

Instituto Politécnico Nacional

Escuela Superior de Cómputo

Materia: Sistemas Distribuidos

Profesor: Carreto Arellano Chadwick

Grupo: 7CM1

Práctica 3. Servidor – Múltiple Cliente

Valdés Castillo Diego - 2021630756

Fecha de Entrega: 10/03/2025

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	10
7. Conclusiones.....	14
8. Referencias Bibliográficas	14

1. Antecedente (Marco teórico)

Un servidor de múltiples clientes es un tipo de arquitectura de software para redes de computadoras donde los clientes, que pueden ser estaciones de trabajo básicas o computadoras personales completamente funcionales, solicitan información de una computadora servidor.

El modelo Cliente-Servidor es una arquitectura de red en la que múltiples clientes pueden solicitar y recibir 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 TCP/IP, que permite el intercambio eficiente de datos y garantiza la entrega de los mensajes.

Para implementar un servidor que sea capaz de atender múltiples clientes de manera concurrente. Se utilizan sockets, que son mecanismos que permiten establecer un enlace entre procesos que se ejecutan de manera independiente dentro de una red. En particular, en el lenguaje de programación Java, la biblioteca `java.net` proporciona las clases necesarias para la comunicación basada en sockets, como `Socket` y `ServerSocket`.

El servidor actúa como un punto central que espera conexiones entrantes de los clientes y establece un canal de comunicación con cada uno de ellos. Cada cliente, por su parte, se conectará al servidor, enviará solicitudes y recibirá respuestas a través de flujos de entrada y salida de datos. Para manejar múltiples clientes simultáneamente, el servidor puede implementar el uso de hilos (threads), lo que permite gestionar cada conexión de manera independiente y eficiente.

Los principales componentes que se utilizan en la implementación son:

- **Socket:** Representa el extremo del cliente en una conexión TCP.
- **ServerSocket:** Se encarga de escuchar solicitudes de conexión y generar un socket para cada cliente.
- **InputStream y OutputStream:** Permiten el intercambio de datos entre el cliente y el servidor.
- **Hilos (Threads):** Facilitan la atención simultánea de múltiples clientes sin afectar el rendimiento del servidor.

Esta arquitectura permite la implementación de aplicaciones en las que múltiples clientes pueden interactuar con un servidor centralizado, lo que es común en sistemas de mensajería, juegos en red, bases de datos distribuidas y otros entornos de comunicación en tiempo real.

2. Planteamiento del Problema

El objetivo de esta práctica es recordar el funcionamiento del modelo Cliente-Servidor y comprender cómo gestionar la comunicación entre múltiples clientes y un servidor a través de sockets en Java. Se busca implementar una aplicación en la que un servidor administre la simulación de una carrera de autos, mientras que múltiples clientes se conectan al servidor para recibir en tiempo real las actualizaciones del estado de la carrera.

El problema se basa en la necesidad de establecer una comunicación confiable entre un servidor y múltiples clientes. El servidor debe manejar múltiples hilos, cada

uno representando un auto en la carrera, y enviar actualizaciones de progreso a los clientes sin retrasos ni pérdidas de datos. Cada cliente debe recibir la información de manera ordenada y en tiempo real, asegurando que la experiencia sea fluida y precisa.

Uno de los principales desafíos es la gestión eficiente de la comunicación mediante sockets, asegurando que cada cliente reciba correctamente los mensajes sin bloqueos ni sobrecarga del sistema. Para esto, el servidor debe manejar múltiples conexiones simultáneamente utilizando hilos (threads), lo que permite que cada cliente tenga una experiencia independiente sin interferencias en la transmisión de datos.

Además, es fundamental garantizar que la estructura Cliente-Servidor refleje el comportamiento esperado de una aplicación en red, con conexiones estables, sincronización en la ejecución de los hilos y un cierre de sesión adecuado para evitar errores o bloqueos inesperados.

3. Propuesta de Solución

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

- **Servidor:**
 - Se crea un `ServerSocket` que escucha conexiones entrantes en el puerto 5000.
 - Se espera la conexión de múltiples clientes antes de iniciar la carrera. Cada cliente representa un auto en la competencia.
 - Una vez que todos los clientes están conectados, el servidor establece canales de comunicación con cada uno.
 - Se crean y ejecutan múltiples hilos representando los autos en la carrera, cada uno con tiempos de pausa aleatorios para simular condiciones de carrera realistas.

- Cada auto envía actualizaciones a su respectivo 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 enviar los resultados finales.
- Al concluir la carrera, el servidor envía a cada cliente su posición final y luego cierra la conexión.
- **Clientes:**
 - Se conectan al servidor utilizando un Socket en el puerto 5000.
 - Una vez conectados, reciben en tiempo real información sobre el avance de su respectivo auto en la carrera.
 - Se muestra en consola la información recibida con mensajes estructurados para facilitar la comprensión del progreso de la carrera.
 - Manejan posibles excepciones para evitar fallos en caso de desconexión inesperada del servidor.
 - Se aseguran de cerrar correctamente la conexión una vez que la carrera concluye.

Esta solución garantiza una simulación fluida de la carrera de autos, implementando la arquitectura Cliente-Servidor con múltiples clientes y utilizando programación concurrente para una experiencia en tiempo real 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 comandos 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 múltiples clientes, cada uno representando un auto en la carrera.
 - Una vez que todos los clientes están conectados, se crean y ejecutan múltiples instancias de la clase Auto.
 - Cada Auto avanza aleatoriamente y envía actualizaciones a su respectivo cliente.
 - Se usa join() para esperar la finalización de todos los autos antes de enviar los resultados finales.
 - Cada cliente se conecta al servidor utilizando un Socket.
 - Los clientes reciben mensajes desde un BufferedReader.
 - Cada cliente muestra en su consola el estado actualizado de la carrera en tiempo real.
 - Se maneja la desconexión limpia cuando la carrera finaliza, enviando a cada cliente su posición final antes de cerrar la conexión.
- Estrategia de programación: Uso de sockets para la comunicación entre el servidor y múltiples clientes, implementación del modelo Cliente-Servidor y empleo de aleatoriedad para simular el tiempo de avance de cada auto en la carrera.

5. Desarrollo de la Solución

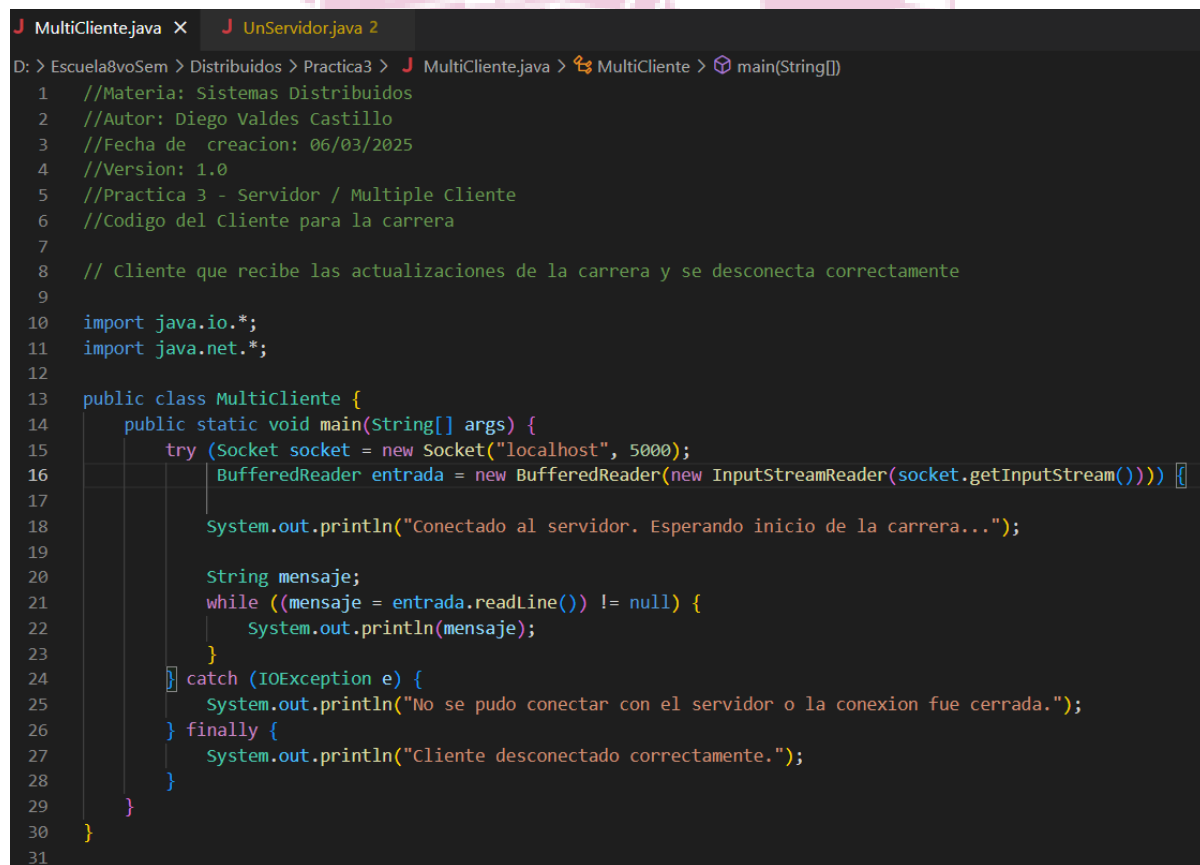
La solución se desarrolla a partir de la implementación de sockets Cliente-Servidor en Java. El servidor utiliza ServerSocket para escuchar conexiones y aceptar múltiples clientes en el puerto 5000. Una vez que todos los clientes están

conectados, el servidor inicia varios hilos, representando autos en la carrera, que envían actualizaciones de estado a cada cliente en tiempo real.

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. Los clientes, al recibir estos mensajes a través de sus respectivos Socket, los muestran en consola para reflejar el avance de su auto en la competencia.

El servidor sincroniza la finalización de los hilos usando `join()`, asegurando que todos los autos terminen antes de enviar los resultados finales a los clientes. Finalmente, el servidor envía un mensaje con la posición final de cada auto a su respectivo cliente y cierra las conexiones 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, 3, 4 y 5:



```
J MultiCliente.java X J UnServidor.java 2
D: > Escuela8voSem > Distribuidos > Practica3 > J MultiCliente.java > MultiCliente > main(String[])
1 //Materia: Sistemas Distribuidos
2 //Autor: Diego Valdes Castillo
3 //Fecha de creacion: 06/03/2025
4 //Version: 1.0
5 //Practica 3 - Servidor / Multiple Cliente
6 //Codigo del Cliente para la carrera
7
8 // Cliente que recibe las actualizaciones de la carrera y se desconecta correctamente
9
10 import java.io.*;
11 import java.net.*;
12
13 public class MultiCliente {
14     public static void main(String[] args) {
15         try (Socket socket = new Socket("localhost", 5000);
16             BufferedReader entrada = new BufferedReader(new InputStreamReader(socket.getInputStream()))) {
17
18             System.out.println("Conectado al servidor. Esperando inicio de la carrera...");
19
20             String mensaje;
21             while ((mensaje = entrada.readLine()) != null) {
22                 System.out.println(mensaje);
23             }
24         } catch (IOException e) {
25             System.out.println("No se pudo conectar con el servidor o la conexion fue cerrada.");
26         } finally {
27             System.out.println("Cliente desconectado correctamente.");
28         }
29     }
30 }
31
```

Figura 1. Código implementado en Java para el Cliente.

```

J UnServidor.java 2 X
D: > Escuela8voSem > Distribuidos > Practica3 > J UnServidor.java > Auto > run()
1 //Materia: Sistemas Distribuidos
2 //Autor: Diego Valdes Castillo
3 //Fecha de creacion: 06/03/2025
4 //Practica 3 - Servidor / Multiple Cliente
5 //Version: 1.0
6 //Codigo del Servidor para la carrera
7 // Servidor que gestiona la carrera con multiples clientes
8
9 import java.io.*;
10 import java.net.*;
11 import java.util.*;
12
13 class Auto extends Thread {
14     private String nombre;
15     private PrintWriter salida;
16     private int progreso = 0;
17     private static Map<String, Integer> posiciones = new LinkedHashMap<>();
18     private static int posicionActual = 1;
19
20     public Auto(String nombre, PrintWriter salida) {
21         this.nombre = nombre;
22         this.salida = salida;
23     }
24
25     @Override
26     public void run() {
27         Random random = new Random();
28         while (progreso < 100) {
29             progreso += random.nextInt(20) + 1;
30             if (progreso > 100) progreso = 100;
31             salida.println(nombre + " ha avanzado a " + progreso + " metros");
32             salida.flush();
33             System.out.println(nombre + " ha avanzado a " + progreso + " metros");
34             try {
35                 Thread.sleep(random.nextInt(1000));

```

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

```

36         } catch (InterruptedException e) {
37             salida.println(nombre + " se ha detenido inesperadamente.");
38             salida.flush();
39             return;
40         }
41     }
42     synchronized (posiciones) {
43         posiciones.put(nombre, posicionActual++);
44     }
45     salida.println(nombre + " ha terminado la carrera!");
46     salida.flush();
47     UnServidor.verificarCarreraFinalizada();
48 }
49
50 public static Map<String, Integer> getPosiciones() {
51     return posiciones;
52 }
53 }
54
55 public class UnServidor {
56     private static final int PUERTO = 5000;
57     private static List<Socket> clientes = new ArrayList<>();
58     private static List<Auto> autos = new ArrayList<>();
59     private static boolean carreraIniciada = false;
60
61     public static synchronized void verificarCarreraFinalizada() {
62         if (autos.stream().allMatch(auto -> Auto.getPosiciones().containsKey(auto.getName())) {
63             System.out.println("\n--- PODIO FINAL ---");
64             Auto.getPosiciones().forEach((auto, posicion) -> System.out.println(posicion + ". " + auto));
65
66             for (int i = 0; i < clientes.size(); i++) {
67                 try {
68                     PrintWriter salida = new PrintWriter(clientes.get(i).getOutputStream(), true);

```

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


```

69         salida.println("\n--- PODIO FINAL ---");
70         Auto.getPosiciones().forEach((auto, posicion) -> salida.println(posicion + ". " + auto));
71         salida.println("\nTu posición final: " + Auto.getPosiciones().get("Auto " + (i + 1)));
72         salida.flush();
73     } catch (IOException e) {
74         System.out.println("Error al enviar el podio a un cliente.");
75     }
76 }
77 }
78 }
79
80 public static void main(String[] args) {
81     try (ServerSocket servidor = new ServerSocket(PUERTO)) {
82         Scanner scanner = new Scanner(System.in);
83         System.out.print("Ingrese el número de autos en la carrera: ");
84         int numAutos = scanner.nextInt();
85
86         System.out.println("Esperando la conexión de " + numAutos + " clientes...");
87
88         for (int i = 1; i <= numAutos; i++) {
89             Socket socket = servidor.accept();
90             clientes.add(socket);
91             PrintWriter salida = new PrintWriter(socket.getOutputStream(), true);
92             Auto auto = new Auto("Auto " + i, salida);
93             autos.add(auto);
94             System.out.println("Cliente " + i + " conectado!");
95         }
96
97         System.out.println("Todos los clientes conectados. Iniciando carrera...");
98         carreraIniciada = true;
99
100         for (Auto auto : autos) {
101             auto.start();

```

Figura 4. Código implementado en Java para el Servidor - 3.

```

102     }
103
104     for (Auto auto : autos) {
105         auto.join();
106     }
107
108     System.out.println("\nCarrera finalizada. Podio de la carrera:");
109     Auto.getPosiciones().forEach((auto, posicion) -> System.out.println(posicion + ". " + auto));
110
111     System.out.println("Cerrando conexiones...");
112     for (Socket cliente : clientes) {
113         cliente.close();
114     }
115 } catch (IOException | InterruptedException e) {
116     System.out.println("Error en el servidor: " + e.getMessage());
117 }
118 }
119 }
120

```

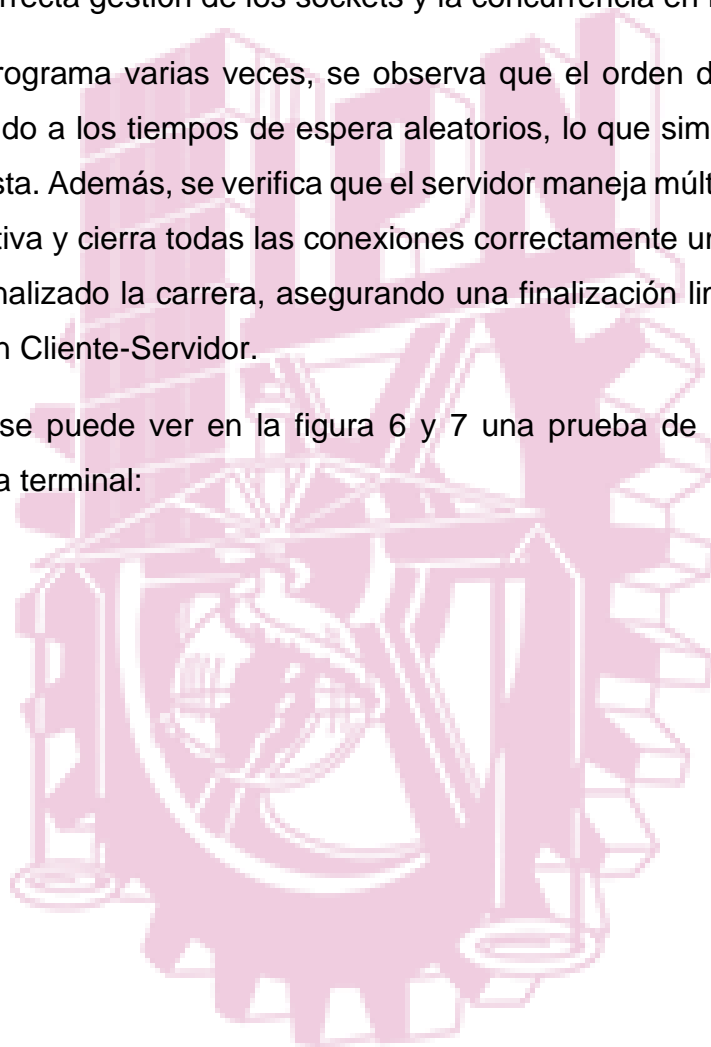
Figura 5. Código implementado en Java para el Servidor - 4.

6. Resultados

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

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 maneja múltiples conexiones de manera efectiva y cierra todas las conexiones correctamente una vez que todos los autos han finalizado la carrera, asegurando una finalización limpia del proceso de comunicación Cliente-Servidor.

A continuación se puede ver en la figura 6 y 7 una prueba de ejecución de los códigos desde la terminal:



```
PS D:\Escuela8voSem\Distribuidos\Practica3> javac UnServidor.java
PS D:\Escuela8voSem\Distribuidos\Practica3> javac MultiCliente.java
PS D:\Escuela8voSem\Distribuidos\Practica3> java UnServidor
Ingrese el número de autos en la carrera: 2
Esperando la conexión de 2 clientes...
Cliente 1 conectado!
Cliente 2 conectado!
Todos los clientes conectados. Iniciando carrera...
Auto 1 ha avanzado a 7 metros
Auto 2 ha avanzado a 4 metros
Auto 2 ha avanzado a 18 metros
Auto 1 ha avanzado a 17 metros
Auto 2 ha avanzado a 33 metros
Auto 2 ha avanzado a 35 metros
Auto 1 ha avanzado a 26 metros
Auto 1 ha avanzado a 33 metros
Auto 1 ha avanzado a 46 metros
Auto 2 ha avanzado a 40 metros
Auto 1 ha avanzado a 56 metros
Auto 1 ha avanzado a 67 metros
Auto 2 ha avanzado a 41 metros
Auto 2 ha avanzado a 44 metros
Auto 1 ha avanzado a 69 metros
Auto 2 ha avanzado a 46 metros
Auto 2 ha avanzado a 53 metros
Auto 1 ha avanzado a 77 metros
Auto 2 ha avanzado a 55 metros
Auto 2 ha avanzado a 61 metros
Auto 1 ha avanzado a 96 metros
Auto 2 ha avanzado a 70 metros
Auto 1 ha avanzado a 100 metros
Auto 2 ha avanzado a 79 metros
Auto 2 ha avanzado a 92 metros
Auto 2 ha avanzado a 100 metros

Carrera finalizada. Podio de la carrera:
1. Auto 1
2. Auto 2
Cerrando conexiones...
PS D:\Escuela8voSem\Distribuidos\Practica3>
```

Figura 6. Prueba de Ejecución 1 – Un Servidor.

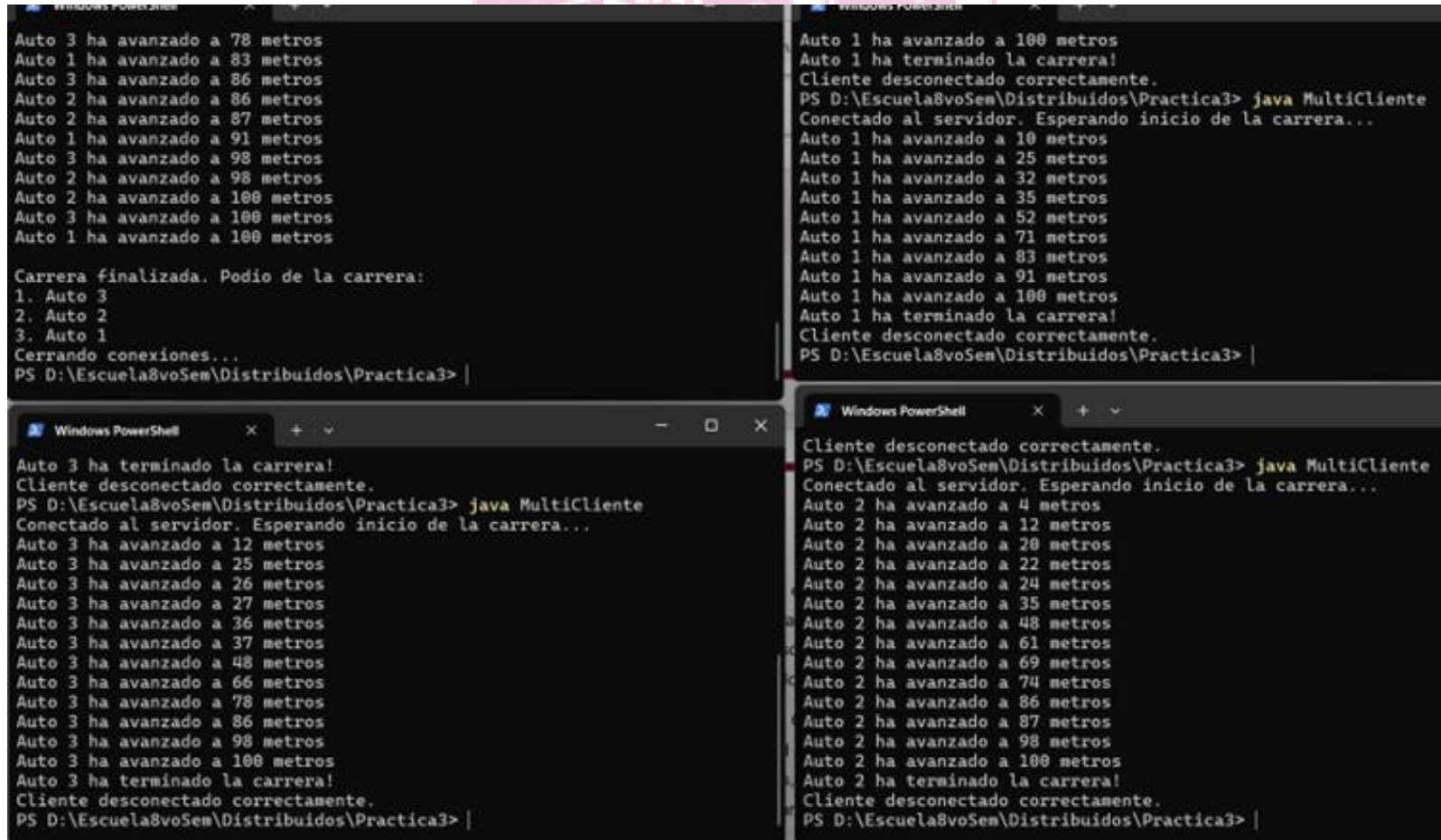
```
PS D:\Escuela8voSem\Distribuidos\Practica3> java MultiCliente
Conectado al servidor. Esperando inicio de la carrera...
Auto 1 ha avanzado a 7 metros
Auto 1 ha avanzado a 17 metros
Auto 1 ha avanzado a 26 metros
Auto 1 ha avanzado a 33 metros
Auto 1 ha avanzado a 46 metros
Auto 1 ha avanzado a 56 metros
Auto 1 ha avanzado a 67 metros
Auto 1 ha avanzado a 69 metros
Auto 1 ha avanzado a 77 metros
Auto 1 ha avanzado a 96 metros
Auto 1 ha avanzado a 100 metros
Auto 1 ha terminado la carrera!
Cliente desconectado correctamente.
PS D:\Escuela8voSem\Distribuidos\Practica3> |

Windows PowerShell
PS D:\Escuela8voSem\Distribuidos\Practica3> java MultiCliente
Conectado al servidor. Esperando inicio de la carrera...
Auto 2 ha avanzado a 4 metros
Auto 2 ha avanzado a 18 metros
Auto 2 ha avanzado a 33 metros
Auto 2 ha avanzado a 35 metros
Auto 2 ha avanzado a 40 metros
Auto 2 ha avanzado a 41 metros
Auto 2 ha avanzado a 44 metros
Auto 2 ha avanzado a 46 metros
Auto 2 ha avanzado a 53 metros
Auto 2 ha avanzado a 55 metros
Auto 2 ha avanzado a 61 metros
Auto 2 ha avanzado a 70 metros
Auto 2 ha avanzado a 79 metros
Auto 2 ha avanzado a 92 metros
Auto 2 ha avanzado a 100 metros
Auto 2 ha terminado la carrera!
Cliente desconectado correctamente.
PS D:\Escuela8voSem\Distribuidos\Practica3>
```

Figura 7. Prueba de Ejecución 1 – Dos Clientes.

En la figura 6 y 7 podemos observar que los códigos se ejecutaron desde la terminal, para esto primero se compilan los códigos con el comando *“javac MultiCliente.java”* y *“javac UnServidor.java”* después se ejecuta primero el servidor con el comando *“java UnServidor”* y luego el cliente con el comando *“java MultiCliente”* , además en esta prueba vemos que la carrera es de 2 autos, es decir, de 2 clientes y en la cual gana el auto 1.

A continuación se puede ver en la figura 8 una segunda prueba de ejecución del código desde la terminal, para observar que la carrera si es aleatoria, ya que ahora gana el auto 3 y además ahora son 3 autos los que compiten, es decir, se conectaron 3 clientes:



```
Auto 3 ha avanzado a 78 metros
Auto 1 ha avanzado a 83 metros
Auto 3 ha avanzado a 86 metros
Auto 2 ha avanzado a 86 metros
Auto 2 ha avanzado a 87 metros
Auto 1 ha avanzado a 91 metros
Auto 3 ha avanzado a 98 metros
Auto 2 ha avanzado a 98 metros
Auto 2 ha avanzado a 100 metros
Auto 3 ha avanzado a 100 metros
Auto 1 ha avanzado a 100 metros

Carrera finalizada. Podio de la carrera:
1. Auto 3
2. Auto 2
3. Auto 1
Cerrando conexiones...
PS D:\Escuela8voSem\Distribuidos\Practica3> |

Auto 1 ha avanzado a 100 metros
Auto 1 ha terminado la carrera!
Cliente desconectado correctamente.
PS D:\Escuela8voSem\Distribuidos\Practica3> java MultiCliente
Conectado al servidor. Esperando inicio de la carrera...
Auto 1 ha avanzado a 10 metros
Auto 1 ha avanzado a 25 metros
Auto 1 ha avanzado a 32 metros
Auto 1 ha avanzado a 35 metros
Auto 1 ha avanzado a 52 metros
Auto 1 ha avanzado a 71 metros
Auto 1 ha avanzado a 83 metros
Auto 1 ha avanzado a 91 metros
Auto 1 ha avanzado a 100 metros
Auto 1 ha terminado la carrera!
Cliente desconectado correctamente.
PS D:\Escuela8voSem\Distribuidos\Practica3> |

Auto 3 ha terminado la carrera!
Cliente desconectado correctamente.
PS D:\Escuela8voSem\Distribuidos\Practica3> java MultiCliente
Conectado al servidor. Esperando inicio de la carrera...
Auto 3 ha avanzado a 12 metros
Auto 3 ha avanzado a 25 metros
Auto 3 ha avanzado a 26 metros
Auto 3 ha avanzado a 27 metros
Auto 3 ha avanzado a 36 metros
Auto 3 ha avanzado a 37 metros
Auto 3 ha avanzado a 48 metros
Auto 3 ha avanzado a 66 metros
Auto 3 ha avanzado a 78 metros
Auto 3 ha avanzado a 86 metros
Auto 3 ha avanzado a 98 metros
Auto 3 ha avanzado a 100 metros
Auto 3 ha terminado la carrera!
Cliente desconectado correctamente.
PS D:\Escuela8voSem\Distribuidos\Practica3> |

Cliente desconectado correctamente.
PS D:\Escuela8voSem\Distribuidos\Practica3> java MultiCliente
Conectado al servidor. Esperando inicio de la carrera...
Auto 2 ha avanzado a 4 metros
Auto 2 ha avanzado a 12 metros
Auto 2 ha avanzado a 20 metros
Auto 2 ha avanzado a 22 metros
Auto 2 ha avanzado a 24 metros
Auto 2 ha avanzado a 35 metros
Auto 2 ha avanzado a 48 metros
Auto 2 ha avanzado a 61 metros
Auto 2 ha avanzado a 69 metros
Auto 2 ha avanzado a 74 metros
Auto 2 ha avanzado a 86 metros
Auto 2 ha avanzado a 87 metros
Auto 2 ha avanzado a 98 metros
Auto 2 ha avanzado a 100 metros
Auto 2 ha terminado la carrera!
Cliente desconectado correctamente.
PS D:\Escuela8voSem\Distribuidos\Practica3> |
```

Figura 8. Prueba de Ejecución 2 – Servidor con 3 Clientes.

7. Conclusiones

El uso de sockets en Java permite establecer una comunicación efectiva entre un servidor y múltiples clientes, 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 con múltiples participantes conectados simultáneamente.

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

El aprendizaje clave de esta práctica es la importancia de los sockets en la comunicación entre procesos, su utilidad en la gestión de múltiples conexiones simultáneas y su relevancia en el desarrollo de sistemas distribuidos.

8. Referencias Bibliográficas

- Web Tutoriales. (2009, 18 de noviembre). Comunicación entre un servidor y múltiples clientes. Web Tutoriales. Recuperado de <https://www.webtutoriales.com/articulos/2009/11/18/comunicacion-entre-un-servidor-y-multiples-clientes/>
- Spiegato. (s.f.). *¿Cómo funciona un servidor de múltiples clientes?*. Spiegato. Recuperado de <https://spiegato.com/es/como-funciona-un-servidor-de-multiples-clientes>