3.3. QUE SON LOS SOCKETS

Los protocolos TCP y UDP utilizan el concepto de sockets para proporcionar los puntos extremos de la comunicación entre aplicaciones o procesos. La comunicación entre procesos consiste en la transmisión de un mensaje entre un conector de un proceso y un conector de otro proceso, a este conector es a lo que llamamos socket.

Para los procesos receptores de mensajes, su conector debe tener asociado dos campos:

■ La dirección IP del host en el que la aplicación está corriendo.

■ El puerto local a través del cual la aplicación se comunica y que identifica el proceso.

Así, todos los mensajes enviados a esa dirección IP y a ese puerto concreto llegaran al proceso receptor. La Figura 3.2 muestra un proceso cliente (envía un mensaje) y un proceso servidor (recibe un mensaje) comunicándose mediante sockets. Cada socket tiene un puerto asociado, el proceso cliente debe conocer el puerto y la IP del proceso servidor. Los mensajes al servidor le deben llegar al puerto acordado. El proceso cliente podrá enviar el mensaje por el puerto que quiera.

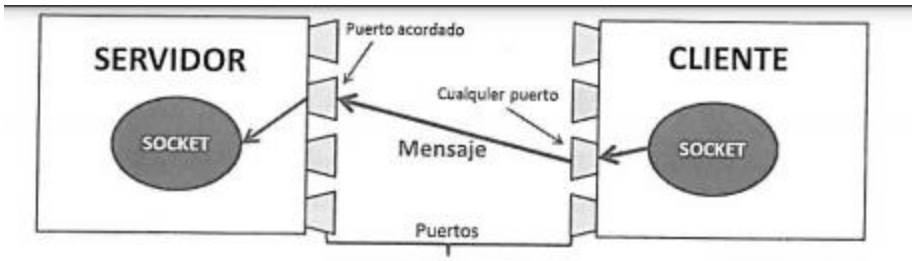

Figura 3.2. Socket y puertos.

Los procesos pueden utilizar un mismo conector tanto para enviar como para recibir mensajes. Cada conector se asocia con un protocolo concreto que puede ser UDP o TCP.

3.3.1. FUNCIONAMIENTO EN GENERAL DE UN SOCKET

Un puerto es un punto de destino que identifica hacía que aplicación o proceso deben dirigirse los datos. Normalmente en una aplicación cliente-servidor, el programa servidor se ejecuta en una maquina específica y tiene un socket que está unido a un numero de puerto especifico, El servidor queda a la espera "escuchando" las solicitudes de conexión de los clientes sobre ese puerto.

El programa cliente conoce el nombre de la maquina en la que se ejecuta el servidor y el número de puerto por el que escucha las peticiones. Para realizar una solicitud de conexión, el cliente realiza la petición a la maquina a través del puerto, el cliente también debe identificarse ante el servidor por lo que durante la conexión se utilizará un puerto local asignado por el sistema,

Si todo va bien, el servidor acepta la conexión. Una vez aceptada, el servidor obtiene un nuevo socket sobre un puerto diferente. Esto se debe a que por un lado debe seguir atendiendo las peticiones de conexión mediante el socket original y por otro debe atender las necesidades del cliente que se conectó.

En el lado del cliente, si se acepta la conexión, se crea un socket y el cliente puede utilizarlo para comunicarse con el servidor. Este socket utiliza un numero de puerto diferente al usado para conectarse al servidor. El cliente y el servidor pueden ahora comunicarse escribiendo y leyendo por sus respectivos sockets.

3.4. TIPOS DE SOCKETS

Hay dos tipos básicos de sockets en redes IP: los que utilizan el protocolo TCP, orientados a conexión; y los que utilizan el protocolo UDP, no orientados a conexión.

3.4.1. SOCKETS ORIENTADOS A CONEXION

La comunicación entre las aplicaciones se realiza por medio del protocolo TCP. Por tanto, es una conexión fiable en la que se garantiza la entrega de los paquetes de datos y el orden en que fueron enviados. TCP utiliza un esquema de acuse de recibo de los mensajes de tal forma que, si el emisor no recibe dicho acuse dentro de un tiempo determinado, vuelve a transmitir el mensaje.

Los procesos que se van a comunicar deben establecer antes una conexión mediante un stream. Un stream es una secuencia ordenada de unidades de información (bytes, caracteres, etc.) que puede fluir en dos direcciones: hacia fuera de un proceso (de salida) o hacia dentro de un proceso (de entrada). Están diseñados para acceder a los datos de manera secuencial.

Una vez establecida la conexión, los procesos leen y escriben en el stream sin tener que preocuparse de las direcciones de Internet ni de los números de puerto. El establecimiento de la conexión implica:

- Una petición de conexión desde el proceso cliente al servidor.
- Una aceptación de la conexión del proceso servidor al cliente.

Los sockets TCP se utilizan en la gran mayoría de las aplicaciones IP. Algunos servicios con sus números de puerto reservados son: FIT (puerto 21), Telnet (23), HTTP (80), SMTP (25).

En Java hay dos tipos de stream sockets que tienen asociadas las clases Socket para implementar el cliente y ServerSocket para el servidor.

3.4.2. SOCKETS NO ORIENTADOS A CONEXION

En este tipo de sockets la comunicación entre las aplicaciones se realiza por medio del protocolo UDP. Esta conexión no es fable y no se garantiza que la información enviada llegue a su destino, tampoco se garantiza el orden de llegada de los paquetes que puede llegar en distinto orden al que se envía. Los datagramas se transmiten desde un proceso emisor a otro receptor sin que se haya establecido previamente una conexión, sin acuse de recibo ni reintentos.

Cualquier proceso que necesite enviar o recibir mensajes debe crear primero un conector asociado a una dirección IP y a un puerto local. El servidor enlazara su conector a un puerto de servidor conocido por los clientes. El cliente enlazara su conector a cualquier puerto local libre. Cuando un receptor recibe un mensaje, se obtiene además del mensaje, la dirección IP y el puerto del emisor, permitiendo al receptor enviar la respuesta correspondiente al emisor.

Los sockets UDP se usan cuando una entrega rápida es más importante que una entrega garantizada, o en los casos en que se desea enviar tan poca información que cabe en un unto datagrama. Se usan en aplicaciones para la transmisión de audio y video en tiempo real donde no es posible el reenvío de paquetes retrasados; algunas aplicaciones como NFS (Network File

System), DNS (Domain Name Server) o SNMP (Simple Network Management Protocol) usan este protocolo.

Para implementar en Java este tipo de sockets se utilizan las clases DatagramSocket y DatagramPacket.

3.5. CLASES PARA SOCKETS TCP

El paquete java.net proporciona las clases ServerSocket y Socket para trabajar con sockets TCP. TCP es un protocolo orientado a conexión por lo que para establecer una comunicación es necesario especificar una conexión entre un par de sockets. Uno de los sockets, el cliente, solicita una conexión, y el otro socket, el servidor, atiende las peticiones de los clientes. Una vez que los dos sockets estén conectados, se pueden utilizar para transmitir datos en ambas direcciones.

Clase ServerSocket.

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

Algunos de los constructores de esta clase son (pueden lanzar la excepción JOException):

CONSTRUCTOR	MISIÓN
ServerSocket()	Crea un socket de servidor sin ningún puerto asociado.
ServerSocket(int port)	Crea un socket de servidor, que se enlaza al puerto especificado.
ServerSocket(int port, int máximo)	Crea un socket de servidor y lo enlaza con el número de puerto local especificado. El parámetro *máximo* especifica, el número máximo de peticiones de conexión que se pueden mantener en cola.
ServerSocket(int port, int máximo, InetAddress direc)	Crea un socket de servidor en el puerto indicado, especificando un máximo de peticiones y conexiones entrantes y la dirección IP local.

Algunos métodos importantes son:

MÉTODOS	MISIÓN
Socket accept ()	El método **accept()** escucha una solicitud de conexión de un cliente y la acepta cuando se recibe. Una vez que se ha establecido la conexión con el cliente, devuelve un objeto de tipo **Socket**, a través del cual se establecerá la comunicación con el cliente. Tras esto, el **ServerSocket** sigue disponible para realizar nuevos ***accept()***. Puede lanzar ***IOException***.
close ()	Se encarga de cerrar el **ServerSocket**.
int getLocalPort ()	Devuelve el puerto local al que está enlazado el **ServerSocket**.

El siguiente ejemplo crea un socket de servidor y lo enlaza al puerto 6000, visualiza el puerto por el que se esperan las conexiones y espera que se conecten 2 clientes:

```
int Puerto= 6000;// Puerto

ServerSocket Servidor= new ServerSocket(Puerto);

System.out.println("Escuchando en " + Servidor.getLocalPort());
```

Socket cliente 1= Servidor.accept();//esperando a un cliente

//realizar acciones con cliente1

Socket cliente 2 = Servidor.accept(); //esperando a otro cliente

//realizar acciones con cliente2

Servidor.close(); //cierro socket servidor

Clase Socket.

La clase Socket implementa un extremo de la conexion TCP. Algunos de sus constructores son (pueden lanzar la excepcion I0Exception):

CONSTRUCTOR	MISIÓN
Socket()	Crea un socket sin ningún puerto asociado.
Socket (InetAddress address, int port)	Crea un socket y lo conecta al puerto y dirección IP especificados.
Socket(InetAddress address, int port, InetAddress localAddr, int localPort)	Permite además especificar la dirección IP local y el puerto local a los que se asociará el socket.
Socket (String host, int port)	Crea un socket y lo conecta al número de puerto y al nombre de host especificados. Puede lanzar ***UnKnownHostException, IOException***

Algunos métodos importantes son:

MÉTODOS	MISIÓN
InputStream getInputStream ()	Devuelve un **InputStream** que permite leer bytes desde el socket utilizando los mecanismos de streams, el socket debe estar conectado. Puede lanzar *IOException*.
OutputStream getOutputStream ()	Devuelve un **OutputStream** que permite escribir bytes sobre el socket utilizando los mecanismos de streams, el socket debe estar conectado. Puede lanzar ***IOException***.

El siguiente ejemplo crea un socket cliente y lo conecta al host local al puerto 6000 (tiene que haber un ServerSocket escuchando en ese puerto). Despues visualiza el puerto local al que esta conectado el socket, y el puerto, host y direction IP de la maquina remota a la que se conecta (en este caso es el host local):

String Host = "localhost";

int Puerto = 6000;//puerto remoto

// ABRIR SOCKET

Socket Cliente = new Socket(Host, Puerto);//conecta

```
InetAddress i= Cliente.getInetAddress();
System.out.println("Puerto local:  "+ Cliente.getLocalPort());
System.out.println ("Puerto Remoto:  "+ Cliente.getPort());
System.out.println("Nombre Host/IP:  "+ Cliente.getInetAddress());
System.out.println("Host Remoto:  "+ i.getHostName().toString());
System.out.println("IP Host Remoto:  "+ i.getHostAddress().toString());

Cliente.close();// Cierra el socket
```

La salida que se genera es la siguiente:

```
Puerto local:    63120
Puerto Remoto:          6000
Nombre Host/IP: localhost/127.0.0.1 Host Remoto: localhost
IP Host Remoto:          127.0.0.1
```

ACTIVIDAD 3.2

Realiza un programa servidor TCP que acepte dos clientes. Muestra por cada cliente conectado sus puertos local y remoto.

Crea también el programa cliente que se conecte a ese servidor. Muestra los puertos locales y remotos a los que está conectado su socket, y la dirección IP de la maquina remota a la que se conecta.