SDCoulourisEsp3ed.pdf

4 Comunicación entre procesos

La interfaz de programación de aplicaciones (API) de Java para la comunicación entre procesos en internet proporciona comunicación tanto por datagramas como por streams. Estas proveen bloques constructivos alternativos para los protocolos de comunicación.

4.1 Introducción

La interfaz del programa de aplicación para UDP proporciona un abstracción del tipo paso de mensajes, la formas mas simple de comunicacion entre procesos. Esto hace que el proceso emisor pueda transmitir un mensaje simple al proceso receptor. Los paquetes independientes que contienen estos mensajes se llaman datagramas. Tanto en Java como en cada API UNIX, el emisor especifica el destino utilizando un zócalo, conector o socket.

La interfaz del programa de aplicación de TCP proporciona la abstracción de un flujo (Stream) de dos dirección entre pares de procesos. La información intercambiada consiste en un stream de items de datos, sin límites entre mensajes. Los stream son un bloque básico para la construcción de la comunicación productor-consumidor. Un productor y un consumidor forman un par de procesos en los cuales el papel del primero es producir items de datos y del segundo consumirlos.

La multidifusión en grupos es una forma de comunicación entre procesos en el cual un proceso de un grupo de procesos transmite el mismo mensaje a todos los miembros del mismo grupo.

4.2.1 Las características de la comunicación entre procesos

El paso de mensajes entre un par de procesos se puede basar en dos operaciones: envía y recibe, definidas en función del destino y del mensaje. Para que un proceso se pueda comunicar con otro, el proceso envía un mensaje a un destino y otro proceso en el destino recibe el mensaje. Esta actividad implica la comunicación de datos desde el proceso emisor al proceso receptor y puede implicar además la sincronización de los dos procesos.

Comunicacion sincrona y asincrona. A cada destino de mensajes se asocia una cola. Los procesos emisores producen mensajes que serán añadidos a las colas remotas mientras que los procesos receptores eliminan mensajes de las colas locales. La comunicación entre los procesos emisor y receptor puede ser sincrona o asincrona. En la forma síncrona, los procesos receptor y emisor se sincronizan con cada mensaje, tanto el envia como recibe son operaciones bloqueantes. En la forma de comunicación asíncrona, la utilización de la operación envia es no bloqueante, de modo que el proceso emisor puede continuar tan pronto como el mensaje haya sido copiado en el buffer local, y la transmisión del mensaje se lleva a cabo en paralelo con el proceso emisor.

Destinos de los mensajes. Los mensaje son enviados a dirección construidas por pares (direcciones Internet, puerto local). Un puerto local es el destino de un mensaje dentro de un compilador, especificado como un número entero. Un puerto tiene exactamente un receptor pero puede tener muchos emisores. Los procesos pueden utilizar múltiples puertos desde los que recibir mensajes.

Generalmente los servidores hacen públicos sus números de puerto para que sean utilizados por los clientes**.** Si el cliente utiliza una dirección Internet fija para referirse a un servicio, entonces ese servicio debe ejecutarse siempre en el mismo computador para que la dirección se considere válida. Esto se puede evitar utilizando algunas de las siguientes aproximaciones que proporcionan transparencia de ubicación:

Los programas cliente se refiere a los servicios por su nombre y utilizan un servidor de nombres o enlazador para trasladar esos nombres a ubicaciones del servidor en tiempo de ejecución. Esto permite reubicar los servicios en otro lugar, pero no migrarlos.
El sistema operativo proporciona identificadores para los destinos de los mensajes independientes de la localización, haciéndoles corresponder con direcciones de más bajo nivel utilizadas para entregar los mensajes en los puertos, permitiendo tanto la migración como la reubicación de los servicios en otro lugar
Fiabilidad. Una comunicación fiables se define en términos de validez e integridad. En lo que concierne a la propiedad de validez, se dice que un servicio de mensajes punto a punto es fiable si se garantiza que los mensajes se entrega a pesar de poder dejar caer o perder un número razonable de ellos
Ordenación Algunas aplicaciones necesitan que los mensajes sean entregados en el orden de su emisión, esto es, en el orden en el que fueron transmitidos por el emisor. La entrega de mensajes desordenados, por esas aplicaciones, es considerada como un fallo.

4.2.2 Sockets

La comunicación mediante los protocolos UDP y TCP se logra gracias a la abstracción de un socket. Su implementación es originaria de UNIX BSD y se encuentran actualmente en todo sistema operativo. Los sockets crean la comunicación entre procesos. En su forma más simple un socket debe tener asociado una dirección de red y un puerto que es por el cual se comunica. Un socket puede tanto recibir como enviar información. El número de puertos disponibles para la comunicación de de 2^16.

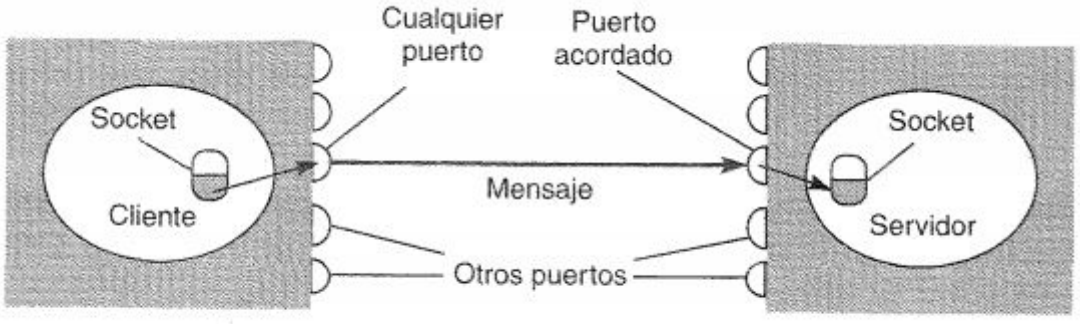

Figura 4.2. Sockets y puertos.

Los sockets en Java cuentan con una clase llamada InetAddress la cual abstrae las direcciones de red.

4.2.3 Comunicación de Datagramas UDP

El protocolo de comunicación de edatagramas (UDP) es usado cuando se requiere hacer una comunicación entre un cliente y un servidor sin tener que recibir mensajes de acuse por parte del cliente.

Aspectos a contemplar en la comunicación de datagramas:

- **Tamaño de mensaje:** el proceso receptor necesita especificar una cadena de bytes de un tamaño concreto sobre el cuál se alamacenará el mensaje recibido. Si el mensaje es demasiado grande, este será truncado. La capa subyacente de IP permite paquetes de hasta 2^{16} bytes, cualquier aplicación que necesite mensajes mayores a este tamaño deberá fracmentarlos.

- **Bloqueo:** la comunicación de datagramas UDP utiliza operaciones de envío no bloqueantes y recepciones bloqueantes. Esto quiere decir que que la operación de envio delvolverá el control cuando el mensaje se ha dirigido a las capas inferiores de UDP e IP. A la llegada, el mensaje será colocado en una cola conectada que está enlazado con el puerto destino, si no existe ningun proceso ligado al conector destino los mensajes serán descartados.

  El método *recibe* produce un bloque hasta que se reciba un datagrama, a menos que se haya establecido un *tiempo límite* (timeout) asociado al conector.

- **Tiempo limite de espera:** El método *recibe* con bloqueo indefinido es adecuado para servidores que están esperando recibir peticiones de sus clientes. En algunos casos no resulta adecuado poner tiempos infenidos por posibles fallas del emisor, por lo cual se establecen tiempos límites de espera (*timeouts)* en los conectores. El tiempo límite de espera debe ser lo suficientemente grande en comparación con el tiempo adecuado de transmición de mensaje.

- **Recibe de cualquiera:** El método *recibe* no especifica el origen de los mensajes. En su lugar, al invocar *recibe* aceptamos mensajes dirigidos a su conector desde cualquier conector. El método *recibe* devuelve la dirección de Internet y el puerto emisor, permitiendo al receptor comprobar de donde viene el mensaje.

Modelo de fallo: El término modelo de fallo define las propiedades de una comunicación fiable en términos de dos propiedades: integridad y validez. La propiedad de integridad requiere que los mensajes no se corrompan ni se dupliquen. Este modelo de fallo puede utilizarse para propioner un modelo de fallo para los datagramas UDP, que padece de la siguientes debilidades:

- *Fallos de omisión:* Los mensajes pueden desecharse ocasionalmente, ya sea porque se ha producido un error detectado por la suma de comprobación o porque no queda espacio en el búfer ya sea en el origen o en el destino.
- *Ordenación: Algunas veces, los mensajes se entregan en desorden con respecto a su orden de emisión.*

Las aplicaciones que utilizan datagramas UDP dependen de sus propias comprobaciones para conseguir la calidad que necesitan respecto a la fiabilidad de la comunicación.

Utilización de UDP: Un ejemplo de la utilización de de un servicio que sea susceptible a sufrir fallos de omisión ocasionales es el Servicio de Nombres de Dominio en Internet

(Domain Name Service, DNS). Los datagramas UDP no padecen las sobrecargas asociadas a la entrega de mensajes garantizada. Existen tres fuentes principales para esa sobrecarga:

1. La necesidad de almacenar información.
2. La transmisión de mensajes extra.
3. La latencia para el emisor.

4.2.4 Comunicación de streams TCP

API para el protocolo TCP originaria de UNIX BSD 4.x.

- ➔ Tamaño de los mensajes: Decide cuántos datos recoge antes de transmitirlos como paquetes IP. En el destino, los datos son proporcionados según se vayan solicitando.
- ➔ Mensajes perdidos: Si el emisor no recibe dicho acuse dentro de un plazo de tiempo, volverá a transmitir el mensaje.
- ➔ Control de flujo: Intenta ajustar las velocidades de los procesos que leen y escriben en un stream. Bloqueo de escritura hasta que la lectura finalice.
- ➔ Duplicación y ordenación de los mensajes: A cada paquete IP se le asocia un identificador; el receptor detecta y rechaza mensajes duplicados, y reordena mensajes que lleguen desordenados.
- ➔ Destinos de los mensajes: Un par de procesos establecen una comunicación, simplemente leen y escriben sin preocuparse por las direcciones de Internet ni de los números de puerto.

El API para la comunicación supone una conexión cliente - servidor, aunque después se comuniquen igual a igual. El cliente crea un stream conector sobre un puerto y envía una petición de conexión con el servidor en su puerto de servicio. El servidor crea un conector de escucha ligado al puerto de servicio y la espera de clientes que soliciten conexiones, manteniendo una cola de peticiones. Cuando un servidor acepta una conexión, crea un nuevo conector para mantener la comunicación con el cliente, mientras reserva el conector del puerto de servicio para escuchar peticiones de conexión de otros clientes.

Ambos conectores se conectan por un par de streams, uno en cada dirección, de entrada y de salida. Un proceso del par cliente-servidor puede enviar información al otro escribiendo en su stream de salida, y el otro puede obtener la información leyendo desde su stream de entrada.

Cuando un proceso finaliza o falla, todos sus conectores se cierran y cualquier proceso que se intente comunicar con él descubrirá que la conexión se ha roto.

- ❏ Concordancia de ítems de datos: Los dos procesos que se comunican deben estar de acuerdo con el tipo de datos que se transmiten por el stream
- ❏ Bloqueo: Cuando un proceso intenta leer datos de un canal de entrada, o bien se extraen los datos de la cola o se bloqueará hasta que existan datos disponibles.

- ❏ Hilos: Cuando un servidor acepta una conexión, generalmente crea un nuevo hilo con el que comunicarse en el nuevo cliente. El servidor se bloquea a la espera de entradas de un cliente sin afectar a los otros.

Modelo de fallo
Los streams TCP utilizan una suma de comprobación para detectar y rechazar paquetes corruptos, y utilizan un número de secuencia para paquetes duplicados. Utilizan timeouts y retransmisión de los paquetes perdidos. Los mensajes tienen garantizada su entregada cuando alguno de sus paquetes se haya perdido. No se garantiza la entrega de mensajes en presencia de algún tipo de problema.

Utilización de TCP
Servicios que se ejecutan sobre conexiones TCP con números de puerto reservados: HTTP, FTP, Telnet, SMTP.

API de Java para los streams TCP
- Clase ServerSocket: Utilizada por un servidor para crear un conector en el puerto de servidor que escucha peticiones de conexión de los clientes. El accept toma una petición connect, la encola, y da como resultado una instancia de Socket, un conector que da acceso a los streams al cliente de entrada y salida.
- Clase Socket: El cliente utiliza un constructor para crear un conector, especificando el host y el puerto del servidor. Excepciones: UnknownHostException, IOException (entrada y salida).

Métodos getInputStream() y getOutputStream() para flujos lectura y escritura, devuelven objetos de tipo InputStream y OutputStream (clases abstractas para leer y escribir bytes). Objetos de tipo DataInputStream y DataOutputStream permiten utilizar representaciones binarias de los tipos de datos primitivos para ser leídos y escritos de forma independiente de la máquina.

Métodos writeUTF() y readUTF() para escribir y leer cadenas de caracteres en sus respectivos streams. Cuando un proceso ha cerrado su conector, no le será posible utilizar sus streams de entrada y salida.

Preguntas

¿Cual es el nombre de los paquetes independientes que permiten que un proceso emisor transmite un mensaje simple a un proceso receptor?
A. Datagramas
B. Tramas
C. Paquetes de red
D. Mensajes simples

¿En qué tipo de comunicación las operaciones envía y recibe son bloqueantes?
A. Síncrona
B. Simple

C. Múltiple
D. Asíncrona

¿Cuáles son los principales componentes en una conexión con el protocolo TCP?

A. Socket cliente, socket servidor, flujos de lectura, flujos de escritura.
B. Puerto, socket cliente, dirección host y flujos de lectura-escritura.
C. Flujos de lectura, flujos de escritura, objeto DataInputStream y objeto DataOutputStream.
D. Conector, método accept, objeto Socket, métodos getInputStream() y getOutputStream()

¿Como se llama la clase que abstrae la comunicación entre procesos?

A. Socket
B. Datagram
C. InetAddress
D. OutputStream e InputStream

¿Por qué resulta atractiva la elección de utilizar datagramas UDP para algunas aplicaciones?

A. La necesidad de almacenar información de estado en el origen y en el destino.
B. La transmisión de mensajes extra.
C. La latencia para el emisor.
D. Todas las anteriores.