Computación Paralela

Departamento de Ciencias de la Computación

Aplicaciones

Sockets

Ing. Carlos Andrés Pillajo B, Msc.
capillajo@espe.edu.ec

30 de noviembre de 2023

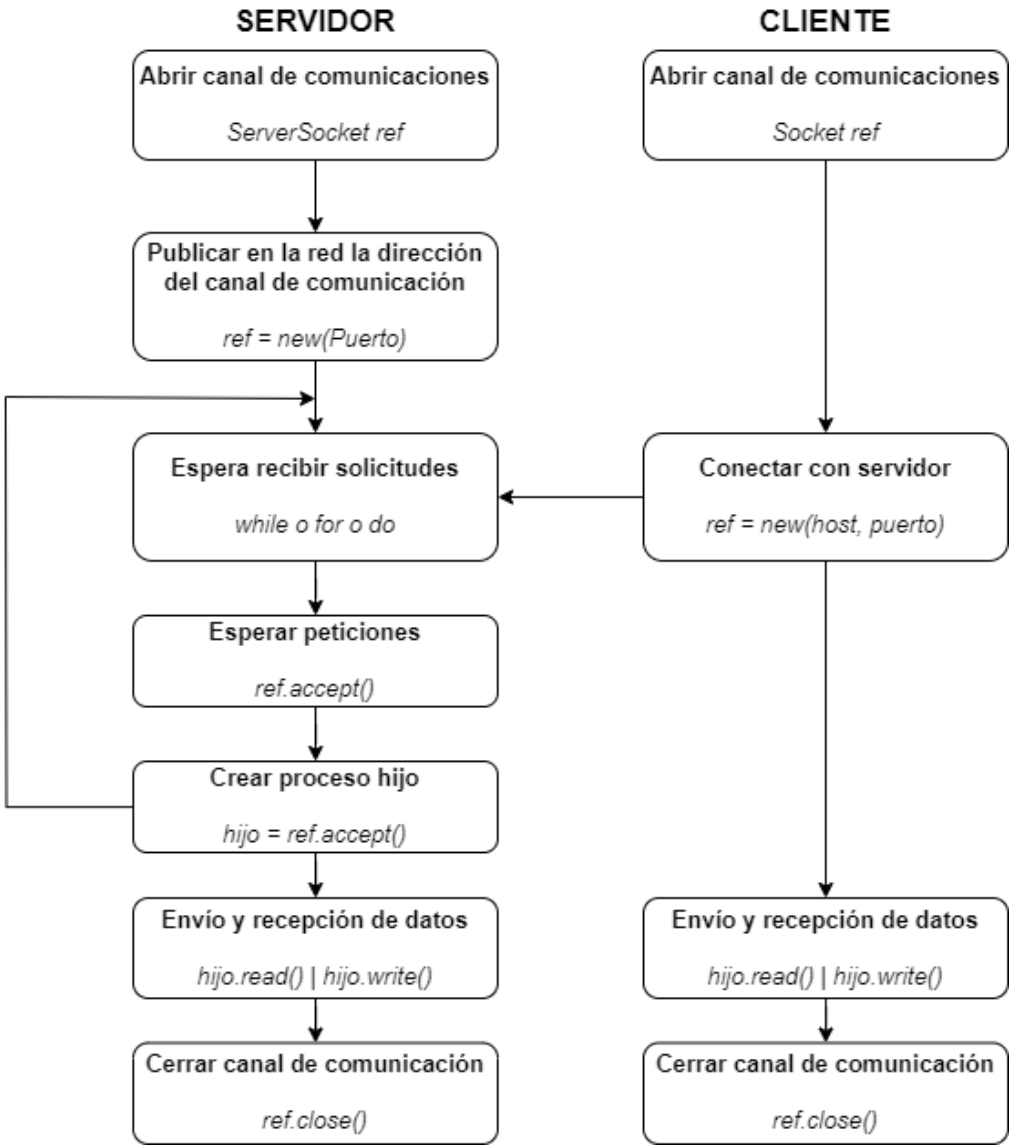
Contenido

Contenido

Fundamentos

- Los sockets son un sistema de comunicaciones entre procesos de diferentes máquinas de una red.
- Es un punto de comunicaciones por el cual un proceso puede enviar o recibir información.
- Son capaces de utilizar el protocolo de streams *TCP (Transfer Control Protocolo)* y el de datagramas *UDP (User Datagram Protocol)*.
- Es utilizado en la Arquitectura Cliente-Servidor.

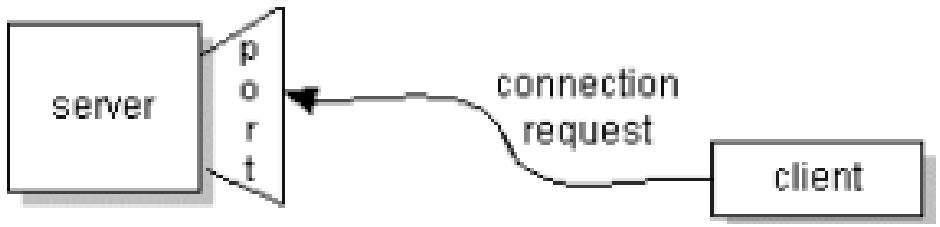
Fundamentos



Contenido

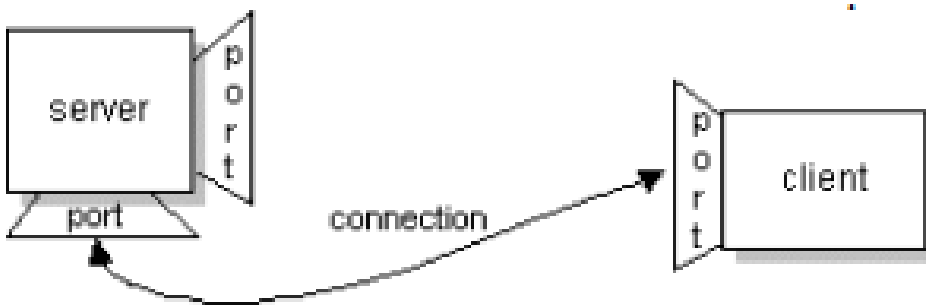
Funcionamiento

- Un servidor se ejecuta bajo una computadora específica y tiene un socket que responde en un puerto específico.
- El servidor únicamente espera, escuchando a través del socket a que un cliente haga una petición.
- El cliente conoce el nombre del host (ej. dirección IP) de la máquina en la cual el servidor se encuentra ejecunado y el número de puerto en el cual el servidor está conectado.
- Para realizar una petición de conexión, el cliente intenta encontrar al servidor en la máquina servidora en el puerto especificado.



Funcionamiento

- Si todo va bien, el servidor acepta la conexión.
- El servidor obtiene un nuevo puerto socket sobre un punto diferente.
- Esto se debe a que necesita un nuevo socket para seguir atendiendo al socket original para peticiones de conexión mientras atiende las necesidades del cliente que se conectó.



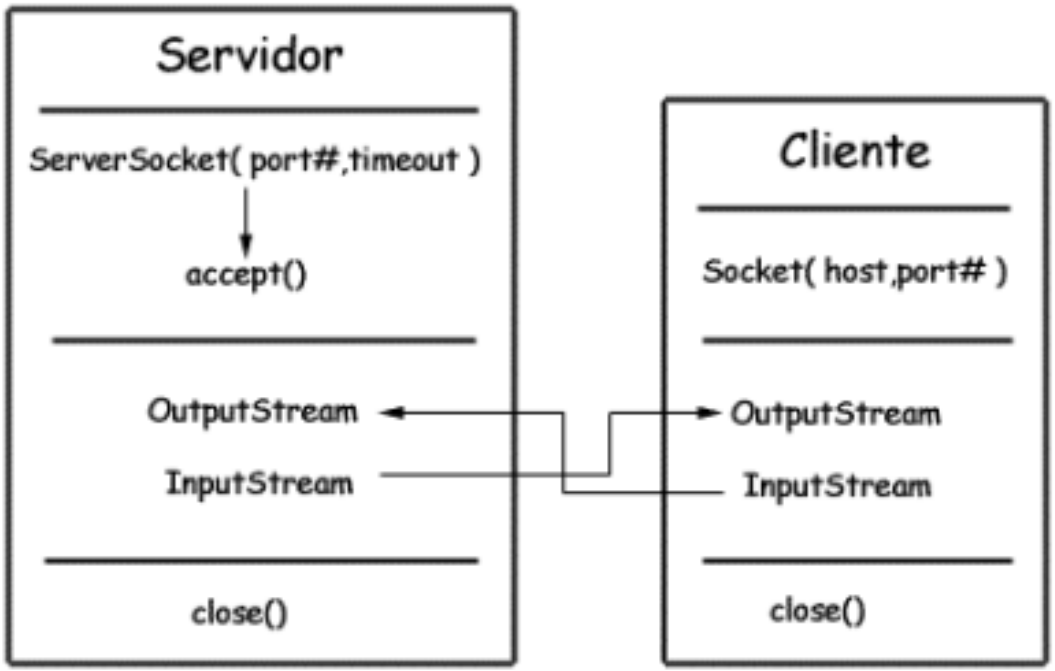
Funcionamiento

- Del lado del cliente, si la conexión es aceptada, un socket se crea de forma satisfactoria y puede usarlo para comunicarse con el servidor.
- Tomar en cuenta que el socket en el cliente no está utilizando el número de puerto usado para realizar la petición al servidor.
- En lugar de este, el cliente asigna un nuevo puerto local a la máquina en la cual está siendo ejecutado.
- Ahora el cliente y el servidor pueden comunicarse escribiendo o leyendo en o desde sus respectivos sockets.

Contenido

Ciclo de vida de un Socket

- **Crear socket:** Apertura del socket
- **Lectura y Escritura:** Recepción y envío de datos.
- **Destrucción:** Cierre del socket



Contenido

Introducción

- El paquete **java.net** de la plataforma Java proporciona una clase **Socket** y una clase **ServerSocket**.
- Usando la clase **java.net.Socket** en lugar de utilizar código nativo de la plataforma, los programas Java pueden comunicarse a través de la red de una forma totalmente independiente de la plataforma.
- La clase **ServerSocket** implementa un socket el cual, los servidores pueden utilizar para escuchar y aceptar peticiones de conexiones de clientes.

Apertura de Sockets

- Si estamos programando un **CLIENTE**, el socket de abre de la forma:

```
Socket cliente = new Socket(ip, puerto);
```

- **ip:** es la dirección IP de la máquina en donde estamos intentando abrir la conexión.
- **puerto:** es el puerto (un número) del servidor que está correindo sobre el cual nos queremos conectar.
  - Los puertos en el rango 0-1023 están reservados para usuarios con privilegios.
  - Estos puertos son los que utilizan los servicios estándar del sistema como email, ftp o http.
  - Para las aplicaciones que se desarrollen, asegurarse de seleccionar un puerto por encima del 1023.

Apertura de Sockets

- Si estamos programando un **SERVIDOR**, la forma de apertura del socket es la siguiente:

```
ServerSocket servidor = new ServerSocket(puerto);
```

- 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 cliente = servidor.accept();
```

Creación de Streams de Entrada

- En la parte **CLIENTE** de la aplicación, se puede utilizar la clase **DataInputStream** para crear un stream de entrada que esté listo a recibir todas las respuestas que el servidor le envíe.

```
DataInputStream entrada = new DataInputStream(cliente.getInputStream());
```

- La clase **DataInputStream** permite la lectura de líneas de texto y tipos de datos primitivos de Java de un modo altamente portable; dispone de métodos para leer todos esos tipos como: *read()*, *readChar()*, *readInt()*, *readDouble()* y *readLine()*.

Creación de Streams de Entrada

- En el lado del **SERVIDOR**, también usaremos **DataInputStream**, pero en este caso para recibir las entradas que se produzcan de los clientes que se hayan conectado

```
DataInputStream entrada = new DataInputStream(cliente.getInputStream());
```

Creación de Streams de Salida

- En el lado del **CLIENTE**, podemos crear un stream de salida para enviar información al socket del servidor utilizando las clases **PrintStream** o **DataOutputStream**

```
DataOutputStream salida = new DataOutputStream(cliente.getOutputStream());
```

Cierre de Sockets

- Siempre deberemos cerrar los canales de entrada y salida que se hayan abierto durante la ejecución de la aplicación.

```
entrada.close();
cliente.close();
```

- Es importante destacar que el orden de cierre es relevante.
- Es decir, se deben cerrar primero los streams relacionados con un socket antes que el propio socket

Servidor

```java
package servidor;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class Servidor {

        private int puerto;

        public Servidor(int puerto) {

                try {
                        ServerSocket servidor = new ServerSocket(puerto);
                        System.out.println("SERVER INICIADO - Esperando conexiones de clientes ...");

                        for (int i = 1; i <= 3; i++) {
                                Socket cliente = servidor.accept();
                                System.out.println("Se conecto el cliente " + i);

                                DataOutputStream salida = new DataOutputStream(cliente.getOutputStream());
                                salida.writeUTF("Hola cliente " + i);
                                salida.close();
                                cliente.close();
                        }

                        servidor.close();
                        System.out.println("SERVER TERMINADO");

                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }

        public static void main(String[] args) {

                new Servidor(10000);

        }
}
```

Cliente

```java
package cliente;
import java.io.DataInputStream;
import java.io.IOException;
import java.net.Socket;

public class Cliente {

        private int puerto;
        private String ip;

        public Cliente(String ip, int puerto) {
                this.ip = ip;
                this.puerto = puerto;

                try {
                        Socket cliente = new Socket(ip, puerto);

                        DataInputStream entrada = new DataInputStream(cliente.getInputStream());

                        System.out.println(entrada.readUTF());

                        entrada.close();
                        cliente.close();

                } catch (IOException e) {
                        e.printStackTrace();
                }
        }

        public static void main(String[] args) {
                new Cliente("localhost", 10000);

        }

}
```