Implementación de una conexión cliente-servidor haciendo uso de sockets en Java

Alejandro Andrade, Allan Panchi & Alex Trejo
Diciembre 2023.

Universidad de la Fuerzas Armadas - ESPE
DCCO
Computación Paralela

Contenido

Definición

En la actualidad los sistemas tecnológicos están interconectados por medio del modelo Cliente-Servidor, en el cual el objetivo es establecer una conexión óptima, eficiente, además de que intenta proveer usabilidad, flexibilidad, interoperabilidad y escalabilidad en la conexión. Para lo cual, se hace uso de un sinnúmero de herramientas para administrar la arquitectura en ambos lados de la comunicación, por un lado el cliente desempeña un equipo demandando servicios de los servidores, pero también puede realizar procesamiento local y por otro lado el servidor que desempeña un equipo ofreciendo un conjunto de servicios a los clientes, tales como manejo de archivos, impresión, páginas web, direccionamiento de correo electrónico, actualización de bases de datos y control de acceso. En otras palabras es una arquitectura de computación en la que se consigue un procesamiento cooperativo de la información por medio de un conjunto de procesadores, de tal forma que uno o varios clientes, distribuidos geográficamente o no, solicitan servicios de computación a uno o más servidores.
De esta forma, y gracias a esta arquitectura, la totalidad de los procesadores, clientes y servidores, trabajan de forma cooperativa para realizar un determinado tratamiento de la información. Para la conexión cliente-servidor se hará a través del uso de sockets en Java es un paradigma de comunicación que permite la transferencia de datos entre dos aplicaciones distintas, una actuando como el servidor y la otra como cliente. Los sockets "son un mecanismo que nos permite establecer un enlace entre dos programas que se ejecutan independientes el uno del otro (generalmente un programa cliente y un programa servidor)" (ProgramaYA, s.f.) Java por medio de la librería java.net nos provee dos clases: Socket para implementar la conexión desde el lado del cliente y ServerSocket que nos permitirá manipular la conexión desde el lado del servidor. En este modelo, el servidor espera pasivamente por solicitudes de conexión entrantes, mientras que el cliente inicia la comunicación. Una vez establecida la conexión, ambas partes pueden enviar y recibir datos de manera sincrónica o asincrónica, dependiendo de la implementación específica.

Objetivos

Objetivo General

Realizar un programa que haga uso de la clase Sockets en la que se establezca una conexión cliente-servidor entre dos computadoras, ya sean conectadas a la misma red o por medio de un cable ethernet, haciendo que una computadora trabaje como servidor y, en este caso, la otra computadora trabaje como un cliente, después de ejecutar la conexión, implementar un algoritmo que para que el servidor (computadora 1) reciba peticiones del cliente (computadora 2) y que el servidor pueda responder, todo este programa se le realizará en el IDE Apache Netbeans

Objetivo Específico:

- Hacer que un cliente pueda realizar peticiones y que reciba una respuesta del servidor a través de una conexión de red.

- Implementar un código funcional y estable para que no caiga el servidor después de que el cliente realice una petición
- Hacer que el servidor responda con éxito a las peticiones hechas por el cliente, las cuales se las harán por otra computadora
- Implementar un mecanismo de manejo de errores eficiente en el código, asegurando que el programa sea capaz de identificar y gestionar situaciones inesperadas durante la conexión cliente-servidor. Esto incluye la detección y manejo de posibles fallos en la red, así como la gestión adecuada de excepciones y errores en la comunicación, con el fin de mantener la estabilidad y robustez del sistema.

Herramientas

- Apache Netbeans (IDE)
  Entorno de desarrollo integrado libre, que facilita la implementación de hilos, además de ser un espacio óptimo para dichos procesos debido a que trabaja principalmente con Java.
- Biblioteca Socket
  Biblioteca ya incorporada en el NetBeans, gracias a esta podemos usar métodos que nos van a ayudar a realizar nuestra conexión cliente-servidor
- IA
  Herramienta utilizada para guiarse y para consultar sobre métodos pertenecientes a la clase Socket

Desarrollo

A continuación se explicarán algunas líneas importantes de código tanto de la clase Servidor como de la clase Cliente y el método utilizado para que un servidor pueda dar respuesta de forma simultánea a varios clientes.

Clase Servidor:

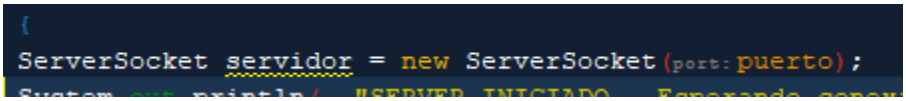


Fig 1. Se crea un ServerSocket que escucha en el puerto especificado.

En resumen, se crea una variable servidor que va a funcionar como un gestor para las conexiones entrantes, aceptar solicitudes y establecer una comunicación entre ellos

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

    DataOutputStream salida = new DataOutputStream(out:cliente.getOutputStream());
    salida.writeUTF("Hola cliente " + i);
    salida.close();
    cliente.close();
}
servidor.close();
System.out.println(x: "SERVER TERMINADO");
```

Fig 2. Bucle for y conexión entre servidor-cliente

Dentro de un bucle for (hecho para admitir hasta tres cliente) espera y acepta la conexión de un cliente, creando un objeto Socket para manejar la comunicación con ese cliente.
Se crea una variable salida en la que se utilizará para enviar datos al cliente a través del flujo de salida asociado al socket.
Con el método writeUTF(), obtiene el flujo de salida del cliente y envía un mensaje
El método close() aplicado en la salida y el cliente es para cerrar el flujo y el socket asociado después de enviar el mensaje, luego con servidor.close() finaliza la conexión después de responder a los 3 clientes

```
public static void main(String[] args) {
    new Servidor(puerto: 10000);
}
```

Fig 3. Implementaciòn de la conexiòn de forma local

En el método main, se instancia un objeto de la clase Servidor con el puerto 10000, lo que inicia la ejecución del servidor.

Clase Cliente

Se va a explicar el try dentro del cliente ya que aquí se genera la conexión con el servidor

```
public Cliente(String ip, int puerto) {
    this.ip = ip;
    this.puerto = puerto;
    try {
        Socket cliente = new Socket(host: ip, port: puerto);
        DataInputStream entrada = new DataInputStream(in: cliente.getInputStream());
        System.out.println(x: entrada.readUTF());
        entrada.close();
        cliente.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Fig 4. Conexión del cliente con el servidor

Primero se crea un nuevo objeto Socket llamado cliente. Esto representa la conexión del cliente al servidor. La dirección ip y el número de puerto se proporcionan como argumentos para especificar la ubicación del servidor al cual el cliente intentará conectarse.

Se crea una objeto DataInputStream el cual se le llama entrada que está asociada al flujo de entrada del socket del cliente, esto permitirá que el cliente pueda recibir datos del servidor

Con el readUTF() para leer una cadena de caracteres desde el flujo de entrada asociado al socket del cliente.

Después de imprimir el mensaje, se cierran los recursos, tanto el cliente como la entrada, lo hacemos para poder liberar

Funcionamiento de la arquitectura Cliente – Servidor sobre una red alámbrica y/o inalámbrica

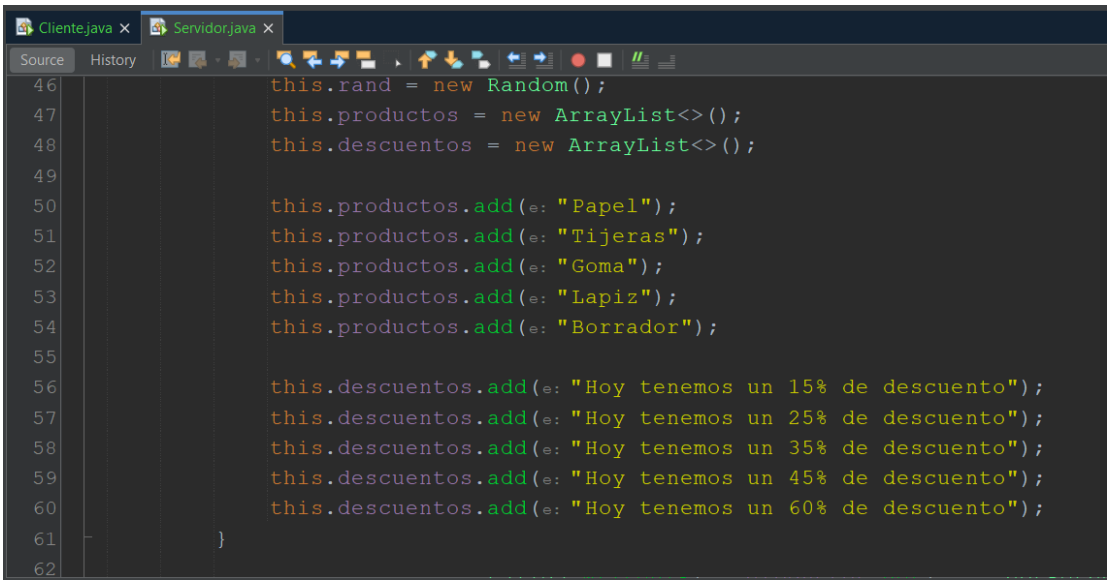
Red alámbrica: para hacer uso de la arquitectura Cliente - Servidor de forma alámbrica se requiere al menos un cable Ethernet que sirva de puente entre el computador que va a funcionar de cliente y el computador que funcionará de servidor. Es importante quitar la conexión Wi-Fi para que este método funcione, posteriormente se desactiva el firewall de windows, después existe la opción de usar la dirección IP asignada automáticamente sin embargo para esta práctica se seleccionaron direcciónes IP específicas tanto para el cliente como para el servidor, en el caso del servidor se utilizó 192.168.1.2 y para el cliente 192.168.1.100 ambas con mascara de subred 255.255.255.0. Cabe destacar que para realizar una conexión alámbrica con más de un cliente es necesario un dispositivo switch.

Red inalámbrica: Para la práctica este método resulta más práctico teniendo en cuenta que no requiere de un dispositivo switch para poder usar más de un cliente para un solo servidor, basta con estar conectados a la misma red Wi-Fi y usar el comando ipconfig en la terminal de windows del servidor para saber la dirección IP a la que los clientes deben conectarse.

Análisis de resultados

Funcionamiento y explicación a solicitudes puntuales (productos-descuentos)

Para esta práctica se crearon 2 ArrayList que almacenen datos respecto a productos y mensajes de descuentos diarios.



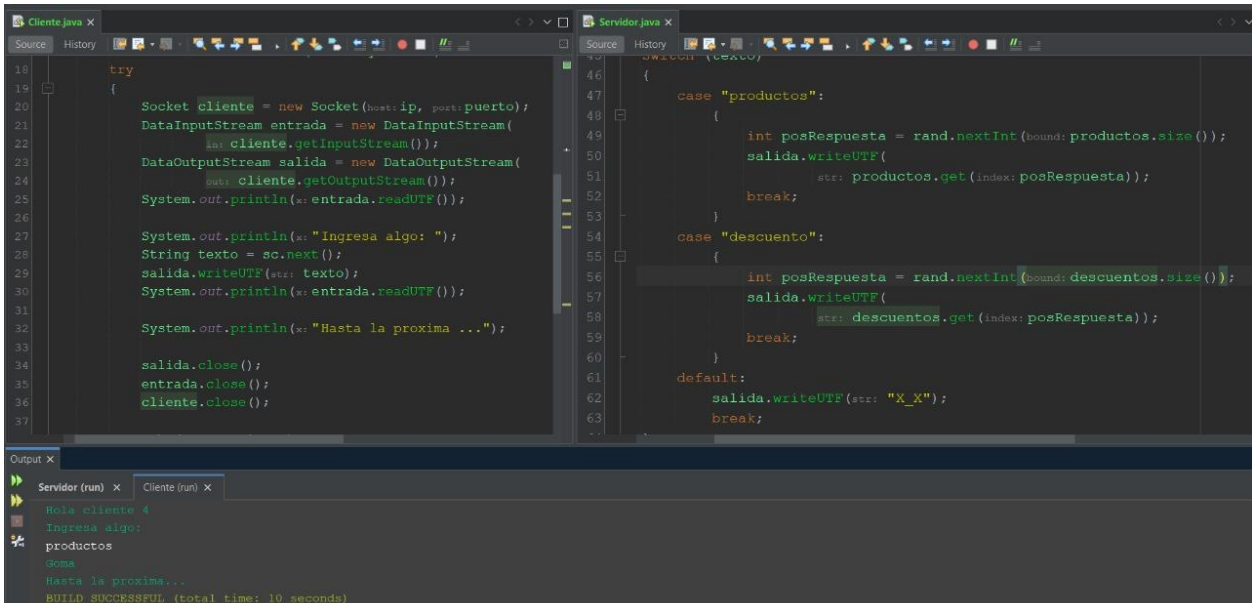
```
this.rand = new Random();
this.productos = new ArrayList<>();
this.descuentos = new ArrayList<>();

this.productos.add("Papel");
this.productos.add("Tijeras");
this.productos.add("Goma");
this.productos.add("Lapiz");
this.productos.add("Borrador");

this.descuentos.add("Hoy tenemos un 15% de descuento");
this.descuentos.add("Hoy tenemos un 25% de descuento");
this.descuentos.add("Hoy tenemos un 35% de descuento");
this.descuentos.add("Hoy tenemos un 45% de descuento");
this.descuentos.add("Hoy tenemos un 60% de descuento");
}
```

Fig. 6. Agregación de datos quemados a cada ArrayList

Para generar una respuesta a partir del valor que ingrese el cliente se hace uso de un switch case en el que para cada caso y dependiendo del valor que envíe el cliente genera una respuesta aleatoria que contenga el ArrayList correspondiente.



```
try
{
    Socket cliente = new Socket(host: ip, port: puerto);
    DataInputStream entrada = new DataInputStream(
            in: cliente.getInputStream());
    DataOutputStream salida = new DataOutputStream(
            out: cliente.getOutputStream());
    System.out.println(x: entrada.readUTF());

    System.out.println(x: "Ingresa algo: ");
    String texto = sc.next();
    salida.writeUTF(str: texto);
    System.out.println(x: entrada.readUTF());

    System.out.println(x: "Hasta la proxima ...");

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

```
{
    case "productos":
        {
            int posRespuesta = rand.nextInt(bound: productos.size());
            salida.writeUTF(
                    str: productos.get(index: posRespuesta));
            break;
        }
    case "descuento":
        {
            int posRespuesta = rand.nextInt(bound: descuentos.size());
            salida.writeUTF(
                    str: descuentos.get(index: posRespuesta));
            break;
        }
    default:
        salida.writeUTF(str: "X_X");
        break;
```

```
Output
Servidor (run)
Cliente (run)
Hola cliente 4
Ingresa algo:
productos
Goma
Hasta la proxima...
BUILD SUCCESSFUL (total time: 10 seconds)
```

Fig. 7. Respuesta a partir de la petición

Más adelante se realiza un ciclo infinito para que se pida ingresar un valor cada justo después de haber dado una respuesta y haciendo uso de la palabra “salir” el cliente puede terminar la ejecución.

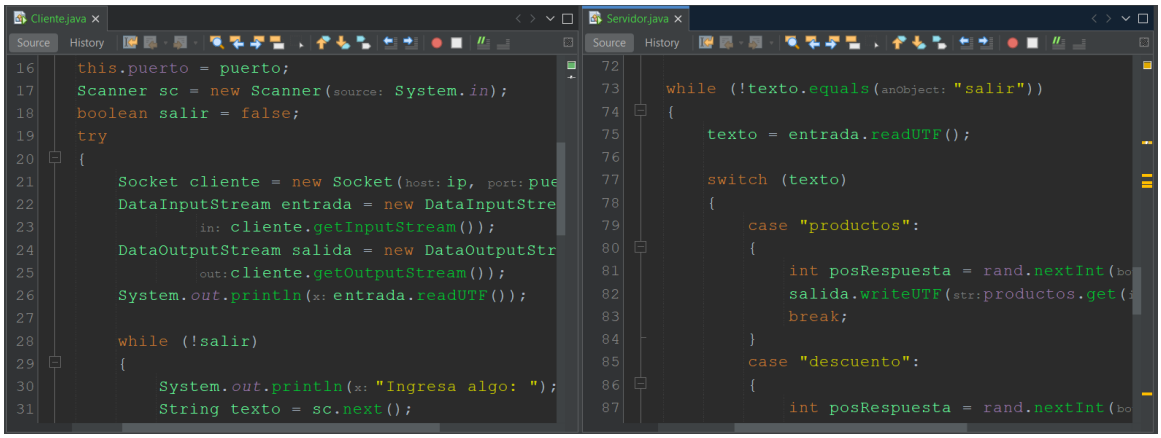


Fig. 8. Bucle para hacer peticiones infinitas.

Aplicación de Hilos:

Dado que se requiere usar un solo servidor y que este genere una respuesta a partir de los datos enviados por el cliente teniendo en cuenta que debe existir respuesta de forma simultánea para cada cliente se hace uso de hilos cada cliente porque en caso de no usar hilos el servidor espera a que se termine la sesión de cada cliente para dar respuesta al resto de clientes. La forma de implementar el hilo es crear una clase que maneje a cada cliente como si fuera un hilo por lo cual cuando se ejecuta el cliente especificando la dirección y el puerto del servidor se llama directamente a iniciar el hilo. Dentro del método run del hilo se encuentran las peticiones y respuestas del servidor. y se utiliza un contador dentro y fuera del hilo para que el cliente como el que administra el servidor sepan qué número de cliente es.

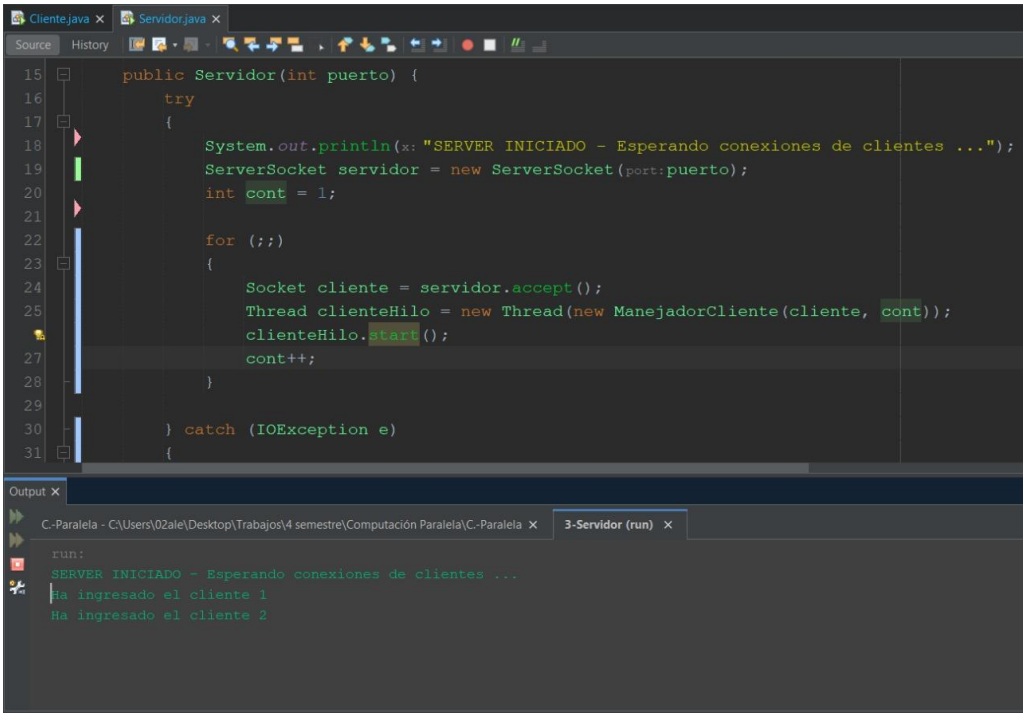


Fig. 5. Creación de hilo para cada cliente

Dentro de la clase ManejadorCliente se encuentran los atributos privados cliente de tipo Socket, cont de tipo int que servirá para saber el número de clientes que han ingresado, productos de tipo ArrayList<String> y descuentos de tipo ArrayList<String>.

Por lo tanto, la modificación del código para lograr la comunicación efectiva entre el Cliente y el Servidor en una red, en donde el cliente envíe solicitudes específicas al servidor, las cuales son respondidas de manera adecuada según el criterio de descuento y producto, sea realizada con éxito utilizando el paradigma de sockets en Java, se crea una arquitectura cliente-servidor que permite la transferencia bidireccional de datos de esta forma se gestiona de manera eficiente y simultánea las solicitudes de un número “n” de clientes.

Respuestas específicas

En la implementación, se buscó permitir que el cliente realice dos tipos de solicitudes puntuales: "descuento" y "productos". Al enviar la solicitud "descuento", el servidor responde de manera aleatoria con una frase que indica el descuento del día, seleccionada de una lista predefinida de frases de descuentos. Asimismo, al solicitar "productos", el servidor responde con el nombre de un producto de una papelería, seleccionado de forma aleatoria desde una lista predeterminada, que se realiza en la clase “ManejadorCliente” del servidor, donde se utiliza un Random para seleccionar aleatoriamente una frase de descuento desde la lista predefinida y de igual forma con los productos.

De modo que, para visualizar la variabilidad de respuestas proporcionadas por el servidor, se incluye una tabla de comparación de respuestas aleatorias para las solicitudes "descuento" y "productos".

Solicitud	Respuesta
descuento	Hoy tenemos un 60% de descuento
producto	Lapiz
descuento	Hoy tenemos un 15% de descuento
producto	Papel

Tabla 1: Respuestas del servidor

Conclusión

Se logró exitosamente desarrollar un programa funcional que establece una comunicación eficiente entre dos computadoras, una actuando como servidor y la otra como cliente. El uso de hilos permitió manejar múltiples clientes simultáneamente, mejorando así la capacidad de respuesta del sistema.

Se alcanzó el objetivo general de establecer una conexión cliente-servidor, y los objetivos específicos también se cumplieron de manera efectiva. El cliente puede realizar solicitudes al servidor y recibir respuestas de manera adecuada. Se implementó un código estable que gestiona

eficientemente las peticiones de los clientes, evitando caídas inesperadas del servidor después de recibir una solicitud.

La respuesta a las solicitudes específicas, como descuentos y productos, se realizó de manera aleatoria y coherente, proporcionando variabilidad en las respuestas y cumpliendo con la funcionalidad esperada. El uso de listas predefinidas para descuentos y productos permitió una fácil expansión y adaptación del sistema a diferentes contextos.

Recomendaciones

Sería beneficioso considerar la implementación de medidas adicionales de seguridad y manejo de excepciones para fortalecer la robustez del sistema, especialmente en entornos de red no controlados. Además, se podría explorar la posibilidad de extender la funcionalidad del sistema, agregando nuevas solicitudes y mejorando la interactividad entre el cliente y el servidor. Por otro lado el implementar un mecanismo de autenticación y autorización para garantizar la seguridad en las conexiones.
La inclusión de un sistema de autenticación sólido ayudaría a prevenir accesos no autorizados y proteger la integridad de los datos transmitidos entre el cliente y el servidor. Además, la aplicación de roles y permisos permitiría un control más granular sobre las acciones que cada cliente puede realizar, aumentando así la seguridad general del sistema.
De igual forma el uso de protocolos seguros, como SSL/TLS, podría proporcionar una capa adicional de protección contra posibles amenazas, asegurando la confidencialidad y la integridad de los datos durante la transmisión.
Asimismo, se podría considerar la optimización del manejo de errores para proporcionar mensajes más descriptivos y amigables al usuario final.
Adicionalmente, implementar un sistema de gestión de conexiones más avanzado que permita manejar un mayor número de clientes de manera eficiente.

Referencias

González, J. D. M. (2021, 20 junio). *Sockets (Cliente-Servidor)*. https://www.programarya.com/Cursos-Avanzados/Java/Sockets

W3Api. (s. f.). *W3API*. https://www.w3api.com/Java/DataOutputStream-java-io/writeUTF/

V.2. Los sockets en java. (s. f.). https://cruzado.info/tutojava/V_2.htm

Villalobos, J., Hernandez, U., del silicon power Armor, R., Reseña, U. S. B., & Marvel, S. P. (2011). Sockets en Java (cliente y servidor). *Codigoprogramación. com*, *7*.