Proyecto FPMADdigital

Recursos digitales y multimedia para Formación Profesional

Comunidad de Madrid

Dirección General de Educación Secundaria, Formación Profesional y Régimen Especial

CONSEJERÍA DE EDUCACIÓN, UNIVERSIDADES, CIENCIA Y PORTAVOCÍA

Unión Europea

Fondo Social Europeo

"El FSE invierte en tu futuro"

Financiado como parte de la respuesta de la Unión a la pandemia de COVID-19

FPMADdigital
Recursos digitales y multimedia para Formación Profesional

CFGS Desarrollo de Aplicaciones Multiplataforma

módulo profesional

0490 - Programación de Servicios y Procesos

unidad didáctica

03 Programación de comunicaciones en red

resultados de aprendizaje

03 Programa mecanismos de comunicación en red empleando sockets y analizando el escenario de ejecución

Comunidad de Madrid
Dirección General de Educación Secundaria, Formación Profesional y Régimen Especial
CONSEJERÍA DE EDUCACIÓN, UNIVERSIDADES, CIENCIA Y PORTAVOCÍA

Unión Europea
Fondo Social Europeo
"El FSE invierte en tu futuro"
Financiado como parte de la respuesta de la Unión a la pandemia de COVID-19

Resultados de aprendizaje y unidades didácticas

Resultados de aprendizaje y unidades didácticas

RESULTADOS DE APRENDIZAJE					UNIDAD DIDÁCTICA
1	2	3	4	5	
X					1.- Programación multiproceso
	X				2.- Programación concurrente y asíncrona
		X			3.- Programación de comunicaciones en red
			X		4.- Generación de servicios en red
				X	5.- Técnicas de programación segura

Unidades didácticas y materiales asociados

Unidades didácticas y materiales multimedia

RRAA					UU.DD	Material Multimedia
1	2	3	4	5		
X					1.- Programación multiproceso	1.1 Contenidos básicos 1.2 Ejemplos aplicados
	X				2.- Programación concurrente y asíncrona	2.1 Contenidos básicos 2.2 Ejemplos aplicados
		X			3.- Programación de comunicaciones en red	3.1 Contenidos básicos 3.2 Ejemplos aplicados
			X		4.- Generación de servicios en red	4.1 Contenidos básicos 4.2 Ejemplos aplicados
				X	5.- Técnicas de programación segura	5.1 Contenidos básicos 5.2 Ejemplos aplicados

Repositorios de materiales y prácticas

Repositorio de materiales y prácticas

Todos los proyectos mostrados, así como otros materiales utilizados en las unidades didácticas los podrás encontrar completos en:

https://github.com/joseluisgs/FP-NextGen-ProgramacionServiciosProcesos

Cualquier error o propuestas de mejora se publicarán en el repositorio indicado.
Gracias por tu colaboración.

Contenidos

1. Protocolos
2. Programación con Sockets
3. Modelo Cliente-Servidor

Protocolos

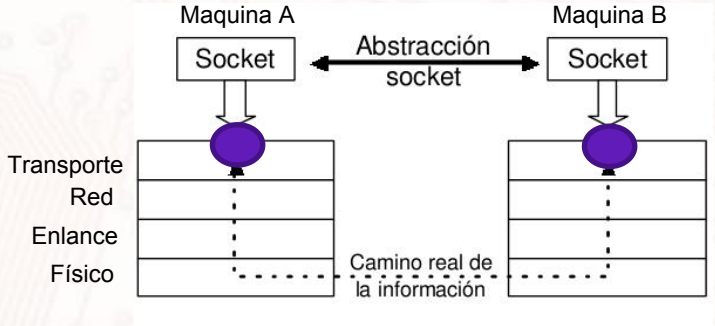
Protocolos

- **TCP** (Transmission Control Protocol). Es un **protocolo orientado a la conexión** que permite que un flujo de bytes originado en una máquina se entregue sin errores en cualquier máquina destino. Este protocolo fragmenta el flujo entrante de bytes en mensajes/**paquetes** y pasa cada uno a la capa de red. En el diseño, el proceso TCP receptor reensambla los mensajes recibidos para formar el flujo de salida. TCP también se encarga del control de flujo para asegurar que un emisor rápido no pueda saturar a un receptor lento con más mensajes de los que pueda gestionar.
- **UDP** (User Datagram Protocol). Es un **protocolo sin conexión**, para aplicaciones que no necesitan la asignación de secuencia ni el control de flujo TCP y que desean utilizar los suyos propios. Este protocolo también se utilizan para las consultas de petición y respuesta del tipo cliente-servidor bajo el uso de **datagramas**, y en aplicaciones en las que la velocidad es más importante que la entrega precisa, como las transmisiones de voz o de vídeo. Uno de sus usos es en la transmisión de audio y vídeo en tiempo real, donde no es posible realizar retransmisiones por los estrictos requisitos de retardo que se tiene en estos casos.

Programación con Sockets

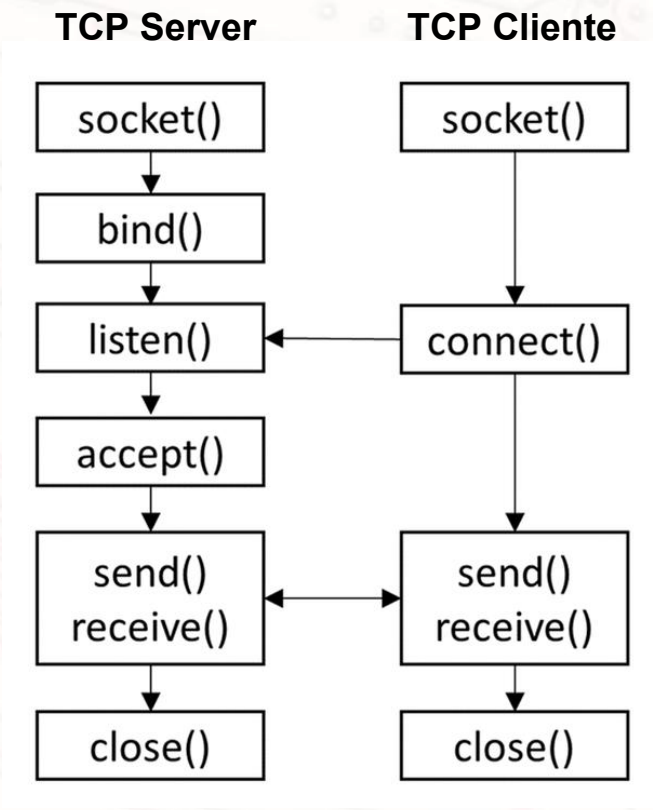
Socket

- Los **sockets** son un sistema de comunicación entre procesos de diferentes máquinas de una red. Más exactamente, un socket es un punto de comunicación por el cual un proceso puede emitir o recibir información.
- Un socket es un **punto final de un proceso de comunicación**. Es una abstracción que permite manejar de una forma sencilla la comunicación entre procesos, aunque estos procesos se encuentren en sistemas distintos, sin necesidad de conocer el funcionamiento de los protocolos de comunicación subyacentes. Tenemos **Stream** (TCP), **Datagramas** (UDP) y **Raw**
- Es así como estos “puntos finales” sirven de enlaces de comunicaciones entre procesos. **Los procesos tratan a los sockets como descriptores de ficheros, de forma que se pueden intercambiar datos con otros procesos transmitiendo y recibiendo a través de sockets**.



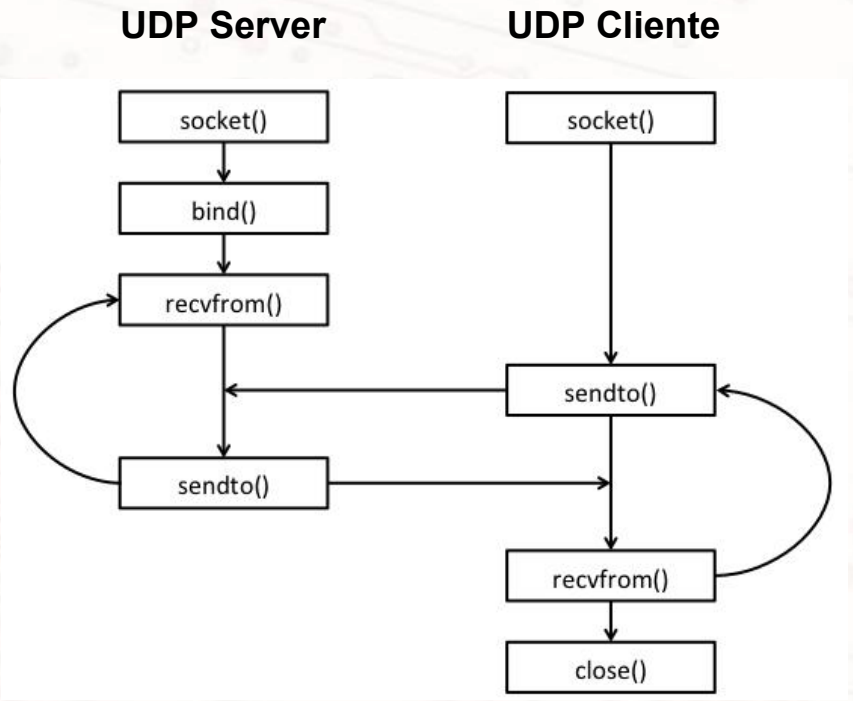
Socket: Stream (TCP)

- Son un servicio **orientado a la conexión**, donde los datos se transfieren sin encuadrarlos en registros o bloques, asegurándose de esta manera que **los datos lleguen al destino en el orden de transmisión**. Si se rompe la conexión entre los procesos, éstos serán informados de tal suceso para que tomen las medidas oportunas, por eso se dice que **están libres de errores**.
- El protocolo de comunicaciones con streams es un protocolo orientado a conexión, ya que para establecer una comunicación utilizando el protocolo **TCP** (Transmission Control Protocol), 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.**



Socket: Datagramas (UDP)

- Son un servicio de transporte **no orientado a la conexión**. Son más eficientes que TCP, pero en su utilización **no está garantizada la fiabilidad**. Los datos se envían y reciben en paquetes, cuya **entrega no está garantizada**. Los paquetes pueden ser duplicados, perdidos o llegar en un orden diferente al que se envió.
- Las comunicaciones a través de datagramas usan **UDP** (User Datagram Protocol), lo que significa que, 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, aunque **tiene la ventaja de que se pueden indicar direcciones globales y el mismo mensaje llegará a muchas máquinas a la vez.**



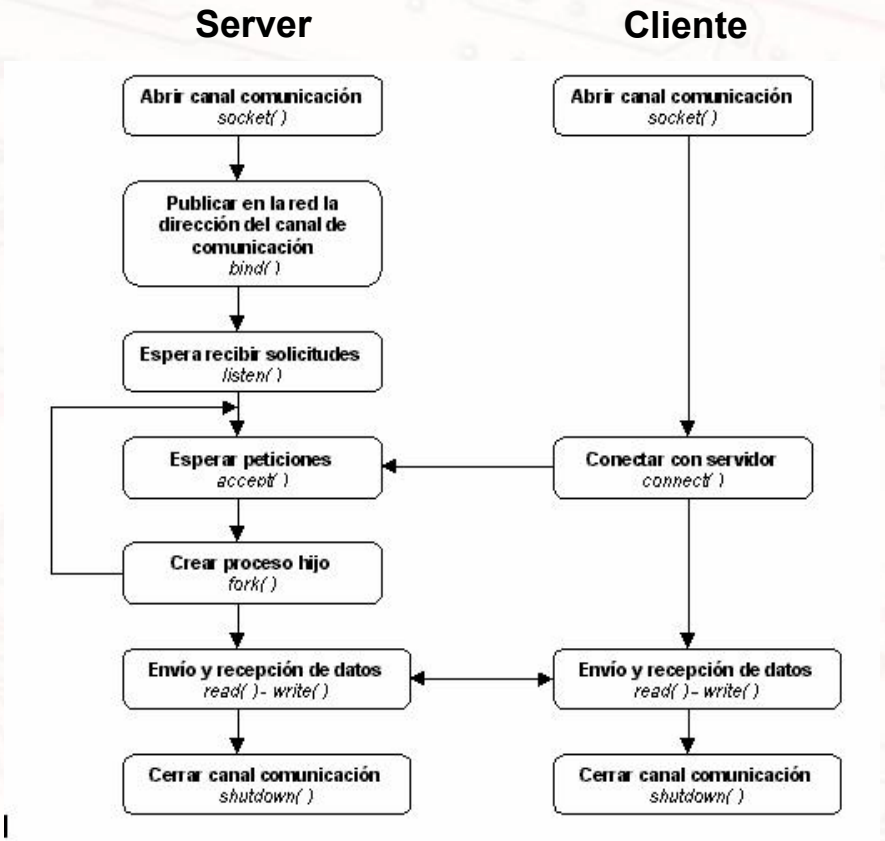
TCP vs UDP

- 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 los mensajes son más grandes que los TCP.
- Como el protocolo **TCP** está orientado a conexión, hay 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 es necesario emplear en UDP.
- En **UDP** hay un **límite de tamaño** de los datagramas, establecido en 64 kilobytes, que se pueden enviar a una localización determinada, mientras que **TCP no tiene límite**; 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 contrario, **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**. Sin embargo, **cuando se requiere un rendimiento óptimo, y está justificado el tiempo adicional que supone realizar la verificación de los datos, la comunicación a través de sockets TCP es un mecanismo realmente útil**.

Modelo Cliente–Servidor

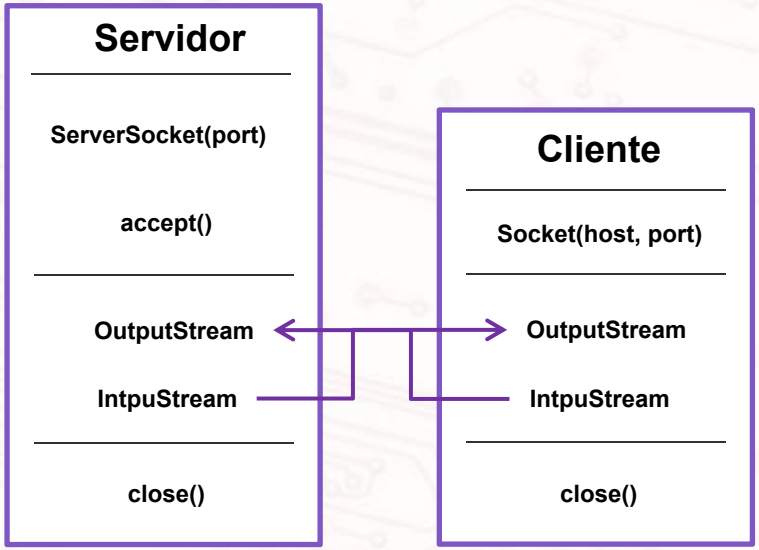
Modelo Cliente-Servidor

- El **servidor** se anuncia con un **Socket fijo**.
- **Cliente** realiza petición de conexión al servidor: **IP:Puerto**.
- Servidor acepta la solicitud y **obtiene un nuevo socket sobre un puerto diferente para seguir atendiendo** al socket original para peticiones de conexión mientras atiende las necesidades del cliente que se conectó.
- Por la parte del **cliente**, si la conexión es aceptada, un socket se crea de forma satisfactoria y puede usarlo para comunicarse con el servidor.
- Ahora el **cliente y el servidor pueden comunicarse escribiendo o leyendo en o desde sus respectivos sockets**.



Modelo Cliente–Servidor: JVM

- El **servidor** establece un **puerto** y espera durante un cierto tiempo (timeout) 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 máquina host a través del puerto que se designe en el parámetro respectivo.
- **El cliente y el servidor se comunican con manejadores InputStream y OutputStream**.
- **Fundametal es poner atencioón en diseñar bien el diálogo de intercambio de mensajes para no acabar bloqueando ni al servidor ni al cliente.**



```java
public class Servidor {

    public static void main(String argv[]) {

        //Definimos los Sockets
        ServerSocket servidor; // Socket de escucha del servidor
        Socket cliente; // Socket para atender a un cliente
        int numCliente = 0; // Contador de clientes
        int PUERTO = 5000; // Puerto para esuchar

        System.out.println("Soy el servidor y empiezo a escuchar peticiones por el puerto: " + PUERTO);

        try {
            // Apertura de socket para escuhar a través de un puerto
            servidor = new ServerSocket(PUERTO);
            // Atendemos a los clientes
            do {
                numCliente++;
                // Aceptamos la conexión
                cliente = servidor.accept();
                System.out.println("\t Llega el cliente: " + numCliente);
                DataOutputStream ps = new DataOutputStream(cliente.getOutputStream());
                ps.writeUTF("Usted es mi cliente: "+numCliente);
                // Y cerramos la conexión porque ya no vamos a oprrar más con él
                cliente.close();
                System.out.println("\t Se ha cerrado la conexión con el cliente: " +numCliente);
            } while (true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

```java
public class Cliente {

    public static void main(String argv[]) {
        // Definimos los parámetros de conexión
        InetAddress direccion; // La IP o Dirección de conexión
        Socket servidor; // Socket para conectarnos a un servidor u otra máquina
        int numCliente = 0; // Mi número de cliente
        int PUERTO = 5000;  // Puerto de conexión

        System.out.println("Soy el cliente e intento conectarme");

        try {
            // Vamos a indicar la dirección de conexión
            direccion = InetAddress.getLocalHost(); // dirección local (localhost)
            // Nos conectamos al servidor: dirección y puerto
            servidor = new Socket(direccion, PUERTO);
            // Operamos con la conexión. En este caso recibimos los datos que nos mandan
            System.out.println("Conexión realizada con éxito");

            // Es inputStream porque los recibimos
            DataInputStream datos = new DataInputStream(servidor.getInputStream());
            // Si queremos leer normal
            //System.out.println(datos.readLine());
            // Si leemos con formato
            System.out.println(datos.readUTF());
            // Cerramos la conexión
            servidor.close();
            System.out.println("Soy el cliente y cierro la conexión");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}

```

Conclusiones

¡Vamos con la práctica!

La programación es una carrera entre los desarrolladores, intentando construir mayores y mejores programas a prueba de idiotas, y el universo, intentanto producir mayores y mejores idiotas. Por ahora va ganando el Universo"

- Rich Cook

Comunidad de Madrid
Dirección General de Educación Secundaria, Formación Profesional y Régimen Especial
CONSEJERÍA DE EDUCACIÓN, UNIVERSIDADES, CIENCIA Y PORTAVOCÍA

Unión Europea
Fondo Social Europeo
"El FSE invierte en tu futuro"
Financiado como parte de la respuesta de la Unión a la pandemia de COVID-19