

Instituto Politécnico Nacional

Escuela Superior de Cómputo

Materia: Sistemas Distribuidos

Profesor: Carreto Arellano Chadwick

Grupo: 7CM1

Práctica 2. Sockets C/S

Valdés Castillo Diego - 2021630756

Fecha de Entrega: 03/03/2025

Índice

1. Antecedente (Marco teórico).....	2
2. Planteamiento del Problema	3
3. Propuesta de Solución.....	4
4. Materiales y Métodos empleados.....	5
5. Desarrollo de la Solución.....	6
6. Resultados	8
7. Conclusiones.....	13
8. Referencias Bibliográficas	13

1. Antecedente (Marco teórico)

El modelo Cliente-Servidor es una arquitectura de red en la que múltiples clientes solicitan y reciben servicios de un servidor central. Este modelo es ampliamente utilizado en aplicaciones web, bases de datos y redes de comunicación. La comunicación entre cliente y servidor puede realizarse mediante diferentes protocolos, como HTTP, FTP o TCP/IP, que permiten el intercambio eficiente de datos.

El modelo cliente-servidor, también conocido como “principio cliente-servidor”, es un modelo de comunicación que permite la distribución de tareas dentro de una red de ordenadores.

Un servidor es un hardware que proporciona los recursos necesarios para otros ordenadores o programas, pero un servidor también puede ser un programa informático que se comunica con los clientes. Un servidor acepta las peticiones del cliente, las procesa y proporciona la respuesta solicitada. También existen diferentes tipos de clientes. Un ordenador o un programa informático se comunica con el servidor, envía solicitudes y recibe respuestas del servidor. En cuanto al modelo cliente-servidor, representa la interacción entre el servidor y el 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) 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.

Un socket es un punto final de comunicación entre dos dispositivos en una red. En Java, la biblioteca java.net proporciona clases como Socket y ServerSocket para manejar la conexión entre procesos de red. La comunicación mediante sockets se basa en el modelo de capas OSI, en particular en la capa de transporte, que usa TCP o UDP para garantizar la entrega de los mensajes.

- **Socket:** Representa el extremo del cliente en una conexión TCP.
- **ServerSocket:** Escucha las solicitudes de conexión entrantes y crea un socket para cada cliente.
- **InputStream y OutputStream:** Se usan para la comunicación de datos entre cliente y servidor.

2. Planteamiento del Problema

El objetivo de esta práctica es recordar el funcionamiento del modelo Cliente-Servidor y comprender cómo la comunicación entre procesos puede gestionarse de manera eficiente a través de sockets en Java. Se busca implementar una aplicación donde un servidor administre la simulación de una carrera de autos y un cliente reciba en tiempo real las actualizaciones del estado de la carrera.

El problema se plantea en la necesidad de establecer una comunicación confiable entre cliente y servidor, donde el servidor debe manejar múltiples hilos que representan autos en una carrera y enviar mensajes de estado al cliente sin retrasos o pérdidas de datos. La ejecución debe permitir que el cliente reciba los mensajes de manera ordenada y en tiempo real.

Entre los desafíos principales está la gestión eficiente de la comunicación mediante sockets, asegurando que los mensajes sean transmitidos correctamente sin

bloqueos ni sobrecarga del sistema. También es importante garantizar que la estructura Cliente-Servidor refleje el comportamiento esperado de una aplicación en red, con conexiones estables y cierre de sesión adecuado.

3. Propuesta de Solución

Para abordar este problema, se propone el diseño una aplicación en Java que consta de dos componentes principales:

- **Servidor:**
 - Se crea un `ServerSocket` que escucha conexiones entrantes en el puerto 5000.
 - Una vez que un cliente se conecta, el servidor establece un canal de comunicación.
 - Se crean y ejecutan tres hilos representando autos en la carrera, cada uno con tiempos de pausa aleatorios para simular condiciones de carrera realistas.
 - Cada auto envía actualizaciones al cliente sobre su progreso en la carrera.
 - Para evitar inconsistencias en la comunicación, se sincroniza el envío de datos asegurando que los mensajes se transmitan de forma ordenada y clara.
 - Se emplea `join()` para garantizar que todos los autos finalicen antes de cerrar la conexión con el cliente.
 - Una vez que la carrera finaliza, el servidor envía un mensaje de cierre y se desconecta.
- **Cliente:**
 - Se conecta al servidor usando un `Socket` en el puerto 5000.
 - Una vez conectado, recibe información en tiempo real sobre el avance de cada auto.
 - Se muestra en consola la información recibida con mensajes estructurados para facilitar la comprensión del progreso de la carrera.

- Maneja posibles excepciones para evitar fallos en caso de desconexión inesperada del servidor.
- Se asegura una correcta finalización de la conexión cuando la carrera concluye.

Esta solución garantiza una simulación fluida de la carrera de autos, implementando la arquitectura Cliente-Servidor y la programación concurrente de manera eficiente.

4. Materiales y Métodos empleados

- Hardware: Laptop ASUS y Teléfono iPhone 16.
- Software: Google Chrome, Word y Acrobat Reader.
- Lenguaje de programación: Java
- Entorno de desarrollo: IDE compatible con Java, como Visual Studio Code con las debidas extensiones de depuración y ejecución, además de la terminal con javac y java.
- Bibliotecas utilizadas: java.net.* para sockets, java.io.* para entrada y salida de datos.
- Métodos empleados:
 - El servidor inicia un ServerSocket en el puerto 5000.
 - El servidor espera la conexión de un cliente.
 - Se crean y ejecutan tres instancias de la clase Auto.
 - Cada Auto avanza aleatoriamente y se envía actualizaciones al cliente.
 - Se usa join() para esperar la finalización de todos los autos.
 - El cliente se conecta al servidor utilizando un Socket.
 - El cliente recibe mensajes desde un BufferedReader.
 - El cliente muestra el estado de la carrera en la consola.
 - Se maneja la desconexión limpia cuando la carrera finaliza.
- Estrategia de programación: Programación con sockets, implementación del modelo cliente/servidor y uso de aleatoriedad para la simulación del tiempo de avance de cada auto.

5. Desarrollo de la Solución

La solución se desarrolla a partir de la implementación de sockets Cliente-Servidor en Java. El servidor utiliza `ServerSocket` para escuchar conexiones y aceptar clientes en el puerto 5000. Una vez conectado, el servidor inicia varios hilos, representando autos en la carrera, que envían actualizaciones de estado al cliente.

Cada auto ejecuta su proceso de manera independiente en un hilo, enviando mensajes de progreso en intervalos de tiempo aleatorios para simular una carrera realista. El cliente, al recibir estos mensajes a través de su `Socket`, los muestra en consola para reflejar el avance de los autos.

El servidor sincroniza la finalización de los hilos usando `join()`, asegurando que todos los autos terminen antes de cerrar la conexión. Finalmente, el servidor envía un mensaje de finalización y se desconecta de manera segura, garantizando una correcta comunicación Cliente-Servidor en Java.

A continuación se puede ver la solución propuesta en código documentado en la figura 1,2 y 3:

```
D:\> Escuela8voSem > Distribuidos > Practica2 > ClienteCarrera.java > ...
1  //Materia: Sistemas Distribuidos
2  //Autor: Diego Valdes Castillo
3  //Fecha de creacion: 27/02/2025
4  //Version: 1.0
5  //Practica 2 - Sockets C/S
6  //Codigo del Cliente para la carrera
7
8  // Cliente que recibe las actualizaciones de la carrera y se desconecta correctamente
9  import java.io.*;
10 import java.net.*;
11
12 public class ClienteCarrera {
13     public static void main(String[] args) {
14         try (Socket socket = new Socket("localhost", 5000);
15             BufferedReader entrada = new BufferedReader(new InputStreamReader(socket.getInputStream()))) {
16
17             String mensaje;
18             while ((mensaje = entrada.readLine()) != null) {
19                 System.out.println("[SERVIDOR]: " + mensaje);
20             }
21         } catch (IOException e) {
22             System.out.println("No se pudo conectar con el servidor o la conexión fue cerrada.");
23         } finally {
24             System.out.println("cliente desconectado correctamente.");
25         }
26     }
27 }
28
```

Figura 1. Código implementado en Java para el Cliente.

```

D: > Escuela8voSem > Distribuidos > Practica2 > J ServidorCarrera.java > ...
1  //Materia: Sistemas Distribuidos
2  //Autor: Diego Valdes Castillo
3  //Fecha de creacion: 27/02/2025
4  //Practica 2 - Sockets C/S //Version: 1.0
5  //Codigo del Servidor para la carrera
6  // Servidor que gestiona la carrera con un solo cliente
7  import java.io.*;
8  import java.net.*;
9
10 class Auto extends Thread {
11     private String nombre;
12     private PrintWriter salida;
13
14     public Auto(String nombre, PrintWriter salida) {
15         this.nombre = nombre;
16         this.salida = salida;
17     }
18
19     @Override
20     public void run() {
21         for (int i = 1; i <= 10; i++) {
22             String mensaje = nombre + " ha avanzado " + (i * 10) + " metros";
23             System.out.println(mensaje);
24             salida.println(mensaje);
25             salida.flush(); // Asegurar que el mensaje se envíe inmediatamente
26             try {
27                 Thread.sleep((int) (Math.random() * 1000));
28             } catch (InterruptedException e) {
29                 salida.println(nombre + " se ha detenido inesperadamente.");
30                 salida.flush();
31             }
32         }
33         salida.println(nombre + " ha terminado la carrera!");
34         salida.flush();
35     }

```

Figura 2. Código implementado en Java para el Servidor - 1.

```

D: > Escuela8voSem > Distribuidos > Practica2 > J ServidorCarrera.java > ...
36     }
37
38     public class ServidorCarrera {
39         public static void main(String[] args) {
40             try (ServerSocket servidor = new ServerSocket(5000)) {
41                 System.out.println("Servidor esperando conexión de un cliente...");
42
43                 try (Socket socket = servidor.accept();
44                     PrintWriter salida = new PrintWriter(socket.getOutputStream(), true)) {
45
46                     System.out.println("Cliente conectado!");
47
48                     Auto auto1 = new Auto("Auto Rojo", salida);
49                     Auto auto2 = new Auto("Auto Azul", salida);
50                     Auto auto3 = new Auto("Auto Verde", salida);
51
52                     auto1.start();
53                     auto2.start();
54                     auto3.start();
55
56                     // Esperar a que los autos terminen la carrera antes de cerrar la conexión
57                     auto1.join();
58                     auto2.join();
59                     auto3.join();
60
61                     salida.println("Carrera finalizada, cerrando conexión...");
62                     salida.flush();
63                     System.out.println("Carrera terminada. Cliente desconectado.");
64                 }
65             } catch (IOException | InterruptedException e) {
66                 System.out.println("Error en el servidor: " + e.getMessage());
67             }
68         }
69     }
70

```

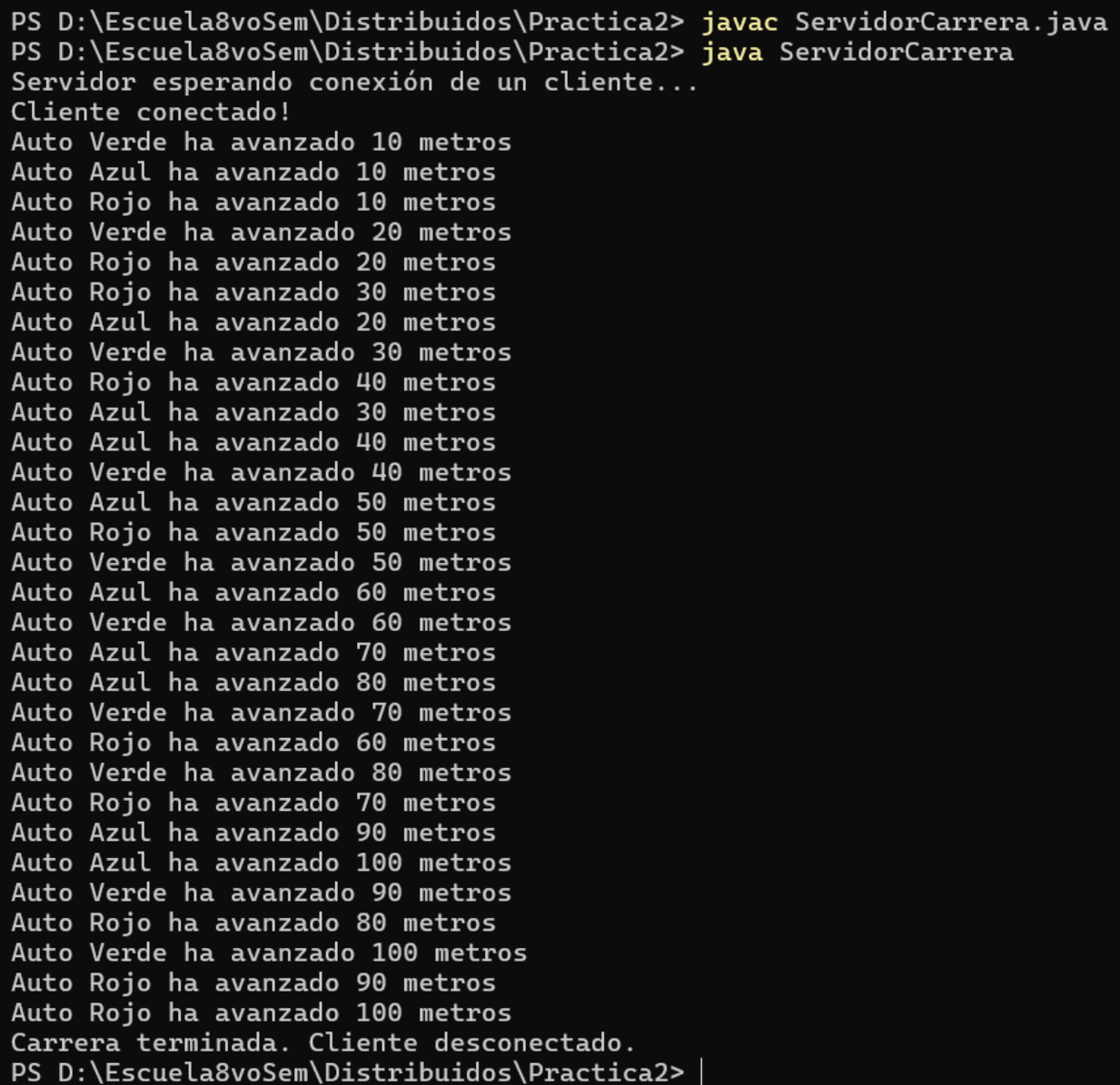
Figura 3. Código implementado en Java para el Servidor - 2.

6. Resultados

Los resultados obtenidos muestran que la comunicación entre el servidor y el cliente se desarrolla de manera eficiente, reflejando el avance de cada auto en tiempo real. Cada mensaje enviado desde el servidor al cliente es recibido y mostrado sin pérdida de información ni bloqueos, lo que demuestra la correcta gestión de los sockets.

Al ejecutar el programa varias veces, se observa que el orden de llegada de los autos varía debido a los tiempos de espera aleatorios, lo que simula un escenario dinámico y realista. Además, se verifica que el servidor cierra la conexión correctamente una vez que todos los autos han finalizado la carrera, asegurando una finalización limpia del proceso de comunicación.

A continuación se puede ver en la figura 4 y 5 una prueba de ejecución de los códigos desde la terminal:



```
PS D:\Escuela8voSem\Distribuidos\Practica2> javac ServidorCarrera.java
PS D:\Escuela8voSem\Distribuidos\Practica2> java ServidorCarrera
Servidor esperando conexión de un cliente...
Cliente conectado!
Auto Verde ha avanzado 10 metros
Auto Azul ha avanzado 10 metros
Auto Rojo ha avanzado 10 metros
Auto Verde ha avanzado 20 metros
Auto Rojo ha avanzado 20 metros
Auto Rojo ha avanzado 30 metros
Auto Azul ha avanzado 20 metros
Auto Verde ha avanzado 30 metros
Auto Rojo ha avanzado 40 metros
Auto Azul ha avanzado 30 metros
Auto Azul ha avanzado 40 metros
Auto Verde ha avanzado 40 metros
Auto Azul ha avanzado 50 metros
Auto Rojo ha avanzado 50 metros
Auto Verde ha avanzado 50 metros
Auto Azul ha avanzado 60 metros
Auto Verde ha avanzado 60 metros
Auto Azul ha avanzado 70 metros
Auto Azul ha avanzado 80 metros
Auto Verde ha avanzado 70 metros
Auto Rojo ha avanzado 60 metros
Auto Verde ha avanzado 80 metros
Auto Rojo ha avanzado 70 metros
Auto Azul ha avanzado 90 metros
Auto Azul ha avanzado 100 metros
Auto Verde ha avanzado 90 metros
Auto Rojo ha avanzado 80 metros
Auto Verde ha avanzado 100 metros
Auto Rojo ha avanzado 90 metros
Auto Rojo ha avanzado 100 metros
Carrera terminada. Cliente desconectado.
PS D:\Escuela8voSem\Distribuidos\Practica2> |
```

Figura 4. Prueba de Ejecución 1 - Servidor.

```

PS D:\Escuela8voSem\Distribuidos\Practica2> javac ClienteCarrera.java
PS D:\Escuela8voSem\Distribuidos\Practica2> java ClienteCarrera
[SERVIDOR]: Auto Verde ha avanzado 10 metros
[SERVIDOR]: Auto Rojo ha avanzado 10 metros
[SERVIDOR]: Auto Azul ha avanzado 10 metros
[SERVIDOR]: Auto Verde ha avanzado 20 metros
[SERVIDOR]: Auto Rojo ha avanzado 20 metros
[SERVIDOR]: Auto Rojo ha avanzado 30 metros
[SERVIDOR]: Auto Azul ha avanzado 20 metros
[SERVIDOR]: Auto Verde ha avanzado 30 metros
[SERVIDOR]: Auto Rojo ha avanzado 40 metros
[SERVIDOR]: Auto Azul ha avanzado 30 metros
[SERVIDOR]: Auto Azul ha avanzado 40 metros
[SERVIDOR]: Auto Verde ha avanzado 40 metros
[SERVIDOR]: Auto Azul ha avanzado 50 metros
[SERVIDOR]: Auto Rojo ha avanzado 50 metros
[SERVIDOR]: Auto Verde ha avanzado 50 metros
[SERVIDOR]: Auto Azul ha avanzado 60 metros
[SERVIDOR]: Auto Verde ha avanzado 60 metros
[SERVIDOR]: Auto Azul ha avanzado 70 metros
[SERVIDOR]: Auto Azul ha avanzado 80 metros
[SERVIDOR]: Auto Verde ha avanzado 70 metros
[SERVIDOR]: Auto Rojo ha avanzado 60 metros
[SERVIDOR]: Auto Verde ha avanzado 80 metros
[SERVIDOR]: Auto Rojo ha avanzado 70 metros
[SERVIDOR]: Auto Azul ha avanzado 90 metros
[SERVIDOR]: Auto Azul ha avanzado 100 metros
[SERVIDOR]: Auto Verde ha avanzado 90 metros
[SERVIDOR]: Auto Azul ha terminado la carrera!
[SERVIDOR]: Auto Rojo ha avanzado 80 metros
[SERVIDOR]: Auto Verde ha avanzado 100 metros
[SERVIDOR]: Auto Rojo ha avanzado 90 metros
[SERVIDOR]: Auto Verde ha terminado la carrera!
[SERVIDOR]: Auto Rojo ha avanzado 100 metros
[SERVIDOR]: Auto Rojo ha terminado la carrera!
[SERVIDOR]: Carrera finalizada, cerrando conexión...
Cliente desconectado correctamente.
PS D:\Escuela8voSem\Distribuidos\Practica2>

```

Figura 5. Prueba de Ejecución 1 - Cliente.

En la figura 4 y 5 podemos observar que los códigos se ejecutaron desde la terminal, para esto primero se compilan los códigos con el comando "javac Nombre Cliente.java" y "javac Nombre Servidor.java" después se ejecuta primero el servidor con el comando "java Nombre Servidor" y luego el cliente con el

comando "java Nombre Cliente" , además en esta prueba vemos que la conexión entre cliente/servidor funciona correctamente y en este caso gana el auto azul.

A continuación se puede ver en la figura 6 y 7 una segunda prueba de ejecución del código desde la terminal, para observar que la carrera si es aleatoria, ya que ahora gana el auto rojo y además el modelo cliente/servidor funciona correctamente con la implementación de sockets:

```
PS D:\Escuela8voSem\Distribuidos\Practica2> java ServidorCarrera
Servidor esperando conexión de un cliente...
Cliente conectado!
Auto Rojo ha avanzado 10 metros
Auto Azul ha avanzado 10 metros
Auto Verde ha avanzado 10 metros
Auto Azul ha avanzado 20 metros
Auto Verde ha avanzado 20 metros
Auto Rojo ha avanzado 20 metros
Auto Verde ha avanzado 30 metros
Auto Azul ha avanzado 30 metros
Auto Rojo ha avanzado 30 metros
Auto Azul ha avanzado 40 metros
Auto Rojo ha avanzado 40 metros
Auto Verde ha avanzado 40 metros
Auto Rojo ha avanzado 50 metros
Auto Rojo ha avanzado 60 metros
Auto Azul ha avanzado 50 metros
Auto Rojo ha avanzado 70 metros
Auto Verde ha avanzado 50 metros
Auto Verde ha avanzado 60 metros
Auto Azul ha avanzado 60 metros
Auto Azul ha avanzado 70 metros
Auto Azul ha avanzado 80 metros
Auto Rojo ha avanzado 80 metros
Auto Verde ha avanzado 70 metros
Auto Azul ha avanzado 90 metros
Auto Rojo ha avanzado 90 metros
Auto Rojo ha avanzado 100 metros
Auto Azul ha avanzado 100 metros
Auto Verde ha avanzado 80 metros
Auto Verde ha avanzado 90 metros
Auto Verde ha avanzado 100 metros
Carrera terminada. Cliente desconectado.
PS D:\Escuela8voSem\Distribuidos\Practica2> |
```

Figura 6. Prueba de Ejecución 2 - Servidor.

```
PS D:\Escuela8voSem\Distribuidos\Practica2> java ClienteCarrera
[SERVIDOR]: Auto Rojo ha avanzado 10 metros
[SERVIDOR]: Auto Verde ha avanzado 10 metros
[SERVIDOR]: Auto Azul ha avanzado 10 metros
[SERVIDOR]: Auto Azul ha avanzado 20 metros
[SERVIDOR]: Auto Verde ha avanzado 20 metros
[SERVIDOR]: Auto Rojo ha avanzado 20 metros
[SERVIDOR]: Auto Verde ha avanzado 30 metros
[SERVIDOR]: Auto Azul ha avanzado 30 metros
[SERVIDOR]: Auto Rojo ha avanzado 30 metros
[SERVIDOR]: Auto Azul ha avanzado 40 metros
[SERVIDOR]: Auto Rojo ha avanzado 40 metros
[SERVIDOR]: Auto Verde ha avanzado 40 metros
[SERVIDOR]: Auto Rojo ha avanzado 50 metros
[SERVIDOR]: Auto Rojo ha avanzado 60 metros
[SERVIDOR]: Auto Azul ha avanzado 50 metros
[SERVIDOR]: Auto Rojo ha avanzado 70 metros
[SERVIDOR]: Auto Verde ha avanzado 50 metros
[SERVIDOR]: Auto Verde ha avanzado 60 metros
[SERVIDOR]: Auto Azul ha avanzado 60 metros
[SERVIDOR]: Auto Azul ha avanzado 70 metros
[SERVIDOR]: Auto Azul ha avanzado 80 metros
[SERVIDOR]: Auto Rojo ha avanzado 80 metros
[SERVIDOR]: Auto Verde ha avanzado 70 metros
[SERVIDOR]: Auto Azul ha avanzado 90 metros
[SERVIDOR]: Auto Rojo ha avanzado 90 metros
[SERVIDOR]: Auto Rojo ha avanzado 100 metros
[SERVIDOR]: Auto Azul ha avanzado 100 metros
[SERVIDOR]: Auto Rojo ha terminado la carrera!
[SERVIDOR]: Auto Azul ha terminado la carrera!
[SERVIDOR]: Auto Verde ha avanzado 80 metros
[SERVIDOR]: Auto Verde ha avanzado 90 metros
[SERVIDOR]: Auto Verde ha avanzado 100 metros
[SERVIDOR]: Auto Verde ha terminado la carrera!
[SERVIDOR]: Carrera finalizada, cerrando conexión...
Cliente desconectado correctamente.
PS D:\Escuela8voSem\Distribuidos\Practica2>
```

Figura 7. Prueba de Ejecución 2 - Cliente.

7. Conclusiones

El uso de sockets en Java permite establecer una comunicación efectiva entre cliente y servidor, facilitando el intercambio de información en tiempo real. En esta práctica, se ha demostrado cómo los sockets pueden emplearse para enviar y recibir datos de manera eficiente, asegurando una simulación fluida de la carrera de autos.

La práctica muestra que la estructura Cliente-Servidor permite que el cliente reciba actualizaciones inmediatas sobre el progreso de los autos en la carrera, sin bloqueos ni pérdida de datos. Además, la implementación concurrente en el servidor garantiza que cada auto pueda avanzar de forma independiente, reflejando un entorno de simulación realista.

El aprendizaje clave de esta práctica es la importancia de los sockets en la comunicación entre procesos y su relevancia en el desarrollo de sistemas distribuidos.

8. Referencias Bibliográficas

- Programarya. (s.f.). Sockets en Java. Programarya. Recuperado el 27 de febrero de 2025, de <https://www.programarya.com/Cursos-Avanzados/Java/Sockets>
- IONOS. (s.f.). Modelo cliente-servidor: definición y características. IONOS Digital Guide. Recuperado el 27 de febrero de 2025, de <https://www.ionos.mx/digitalguide/servidores/know-how/modelo-cliente-servidor/>
- Oracle. (s.f.). Class Socket (Java SE 8 & JDK 8). Oracle. Recuperado el 27 de febrero de 2025, de <https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html>