Universidad de Málaga

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

Programación Distribuida: Sockets en Java

Profesores:

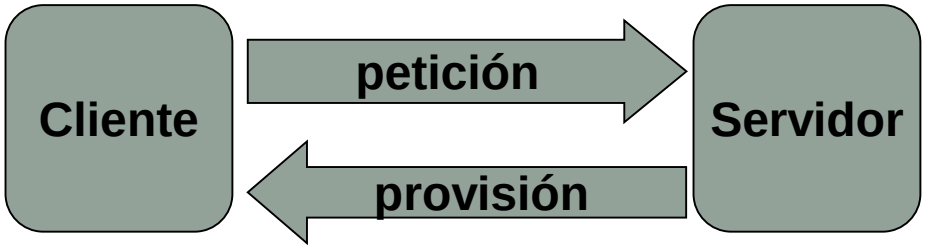
Mercedes Amor Pinilla

Francisco Chicano

Gabriel Luque Polo

Paradigma Cliente/Servidor

- La programación de aplicaciones distribuidas sigue el paradigma cliente/servidor.
- Vamos a diferenciar entre procesos clientes y servidores
- La estructura general de estas aplicaciones es:



Cliente:	**Servidor:**
... Petición_servicio() Servicio(...)	WHILE TRUE DO Espera_petición(...) Servicio(...) END

Paradigma Cliente/Servidor

- Los procesos servidores:
  - Son procesos permanentemente activos que ofrecen un servicio concreto siempre disponible para los usuarios del servicio.
  - El servidor tiene una dirección fija y conocida denominada dirección IP.
- Los procesos clientes
  - piden un servicio en un momento dado.
  - Se comunican sólo con el servidor.
- Clientes y servidores de una misma aplicación se comunicarán mediante el intercambio de mensajes utilizando los servicios del nivel de Transporte
- El envío y recepción de mensajes se realiza a través de sockets

Programación con sockets

- Definición de socket
  - Un socket es un punto final (origen o destino) de comunicación entre procesos que se ejecutan en ordenadores diferentes
    - El canal que se establece es bidireccional (dúplex)
  - La interfaz socket es una API (*Application Programming Interface*) para realizar aplicaciones distribuidas sobre una red
- Dos tipos básicos de sockets
  - Sockets TCP: comunicación orientada a la conexión, fiable
  - Sockets UDP: comunicación no orientada a la conexión, no fiable

Alternativas para programa con sockets

- C/C++
  - La interfaz socket original estaba diseñada para ser usada desde C sobre sistemas UNIX
  - Tiene el inconveniente de ser compleja de usar
  - Problemas de portabilidad
- Java
  - Funciones de manejo de sockets simplificadas respecto a C
  - Código más corto y más legible
  - Código portable
- Otras alternativas
  - C# (.NET), Perl, etc.

Programación con sockets TCP en Java

- Características
  - Se utiliza el modelo cliente/servidor para establecer las conexiones
  - Una vez creado el canal de comunicaciones, los roles del servidor y del cliente dependen del programador
  - El servidor tiene que estar activo para que el cliente pueda establecer la conexión
  - Proporcionado en el paquete java.net
- Conceptos
  - ServerSocket: clase que usa el servidor para aceptar conexiones de los clientes
  - Socket de conexión (o conectado): socket que se crea cuando se establece una conexión entre un cliente y un servidor

Programación con sockets TCP en Java

- Acciones del servidor
  - Crear un `ServerSocket`
  - Esperar una solicitud de conexión
  - Crear el socket de conexión
  - Enviar/recibir usando el socket de conexión
  - Cerrar la conexión
- Acciones del cliente
  - Conectarse con el servidor (crea el socket de conexión)
  - Enviar/recibir usando el socket de conexión
  - Cerrar la conexión

Clase ServerSocket

- Socket para un servidor orientado a la conexión.
- Constructores:
  - **public ServerSocket (int port)**: crea un socket que recibe peticiones por el puerto indicado (0 = cualquiera disponible). Máximo 50 peticiones en cola pendientes.
  - **public ServerSocket (int port, int count)**: igual que el anterior pero especifica el número máximo de peticiones en cola pendientes.
- Métodos:
  - **public Socket accept ()**: saca una petición de la cola de pendientes (se bloquea si no hay) y crea un socket conectado al cliente.
  - **public void close()**: cierra el socket.
  - **public int getLocalPort ()** : indica en que puerto está escuchando.

Clase Socket

- Socket para una conexión C/S orientada a la conexión.
- Constructor:
  - **public Socket (InetAddress address, int port)** : crea un socket y lo conecta a la dirección IP y puerto indicados.
- Métodos:
  - **public InputStream getInputStream ()**: devuelve el stream para la recepción de datos.
  - **public OutputStream getOutputStream ()** : devuelve el stram para el envío de datos.
  - **public InetAddress getInetAddress ()** : dirección IP remota del socket.
  - **public int getLocalPort ()** : puerto local.
  - **public int getPort ()** : puerto remoto.

Envío y recepción a través de la clase Socket

- Una vez tengamos el socket conectado podemos utilizar los métodos getInputStream() y getOutputStream() para obtener los flujos asociados.
- Recepción:
  - Usaremos la clase **BufferedReader** o la clase **Scanner**. Ejemplo:

```
BufferedReader r = new BufferedReader(
                        new InputStreamReader(socket.getInputStream()));
Scanner sc = new Scanner(System.in);
```

  - La primera permite los métodos **read** (recibe uno o varios bytes) y **readLine** (recibe líneas completas y devuelve un String).
  - La segunda **nextByte**, **nextLine**
- Envío:
  - Usaremos la clase **PrintWriter**. Ejemplo:

```
PrintWriter s = new PrintWriter(socket.getOutputStream());
```

  - Esta clase permite los métodos **print, println y write.**

Clase InetAdress

- La clase `InetAdress` nos facilita el manejo de direcciones IP.
- Mediante los métodos `getByName/getLocalHost` podemos obtener la conversión de direcciones a su valor numérico:

```
try {
   InetAddress utopia = InetAddress.getByName("utopia.poly.edu");
   InetAddress duke = InetAddress.getByName("128.238.2.92");
   InetAddress localIP = InetAddress.getLocalHost();
   //... }
catch (UnknownHostException e)
{ System.err.println(ex); }
```

- También tenemos métodos para hacer la conversión inversa: `getHostName`, `getAddress` y `getHostAddress`
- Puede escribirse al disponer del método `toString`

Programación con sockets UDP en Java

- Características
  - UDP proporciona un servicio sin conexión
  - No existe un canal de comunicaciones preestablecido
  - No existen los roles de servidor y cliente
  - Los procesos envían mensajes (datagramas)
  - No se garantiza que los datagramas lleguen, ni el orden, ni tal vez la producción de datagramas duplicados
  - Es más eficiente que TCP
  - Los problemas de fiabilidad se producen con muy escasa probabilidad en redes locales

Programación con sockets UDP en Java

- Acciones del servidor
  - Crear un `DatagramSocket`
  - Recibir un mensaje (datagrama)
  - Enviar un mensaje de respuesta
- Acciones del cliente
  - Crear un `DatagramSocket`
  - Enviar un mensaje (datagrama)
  - Recibir mensaje de respuesta
  - Cerrar el socket

Clase DatagramSocket

- Socket para el envío/recepción de datagramas
- Constructores:
  - **public DatagramSocket ()**: crea un socket para el envío/recepción de datagramas. Lo asocia al primer puerto libre.
  - **public DatagramSocket (int port)**: igual que el anterior pero asociado al puerto indicado.
  - **public DatagramSocket (int port, InetAddress ip)**: igual que el anterior pero además se indica el interfaz local al que está asociado.
- Métodos:
  - **public void close()**: cierra el socket.
  - **public int getLocalPort ()** : puerto local asociado.
  - **public void receive (DatagramPacket p)** : recepción.
  - **public void send (DatagramPacket p)** : envío.

Clase DatagramPacket

- Información enviada en el datagrama (IP,puerto, datos,…)
- Constructores:
  - **public DatagramPacket(byte ibuf[], int ilength)** : crea un datagrama para la recepción.
  - **public DatagramPacket(byte ibuf[], int ilength, InetAddress iaddr, int iport)** : crea un datagrama para el envío de información.
- Métodos:
  - **public InetAddress getAddress()**: IP al que se envía o del que se recibe.
  - **public byte [] getData ()** : datos a enviar o recibidos.
  - **public int getLength()** : longitud de los datos anteriores.
  - **public int getPort()** : puerto destino o del que se recibió.

Clase MulticastSocket

- La clase MulticastSocket facilita la recepción de datagramas enviados a direcciones IP de grupo (clase D)
- Creación:

```
MulticastSocket socket = new MulticastSocket(4446);
InetAddress address = InetAddress.getByName("230.0.0.1");
socket.joinGroup(address);
...
```

- Recepción

```
packet = new DatagramPacket(buf, buf.length);
socket.receive(packet);
```

- Envío

```
InetAddress group = InetAddress.getByName("230.0.0.1");
DatagramPacket packet = new DatagramPacket(buf, buf.length,
 group, 4446);
socket.send(packet);
```

Opciones en sockets (I)

- El paquete java.net define una interfaz para establecer y obtener las opciones de un socket (Socket, ServerSocket, DatagramSocket o MulticastSocket)
- Las opciones de sockets dan al programador un mayor control sobre el comportamiento de los sockets
- Get/setSoTimeout()
  - Para las clases Socket, ServerSocket, DatagramSocket
  - Antes de una lectura
  - La lectura lanza una excepción SocketTimeoutException cuando expira el temporizador
- get/setReceiveBufferSize() – get/setSendBufferSize()
  - Cambia el tamaño del buffer de lectura/escritura
  - Un objeto Socket y DatagramSocket tiene buffers de envío y recepción
  - Un objeto ServerSocket tiene sólo buffer de recepción

Opciones en sockets (II)

- get/setBroadcast()
  - Permite el la difusión en un DatagramSocket
  - Por defecto está habilitado
- Get/setReuseAddress()
  - En ServerSocket, DatagramSocket
  - Permite reutilizar la dirección (puerto)
- Get/setOOBInline()
  - Habilita en envío de datos fuera de línea (urgentes)
  - En Socket
  - Por defecto no está habilitada
- Get/setSoLinger()
  - En Socket
  - Habilita una espera antes del cierre
- Set/getTcpNoDelay()
  - En Socket
  - Deshabilita el algoritmo de Nagle en TCP