

Pontificia Universidad Javeriana



Laboratorio Cliente Servidor

Autor:

Juan Santiago Saavedra Holguín

29 de agosto de 2025

Bogotá D.C.

Índice

1. Resumen	3
2. Arquitectura	4
3. Código fuente	5
4. Ejecución y resultados	8
5. Análisis	9
6. Conclusión	10

1. Resumen

En este laboratorio se desarrolló una aplicación basada en el modelo cliente-servidor utilizando sockets TCP en el lenguaje de programación Java. El enfoque principal fue implementar un servidor multihilo capaz de aceptar múltiples conexiones de clientes de manera concurrente.

El sistema consiste en un servidor que escucha en un puerto específico y genera un hilo independiente para cada cliente que se conecta. Cada cliente envía un número entero, y el servidor responde devolviendo el cuadrado de dicho número.

Durante las pruebas, se verificó el funcionamiento tanto en conexiones locales como en clientes remotos, logrando observar la correcta concurrencia. También se identificaron algunas limitaciones, principalmente asociadas al manejo de entradas no válidas y desbordamientos numéricos.

Este ejercicio permitió afianzar conocimientos de programación concurrente y comunicación en red, elementos fundamentales en la formación de un ingeniero en sistemas.

2. Arquitectura

La arquitectura del sistema se basa en el modelo clásico de cliente-servidor, pero extendido con manejo de concurrencia para atender múltiples clientes de forma simultánea.

- **Servidor principal** (`MultithreadedSocketServer`): se encarga de inicializar el socket de escucha en el puerto 8888 y aceptar las conexiones entrantes. Por cada nuevo cliente, lanza un hilo independiente.
- **Hilo por cliente** (`ServerClientThread`): gestiona la comunicación individual con cada cliente. Recibe el número, calcula el cuadrado y envía la respuesta.
- **Cliente** (`TCPClient`): solicita un número al usuario mediante consola, lo envía al servidor y muestra la respuesta obtenida.

De esta manera, se logra un sistema escalable donde múltiples usuarios pueden interactuar con el servidor sin bloquearse entre sí.

3. Código fuente

A continuación, se presenta el código implementado. Este fue documentado y estructurado para facilitar su comprensión.

MultithreadedSocketServer.java

```
1  /**
2   * Pontificia Universidad Javeriana
3   * Autor: Juan Santiago Saavedra Holgu n
4   * Materia: Sistemas Operativos
5   * Tema: Cliente-Servidor TCP multihilo
6   * Fichero: MultithreadedSocketServer.java
7   */
8  import java.net.*;
9  import java.io.*;
10
11 public class MultithreadedSocketServer {
12     public static void main(String[] args) {
13         try {
14             ServerSocket server = new ServerSocket(8888);
15             int counter = 0;
16             System.out.println("Server Started ....");
17             while (true) {
18                 counter++;
19                 Socket serverClient = server.accept();
20                 System.out.println(" >> Client No:" + counter + " started!
21                                     ");
22                 ServerClientThread sct = new ServerClientThread(
23                     serverClient, counter);
24                 sct.start();
25             }
26         } catch (Exception e) {
27             System.out.println(e);
28         }
29     }
30 }
```

ServerClientThread.java

```
1  /**
2   * Pontificia Universidad Javeriana
3   * Autor: Juan Santiago Saavedra Holgu n
4   * Materia: Sistemas Operativos
5   * Tema: Cliente-Servidor TCP multihilo
6   * Fichero: ServerClientThread.java
7   */
8  import java.io.*;
9  import java.net.*;
10
11  class ServerClientThread extends Thread {
12      Socket serverClient;
13      int clientNo;
14      int sqre;
15
16      ServerClientThread(Socket inSocket, int counter){
17          serverClient = inSocket;
18          clientNo = counter;
19      }
20
21      public void run(){
22          try {
23              DataInputStream inStream = new DataInputStream(serverClient.
24                  getInputStream());
25              DataOutputStream outStream = new DataOutputStream(serverClient
26                  .getOutputStream());
27              String clientMessage = "", serverMessage = "";
28              while(!clientMessage.equals("bye")){
29                  clientMessage = inStream.readUTF();
30                  System.out.println("From Client-" + clientNo + ": Number
31                      is : " + clientMessage);
32                  int number = Integer.parseInt(clientMessage);
33                  sqre = number * number;
34                  serverMessage = "From Server to Client-" + clientNo + ":
35                      Square of " + number + " is " + sqre;
36                  outStream.writeUTF(serverMessage);
37                  outStream.flush();
38              }
39              inStream.close();
40              outStream.close();
41              serverClient.close();
42          } catch (Exception ex){
43              System.out.println(ex);
44          } finally{
45              System.out.println("Client -" + clientNo + " exit!! ");
46          }
47      }
48  }
```

TCPClient.java

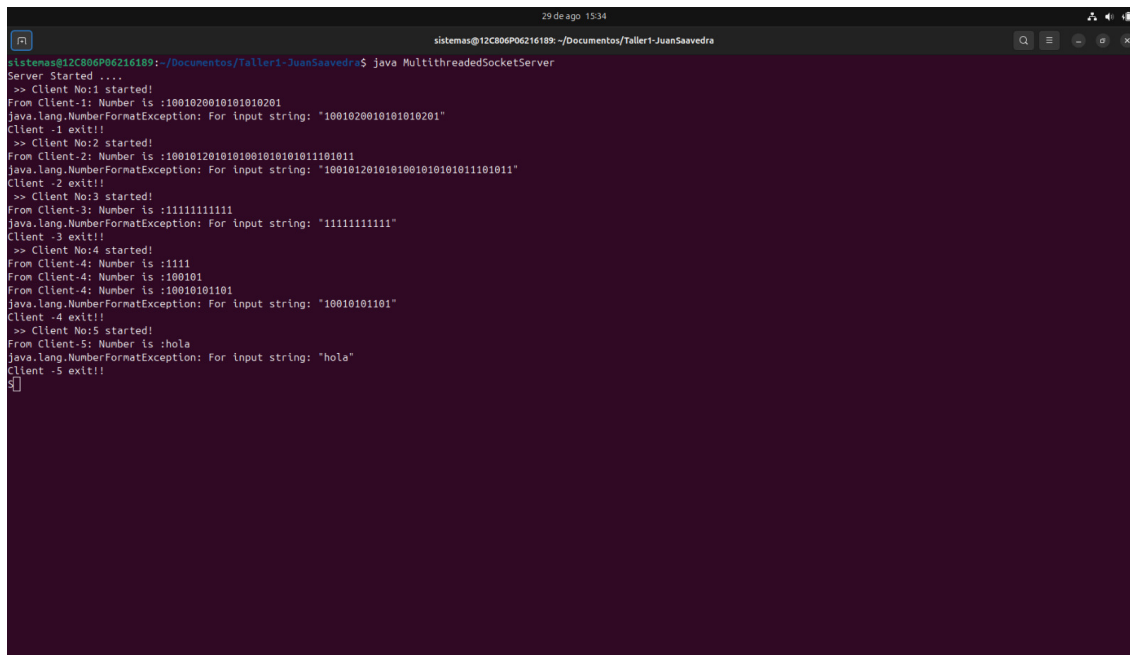
```
1  /**
2   * Pontificia Universidad Javeriana
3   * Autor: Juan Santiago Saavedra Holgu n
4   * Materia: Sistemas Operativos
5   * Tema: Cliente-Servidor TCP multihilo
6   * Fichero: TCPClient.java
7   */
8  import java.net.*;
9  import java.io.*;
10
11 public class TCPClient {
12     public static void main(String[] args) {
13         try {
14             Socket socket = new Socket("127.0.0.1", 8888);
15             DataInputStream inStream = new DataInputStream(socket.
16                 getInputStream());
17             DataOutputStream outStream = new DataOutputStream(socket.
18                 getOutputStream());
19             BufferedReader br = new BufferedReader(new InputStreamReader(
20                 System.in));
21             String clientMessage = "", serverMessage = "";
22             while(!clientMessage.equals("bye")) {
23                 System.out.println("Enter number :");
24                 clientMessage = br.readLine();
25                 outStream.writeUTF(clientMessage);
26                 outStream.flush();
27                 serverMessage = inStream.readUTF();
28                 System.out.println(serverMessage);
29             }
30             inStream.close();
31             outStream.close();
32             socket.close();
33         } catch(Exception e) {
34             System.out.println(e);
35         }
36     }
37 }
```

4. Ejecución y resultados

En esta sección se presentan las evidencias gráficas de la ejecución del sistema. Se realizaron pruebas con varios clientes que enviaban números al servidor y recibían la respuesta correspondiente.

```
^Csistemas@12C806P08216205:~/Descargas$ java TCPClient
Enter number :
98888
From Server to Client-2 Square of 98888 is 1188901952
Enter number :
^Csistemas@12C806P08216205:~/Descargas$ java TCPClient
Enter number :
10010101001
java.io.EOFException
sistemas@12C806P08216205:~/Descargas$ java TCPClient
Enter number :
1001020010101010201
hol
hola nuevo
^Csistemas@12C806P08216205:~/Descargas$ java TCPClient
Enter number :
1001012010101001010101011101011
^Csistemas@12C806P08216205:~/Descargas$
sistemas@12C806P08216205:~/Descargas$ java TCPClient
Enter number :
11111111111
^Csistemas@12C806P08216205:~/Descargas$ java TCPClient
Enter number :
1111
From Server to Client-4 Square of 1111 is 1234321
Enter number :
100101
From Server to Client-4 Square of 100101 is 1430275609
Enter number :
10010101101
^Csistemas@12C806P08216205:~/Descargas$ java TCPClient
Enter number :
hola
sistemas@12C806P08216205:~/Descargas$
```

Figura 1: Cliente enviando números y recibiendo respuesta.



```
sistemas@12C806P06216189: ~/Documentos/Taller1-JuanSaavedra$ java MultithreadedSocketServer
Server Started ....
>> Client No:1 started!
From Client-1: Number is :1001020010101010201
java.lang.NumberFormatException: For input string: "10010200101010201"
Client -1 exit!!!
>> Client No:2 started!
From Client-2: Number is :10010120101010010101011101011
java.lang.NumberFormatException: For input string: "10010120101010010101011101011"
Client -2 exit!!!
>> Client No:3 started!
From Client-3: Number is :111111111111
java.lang.NumberFormatException: For input string: "111111111111"
Client -3 exit!!!
>> Client No:4 started!
From Client-4: Number is :1111
From Client-4: Number is :100101
From Client-4: Number is :10010101101
java.lang.NumberFormatException: For input string: "10010101101"
Client -4 exit!!!
>> Client No:5 started!
From Client-5: Number is :hola
java.lang.NumberFormatException: For input string: "hola"
Client -5 exit!!!
```

Figura 2: Servidor aceptando múltiples clientes y procesando cálculos.

5. Análisis

Desde el punto de vista de un ingeniero en formación, este laboratorio representó un reto importante para comprender cómo se gestiona la comunicación en red a bajo nivel mediante sockets.

El sistema cumplió su propósito básico: demostrar que un servidor puede atender múltiples clientes de forma concurrente. Sin embargo, también permitió evidenciar ciertas debilidades que son comunes en aplicaciones reales:

- Manejo limitado de errores: la aplicación puede fallar al recibir caracteres no numéricos.
- Problemas de **overflow**: al elevar números grandes al cuadrado, el tipo `int` no es suficiente.
- Experiencia del usuario: la interacción en consola es funcional pero poco amigable.

Estos hallazgos son valiosos porque muestran que en ingeniería de software no basta con que un programa “funcione”, sino que debe ser robusto y escalable.

6. Conclusión

La práctica realizada permitió fortalecer conocimientos de:

- Programación concurrente en Java.
- Comunicación cliente-servidor mediante sockets TCP.
- Identificación y análisis de errores en aplicaciones distribuidas.

En conclusión, este laboratorio no solo evidenció la correcta implementación del sistema, sino que también dejó claro que los problemas de validación de entradas, control de excepciones y gestión de recursos son aspectos fundamentales que deben abordarse en cualquier desarrollo real.

Como estudiante de ingeniería en formación, considero que este ejercicio es un paso importante para acercarse a los retos del mundo profesional, donde los sistemas distribuidos y la concurrencia son cada vez más comunes.

Compilación y ejecución

Para compilar y ejecutar el sistema se deben seguir los siguientes pasos:

Compilar

```
javac MultithreadedSocketServer.java ServerClientThread.java TCPClient.java
```

Ejecutar el servidor

```
java MultithreadedSocketServer
```

Ejecutar un cliente (en otra terminal o máquina)

```
java TCPClient
```

Se recomienda iniciar el servidor en una terminal y luego ejecutar varios clientes en diferentes ventanas para comprobar la concurrencia.