SOCKETS

Comunicaciones en Java.

Como le voy a hacer para comunicar algo en 9 dias con java, me lleva la búrguer, pero bueno, máximo esfuerzo  motivación, hay que motivarse :c, animo.

Tengo que estudiar los temas que hay en el temario. Eso dijo el profe

Comunicaciones en UNIX

El sistema de Entrada/Salida de Unix sigue el paradigma que normalmente se designa como Abrir-Leer-Escribir-Cerrar. Antes de que un proceso de usuario pueda realizar operaciones de entrada/Salida, debe hacer una llamada a Abrir(**open**) para indicar, y obtener permisos para su uso, el fichero o dispositivo que quiere utilizar. Una vez que el objeto esta abierto, el proceso de usuario realiza una o varias llamadas a Leer (**read**) y Escribir (**write**), para conseguir leer y escribir datos. Leer coge datos desde el objeto y los transfierre al proceso de usuario, mientras que Escribir transfiere datos desde el proceso de usuario al objeto. Una vez que todos estos intercambios de información estén concluidos, el proceso de usuario llamara a Cerrar (**close**) para informar al sistema operativo que ha finalizado la utilización del objeto que antes había abierto.

Cuando se incorporan las características a Unix de comunicación entre procesos (**IPC**) y el manejo de redes, la idea fue implementar la interface con IPC similar a la que se estaba utilizando para la entrada/salida de ficheros, es decir, siguiendo el paradigma del párrafo anterior. En Unix, un proceso tiene un conjunto de descriptores de entrada/salida desde donde Leer y por donde Escribir. Estos descriptores pueden estar referidos a ficheros, dispositivos, o canales de comunicaciones (sockets). El ciclo de vida de un descriptor, aplicando a un canal de comunicación (socket), esta determinado por tres fases (siguiendo el paradigma):

- Creación, apertura del socket
- Lectura y Escritura, recepción y envio de datos por el socket
- Destrucción, cierre del socket.

La interface IPC en Unix-BSD esta implementada sobre los protocolos de red TP y UDP. Los destinatarios de los mensajes se especifican como direcciones de socket; cada dirección de socket es un identificador de comunicacio que consiste en una dirección Internet y un numero de puerto.

Las operaciones IPC e basan en pares de sockets. Se intercambian información transmitiendo datos a través de mensajes que circulan entre un socket en un proceso y otro socket en otro proceso. Cuando los mensajes son enviados, se encolan en el socket hasta que el protocolo de red los haya tansmitido. Cuando llegan, los mensajes son encolados en el socket de recepción hasta que el proceso que tiene que recibirlos haga las llamadas necesarias para recoger esos datos.

SOCKETS

Los sockets son puntos finales de enlaces de comunicaciones entre procesos. Los procesos los tratan como descriptores de ficheros, de forma que se pueden intercambiar datos con otros procesos transmitiendo y recibiendo a través de sockets.

El tipo de sockets describe la forma en la que se transfiere información a través de ese socket.

**Sockets Stream (TCP, Transport Control Protocol)**

Son un servicio orientado a conexión donde los datos se transfieren sin encuadrarlos en registro o bloques. Si se rompe la conexión entre los procesos, estos serán informados.

El protocolo de comunicaciones con streams es un protocolo orientado a conexión, ya que para establecer una comunicación utilizando el protocolo TCP, hay que establecer en primer lugar una conexión entre un par de sockets. Mientras uno de los sockets atiende peticiones de conexión (servidor), el otro solicita una conexión (cliente). Una vez que los dos sockets estén conectados, se pueden utilizar para transmitir datos en ambas direcciones.

**Sockets Datagrama (UDP, User Datagram Protocol)**

Son un servicio de transporte sin conexión. Son mas eficientes que TCP, pero no esta garantizada la fiabilidad. Los datos se envían y reciben en paquetes, cuya entrega no esta garantizada. Los paquetes pueden ser duplicados, perdidos o llegar en un orden diferente al que se envio.

El protocolo de comunicaciones con datagramas es un protocolo sin conexión, es decir, cada vez que se envíen datagramas es necesario enviar el descriptor del socket local y la dirección del socket que debe recibir el datagrama. Como se puede ver, hay que enviar datos adicionales cada vez que se realice una comunicación.

**Socket Raw**

Son sockets que dan acceso directo a la capa de software de red subyacente o a protocolos de mas bajo nivel. Se utilizan sobre todo para la depuración del código de los protocolos

**Diferencias entre sockets stream y datagrama**

Ahora se nos presenta un problema, ¿Qué protocolo, o tipo de sockets, debemos usar – UDP o TCP? La decisión depende de la aplicación cliente/servidor que estemos escribiendo. Vamos a ver algunas diferencias entre los protocolos para ayudar en la decisión.

En UDP, cada vez que se envía un datagrama, hay que enviar también el descriptor del socket local y la dirección del socket que va a recibir el datagrama, luego estos son mas grandes que los TCP. Como el protocolo TCP está orientado a conexión, tenemos que establecer esta conexión entre los dos sockets antes de nada, lo que implica un cierto tiempo empleado en el establecimiento de la conexión, que no existe en UDP.

En UDP hay un limite de tamaño de los datagramas, establecido en 64kilobytes, que se pueden enviar a una localización determinada, mientras que TCP no tiene limite;  una vez que se ha establecido la conexión, el par de sockets funciona como los streams: todos los datos se leen inmediatamente, en el mismo orden en que se van recibiendo.

UDP es un protocolo desordenado, no garantiza que los datagramas que se hayan enviado sean recibidos en el mismo orden por el socket de recepción. Al conrario, TCP es un protocolo ordenado, garantiza que todos los paquetes que se envíen serán recibidos en el socket destino en el mismo orden en que se han enviado.

Los datagramas son bloques de información del tipo lanzar y olvidar. Para la mayoría de los programas que utilicen la red, el usar un flujo TCP en vez de un datagrama UDP es mas sencillo y hay menos posibilidades de tener problemas. Sin embargo, cuando se requiere un rendimiento optimo, y esta justificando el tiempo adicional que supone realizar la verificación de los datos, los datagramas son un mecanismo realmente útil.

En resumen, TCP parece mas indicado para la implementación de servicios de red como un control remoto (rlogin, telnet) y transmisión de ficheros (ftp): que necesitan transmitir datos de longitud indefinida. UDP es menos complejo y tiene una menor sobrecarga sobre la conexión; esto hace que sea el indicado en la implementación de aplicaciones cliente/servidor en sistemas distribuidos montados sobre redes de área local.

**USO DE SOCKETS**

Podemos pensar que un Servidor Intenet es un conjunto de sockets que proporciona capacidades adicionales del sistema, los llamados servicios.

**Puertos y Servicios**

Cada servicio esta asociado a un puerto. Un puerto es una dirección numérica a través de la cual se procesa el servicio. Sobre un sistema Unix, los servicios que proporciona este sistema se indican en el fichero /etc/services, y algunos ejemplos son:

```
daytime 13/udp
ftp     21/tcp
telnet  23/tcp      telnet
smtp    25/tcp      mail
http    80/tcp
```

La primera columna indica el nombre del servicio. La segunda columna indica el puerto y el protocolo que está asociado al servicio. La tercera columna es un alias del servicio; por ejemplo, el servicio smtp, también conocido como mail, es la implementación del servicio de correo electrónico.

Las comunicaciones de información relacionada con Web tienen lugar a través del puerto 80 mediante protocolo TCP. Para emular esto en Java, usaremos la clase socket. La fecha (daytime). Sin embargo, el servicio que coge la fecha y la hora del sistema, esta ligado al puerto 13 utilizando el protocolo UDP. Un servidor que lo emule en Java usaría un objeto DatagramSocket.

**LA CLASE URL**

La clase URL contiene constructores y métodos para la manipulación de URL (Universal Resource Locator): un objeto o servicio en Internet. El protocolo TCP necesita dos tipos de información: la dirección IP y el numero de puerto. Vamos a ver como podemos recibir pues la pagina Web principal de nuestro buscador favorito al teclear:

http://www.yahoo.com

En primer lugar, yahoo tiene registrado su nombre, permitiendo que se use yahoo.com como su dirección IP, o lo que es lo mismo, cuando indicamos yahoo.com es como si hubiésemos indicado 205.216.146.71, su dirección IP real.

La verdad es que la cosa es un poco mas complicada que eso. Hay un servicio, el DNS (Domain Name Service), que traslada www.yahoo.com a 205.216.146.71, lo que nos permite teclear www.yahoo.com, en lugar de tener que recordar su dirección IP.

Si queremos obtener la dirección IP real de la red en que estamos corriendo, podemos realizar llamadas a los métodos getLocalHost() y getAddress(). Primero, getLocalHost() nos devuelve un

objeto iNEtAddress, que si usamos con getAddress() generara un array con los cuatro bytes de la dirección IP, por ejemplo:

```
InetAddress dirección = InetAddress.getLocalHost();

Byte direccionIp[] = dirección.getAddress();
```

Su la dureccion de la maquina en que estamos corriendo es 150.150.112.145, entonces:

```
direccionIP[0] = 150

direccionIP[0] = 150

direccionIP[0] = 112

direccionIP[0] = 145
```

Una cosa interesante en este punto es que una red puede mapear muchas direcciones IP. Esto puede ser necesario para un Servidor Web, como Yahoo, que tiene que soportar grandes cantidades de trafico y necesita mas de una dirección IP para poder atender a todo ese trafico. El nombre interno para la dirección 205.216.146.71, por ejemplo, es www7.yahoo.com El DNS puede trasladar una lista de direcciones IP asignadas a Yahoo en www.yahoo.com. Esto es una cualidad útil, pero por ahora abre un agujero en cuestión de seguridad.

Ya conocemos la dirección IP, nos falta el numero del puerto. Si no se indica nada, se utilizara el que se haya definido por defecto en el dichero de configuración de los servicios del sistema. En Unix se indican en el fichero /etc/services, en Windows-NT en el dichero services y en otros sistemas puede ser diferente.

El puerto habitual de los servicios Web es el 80, así que si no indicamos nada, entraremos en el servidor de Yahoo por el puerto 80. Si tecleamos la URL siguiente en un navegador:

```
http://www.yahoo.com:80
```

también recibiremos la pagina principal de Yahoo. No hay nada que nos impida cambiar el puerto en el que residirá el servidor Web; sin embargo, el uso del puerto 80 es casi estándar, porque elimina pulsaciones en el teclado y, además, las direcciones URL son lo suficientemente difíciles de recordar como para añadirle encima el numero del puerto.

Si necesitamos otro protocolo, como:

```
ftp://ftp.microsoft.com
```

el puerto se derivara de ese protocolo. Así el puerto FTP de Microsoft es el 21, según su fichero services. La primera parte, antes de los dos puntos, de la URL, indica el protocolo que se quiere utilizar en la conexión con el servidor. El protocolo http (HyperText Transmission Protocol), es el utilizado para manipular documentos Web. Y si no se especifica ningún documento, muchos servidores están configurados para devolver un documento de nombre index.html

Con todo esto, Java permite los siguientes cuatro constructores para la clase URL:

Public URL( String spec ) throws MalformedURLException;

Public URL( String protocol, String host, int port, String file ) throws MAlformedURLException;

Public URL( String protocol, String host, String file ) throws MalformedURLException;

Public URL( URL context, String spec ) throws MalformedURLException;

Asi que podriamos especificar todos los componentes del URL como en:

URL( “http”, “www.yahoo.com”, “80”, “index.html” );

O dejar que los sitemas utilicen todos los valores por defecto que tienen definidos, como en:

URL( “http://www.yahoo.com” ) ;

Y en los dos casos obtendríamos la visualización de la pagina principal de Yahoo en nuestro navegador.

**DOMINIOS DE COMUNICACIONES**

El mecanismo de sockets esta diseñado para ser todo lo genérico posible. El socket por si mismo no contiene información suficiente para describir la comunicación entre procesos. Los sockets operan dentro de dominios de comunicación, entre ellos se define si los dos procesos que se comunican se encuentran en el mismo sistema o en sistemas diferentes y como pueden ser direccionados.

**Dominio Unix**

Bajo Unix, hay dos dominios, uno para comunicaciones internas al sistema y otro para comunicaciones entre sistemas.

Las comunicaciones intrasistema (entre dos procesos en el mismo sistema) ocurren (en una maquina Unix) en el dominio Unix. Se permiten tanto los sockets stream como los datagrama. Los sockets de dominio Unix bajo Solaris 2.x se implementan sobre TLI (Transport Level Interface).

En el dominio Unix no se permiten sockets de tipo Raw

**Dominio Internet**

Las comunicaciones intersistemas proporcionan acceso a TCP, ejecutando sobre IP(Internet Protocol). De la misma forma que el dominio Unix, el dominio Internet permite tanto sockets stream como datagrama, pero además permite sockets de tipo Raw.

Los sockets stream permiten a los procesos comunicarse a través de TCP. Una vez establecidas las conexiones, los datos se pueden leer y escribir a/desde los sockets como un flujo (stream) de bytes. Algunas aplicaciones de servicios TCP son:

- File Transfer Protocol, FTP
- Simple Mail Transfer Protocol, SMTP
- TELNET, servicio de conexión de terminal remoto.

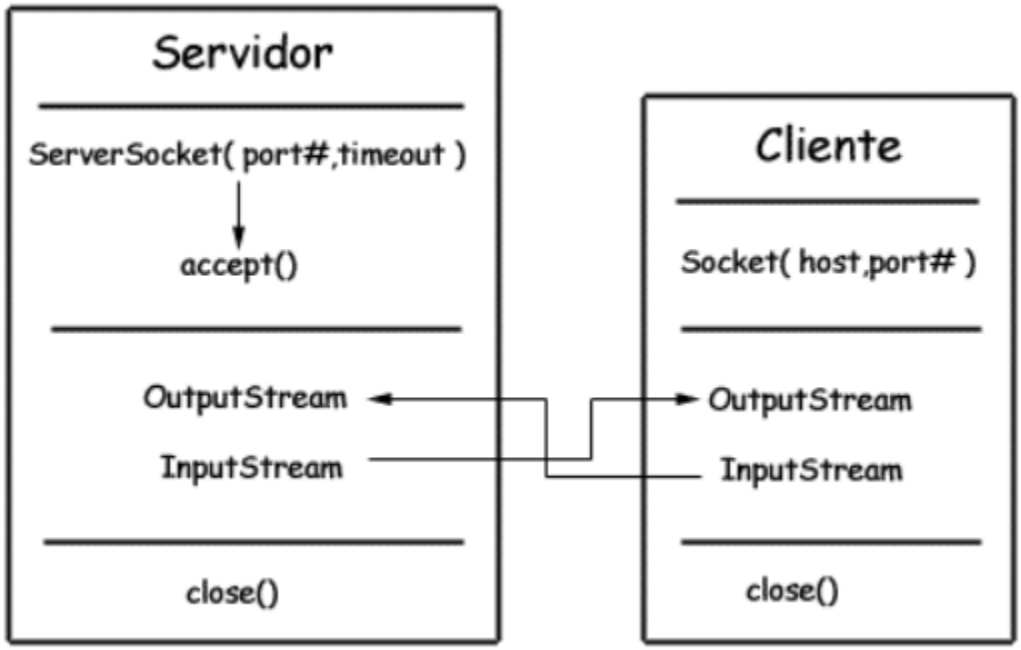
Los sockets datagrama permiten a los procesos utilizar el protocolo UDP para comunicarse a y desde esos sockets por medio de bloques. UDP es un protocolo no fiable y la entrega de los paquetes no esta garantizada. Servicios UDP son:

- Simple Network Management Protocol, SNMP
- Trivial File Transfer Protocol, TFTP (Version FTP sin conexion)
- Versatile Message Transaction Protocol, VMTP (servicio fiable de entrega punto a punto de datagramas independiente de TCP)

Los sockets raw proporcionan acceso a internet control message protocol, ICMP, y se utiliza para comunicarse entre varias entidades IP.

**MODELO DE COMUNICACIONES CON JAVA**

En java, crear una conexión socket TCP/IP se realiza directamente con el paquete java.net. A continuación mostramos un diagrama de los que ocurre en el lado del cliente y del servidor:




El modelo de sockets mas simple es:

- El servidor establece un puerto y espera durante un cierto tiempo (timeout segundos), a que el cliente establezca la conexión. Cuando el cliente solicite una conexión , el servidor abrirá la conexión socket con el método accept().
- El cliente establece una conexión con la maquina hos a través del puerto que se designe en puerto#
- El cliente y el servidor se comunican con manejadores InputStream y OutputStream

Hay una cuestión al respecto de los sockets, que viene impuesta por la implementación del sistema de seguridad de Java. Actualmente, los applets solo pueden establecer conexiones con el nodo desde el cual se transfirió su código. Esto esta implementado en el JDK y en el interprete de Java de Netscape. Esto reduce en gran manera la flexibilidad de las fuentes de datos disponibles para los applets. El problema si se permite que un applet se conecte a cualquier maquina de la red, es que entonces se podrían utilizar los applets para inundar la red desde un ordenador con un cliente Netscape del que no se sopecha y sin ninguna posibilidad de rastreo.

**APERTURA DE SOCKETS**

Si estamos programando un cliente, el socket se abre de la forma:

Socket miCliente;

miCliente = new Socket( “maquina”, numeroPuerto );

Donde maquina es el nombre de la maquina en donde estamos intentando abrir la conexión y numeroPuerto es el puerto (un numero) del servidor que esta corriendo sobre el cual nos queremos conectar. Cuando se selecciona un numero de puerto, se debe tener en cuenta que los puertos en el rango 0-1033 estan reservados para usuarios con muchos privilegios (superusuarios o root). Estos puertos son los que utilizan los servicios estándar del sistema como email, ftp p http. Para las aplicaciones que se desarrollen, asegurarse de seleccionar un puerto por encima del 1023-

En el ejemplo anterior no se usan excepciones: sin embargo, es una gran idea la captura de excepciones cuando se esta trabajando con sockets. El mismo ejemplo quedaría como:

```
    Socket miCliente;
try {
    miCliente = new Socket( "maquina",numeroPuerto );
} catch( IOException e ) {
    System.out.println( e );
    }
```

Si estamos programando un ***servidor***, la forma de apertura del socket es la que muestra el siguiente ejemplo:

```
Socket miServicio;
try {
    miServicio = new ServerSocket( numeroPuerto );
} catch( IOException e ) {
    System.out.println( e );
    }
```

A la hora de la implementación de un servidor también necesitamos crear un objeto socket desde el ServerSocket para que esté atento a las conexiones que le puedan realizar clientes potenciales y poder aceptar esas conexiones:

```
Socket socketServicio = null;
try {
    socketServicio = miServicio.accept();
} catch( IOException e ) {
    System.out.println( e );
    }
```