Actualmente debido a la proliferación de ordenadores personales y al uso de Internet, existen múltiples ordenadores que colaboran entre sí, comunicándose a través de la red, que llamamos sistemas informáticos distribuidos. Todo el mundo quiere estar conectado e informado al momento y cada vez más desde los teléfonos móviles (jugar on-line, consultar contenidos multimedia, información actualizada, etc…).

Las características de un sistema informático distribuido son:

1. Tener más de un elemento informático distinto e independiente. Como puede ser un procesador en un ordenador dedicado a una única tarea, un ordenador dentro de una red, una aplicación funcionando a través de internet, etc… Ninguno de estos elementos comparten memoria con el resto.
2. Los elementos no están sincronizados entre ellos.
3. Todos los elementos del sistema están conectados a una red de comunicaciones.

Algunos de los sistemas distribuidos son: Cloud Computing (la nube), DropBox, Spotify, Google Docs/Drive, …

Para que las diferentes aplicaciones que forman un sistema distribuido puedan comunicarse, existen una serie de mecanismos que lo permiten, tanto a nivel de hardware (interfaces de red, routers o encaminadores, …) como de software (herramientas, bibliotecas de programación, componentes del Sistema Operativo, …)

El protocolo estándar de comunicaciones a través de Internet de los equipos informáticos es el protocolo TCP/IP. Que básicamente se encarga de dividir el mensaje en paquetes para ser enviados por medio de la red informática hacia otro u otros equipos.

Este protocolo tiene 4 capas o niveles, estructurados de forma jerárquica y cada una de estas capas tiene una tarea muy específica y que depende de los otros niveles para su perfecto funcionamiento. Están aquí indicados del nivel más transparente para el usuario al más complejo y que tiene directamente contacto con el hardware. Son:

1. **Aplicación**, en él están las aplicaciones de comunicación para los usuarios como pueden ser: **ftp**, **smtp** (Simple Mail Transfer Protocol), **telnet**, **http**, …
2. **Transporte**, son elementos del software cuya función es crear el canal de comunicación, descomponer el mensaje en paquetes y gestionar la transmisión entre el emisor y el receptor. Nos da el servicio de comunicaciones y utiliza dos tipos de protocolos que son TCP y UDP.

   TCP, garantiza que los datos enviados desde un ordenador llegan correctamente al equipo indicado y en el mismo orden en que se enviaron, de lo contrario envía un mensaje de error.

   UDP, envía los paquetes de datos independientes, llamados **datagramas**, y no controla ni el orden de recepción ni garantiza su llegada.
3. **Red**, se encarga de elegir la mejor ruta para enviar los paquetes por la red. Su principal protocolo es IP (protocolo de internet).
4. **Enlace**, recibe la información del nivel anterior y lo transmite al hardware de la red informática.

El programador en la mayoría de los casos no tiene que prestar atención a los distintos niveles de este protocolo, tan solo utiliza el de aplicación.

Java dispone de clases para establecer conexiones, crear servidores, enviar y recibir datos. Los hilos nos van a permitir la manipulación simultánea de múltiples conexiones.

La librería **java.net** nos proporciona las clases para implementar nuestras aplicaciones de red. Podemos dividirlas en dos secciones:

a) **API de bajo nivel**, que se encarga de las direcciones IP, Sockets y de los interfaces.
b) **API de alto nivel**, se encargan de las URI(identificador de recurso universal), URL(localizador de recurso universal) y de las conexiones.

Veamos algunas de las clases de la librería java.net:

a. Clase **InetAddress**, está relacionada con las direcciones IP, podríamos especificar si es de tipo 4 o 6 (Inet4Adress ó Inet6Adress), pero no hace falta indicarlo. La excepción que podemos tener es **UnKnownHostName**. Algunos de los métodos que tenemos son:

   - **getLocalHost( )**, nos devuelve el nombre y la IP del equipo.
   - **getByName(String IP)**, devuelve el nombre de la máquina y la IP.
   - **InetAddress[ ] getAllByName(String IP)**, devuelve un array de objetos, son las distintas IPs que tiene la máquina indicada.
   - **String getHostAddress( )**, devuelve la IP como una cadena.
   - **String getHostName( )**, devuelve el nombre del host.
   - **String getCanonicalHostName( )**, devuelve el nombre canónico completo de la máquina.

   Ejemplo IP.

b) Clase **URL**, es un puntero a un recurso WEB. Una URL se puede descomponer en los siguientes elementos:

   Protocolo ://nombredelamaquina:puerto/directorio/fichero
   http://www.google.es:80/BCUM/carpeta/trabajo.html

Por defecto el protocolo http suele utilizar el puerto 80.

La clase URL tiene varios constructores definidos, algunos de ellos son:

- **URL(String url)**, crea un objeto url a partir de la String dada.
- **URL(String protocolo, String host, String fichero)**, crea el objeto url con los datos dados.

Los constructores no comprueban que la URL exista o sea correcta.
La excepción que puede aparecer es **MalformedURLException**.

Veamos algunos de los métodos que podemos utilizar en esta clase:

- **int getPort( )**, devuelve el puerto utilizado, si lo indicas en la URL. Devolverá -1 si el puerto no viene especificado.
- **Int getDefaultPort( )**, nos devuelve el puerto que el utiliza.
- **String getPath( )**, devuelve la ruta.
- **String getUserInfo( )**, devuelve el usuario o la palabra null sino existe usuario.
- **String getQuery( )**, devuelve null si no existe consulta.
- **String getProtocol( )**,nos da el protocolo de la URL (http, https,..).
- **String getHost( )**, da la dirección de la página.
- **String getFile( )**, nos da el fichero a consultar con su ruta si la hemos puesto.
- **String getAuthority( )**, suele dar la dirección de la página.

Ejemplo: EjemploURL y EjemploURL2.

c. Clase **URLConnection**, para poder establecer conexión con un equipo. Nos permite la comunicación entre la aplicación y la URL. Podemos leer o escribir en el recurso referenciado. La excepción que puede producirse es **IOException**.

openConnection( ), obtenemos una conexión al objeto URL indicado. Esta instancia se puede utilizar tanto para leer como para escribir al recurso referenciado por la URL.

La diferencia entre este ejemplo y el anterior EjemploURL, es que este abre una conexión con la URL y el otro abre un stream desde la URL.

Ejemplo: EjemploConexion2.

d. **Sockets**, es el mecanismo de comunicación fundamental para la transferencia de información entre aplicaciones a través de las redes internas o Internet. Nos proporcionan los puntos de comunicación entre aplicaciones y procesos. Necesitamos una IP y el puerto local por el que se van a comunicar e identificar el proceso. El cliente manda un mensaje desde cualquier puerto y los protocolos para el envío de la información será TCP o UDP.
El cliente solicitara la petición de comunicación y el servidor la tendrá que aceptar para establecer la comunicación. Una vez establecida la comunicación de ambos se crea un socket de comunicación.
En función del tipo de protocolo que utilicemos para la comunicación tendremos:

- ✓ Conexión fiable con TCP, garantiza la entrega de los paquetes de información y en el orden deseado, esta comunicación tiene acuse de recibo; si en un tiempo no ha llegado la información la vuelve a enviar de nuevo. La comunicación se establece mediante una canal que es el Stream.
  Para comunicarnos debemos realizar:
  - Una petición del cliente de conexión al servidor.

- Aceptación de la conexión por parte del servidor.

Según la herramienta que utilicemos tenemos reservados por defecto una serie de puertos: FTP es el puerto 21, HTTP utiliza el puerto 80, Telnet emplea el puerto 23 y SMTP usa el puerto 25.

Tenemos la clase **Socket** para el cliente y **ServerSocket** para el servidor.

Clase ServerSocket

Se utiliza para implementar el extremo de la conexión que corresponde al servidor, donde se crea un conector en el puerto servidor que escucha las peticiones de conexión.

Algunos de los constructores de esta clase lanzan la excepción **IOException**, son:

- ✓ **SeverSocket( )**, crea un socket sin ningún puerto asociado.
- ✓ **ServerSocket(int puerto)**, crea un socket e indicamos porque puerto.
- ✓ **ServerSocket(int puerto, int máximo)**, crea el socket asociado a un puerto e indica el número máximo de peticiones de conexiones.
- ✓ **ServerSocket(int puerto, int máximo, InetAddress dirección)**, además de hacer lo del constructor anterior indica la dirección IP local.

Algunos de los métodos que tenemos son:

- **Socket accept( )**, escucha una solicitud de conexión de un cliente y la acepta. Una vez establecida la conexión devuelve un objeto tipo **Socket** por el que se creara la conexión. El **ServerSocket** sigue disponible para aceptar nuevas conexiones.
- **bind(SocketAddress)**, asigna al socket una dirección y el puerto indicado.
- **close( )**, cierra el **ServerSocket**.
- **int getLocalPort( )**, devuelve el puerto local al que esta enlazado el **ServerSocket**.

Clase Socket

Es la clase para el cliente, algunos constructores y métodos son iguales y tiene otros distintos.

Algunos de los constructores:

- **Socket( )**, crea un socket sin ningún puerto asociado.
- **Socket(String host, int port)**, crea un socket al puerto indicado y al nombre del host indicado. Puede lanzar las excepciones **UnKnownHostException**, **IOException**.
- **Socket(InetAddress ip, int port)**, indicamos la ip y el puerto, para conectarnos.

Veamos algunos de los métodos:

- **InputStream getInputStream( )**, permite leer bytes desde el socket.
- **OutputStream getOutputStream( )**, permite escribir bytes sobre el socket
- **InetAddress getInetAddress( )**, devuelve la dirección IP y el puerto al que está conectado.
- **Int getPort( )**, devuelve el puerto remoto al que está conectado.
- **Int getLocalPort( )**.

- **Connet(SocketAddress addr)**, establece la conexión con la dirección y el puerto destino.
- **close( )**.

Los puertos TCP van del número 0 al 65535. Los puertos del 0 al 1023 están reservados para servicios privilegiados. Los puertos del 1024 al 49151, están reservados para aplicaciones concretas (por ejemplo el 3306 lo suele utilizar MySQL y el 1521 Oracle) y los puertos del 49152 al 65535 no están reservados para ninguna aplicación en concreto.

DataInputStream, permite la lectura de líneas de texto y tipos primitivos de Java. Podemos utilizar los métodos **readInt( )**, **readDouble( )**, **readLine( )**, **readUTF( )**,…

DataOutputStream, permite la escritura de tipos primitivos de Java. Podemos utilizar los métodos **writeInt( )**, **writeDouble( )**, **writeLine( )**, **writeUTF( )**,…

Debemos cerrar primero los streams relacionados con los sockets antes que el socket.

Ejemplo: EjemploServidor y EjemploCliente.

- ✓ Con UDP la conexión no es fiable, al no garantizar la llegada de la información, ni el orden correcto de esta. Simplemente envía la información sin haber establecido conexión previa. Este tipo se utiliza por rapidez y cuando la información es corta (64 Kb como máximo) y solo se envía un datagrama. Es muy utilizado.

  Cada vez que se envía un datagrama el emisor debe indicar la dirección IP y el puerto de destino de cada paquete y el receptor debe extraer la IP y el puerto del emisor del paquete.

  Cada paquete del datagrama tiene: una secuencia de bytes del mensaje, la longitud del mensaje, la dirección IP destino y el puerto destino.
  Las herramientas que utilizan este tipo de envió son: NFS(Network File System), DNS(Domain Name Server) y SNMP(Simple Network Management Protocol).
  Para ello tenemos dos clase que son: **DatagramSocket** y **DatagramPacket**.

Clase DatagramPacket

Es para los datagramas recibidos y para los datagramas enviados. Los constructores que podemos tener son:

- ❖ **DatagramPacket(byte[] buffer, int length)**, para datagramas recibidos. Indicamos la cadena de bytes donde va el mensaje y la longitud de la misma.
- ❖ **DatagramPacket(byte[] buffer, int length, InetAddress host, int port)**, además de lo anterior indicamos la dirección destino y el número de puerto.

Algunos de los métodos que tenemos:

- **InetAddress getAddress()**, devuelve la dirección IP del host al que se le envía o recibe el datagrama.
- **Byte[] getData()**, devuelve el contenido del datagrama enviado o recibido.
- **int getLength()**, devuelve la longitud de los datos a enviar o recibir.
- **int getPort()**, devuelve el puerto de la máquina del cual recibimos o enviamos el datagrama.
- **setAddress(InetAddress ip)**, establece la dirección IP de la máquina a la que se envía el datagrama.
- **setData(byte[buffer])**, establece el buffer de datos para este paquete.
- **setLength(int length)**, ajusta la longitud del paquete.
- **setPort(int Port)**, establece el número de puerto del host remoto al que este datagrama envía.

Clase DatagramSocket

Da soporte para el envío y recepción de datagramas UDP, algunos de los constructores de esta clase:

- **DatagramSocket( )**, construye un socket para el envío del datagrama y elige un puerto libre.
- **DatagramSocket(int port)**, construye el socket al puerto indicado.
- **DatagramSocket(int port, InetAddress ip)**, indicamos el puerto y la dirección ip.

pueden lanzar la excepción **SocketException**.

Algunos de los métodos que podemos utilizar son:

- **receive(DatagramPacket paquete)**, recibe el datagrama.
- **send(DatagramPacket paquete)**, envía un datagrama.
- **connect(InetAddress ip, int port)**, conecta el socket a los datos indicados.
- **setSoTimeout(int timeout)**, establece un tiempo de espera límite, el valor está en milisegundos. Y el método **receive()** se bloquea durante el tiempo indicado. Se puede producir la excepción **InterruptedIOException**.

Ejemplo: servidorUDP y clienteUDP.

Multicast, es cuando realizamos operaciones de comunicación en grupo. Se envía el mismo mensaje a varios destinatarios de forma simultánea utilizando una dirección IP especial multicast. Todos los procesos receptores deben tener la misma IP multicast. En IPv4 se reservan una serie de direcciones específicas que van de la 224.0.0.0 a la 239.255.255.255. Este tipo de comunicación supone un riesgo para la seguridad de la

red, ya que cualquiera puede enviar cientos de mensajes y saturar a todos los miembros del grupo. Multicast está restringido en Internet y solo se usa en redes de área local.

Tenemos la clase MulticastSocket, que lanza la excepción IOException.

Algunos de los constructores:

- **MulticastSocket( )**, elige un puerto de los que estén libres.
- **MulticastSocket(int port)**, lo conecta al puerto indicado.

Algunos de los métodos que podemos tener son:

- **joinGroup(InnetAddress)**, para unirse al grupo multicast.
- **leaveGroup(InnetAddress)**, un elemento abandona el grupo.
- **send(DatagramPacket p)**, envía el datagrama al grupo.
- **receive(DatagramPacket p)**, recibe el datagrama del grupo.

Tenemos 4 tipos de envíos de mensajes:

a) **Unicast**, es el envío de mensajes entre emisor y receptor, con Socket TCP o UDP.
b) **Multicast**, envío de mensajes a un grupo con una dirección específica.
c) **Broadcast**, es el envío simultaneo de un mensaje a todos los miembros de la red local.
d) **Anycast**, solo se envía el mensaje al destinatario más cercano en la red.

Ejemplo: ServidorMultiCast.java y ClienteMultiCast.java

Podemos enviar objetos a través de los Sockets TCP. Para ello se utilizará la clase **ObjectInputStream** y **ObjectOutputStream**, y los métodos **readObject( )** para leer y **writeObject( )** para escribir el objeto al **stream**.

```
ObjectOutputStream objetosalida = new ObjectOutputStream(socket.getOutputStream());

ObjectInputStream objetosalida = new ObjectInputStream(socket.getIntputStream());
```

Lo ideal sería que el servidor pueda atender a múltiples clientes de forma simultánea, para ello deberíamos utilizar programación multihilo, para que cada cliente sea atendido por un hilo.

Tipos de redes

Según la IP las redes pueden ser clasificadas en 5 grupos según IANA(Internet Assigned Numbers Authority) para IPv4, es de 32 bits. Ambas trabajan con 4 octetos para ser identificadas. Los grupos son:

1. **Clase A**, solo utiliza los 8 primeros bit para identificar las redes, puede distinguir 126 redes y tener 16.777.214 ordenadores conectados. Su numeración va para el primer octeto es del 0 al 126.
2. **Clase B**, utiliza 16 bits o dos octetos para identificar las redes, con ello podemos tener 16.384 redes distintas y tener conectados 65.534 ordenadores. La IP empieza por 128.0 y puede llegar hasta el 191.255.
3. **Clase C**, emplea tres octetos para las redes y el último octeto para ordenadores. Podemos tener 2.097.152 redes y 254 ordenadores por red. Sus IPs van del 192.0.1 al 223.255.255.
4. **Clase D**, emplea todos los octetos para la red y van del 224.0.0.0 al 239.255.255.255. Este tipo de red está reservada para el multicast (envía un paquete único a múltiples destinos).
5. **Clase Y**, es similar a la de tipo D y sus IPs van del 240.0.0.0 al 255.255.255.255, está reservada por IANA para uso futuro.

La IPs 127 estan reservadas para pruebas internas del equipo. Como es la 127.0.0.1 que es la del propio equipo.

IPv6, utiliza 64 bits y tiene 4 octetos de 4 dígitos en hexadecimal para poder tener acceso a más direcciones.

ECO ENTRANTE ver documento