
Capítulo 5. Programación de aplicaciones de red

Redes de Ordenadores
2º Grado en Ingeniería en Tecnologías de Telecomunicación



1

Índice

Hora 1

- 1 API de sockets BSD
- 2 Sockets TCP
 - 2.1 Cliente TCP
 - 2.2 Servidor TCP
 - 2.3 Detalles de sockets TCP

Hora 2

- 3 Sockets UDP
 - 3.1 Cliente UDP
 - 3.2 Servidor UDP
 - 3.3 Detalles de sockets UDP
- 4 Otras funcionalidades del API BSD
- 5 Excepciones

Hora 3

- 6 Streams
- 7 Servidores concurrentes
 - 7.1 Sockets no bloqueantes
 - 7.2 Selectores
 - 7.3 Threads

Índice hora 1

Hora 1

- 1 API de sockets BSD
- 2 Sockets TCP
 - 2.1 Cliente TCP
 - 2.2 Servidor TCP
 - 2.3 Detalles de sockets TCP

Hora 2

- 3 Sockets UDP
 - 3.1 Cliente UDP
 - 3.2 Servidor UDP
 - 3.3 Detalles de sockets UDP
- 4 Otras funcionalidades del API BSD
- 5 Excepciones

Hora 3

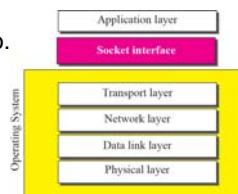
- 6 Streams
- 7 Servidores concurrentes
 - 7.1 Sockets no bloqueantes
 - 7.2 Selectores
 - 7.3 Threads

Objetivos

- Revisar las bases del API de sockets BSD
- Aprender a programar aplicaciones cliente/servidor TCP

1 API de sockets BSD

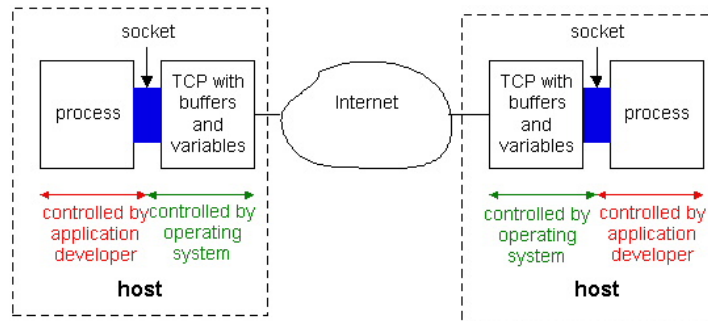
- El API (Application Programming Interface) de sockets BSD es el estándar de facto para el desarrollo de aplicaciones de red en la mayor parte de sistemas operativos (Linux, Windows, MacOSX, etc.) y la mayor parte de lenguajes de programación (C, C++, C#, Java, Python, etc)
- Se distribuyó por primera vez como parte del sistema operativo UNIX BSD4.1, 1981
- Es una librería que provee las funcionalidades para desarrollar aplicaciones que utilicen la familia de protocolos TCP/IP que provee el sistema operativo (kernel).
 - Las aplicaciones trabajan como proceso de usuario.
 - Soporta el paradigma cliente/servidor



API de sockets BSD

- El socket es el interfaz entre la aplicación y el sistema operativo que provee la capa de transporte (UDP o TCP)
 - Local a cada máquina
 - Creado por la aplicación
 - Controlado por el sistema operativo
 - Que permite a las aplicaciones enviar y recibir mensajes por la red a otra aplicación corriendo en otra máquina
 - Estará asociado siempre a la pareja [dirección IP, puerto]
 - En el caso de una máquina con varias direcciones IP, el socket podrá hacer referencia a todas o alguna de ellas.
- Revisaremos la programación de sockets en Java, pero es muy similar para otros lenguajes de programación.
 - Para utilizar el API de sockets necesitaremos importar el paquete:
 - `import java.net.*;`
 - Este paquete incorpora muchas otras funcionalidades como la URL, URLConnection que implementa el protocolo de la web

API de sockets BSD



- Distingue dos tipos de servicio de transporte vía sockets
 - SOCK_DGRAM: datagramas no fiable – UDP – DatagramSocket()
 - SOCK_STREAM: orientado a stream, fiable – TCP – Socket(), serverSocket()

Llamadas básicas del API sockets BSD

- socket (): crea el socket
 - bind(): asocia el socket a un puerto y/o dirección IP local
 - listen(): espera pasiva de conexiones
 - connect(): inicia la conexión con otro socket
 - accept(): acepta una nueva conexión
 - write(): escribe datos al socket
 - read(): recibe datos del socket
 - sendto(): envía datagramas a otro socket UDP
 - recvfrom(): lee datagramas del socket UDP
 - close(): cierra el socket y con ello la conexión
- } Java incorpora clases que permiten englobar varios de estos pasos

2 Sockets TCP

- Para la aplicación, TCP permite transferir bytes de manera confiable, en orden (“un tubo”), entre el cliente y el servidor.
- Una conexión TCP unirá un cliente y un servidor
- El servidor
 - Crea un **socket servidor**
 - Queda a la escucha sobre ese socket servidor, esperando conexiones de clientes
 - Cuando recibe la conexión de un cliente, crea un nuevo **socket de conexión** que permitirá la comunicación con ese cliente.
 - Esto permitirá al servidor establecer varios sockets secundarios simultáneamente para comunicarse con varios clientes
- El cliente
 - Crea un **socket cliente**, especificando la dirección IP y puerto del servidor al que se quiere conectar
 - Al crear el socket, el cliente establece automáticamente la conexión TCP con el servidor

Sockets TCP

- Clases Java relacionadas con sockets TCP:
 - `java.net.Socket`: socket general y socket cliente
 - `java.net.ServerSocket`: socket servidor

Sockets TCP

Servidor (ejecutando en `hostid`)

crea socket,
port=`x`, para
recibir solicitudes:
`welcomeSocket =
ServerSocket()`

espera solicitudes
de conexión
`connectionSocket =
welcomeSocket.accept()`

lee solicitudes desde
`connectionSocket`

escribe las respuestas en
`connectionSocket`

cierra
`connectionSocket`

Cliente

crea socket,
se conecta a `hostid`, port=`x`
`clientSocket =
Socket()`

envía solicitudes usando
`clientSocket`

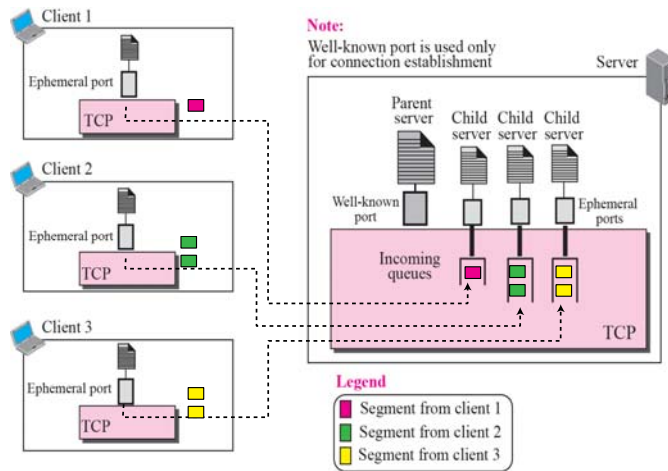
lee respuestas de
`clientSocket`

cierra
`clientSocket`

establece conexión
TCP

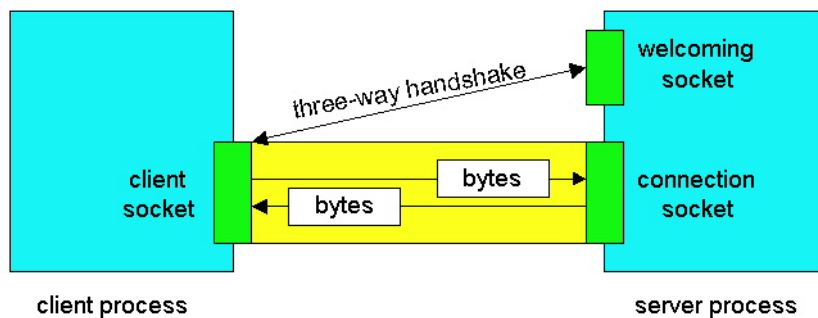
cierra conexión
TCP

Sockets TCP



Sockets TCP

- 2 tipos de sockets en el lado de servidor
 - Socket servidor
 - Socket de conexión generado a partir del socket servidor

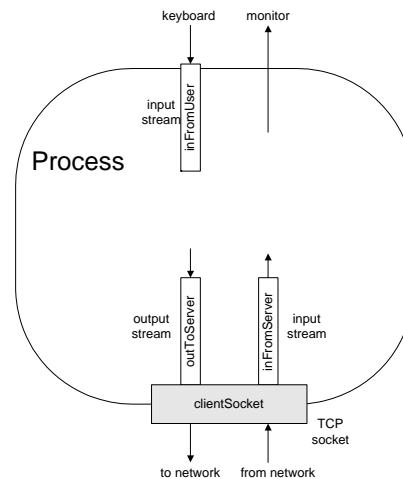


Streams

- Un stream es un flujo de bytes bidireccional.
- Se puede decidir separar el stream en sus dos sentidos:
 - Input stream: interfaz por el que se obtienen los datos.
 - Output stream: interfaz por el que se tienen que volcar los datos.
- El socket TCP provee un stream de bytes con los datos intercambiados con la red, pero un stream puede asociarse también a un fichero, entrada por teclado, etc. (cualquier fuente/destino de bytes)
- Sin embargo UDP no da un interfaz a nivel de byte sino a nivel de paquete sin fiabilidad, por lo que no es posible asociarlo a un stream.
- En java los streams se implementan en el paquete:
 - `import java.io.*;`

Ejemplo de aplicación cliente-servidor TCP

1. El cliente lee una línea desde el dispositivo de entrada estándar (stream inFromUser) y lo envía al servidor a través de un socket (stream outToServer)
2. El servidor lee la línea desde un socket
3. El servidor convierte la línea a mayúsculas, y la devuelve al cliente
4. El cliente lee la línea modificada que lee desde el socket (stream inFromServer) y la imprime en pantalla



2.1 Cliente TCP

```
import java.io.*;
import java.net.*;
class TCPClient {
    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        //Crea input stream (de texto) asociado al teclado
        BufferedReader inFromUser = new BufferedReader(new
        InputStreamReader(System.in));
        //Crea el socket cliente y conecta al servidor
        Socket clientSocket = new Socket("server.com", 6789);
        //Crea output stream (de bytes) asociado al socket
        DataOutputStream outToServer = new
        DataOutputStream(clientSocket.getOutputStream());
        //Crea input stream (de texto) asociado al socket
        BufferedReader inFromServer = new BufferedReader(new
        InputStreamReader(clientSocket.getInputStream()));
        //Lee una línea de teclado
        sentence = inFromUser.readLine();
        //Envía la línea al servidor
        outToServer.writeBytes(sentence + '\n');
        //Lee la línea devuelta por el servidor
        modifiedSentence = inFromServer.readLine();
        System.out.println("FROM SERVER: " + modifiedSentence);
        //Cierra del socket
        clientSocket.close();
    }
}
```


2.2 Servidor TCP

```
import java.io.*;
import java.net.*;
class TCPServer {
    public static void main(String argv[]) throws Exception
    {
        String clientSentence;
        String capitalizedSentence;

        //Crea el socket servidor en cierto puerto y se queda a la escucha
        ServerSocket welcomeSocket = new ServerSocket(6789);
        //Bucle infinito esperando conexiones de clientes
        while(true) {
            //Espera conexión de un cliente, y cuando la hay genera un socket secundario asociado al cliente
            Socket connectionSocket = welcomeSocket.accept();
            //Crea un input stream (de texto) asociado al socket secundario
            BufferedReader inFromClient = new BufferedReader(new
            InputStreamReader(connectionSocket.getInputStream()));
            //Crea un output stream asociado (de bytes) al socket secundario
            DataOutputStream outToClient = new
            DataOutputStream(connectionSocket.getOutputStream());
            //Lee la línea que manda el cliente
            clientSentence = inFromClient.readLine();
            //Convierte la línea a mayúsculas
            capitalizedSentence = clientSentence.toUpperCase() + '\n';
            //Escribe la línea hacia el cliente
            outToClient.writeBytes(capitalizedSentence);
        }
    }
}
```

2.3 Detalles de sockets TCP

- Creación y conexión del socket (java.net.Socket).
 - Al llamar al constructor se crea el socket y si se indica la dirección y puerto del servidor ya se conecta con la máquina y puerto indicados.
 - Constructores:
 - Socket()
 - Socket(InetAddress dir, int puerto)
 - Socket(InetAddress dir, int puerto, InetAddress dirLocal, int puertoLocal)
 - Socket(String nombre, int puerto)
 - Socket(String nombre, int puerto, InetAddress dirLocal, int puertoLocal)
 - Métodos
 - void bind(SocketAddress bindpoint)
 - void connect(SocketAddress endpoint)
 - void connect(SocketAddress endpoint, int timeout)

Detalles de sockets TCP

- Creación del socket servidor (java.net.ServerSocket)
 - Al llamar al constructor se crea el socket servidor y si se indica el puerto ya se asocia a ese puerto.
 - Constructores:
 - ServerSocket()
 - ServerSocket(int puerto)
 - ServerSocket(int puerto, int backlog)
 - backlog: Número de máximo de conexiones pendientes que aceptará el socket.
 - ServerSocket(int puerto, int backlog, InetAddress dirIP)
 - dirIP: Dirección por la que va a aceptar conexiones (en caso de que la máquina del servidor tenga más de una dirección IP).
 - Métodos:
 - void bind(SocketAddress endpoint)
 - void bind(SocketAddress endpoint, int backlog)
 - Socket accept()
 - Devuelve un Socket conectado al cliente que realizó la conexión.

Detalles de sockets TCP

- Envío/recepción mediante streams
 - Para obtener los dos streams entrada/salida de un socket:
 - InputStream getInputStream()
 - OutputStream getOutputStream()
 - Ofrecen un interfaz a nivel de byte
 - Sobre ellos se pueden definir buffers y conversores que nos faciliten la tarea. Para texto por ejemplo:
 - Para la entrada de datos: BufferedReader – InputStreamReader
 - Método readLine().
 - Para la salida de datos: PrintWriter
 - Método println().

Resumen

- API de sockets BSD: librería para el desarrollo de aplicaciones de red
 - Socket: interfaz entre la aplicación y el sistema operativo para intercambiar información por la red
- Clases básicas para el desarrollo de aplicaciones TCP
 - java.net.Socket: socket general y socket cliente
 - Socket(InetAddress dir, int puerto)
 - Crea un socket cliente, y lo conecta con cierta dirección IP y puerto destino (realiza el proceso de conexión TCP)
 - java.net.ServerSocket: socket servidor
 - ServerSocket(int puerto)
 - Crea un socket servidor escuchando en determinado puerto

Referencias

- [Forouzan]
 - Capítulo 2, “Application Layer”, sección 2.7
- Manual en línea Java 1.6,
<http://docs.oracle.com/javase/6/docs/api/overview-summary.html>
- “All About Sockets” (Tutorial Oracle Java),
<http://docs.oracle.com/javase/tutorial/networking/sockets/index.html>
- “I/O Streams” (Tutorial Oracle Java),
<http://docs.oracle.com/javase/tutorial/essential/io/streams.html>
- “Socket Programming in Java: a tutorial,”
<http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets.html>