

**Instituto Politécnico Nacional**

**Escuela Superior de Cómputo**

**Materia: Sistemas Distribuidos**

**Profesor: Carreto Arellano Chadwick**

**Grupo: 7CM1**

**Práctica 4 – Múltiple Servidor / Múltiple Cliente**

**Valdés Castillo Diego - 2021630756**

**Fecha de Entrega: 19/03/2025**

# Índice

1. Antecedente (Marco teórico).....	2
2. Planteamiento del Problema .....	4
3. Propuesta de Solución.....	5
4. Materiales y Métodos empleados.....	7
5. Desarrollo de la Solución.....	9
6. Resultados .....	13
7. Conclusiones.....	20
8. Referencias Bibliográficas .....	20

## 1. Antecedente (Marco teórico)

El modelo MultiServidor-MultiCliente es una arquitectura distribuida en la que múltiples clientes pueden conectarse a múltiples servidores, permitiendo distribución de procesos, distribución de tareas y procesamiento eficiente de datos. Esta arquitectura es utilizada en aplicaciones de alto rendimiento como sistemas de mensajería, juegos en línea, servicios en la nube y procesamiento distribuido de información.

A diferencia del modelo tradicional Cliente-Servidor, donde un único servidor atiende múltiples clientes, en esta arquitectura existe un distribuidor que asigna dinámicamente los clientes a diferentes servidores disponibles. Cada servidor se encarga de procesar las solicitudes de un conjunto de clientes de manera independiente, asegurando que el sistema pueda escalar sin afectar el rendimiento.

El distribuidor actúa como intermediario entre clientes y servidores. Recibe solicitudes de conexión de los clientes y los asigna a un servidor disponible utilizando un mecanismo de distribución equitativo. Además, centraliza la recepción de resultados.

Los Servidores de Carrera son instancias independientes que gestionan carreras de autos con múltiples clientes. Cada servidor maneja sus clientes de manera concurrente y envía los resultados al distribuidor.

Los Clientes se conectan al distribuidor y, posteriormente, a un servidor específico para recibir actualizaciones en tiempo real del estado de la carrera.

Para coordinar las acciones entre servidores y clientes, se requiere una estrategia de sincronización centralizada que evite inconsistencias en la comunicación y en el procesamiento de eventos. En este caso, el distribuidor actúa como punto central para recibir y consolidar los resultados de múltiples servidores.

Este enfoque permite:

- Distribución de procesos: Se distribuyen las conexiones de los clientes equitativamente entre los servidores.
- Coordinación de eventos: Asegura que los servidores inicien y finalicen las carreras de manera ordenada.
- Gestión eficiente de la comunicación: Permite la recopilación y presentación de resultados de cada servidor sin interferencias.

En sistemas distribuidos, el Reloj de Lamport es un mecanismo lógico utilizado para asignar marcas de tiempo a eventos y asegurar un orden consistente en la ejecución de procesos concurrentes. En esta implementación, cada vez que un auto avanza en la carrera, se incrementa un contador global de reloj, representando la secuencia de eventos.

Este mecanismo es clave para:

- Mantener coherencia en la ejecución de los autos, asegurando que los mensajes sean procesados en el orden correcto.
- Evitar condiciones de carrera y desincronización, garantizando que los resultados de cada cliente sean reflejados correctamente en los servidores.

- Facilitar la depuración y monitoreo, ya que cada actualización del estado de los autos incluye su respectiva marca de tiempo ( $T=$ ) generada por el reloj de Lamport.

La arquitectura MultiServidor-MultiCliente combinada con un mecanismo de sincronización centralizado y el uso del Reloj de Lamport permite la implementación eficiente de sistemas distribuidos. En este modelo, los clientes pueden conectarse a diferentes servidores de manera distribuida, los servidores pueden procesar eventos concurrentes sin interferencias y los resultados se consolidan de forma sincronizada, garantizando una ejecución fluida y coherente del sistema.

## **2. Planteamiento del Problema**

Esta práctica tiene como finalidad reforzar el conocimiento sobre el modelo MultiServidor-MultiCliente, explorando la gestión de comunicación entre múltiples clientes y servidores mediante sockets en Java. Para ello, se implementará un sistema en el que un distribuidor asigna dinámicamente a los clientes a diferentes servidores. Cada servidor, a su vez, administra una carrera de autos y transmite en tiempo real las actualizaciones de estado a los clientes conectados.

El principal reto radica en garantizar una comunicación eficiente y confiable entre las distintas partes del sistema. Cada servidor debe gestionar múltiples hilos para simular la carrera, enviando periódicamente el progreso de los autos a los clientes sin generar retrasos ni pérdida de datos. Al mismo tiempo, el distribuidor se encarga de distribuir las conexiones de manera equitativa y consolidar los resultados obtenidos.

Uno de los aspectos clave en esta implementación es la gestión óptima de la concurrencia y la comunicación, evitando bloqueos o sobrecarga en los servidores. Para lograrlo, el distribuidor asigna clientes a los servidores, mientras que cada servidor maneja múltiples conexiones simultáneamente mediante hilos (threads). Esto permite que cada cliente reciba la información de su auto sin interferencias y de manera ordenada.

Por último, es crucial que la arquitectura MultiServidor-MultiCliente funcione de manera estable, con procesos bien sincronizados, centralizados, con tiempos y mecanismos adecuados para el cierre de conexiones. También se debe asegurar una correcta recopilación y presentación de los resultados, garantizando que la comunicación entre clientes, servidores y el distribuidor sea fluida y libre de errores.

### 3. Propuesta de Solución

Para abordar este problema, se propone diseñar una aplicación en Java que implemente una arquitectura MultiServidor-MultiCliente, con tres componentes principales:

- **Distribuidor:**
  - Se implementa un distribuidor que actúa como intermediario entre los clientes y los servidores de carrera.
  - Se crea un ServerSocket en el puerto 6000 para recibir conexiones de los clientes.
  - Cuando un cliente se conecta, el distribuidor le asigna dinámicamente uno de los servidores disponibles, equilibrando la carga de trabajo.
  - Una vez asignado el servidor, el distribuidor envía al cliente la dirección y el puerto correspondientes para que establezca la conexión.
  - Además, el distribuidor centraliza la recolección de los resultados de las carreras y los muestra en la consola.
- **Servidores de Carrera:**
  - Se crean múltiples servidores de carrera, cada uno ejecutándose en un puerto diferente (por ejemplo, 5001, 5002, etc.).
  - Cada servidor espera la conexión de un grupo de clientes antes de iniciar la carrera.
  - Una vez que 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, con tiempos de pausa aleatorios para simular condiciones realistas.
- Cada auto envía actualizaciones a su respectivo cliente sobre su progreso en la carrera.
- Se utiliza `join()` para asegurar que todos los autos finalicen antes de enviar los resultados finales.
- Al concluir la carrera, el servidor envía al distribuidor un resumen del podio y cierra las conexiones.
- **Clientes:**
  - Los clientes primero se conectan al distribuidor, que les proporciona la dirección del servidor de carrera al que deben conectarse.
  - Una vez asignado un servidor, establecen una conexión con él utilizando un Socket.
  - Reciben en tiempo real información sobre el avance de su auto en la carrera.
  - Muestran en consola los mensajes recibidos para reflejar el progreso de la carrera.
  - Manejan posibles excepciones para evitar fallos en caso de desconexión inesperada.
  - Una vez finalizada la carrera, reciben la posición final y cierran la conexión correctamente.

Esta arquitectura mejora la eficiencia del sistema al permitir la ejecución de múltiples carreras simultáneamente, distribuyendo los procesos entre los servidores y asegurando que ningún servidor esté sobrecargado. Además, la comunicación entre los clientes y los servidores se mantiene ordenada y libre de interferencias, gracias al esquema centralizado y el reloj Lamport que se implementa, lo que garantiza una simulación fluida y en tiempo real de la carrera de autos.

## 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 la gestión de sockets.
  - java.io.\* para la entrada y salida de datos.
- Métodos empleados:
  - Distribuidor:
    - Inicia un ServerSocket en el puerto 6000 para gestionar las conexiones de los clientes.
    - Asigna dinámicamente un servidor disponible a cada cliente y le envía su dirección y puerto de conexión.
    - Recibe y consolida los resultados de las carreras desde los servidores de carrera.
  - Servidores de Carrera:
    - Cada servidor inicia un ServerSocket en un puerto específico (5001, 5002, etc.).
    - 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.

- Al concluir la carrera, el servidor envía los resultados al distribuidor, que los consolida y muestra en consola.
- Clientes:
  - Se conectan primero al distribuidor, quien les asigna un servidor de carrera.
  - Luego, establecen una conexión con el servidor utilizando un Socket.
  - Reciben en tiempo real información sobre el avance de su auto en la carrera a través de un `BufferedReader`.
  - Muestran en consola los mensajes recibidos con el estado actualizado de la carrera.
  - Manejan posibles excepciones para evitar fallos en caso de desconexión inesperada.
  - Una vez finalizada la carrera, reciben su posición final y cierran la conexión correctamente.
- Estrategia de programación: La implementación del sistema sigue un enfoque basado en el uso de sockets para establecer la comunicación entre el distribuidor, los servidores y los clientes, asegurando una correcta transmisión de datos en tiempo real.

Se adopta un modelo `MultiServidor-MultiCliente` que distribuye los procesos, asignando clientes a diferentes servidores de manera equitativa. Para simular el comportamiento realista de una carrera, se implementa la aleatoriedad en los tiempos de avance de los autos, garantizando una competencia dinámica y sin patrones predefinidos.

Para mantener la coherencia en la ejecución de los eventos y el orden en el envío de mensajes, se utiliza el reloj de Lamport, lo que permite evitar conflictos en la recepción de actualizaciones y asegurar que el estado de la carrera se refleje de manera precisa en cada cliente.



## 5. Desarrollo de la Solución

Para llevar a cabo la implementación de esta solución, se diseñó un sistema basado en sockets en Java con una arquitectura MultiServidor-MultiCliente. En este modelo, un distribuidor se encarga de gestionar las conexiones de los clientes y asignarlos a diferentes servidores de carrera, los cuales son responsables de simular la competencia y enviar actualizaciones en tiempo real.

El proceso comienza cuando un cliente se conecta al distribuidor, que escucha las solicitudes en el puerto 6000. Una vez establecida la conexión, el distribuidor le asigna dinámicamente un servidor de carrera disponible y le proporciona la información necesaria para conectarse a él. Cada servidor opera de manera independiente en un puerto específico y administra un grupo de clientes que representarán autos en la competencia.

Cuando todos los clientes se han conectado a su servidor correspondiente, se inicia la carrera. Cada auto avanza de forma independiente dentro de un hilo, generando pausas aleatorias para simular un desarrollo impredecible. Durante la ejecución, los servidores envían constantemente actualizaciones de estado a los clientes, permitiéndoles visualizar el progreso de la carrera en tiempo real.

Para garantizar un flujo de ejecución ordenado, se emplea sincronización mediante `join()`, lo que asegura que todos los autos finalicen antes de procesar los resultados. Al concluir la carrera, cada servidor envía el podio al distribuidor, que consolida la información y la muestra en consola. Finalmente, los clientes reciben su posición final antes de que la conexión se cierre correctamente, asegurando una comunicación eficiente y sin interrupciones.

A continuación se puede ver la solución propuesta en código documentado en la figura 1, 2, 3, 4, 5 y 6:

```

D: > Escuela8voSem > Distribuidos > Practica4 > J Cliente.java > ...
1 //Materia: Sistemas Distribuidos
2 //Autor: Diego Valdes Castillo
3 //Fecha de creacion: 12/03/2025
4 //Version: 1.0
5 //Practica 4 Multiple Servidor / Multiple Cliente
6 //Codigo del Cliente para la carrera
7
8 // Cliente.java - Cliente que se conecta al balanceador y luego al servidor de carrera
9 import java.io.*;
10 import java.net.*;
11
12 public class Cliente {
13     public static void main(String[] args) {
14         try (Socket balanceador = new Socket("localhost", 6000);
15             BufferedReader entrada = new BufferedReader(new InputStreamReader(balanceador.getInputStream()));
16             String direccionServidor = entrada.readLine();
17             String[] partes = direccionServidor.split(":");
18             String host = partes[0];
19             int puerto = Integer.parseInt(partes[1]);
20
21             try (Socket socket = new Socket(host, puerto);
22                 BufferedReader entradaServidor = new BufferedReader(new InputStreamReader(socket.getInputStream()));
23                 System.out.println("Conectado a " + direccionServidor);
24                 String mensaje;
25                 while ((mensaje = entradaServidor.readLine()) != null) {
26                     System.out.println(mensaje);
27                 }
28             } catch (IOException e) {
29                 System.out.println("Error en el cliente: " + e.getMessage());
30             }
31         }
32     }
33 }
34

```

**Figura 1. Código implementado en Java para el Cliente.**

```

D: > Escuela8voSem > Distribuidos > Practica4 > J Distribuidor.java > ...
1 //Materia: Sistemas Distribuidos
2 //Autor: Diego Valdes Castillo
3 //Fecha de creacion: 12/03/2025
4 //Version: 1.0
5 //Practica 4 Multiple Servidor / Multiple Cliente
6 //Codigo del Distribuidor para la carrera
7
8 // Distribuidor.java - Distribuye clientes a servidores y recibe resultados
9 import java.io.*;
10 import java.net.*;
11 import java.util.*;
12
13 public class Distribuidor {
14     private static final int PUERTO = 6000;
15     private static List<String> servidores = Arrays.asList(
16         "localhost:5001", "localhost:5002"
17     );
18     private static int indice = 0;
19     private static List<String> resultados = new ArrayList<>();
20
21     public static void main(String[] args) {
22         new Thread(Distribuidor::recibirResultados).start();
23
24         try (ServerSocket serverSocket = new ServerSocket(PUERTO)) {
25             System.out.println("Distribuidor activo en el puerto " + PUERTO);
26             while (true) {
27                 Socket cliente = serverSocket.accept();
28                 String servidorAsignado = servidores.get(indice);
29                 indice = (indice + 1) % servidores.size();
30
31                 PrintWriter salida = new PrintWriter(cliente.getOutputStream(), true);
32                 salida.println(servidorAsignado);
33                 cliente.close();
34             }
35         } catch (IOException e) {
36             System.out.println("Error en el distribuidor: " + e.getMessage());
37         }
38     }
39
40     private static void recibirResultados() {
41         // Implementación del método recibirResultados
42     }
43 }

```

**Figura 2. Código implementado en Java para el Distribuidor - 1.**

```

37     }
38 }
39
40 private static void recibirResultados() {
41     try (ServerSocket serverSocket = new ServerSocket(7000)) {
42         while (true) {
43             Socket servidor = serverSocket.accept();
44             BufferedReader entrada = new BufferedReader(new InputStreamReader(servidor.getInputStream()));
45             String resultado;
46             StringBuilder podio = new StringBuilder();
47             while ((resultado = entrada.readLine()) != null) {
48                 synchronized (resultados) {
49                     resultados.add(resultado);
50                     podio.append(resultado).append("\n");
51                 }
52             }
53             System.out.println("\nResumen de la carrera recibida:");
54             System.out.println(podio.toString());
55             servidor.close();
56         }
57     } catch (IOException e) {
58         System.out.println("Error al recibir resultados: " + e.getMessage());
59     }
60 }
61 }

```

**Figura 3. Código implementado en Java para el Distribuidor - 2.**

```

D: > Escuela8voSem > Distribuidos > Practica4 > J ServidorCarrera.java > ...
1 //Materia: Sistemas Distribuidos
2 //Autor: Diego Valdes Castillo
3 //Fecha de creacion: 12/03/2025
4 //Version: 1.0
5 //Practica 4 Multiple Servidor / Multiple Cliente
6 //Codigo de los Servidores para la carrera
7 // ServidorCarrera.java - Servidores de carrera que manejan clientes y envian resultados
8 import java.io.*;
9 import java.net.*;
10 import java.util.*;
11
12 class Auto extends Thread {
13     private String nombre;
14     private PrintWriter salida;
15     private int progreso = 0;
16     private static int relojLamport = 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     private synchronized void incrementarReloj() {
26         relojLamport++;
27     }
28
29     @Override
30     public void run() {
31         Random random = new Random();
32         while (progreso < 100) {
33             progreso += random.nextInt(20) + 1;
34             if (progreso > 100) progreso = 100;
35             incrementarReloj();
36             salida.println("[T=" + relojLamport + "] " + nombre + " ha avanzado a " + progreso + " metros");

```

**Figura 4. Código implementado en Java para el Servidor - 1.**

```

37         salida.flush();
38         try {
39             Thread.sleep(random.nextInt(1000));
40         } catch (InterruptedException e) {
41             return;
42         }
43     }
44     synchronized (posiciones) {
45         posiciones.put(nombre, posicionActual++);
46     }
47     salida.println("[T=" + relojLamport + "] " + nombre + " ha terminado la carrera!");
48     salida.flush();
49 }
50
51 public static Map<String, Integer> getPosiciones() {
52     return posiciones;
53 }
54 }
55
56 public class ServidorCarrera {
57     private static int clientesConectados = 0;
58     private static List<Auto> autos = new ArrayList<>();
59     private static List<Socket> clientes = new ArrayList<>();
60
61     public static void main(String[] args) {
62         int puerto = args.length > 0 ? Integer.parseInt(args[0]) : 5001;
63         try (ServerSocket servidor = new ServerSocket(puerto)) {
64             System.out.println("Servidor de carrera activo en puerto " + puerto);
65             while (clientesConectados < 3) {
66                 Socket cliente = servidor.accept();
67                 clientes.add(cliente);
68                 PrintWriter salida = new PrintWriter(cliente.getOutputStream(), true);
69                 Auto auto = new Auto("Auto " + (clientesConectados + 1), salida);

```

**Figura 5. Código implementado en Java para el Servidor - 2.**

```

70         autos.add(auto);
71         clientesConectados++;
72         System.out.println("Cliente conectado en puerto " + puerto);
73     }
74
75     System.out.println("Iniciando carrera en puerto " + puerto);
76     for (Auto auto : autos) {
77         auto.start();
78     }
79     for (Auto auto : autos) {
80         auto.join();
81     }
82
83     enviarPodio(puerto);
84     System.out.println("Carrera finalizada en puerto " + puerto);
85     for (Socket cliente : clientes) {
86         cliente.close();
87     }
88 } catch (IOException | InterruptedException e) {
89     System.out.println("Error en el servidor de carrera: " + e.getMessage());
90 }
91
92
93 private static void enviarPodio(int puerto) {
94     try (Socket socket = new Socket("localhost", 7000);
95         PrintWriter salida = new PrintWriter(socket.getOutputStream(), true)) {
96         salida.println("\n--- PODIO FINAL EN SERVIDOR " + puerto + " ---");
97         Auto.getPosiciones().forEach((auto, posicion) -> salida.println(posicion + ". " + auto));
98         salida.flush();
99     } catch (IOException e) {
100         System.out.println("Error al enviar resultados: " + e.getMessage());
101     }
102 }
103 }

```

**Figura 6. Código implementado en Java para el Servidor - 3.**

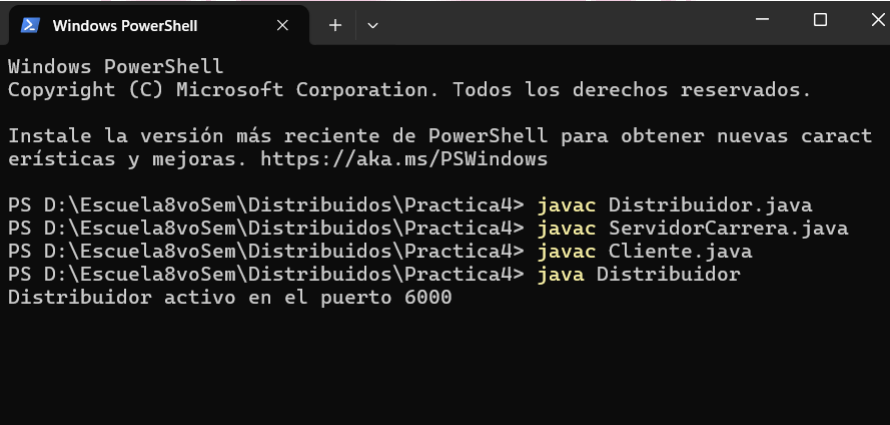
## 6. Resultados

Los resultados obtenidos evidencian que la comunicación entre los servidores y los múltiples clientes se lleva a cabo de manera eficiente, permitiendo que cada cliente reciba actualizaciones en tiempo real sobre el progreso de su auto en la carrera. Los mensajes enviados por los servidores son correctamente entregados y mostrados en los clientes sin pérdidas de información ni bloqueos, lo que confirma una gestión adecuada de los sockets y una ejecución concurrente estable.

Al ejecutar el programa en distintas ocasiones, se observa que el orden de llegada de los autos varía debido a los tiempos de espera aleatorios, logrando así una simulación dinámica y realista de la competencia. Además, se comprueba que los servidores administran las conexiones de manera eficiente, asegurando que todas las carreras finalicen correctamente y que las conexiones se cierren de forma ordenada al concluir la simulación, garantizando una comunicación estable y sin interrupciones dentro de la arquitectura MultiServidor-MultiCliente.

A continuación se puede ver en la figura 7, 8, 9 y 10 una prueba de ejecución de los códigos desde la terminal:

En la figura 7 podemos observar que los códigos se ejecutaron desde la terminal, para esto primero se compilan los códigos con el comando *“javac Distribuidor.java”*, *“javac ServidorCarrera.java”* y *“javac Cliente.java”* después se ejecuta primero el distribuidor con el comando *“java Distribuidor”* el cual se activa en el puerto 6000.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS D:\Escuela8voSem\Distribuidos\Practica4> javac Distribuidor.java
PS D:\Escuela8voSem\Distribuidos\Practica4> javac ServidorCarrera.java
PS D:\Escuela8voSem\Distribuidos\Practica4> javac Cliente.java
PS D:\Escuela8voSem\Distribuidos\Practica4> java Distribuidor
Distribuidor activo en el puerto 6000
```

**Figura 7. Prueba de Ejecución 1 – Distribuidor.**

Ahora en la figura 8 podemos ver que se ejecuta un servidor primero en el localhost 5001 y espera la conexión de 3 clientes para poder iniciar la carrera, en esta prueba gano el auto 1, en la posición 2 quedo el auto 3 y en último lugar llego el auto 2.

```

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas caracter
ísticas y mejoras. https://aka.ms/PSWindows

PS D:\Escuela8voSem\Distribuidos\Practica4> java ServidorCarrera 5001
Servidor de carrera activo en puerto 5001
Cliente conectado en puerto 5001
Cliente conectado en puerto 5001
Cliente conectado en puerto 5001
Iniciando carrera en puerto 5001
Carrera finalizada en puerto 5001
PS D:\Escuela8voSem\Distribuidos\Practica4> |

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas caracter
ísticas y mejoras. https://aka.ms/PSWindows

PS D:\Escuela8voSem\Distribuidos\Practica4> java Cliente
Conectado a localhost:5001
[T=1] Auto 1 ha avanzado a 12 metros
[T=4] Auto 1 ha avanzado a 32 metros
[T=5] Auto 1 ha avanzado a 44 metros
[T=8] Auto 1 ha avanzado a 56 metros
[T=9] Auto 1 ha avanzado a 75 metros
[T=12] Auto 1 ha avanzado a 86 metros
[T=16] Auto 1 ha avanzado a 99 metros
[T=21] Auto 1 ha avanzado a 100 metros
[T=24] Auto 1 ha terminado la carrera!
PS D:\Escuela8voSem\Distribuidos\Practica4> |

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas caracter
ísticas y mejoras. https://aka.ms/PSWindows

PS D:\Escuela8voSem\Distribuidos\Practica4> java Cliente
Conectado a localhost:5001
[T=3] Auto 2 ha avanzado a 7 metros
[T=7] Auto 2 ha avanzado a 11 metros
[T=11] Auto 2 ha avanzado a 13 metros
[T=13] Auto 2 ha avanzado a 23 metros
[T=15] Auto 2 ha avanzado a 28 metros
[T=18] Auto 2 ha avanzado a 37 metros
[T=19] Auto 2 ha avanzado a 55 metros
[T=22] Auto 2 ha avanzado a 56 metros
[T=23] Auto 2 ha avanzado a 60 metros
[T=25] Auto 2 ha avanzado a 61 metros
[T=26] Auto 2 ha avanzado a 69 metros
[T=29] Auto 2 ha avanzado a 84 metros
[T=30] Auto 2 ha avanzado a 92 metros
[T=32] Auto 2 ha avanzado a 95 metros
[T=33] Auto 2 ha avanzado a 100 metros
[T=33] Auto 2 ha terminado la carrera!
PS D:\Escuela8voSem\Distribuidos\Practica4> |

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas caracter
ísticas y mejoras. https://aka.ms/PSWindows

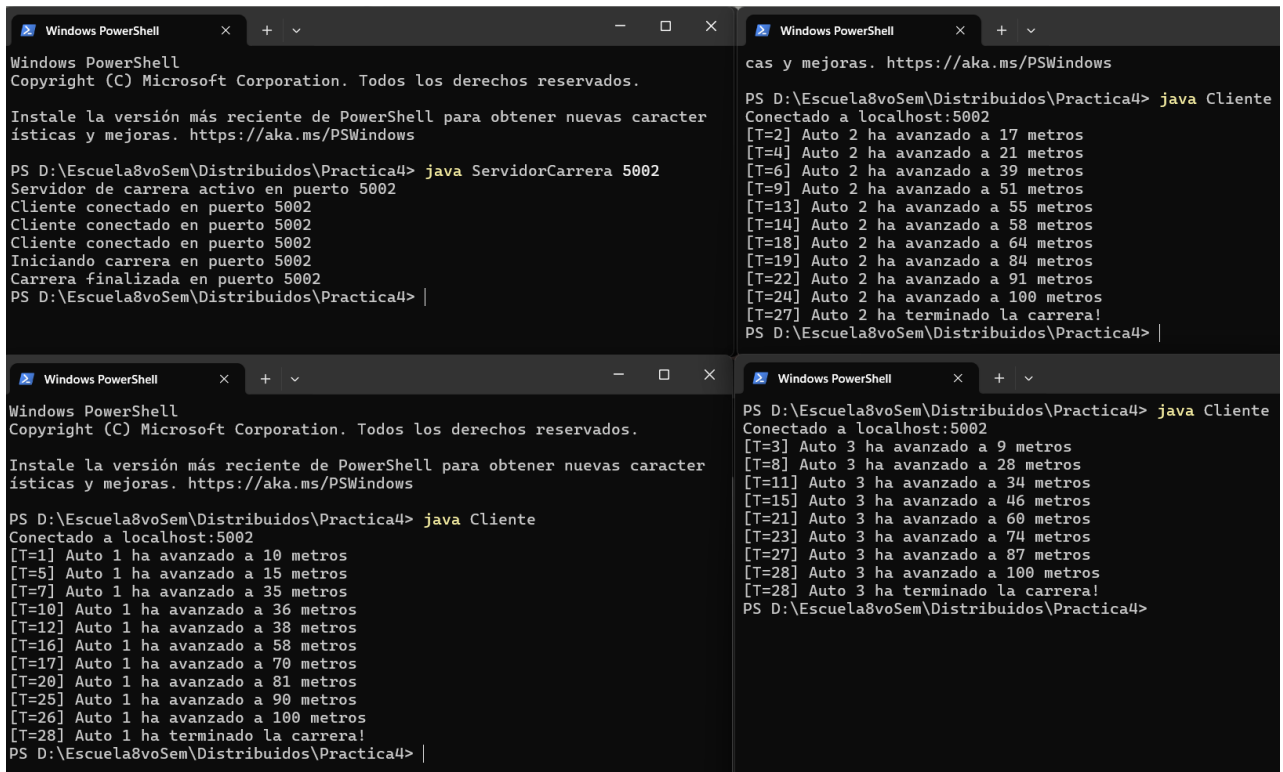
PS D:\Escuela8voSem\Distribuidos\Practica4> java Cliente
Conectado a localhost:5001
[T=2] Auto 3 ha avanzado a 16 metros
[T=6] Auto 3 ha avanzado a 31 metros
[T=10] Auto 3 ha avanzado a 40 metros
[T=14] Auto 3 ha avanzado a 57 metros
[T=17] Auto 3 ha avanzado a 61 metros
[T=20] Auto 3 ha avanzado a 62 metros
[T=24] Auto 3 ha avanzado a 82 metros
[T=27] Auto 3 ha avanzado a 84 metros
[T=28] Auto 3 ha avanzado a 90 metros
[T=31] Auto 3 ha avanzado a 100 metros
[T=33] Auto 3 ha terminado la carrera!
PS D:\Escuela8voSem\Distribuidos\Practica4>

```

**Figura 8. Prueba de Ejecución 1 – Servidor 5001 con 3 Clientes.**

Después tenemos que ver como quedo la carrera en el servidor con el localhost 5002, en la figura 9 vemos que se realiza correctamente la conexión en el puerto 5002 y espera la conexión de 3 clientes para poder iniciar la carrera, en esta prueba gano el auto 2, en la posición 2 quedo el auto 1 y en último lugar llego el auto 3.

Además en esta prueba de ejecución podemos ver que el distribuidor conectado en el puerto 6000 se encarga de conectar a los clientes en los diferentes servidores disponibles en los puertos 5001 y 5002, conforme el cliente y el proceso llega al distribuidor, este se encarga de mandar la dirección de un servidor disponible a cada cliente para posteriormente realizar la carrera.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS D:\Escuela8voSem\Distribuidos\Practica4> java ServidorCarrera 5002
Servidor de carrera activo en puerto 5002
Cliente conectado en puerto 5002
Cliente conectado en puerto 5002
Cliente conectado en puerto 5002
Iniciando carrera en puerto 5002
Carrera finalizada en puerto 5002
PS D:\Escuela8voSem\Distribuidos\Practica4> |

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS D:\Escuela8voSem\Distribuidos\Practica4> java Cliente
Conectado a localhost:5002
[T=2] Auto 2 ha avanzado a 17 metros
[T=4] Auto 2 ha avanzado a 21 metros
[T=6] Auto 2 ha avanzado a 39 metros
[T=9] Auto 2 ha avanzado a 51 metros
[T=13] Auto 2 ha avanzado a 55 metros
[T=14] Auto 2 ha avanzado a 58 metros
[T=18] Auto 2 ha avanzado a 64 metros
[T=19] Auto 2 ha avanzado a 84 metros
[T=22] Auto 2 ha avanzado a 91 metros
[T=24] Auto 2 ha avanzado a 100 metros
[T=27] Auto 2 ha terminado la carrera!
PS D:\Escuela8voSem\Distribuidos\Practica4> |

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS D:\Escuela8voSem\Distribuidos\Practica4> java Cliente
Conectado a localhost:5002
[T=1] Auto 1 ha avanzado a 10 metros
[T=5] Auto 1 ha avanzado a 15 metros
[T=7] Auto 1 ha avanzado a 35 metros
[T=10] Auto 1 ha avanzado a 36 metros
[T=12] Auto 1 ha avanzado a 38 metros
[T=16] Auto 1 ha avanzado a 58 metros
[T=17] Auto 1 ha avanzado a 70 metros
[T=20] Auto 1 ha avanzado a 81 metros
[T=25] Auto 1 ha avanzado a 90 metros
[T=26] Auto 1 ha avanzado a 100 metros
[T=28] Auto 1 ha terminado la carrera!
PS D:\Escuela8voSem\Distribuidos\Practica4> |

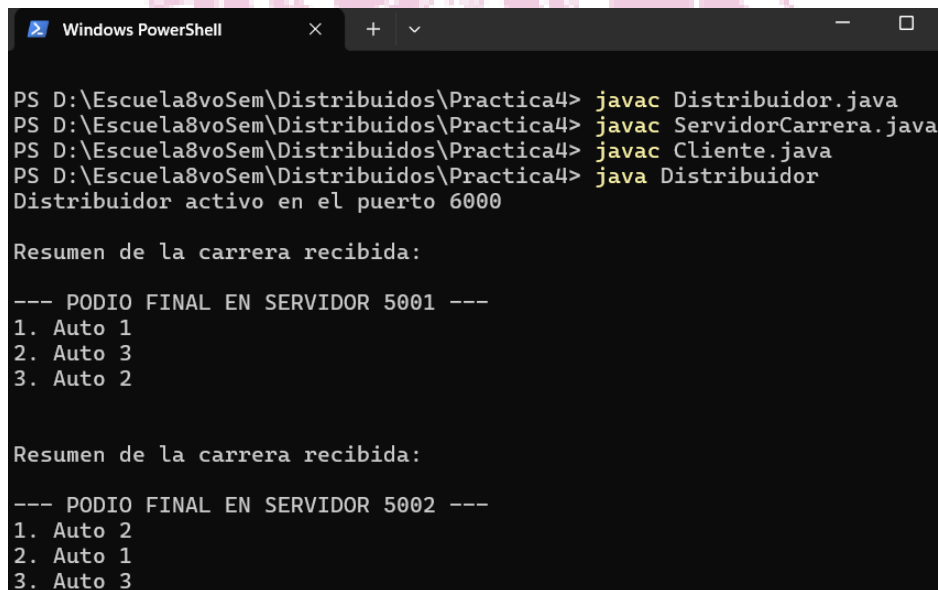
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS D:\Escuela8voSem\Distribuidos\Practica4> java Cliente
Conectado a localhost:5002
[T=3] Auto 3 ha avanzado a 9 metros
[T=8] Auto 3 ha avanzado a 28 metros
[T=11] Auto 3 ha avanzado a 34 metros
[T=15] Auto 3 ha avanzado a 46 metros
[T=21] Auto 3 ha avanzado a 60 metros
[T=23] Auto 3 ha avanzado a 74 metros
[T=27] Auto 3 ha avanzado a 87 metros
[T=28] Auto 3 ha avanzado a 100 metros
[T=28] Auto 3 ha terminado la carrera!
PS D:\Escuela8voSem\Distribuidos\Practica4>
```

**Figura 9. Prueba de Ejecución 1 – Servidor 5002 con 3 Clientes.**

Finalmente en la figura 10, podemos ver en el distribuidor el podio final de las carreras realizadas en los diferentes servidores utilizados.



```
Windows PowerShell

PS D:\Escuela8voSem\Distribuidos\Practica4> javac Distribuidor.java
PS D:\Escuela8voSem\Distribuidos\Practica4> javac ServidorCarrera.java
PS D:\Escuela8voSem\Distribuidos\Practica4> javac Cliente.java
PS D:\Escuela8voSem\Distribuidos\Practica4> java Distribuidor
Distribuidor activo en el puerto 6000

Resumen de la carrera recibida:

--- PODIO FINAL EN SERVIDOR 5001 ---
1. Auto 1
2. Auto 3
3. Auto 2

Resumen de la carrera recibida:

--- PODIO FINAL EN SERVIDOR 5002 ---
1. Auto 2
2. Auto 1
3. Auto 3
```

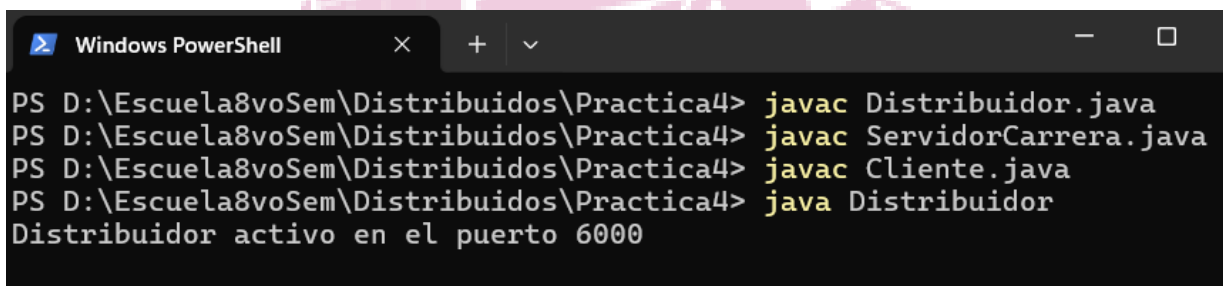
**Figura 10. Prueba de Ejecución 1 – Distribuidor con el podio de cada servidor.**



A continuación se puede ver en la figura 11, 12, 13, 14 y 15 una segunda prueba de ejecución del código desde la terminal, para observar que la carrera si es aleatoria y ahora conectaremos clientes a 3 servidores en lugar de 2 como se vio en la prueba de ejecución 1:

Para esto solo se modifica el código con el nombre Distribuidor.java en la parte del arreglo que contiene los localhost, se agrega el 5003 para ahora poder vincular clientes con 3 servidores.

En la figura 11 podemos observar que los códigos se ejecutaron desde la terminal, para esto primero se compilan los códigos con el comando "javac Distribuidor.java", "javac ServidorCarrera.java" y "javac Cliente.java" después se ejecuta primero el distribuidor con el comando "java Distribuidor" el cual se activa en el puerto 6000.

A screenshot of a Windows PowerShell terminal window. The title bar shows 'Windows PowerShell' with standard window controls. The terminal text shows the following commands and output:

```
PS D:\Escuela8voSem\Distribuidos\Practica4> javac Distribuidor.java
PS D:\Escuela8voSem\Distribuidos\Practica4> javac ServidorCarrera.java
PS D:\Escuela8voSem\Distribuidos\Practica4> javac Cliente.java
PS D:\Escuela8voSem\Distribuidos\Practica4> java Distribuidor
Distribuidor activo en el puerto 6000
```

**Figura 11. Prueba de Ejecución 2 – Distribuidor.**

Ahora en la figura 12 podemos ver que se ejecuta un servidor primero en el localhost 5001 y espera la conexión de 3 clientes para poder iniciar la carrera, en esta prueba gano el auto 3, en la posición 2 quedo el auto 1 y en último lugar llego el auto 2.



```
PS D:\Escuela8voSem\Distribuidos\Practica4> java ServidorCarrera 5001
Servidor de carrera activo en puerto 5001
Cliente conectado en puerto 5001
Cliente conectado en puerto 5001
Cliente conectado en puerto 5001
Iniciando carrera en puerto 5001
Carrera finalizada en puerto 5001
PS D:\Escuela8voSem\Distribuidos\Practica4>

PS D:\Escuela8voSem\Distribuidos\Practica4> java Cliente
Conectado a localhost:5001
[T=1] Auto 1 ha avanzado a 18 metros
[T=4] Auto 1 ha avanzado a 20 metros
[T=8] Auto 1 ha avanzado a 39 metros
[T=13] Auto 1 ha avanzado a 43 metros
[T=16] Auto 1 ha avanzado a 61 metros
[T=20] Auto 1 ha avanzado a 64 metros
[T=21] Auto 1 ha avanzado a 77 metros
[T=25] Auto 1 ha avanzado a 85 metros
[T=27] Auto 1 ha avanzado a 100 metros
[T=27] Auto 1 ha terminado la carrera!
PS D:\Escuela8voSem\Distribuidos\Practica4>

PS D:\Escuela8voSem\Distribuidos\Practica4> java Cliente
Conectado a localhost:5001
[T=2] Auto 2 ha avanzado a 3 metros
[T=5] Auto 2 ha avanzado a 21 metros
[T=9] Auto 2 ha avanzado a 36 metros
[T=11] Auto 2 ha avanzado a 49 metros
[T=12] Auto 2 ha avanzado a 56 metros
[T=18] Auto 2 ha avanzado a 61 metros
[T=22] Auto 2 ha avanzado a 73 metros
[T=23] Auto 2 ha avanzado a 74 metros
[T=26] Auto 2 ha avanzado a 78 metros
[T=28] Auto 2 ha avanzado a 82 metros
[T=29] Auto 2 ha avanzado a 84 metros
[T=30] Auto 2 ha avanzado a 94 metros
[T=31] Auto 2 ha avanzado a 99 metros
[T=32] Auto 2 ha avanzado a 100 metros
[T=32] Auto 2 ha terminado la carrera!
PS D:\Escuela8voSem\Distribuidos\Practica4>

PS D:\Escuela8voSem\Distribuidos\Practica4> java Cliente
Conectado a localhost:5001
[T=3] Auto 3 ha avanzado a 7 metros
[T=6] Auto 3 ha avanzado a 17 metros
[T=7] Auto 3 ha avanzado a 25 metros
[T=10] Auto 3 ha avanzado a 36 metros
[T=14] Auto 3 ha avanzado a 56 metros
[T=15] Auto 3 ha avanzado a 73 metros
[T=17] Auto 3 ha avanzado a 80 metros
[T=19] Auto 3 ha avanzado a 91 metros
[T=24] Auto 3 ha avanzado a 100 metros
[T=25] Auto 3 ha terminado la carrera!
PS D:\Escuela8voSem\Distribuidos\Practica4>
```

**Figura 12. Prueba de Ejecución 2 – Servidor 5001 con 3 Clientes.**

Después tenemos que ver como quedo la carrera en el servidor con el localhost 5002, en la figura 13 vemos que se realiza correctamente la conexión en el puerto 5002 y espera la conexión de 3 clientes para poder iniciar la carrera, en esta prueba gano el auto 3, en la posición 2 quedo el auto 2 y en último lugar llego el auto 1.

```
PS D:\Escuela8voSem\Distribuidos\Practica4> java ServidorCarrera 5002
Servidor de carrera activo en puerto 5002
Cliente conectado en puerto 5002
Cliente conectado en puerto 5002
Cliente conectado en puerto 5002
Iniciando carrera en puerto 5002
Carrera finalizada en puerto 5002
PS D:\Escuela8voSem\Distribuidos\Practica4>

PS D:\Escuela8voSem\Distribuidos\Practica4> java Cliente
Conectado a localhost:5002
[T=2] Auto 2 ha avanzado a 16 metros
[T=6] Auto 2 ha avanzado a 25 metros
[T=7] Auto 2 ha avanzado a 32 metros
[T=11] Auto 2 ha avanzado a 33 metros
[T=14] Auto 2 ha avanzado a 46 metros
[T=18] Auto 2 ha avanzado a 66 metros
[T=19] Auto 2 ha avanzado a 86 metros
[T=21] Auto 2 ha avanzado a 93 metros
[T=23] Auto 2 ha avanzado a 100 metros
[T=25] Auto 2 ha terminado la carrera!
PS D:\Escuela8voSem\Distribuidos\Practica4>

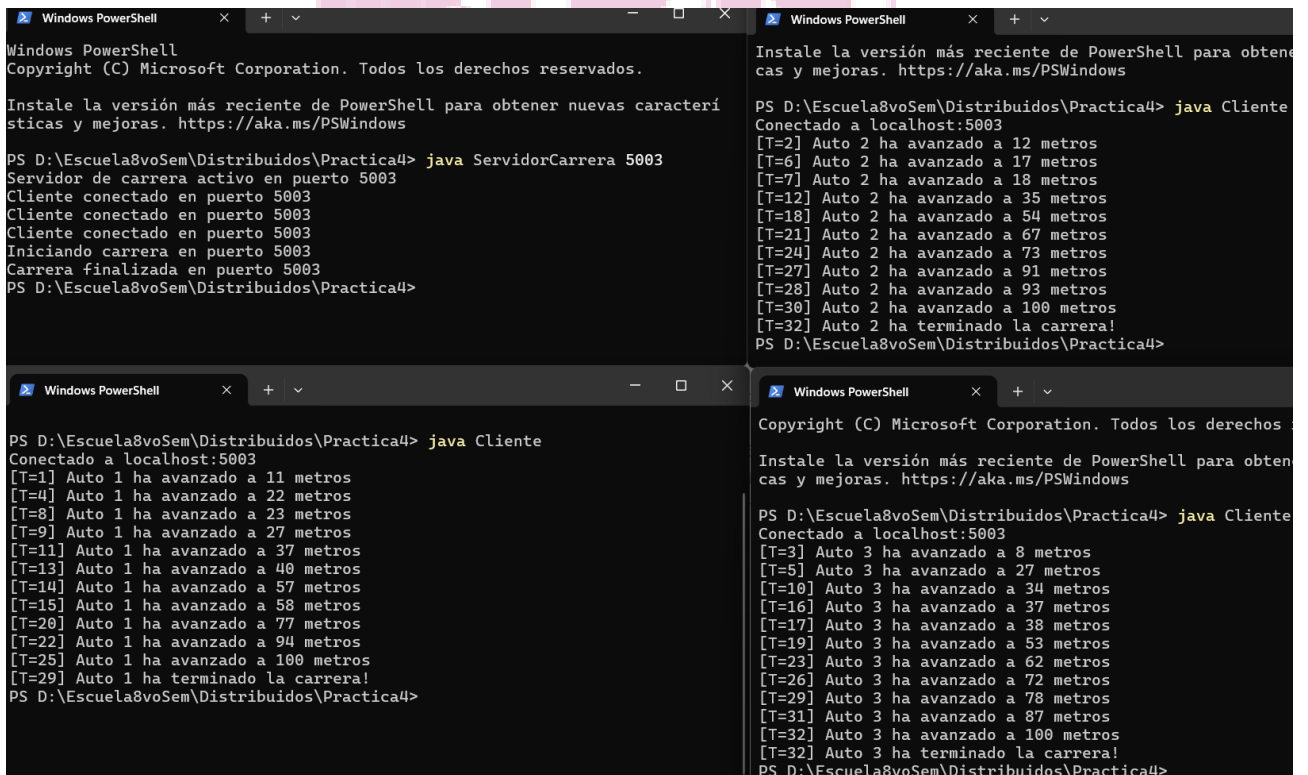
PS D:\Escuela8voSem\Distribuidos\Practica4> java Cliente
Conectado a localhost:5002
[T=1] Auto 1 ha avanzado a 15 metros
[T=4] Auto 1 ha avanzado a 22 metros
[T=9] Auto 1 ha avanzado a 35 metros
[T=12] Auto 1 ha avanzado a 45 metros
[T=16] Auto 1 ha avanzado a 59 metros
[T=20] Auto 1 ha avanzado a 62 metros
[T=22] Auto 1 ha avanzado a 74 metros
[T=24] Auto 1 ha avanzado a 90 metros
[T=25] Auto 1 ha avanzado a 100 metros
[T=25] Auto 1 ha terminado la carrera!
PS D:\Escuela8voSem\Distribuidos\Practica4>

PS D:\Escuela8voSem\Distribuidos\Practica4> java Cliente
Conectado a localhost:5002
[T=2] Auto 3 ha avanzado a 9 metros
[T=3] Auto 3 ha avanzado a 25 metros
[T=5] Auto 3 ha avanzado a 45 metros
[T=8] Auto 3 ha avanzado a 59 metros
[T=10] Auto 3 ha avanzado a 61 metros
[T=13] Auto 3 ha avanzado a 80 metros
[T=15] Auto 3 ha avanzado a 96 metros
[T=17] Auto 3 ha avanzado a 100 metros
[T=21] Auto 3 ha terminado la carrera!
PS D:\Escuela8voSem\Distribuidos\Practica4>
```

**Figura 13. Prueba de Ejecución 2 – Servidor 5002 con 3 Clientes.**

Ahora tenemos que ver como quedo la carrera en el servidor con el localhost 5003, en la figura 14 vemos que se realiza correctamente la conexión en el puerto 5003 y espera la conexión de 3 clientes para poder iniciar la carrera, en esta prueba gano el auto 1, en la posición 2 quedo el auto 2 y en último lugar llego el auto 3.

Además en esta prueba de ejecución podemos ver que el distribuidor conectado en el puerto 6000 se encarga de conectar a los clientes en los diferentes servidores disponibles en los puertos 5001, 5002 y 5003, conforme el cliente y el proceso llega al distribuidor, este se encarga de mandar la dirección de un servidor disponible a cada cliente para posteriormente realizar la carrera.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS D:\Escuela8voSem\Distribuidos\Practica4> java ServidorCarrera 5003
Servidor de carrera activo en puerto 5003
Cliente conectado en puerto 5003
Cliente conectado en puerto 5003
Cliente conectado en puerto 5003
Iniciando carrera en puerto 5003
Carrera finalizada en puerto 5003
PS D:\Escuela8voSem\Distribuidos\Practica4>

Windows PowerShell
Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS D:\Escuela8voSem\Distribuidos\Practica4> java Cliente
Conectado a localhost:5003
[T=2] Auto 2 ha avanzado a 12 metros
[T=6] Auto 2 ha avanzado a 17 metros
[T=7] Auto 2 ha avanzado a 18 metros
[T=12] Auto 2 ha avanzado a 35 metros
[T=18] Auto 2 ha avanzado a 54 metros
[T=21] Auto 2 ha avanzado a 67 metros
[T=24] Auto 2 ha avanzado a 73 metros
[T=27] Auto 2 ha avanzado a 91 metros
[T=28] Auto 2 ha avanzado a 93 metros
[T=30] Auto 2 ha avanzado a 100 metros
[T=32] Auto 2 ha terminado la carrera!
PS D:\Escuela8voSem\Distribuidos\Practica4>

Windows PowerShell
PS D:\Escuela8voSem\Distribuidos\Practica4> java Cliente
Conectado a localhost:5003
[T=1] Auto 1 ha avanzado a 11 metros
[T=4] Auto 1 ha avanzado a 22 metros
[T=8] Auto 1 ha avanzado a 23 metros
[T=9] Auto 1 ha avanzado a 27 metros
[T=11] Auto 1 ha avanzado a 37 metros
[T=13] Auto 1 ha avanzado a 40 metros
[T=14] Auto 1 ha avanzado a 57 metros
[T=15] Auto 1 ha avanzado a 58 metros
[T=20] Auto 1 ha avanzado a 77 metros
[T=22] Auto 1 ha avanzado a 94 metros
[T=25] Auto 1 ha avanzado a 100 metros
[T=29] Auto 1 ha terminado la carrera!
PS D:\Escuela8voSem\Distribuidos\Practica4>

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS D:\Escuela8voSem\Distribuidos\Practica4> java Cliente
Conectado a localhost:5003
[T=3] Auto 3 ha avanzado a 8 metros
[T=5] Auto 3 ha avanzado a 27 metros
[T=10] Auto 3 ha avanzado a 34 metros
[T=16] Auto 3 ha avanzado a 37 metros
[T=17] Auto 3 ha avanzado a 38 metros
[T=19] Auto 3 ha avanzado a 53 metros
[T=23] Auto 3 ha avanzado a 62 metros
[T=26] Auto 3 ha avanzado a 72 metros
[T=29] Auto 3 ha avanzado a 78 metros
[T=31] Auto 3 ha avanzado a 87 metros
[T=32] Auto 3 ha avanzado a 100 metros
[T=32] Auto 3 ha terminado la carrera!
PS D:\Escuela8voSem\Distribuidos\Practica4>
```

**Figura 14. Prueba de Ejecución 2 – Servidor 5003 con 3 Clientes.**

Finalmente en la figura 15, podemos ver en el distribuidor el podio final de las carreras realizadas en los diferentes servidores utilizados.

```
Windows PowerShell
PS D:\Escuela8voSem\Distribuidos\Practica4> javac Distribuidor.java
PS D:\Escuela8voSem\Distribuidos\Practica4> javac ServidorCarrera.java
PS D:\Escuela8voSem\Distribuidos\Practica4> javac Cliente.java
PS D:\Escuela8voSem\Distribuidos\Practica4> java Distribuidor
Distribuidor activo en el puerto 6000

Resumen de la carrera recibida:

--- PODIO FINAL EN SERVIDOR 5001 ---
1. Auto 3
2. Auto 1
3. Auto 2

Resumen de la carrera recibida:

--- PODIO FINAL EN SERVIDOR 5002 ---
1. Auto 3
2. Auto 2
3. Auto 1

Resumen de la carrera recibida:

--- PODIO FINAL EN SERVIDOR 5003 ---
1. Auto 1
2. Auto 2
3. Auto 3
```

**Figura 15. Prueba de Ejecución 2 – Distribuidor con el podio de cada servidor.**

## 7. Conclusiones

La implementación de sockets en Java ha demostrado ser una herramienta efectiva para establecer comunicación en tiempo real dentro de una arquitectura MultiServidor-MultiCliente. Se logró diseñar un sistema en el que múltiples clientes se conectan a servidores de carrera a través de un distribuidor, el cual gestiona las asignaciones de manera eficiente. Gracias a esta estructura, se facilita la distribución equitativa de los clientes entre los servidores disponibles, optimizando el uso de los recursos y evitando la sobrecarga de un único servidor.

Los resultados obtenidos reflejan que la arquitectura distribuida permite que cada cliente reciba actualizaciones inmediatas sobre el progreso de su auto en la carrera, sin bloqueos ni pérdida de datos. Además, la ejecución concurrente en cada servidor garantiza que los autos avancen de manera independiente, generando una simulación realista y dinámica. Para asegurar el correcto funcionamiento del sistema, se han implementado mecanismos de control, como el método centralizado para consolidar resultados y el uso del reloj de Lamport para coordinar el orden de los eventos dentro de los servidores, evitando inconsistencias en la recepción de datos.

## 8. Referencias Bibliográficas

- IBM. (s.f.). Ejemplo: utilizar sockets para la comunicación entre procesos. Recuperado de <https://www.ibm.com/docs/es/i/7.3?topic=communication-example-using-sockets-interprocess>
- Nacimiento, M. (2020, 31 de mayo). Algoritmos de sincronización de relojes. DEV Community. Recuperado de <https://dev.to/martinnacimiento/algoritmos-de-sincronizacion-de-relojes-56e8>
- Píldoras Informáticas. (2014, 24 de marzo). Comunicación con Sockets en Java (1/2) [Video]. YouTube. Recuperado de [https://www.youtube.com/watch?v=tX\\_RDW5QYMY](https://www.youtube.com/watch?v=tX_RDW5QYMY)