAIL FROM: origen	A la derecha se indica quien envía el mensaje.
RCPT TO: destino	A la derecha se indica el destinatario del mensaje
DATA mensaje	Se utiliza para indicar el comienzo del mensaje, este finalizará cuando haya una línea únicamente con un punto.
QUIT	Cierra la sesión
HELP	Muestra la lista de comandos que el servidor admite.

A continuación vemos las líneas de código de cómo se enviaría un correo.

```
HELO
250 Welcome [127.0.0.1], pleased to meet you
MAIL FROM: irenes@unizar.es
```

```
250 Sender irenes@unizar.es OK…
RCPT TO: irenes@unizar.es
250 Recipient “irenes@unizar.es” OK…
DATA
354 Enter mail, end with ”.” on a line by itself
from: irenes@unizar.es
to: irenes@unizar.es
subject: probando
Hola yo,
Esto es una prueba sencilla.
Adiós.
.
250 Message accepted for delivery.
s1012qpd0infsjm.123420131025@127.0.0.1
QUIT
```

4.3. Uso de Java para comunicar con un servidor SMTP

La librería Apache Commons NET™ proporciona la clase SMTPClient (extiende SMTP)que encapsula toda la funcionalidad necesaria para enviar ficheros a través de un servidor SMTP. Esta clase se encarga de todos los detalles de bajo nivel de interacción con un servidor SMTP.

Antes de hacer cualquier operación es necesario conectarse al servidor y una vez finalizada la interacción es necesario desconectarse. Normalmente, una vez conectados es necesario comprobar el código de respuesta SMTP para ver si la conexión se ha realizado correctamente.

SMTP presenta dos tipos de constructores:

Constructor	Descripción
SMTPClient()	Constructor por defecto
SMTPClient (String codificacion)	Se establece una codificación en le constructor(BASE64, BINARY,…)

La clase utiliza el método connect() de la clase SocketClient para conectarse al servidor; y el método disconnect() de la clase SMTP para desconectarse del servidor SMTP. Para conectarnos a un servidor SMTP cuyo puerto de escucha sea distinto, tendríamos que indicar en la conexión el número de puerto: *connect(host, puerto).*

Para escribir un simple mensaje a un destinatario podemos escribir las siguientes líneas:

```
cliente.login(); //inicio de sesión HELO
String destinatario: “irenes@unizar.es”;
String mensaje=“Hola. \n\t y adiós.”;
String remitente= “yo@localhost.es”;
client.sendSimpleMessage(remitente, destinatario, mensaje);
```

```
client.logout(); //final de sesión QUIT
```

En algunos servidores de correo, por ejemplo en el servidor SMTP de GMAIL, este mensaje no será admitido y no llegaría a su destino porque no está bien construido. Nuestro servidor SMTP mostraría un mensaje de error. En otros, posiblemente llegará como correo spam. Para solucionar esto tenemos la clase SimpleSMTPHeader que se utilizar para crear una cabecera mínima aceptable para el envío de un mensaje de correo electrónico. El constructor es el siguiente

```
SimpleSMTPHeader( String from, String to, Strig Subject)
```

Autenticación.

La autenticación SMTP se configura con el fin de elevar los niveles de seguridad y eficacia del servicio de correo electrónico y con el objetivo de prevenir que nuestra dirección de correo sea utilizada sin autorización, evitando el posible envío de correos no deseados a otras personas con fines perjudiciales. La autenticación se realiza a través de la verificación de nombre de usuario y contraseña.

Apache Commons Net™ proporciona la clase AuthenticatingSMTPClient (extiende SMTPSClient) con soporte de autenticación SMTP. La clase SMTPSClient proporciona soporte SMTP sobre el protocolo SSL *(Secure Socket Layer*- capa de conexión segura). SSL es un protocolo criptográfico empleado para realizar conexiones seguras entre un cliente y un servidor. TLS (*Transport Layer Security*-seguridad de la capa de transporte) es el protocolo sucesor de SSL.

4.4. Acceso a los mensajes desde un servidor SMTP

En el correo electrónico se usan otros protocolos, además del SMTP, para funciones adicionales. Entre los más utilizados están:

-MIME (*Multipurpose Internet Mail Extensions*- extensiones multipropósito de correo de internet): define una serie de especificaciones para expandir las capacidades limitadas del correo electrónico y en particular para permitir la inserción de documentos (como imágenes, sonido y texto)en un mensaje. Su versión segura se denomina S/MIME.

-POP (*Post Office Protocol*- protocolo de oficina de correo): proporciona acceso a los mensajes de los servidores SMTP. En general cuando hacemos referencia al término POP, nos estamos refiriendo a POP3 que es la última versión. Se basa en el protocolo de transporte TCP y proporciona peticiones básicas para acceso, descarga y borrado de mensajes. El número de puerto por defecto para servidores POP3 es el 110.

-IMAP(*Ineternet Message Access Protocol*-protocolo de acceso a mensajes de Internet): permite acceder a los mensajes de correo electrónico almacenados en los servidores SMTP. Permite que los usuarios accedan a su correo desde cualquier equipo que tenga una conexión a internet. Tiene alguna ventaja sobre POP, poer ejemplo, los mensajes continúan siempre almacenados en el servidor o los usuarios pueden organizar los mensajes en carpetas.