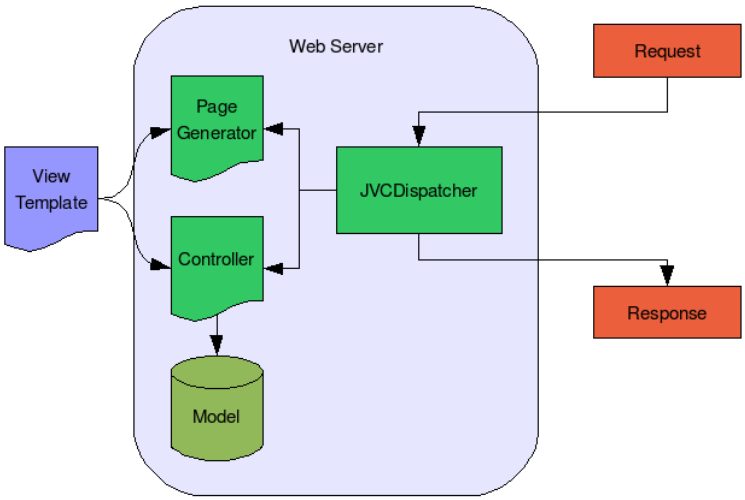
INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Desarrollo de Sistemas Distribuidos

Práctica 4 ‘Multiclientes - Multiservidores’



Docente:

Carreto Arellano Chadwick

Grupo:

4CM11

Alumnos:

Rodríguez Hernández Marlene Guadalupe

Fecha: martes 7, octubre 2025

Antecedentes

Multiclientes: muchos clientes pueden conectarse al servidor

Multiservidor: debe haber varios servidores trabajando al mismo tiempo, para distribuir la carga o manejar diferentes tareas en paralelo

Planteamiento del problema

Retomando nuevamente la carrera de tortugas pero haciendo la implementación de multiclientes multi servidores, lograremos que cada servidor maneje una carrera de tortugas, y cada tortuga podrá conectarse a un servidor específico (a una carrera en particular), de esta forma se podrán ejecutar múltiples servidores y múltiples clientes al mismo tiempo.

Propuesta de solución

Con base en el código anterior vamos a modificar un poco para que ahora podamos crear más servidores en puertos distintos, de esta forma cada servidor maneja su propia carrera de tortugas (implementando 3 carreras, la principal de 100 pasos, una de 50 pasos y otra de 25 pasos), y a su vez vamos a modificar el codigo de tortugas.java (cliente) para que se pueda conectar a diferentes servidores, de esta forma al ejecutar varias instancias de tortugas las veremos correr en diferentes servidores, tomando en cuenta que para esto tendremos un servidor principal (que funcionara como un coordinador) y los servidores secundarios, teniendo en cuenta que:

- cada servidor se mantiene abierto hacia los demás servidores
- cuando una tortuga avanza, su servidor envía mensaje de actualización de su posición a los demás servidores logrando así que todos los servidores tengan un estado global de la carrera

de esta forma cuando alguna tortuga gane la carrera avisara a su servidor y este al servidor principal para coordinar y avisar a los demás servidores que alguien ya ganó la carrera y terminarla.

Desarrollo de la solución

Comenzaré explicando un poco sobre el servidor coordinador, este lleva el control global de todas las carreras, marca el inicio de cada carrera cuando se cumple el mínimo de tortugas o pasa el tiempo de espera para que estas se conecten, también

indica al ganador de cada una de las carreras y envía mensaje a los servidores secundarios de cada carrera para iniciar las carreras y esperar a que estos tengan tortugas conectadas para poder dar inicio a las carreras, finalmente muestra un resumen de los ganadores por cada carrera.

```
public static void main(String[] args) {
    System.out.println("Coordinador arrancando en puerto " + PORT);
    try (ServerSocket serverSocket = new ServerSocket(PORT)) {
        while (true) {
            Socket s = serverSocket.accept();
            pool.execute(new HandlerServidor(s));
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        pool.shutdown();
    }
}
```

```
String line;
while ((line = in.readLine()) != null) {
    String[] parts = line.trim().split(regex:"\\s+");
    if (parts.length == 0) continue;
    String cmd = parts[0];

    if ("REGISTER".equalsIgnoreCase(cmd) && parts.length >= 4) {
        // REGISTRAR <name> <port> <meta>
        name = parts[1];
        servidores.put(name, out);
        System.out.println("Servidor registrado: " + name + " (port=" + parts[2] + ", meta=" + parts[3] + ")")
        out.println(x:"OK REGISTRADO");
    } else if ("LISTO".equalsIgnoreCase(cmd) && parts.length >=2) {
        String race = parts[1];
        System.out.println("Coordinador: servidor LISTO para carrera " + race + " -> autorizando INICIAR");
        out.println(x:"INICIAR");
    } else if ("GANADOR".equalsIgnoreCase(cmd) && parts.length >=3) {
        String race = parts[1];
        String winner = parts[2];
        ganadores.put(race, winner);
        System.out.println("Coordinador: recibio GANADOR para " + race + " -> " + winner);
        // Si todos los servidores tienen ganadores se imprime un breve resumen
        maybePrintSummary();
    } else {
        System.out.println("Coordinador recibio: " + line);
    }
}
atch (IOException e) {
System.out.println("Conexion con servidor cerrada: " + name);
```

```java
// En caso de que todos los servidores tengan ganadores, se muestran
if (!servidores.isEmpty() && ganadores.keySet().containsAll(servidores.keySet())) {
    System.out.println(x:"=== RESUMEN DE GANADORES ===");
    for (String race : ganadores.keySet()) {
        System.out.println("Carrera " + race + " -> Ganador: " + ganadores.get(race));
    }
    System.out.println(x:"============================");
}
```

Después tenemos a los servidores secundarios (al servidor que lleva la carrera de 50 pasos y al que lleva la carrera corta de 25 pasos), estos reciben a las tortugas, simulan los pasos que dan en cada una de las carreras y envían mensaje al coordinador de sus posiciones e informan a sus tortugas cuando se inicia la carrera, sus avances y el resultado de quien gane la carrera, pero esta en un solo archivo llamado ServerRace, y coordina las carreras en distintos puertos. Tiene funciones para aceptar a las tortugas en la carrera, esperar el tiempo limite para que se unan mas tortugas y se avise al coordinador y este indique el inicio de la carrera, y al momento de que esta de inicio muestre que tortugas se han unido (indicado con un numero) marque el inicio, fin y ganador de la carrera.

```java
private void acceptClients() {
    System.out.println(raceName + " escuchando tortugas en puerto " + listenPort);
    try (ServerSocket serverSocket = new ServerSocket(listenPort)) {
        while (!raceStarted) {
            Socket client = serverSocket.accept();
            clients.add(client);
            positions.put(client, value:0);
            System.out.println("[" + raceName + "] Tortuga conectada. Total: " + clients.size());
        }
    } catch (IOException e) {
        System.err.println("[" + raceName + "] Error accept: " + e.getMessage());
    }
}
```

```java
private void waitForClientsAndNotifyCoordinator() {
    long start = System.currentTimeMillis();
    while (true) {
        long elapsed = System.currentTimeMillis() - start;
        if (clients.size() >= minClients) break;
        if (elapsed >= waitMillis) break;
        try { Thread.sleep(millis:500); } catch (InterruptedException ignored) {}
    }
    // envia mensaje al coordinador de que esta listo
    System.out.println("[" + raceName + "] NOTIFICANDO LISTO al coordinador (clients=" + clients.size() + ")");
    coordOut.println("LISTO " + raceName);
}
```

```java
private void waitForCoordinatorStart() {
    System.out.println("[" + raceName + "] esperando instruccion INICIAR del coordinador...");
    try {
        String line;
        while ((line = coordIn.readLine()) != null) {
            if ("INICIAR".equalsIgnoreCase(line.trim())) {
                System.out.println("[" + raceName + "] Coordinador autorizo INICIAR");
                raceStarted = true;
                break;
            } else {
                // ignnorar mensajes
            }
        }
    } catch (IOException e) {
        System.err.println("[" + raceName + "] Error leyendo coordinador: " + e.getMessage());
    }
}
```

```java
private void runRace() {
    if (clients.isEmpty()) {
        System.out.println("[" + raceName + "] No hay tortugas para correr. Abortando.");
        return;
    }
    System.out.println("[" + raceName + "] INICIANDO carrera con " + clients.size() + " tortugas (meta=" + meta + ")");
    ExecutorService pool = Executors.newFixedThreadPool(Math.max(a:1, clients.size()));
    for (Socket client : new ArrayList<>(clients)) {
        pool.execute(() -> handleClientRace(client));
    }
    pool.shutdown();
    try { pool.awaitTermination(timeout:10, TimeUnit.MINUTES); } catch (InterruptedException ignored) {}
    System.out.println("[" + raceName + "] Carrera finalizada localmente.");
}
```

```java
private void handleClientRace(Socket client) {
    String tortugaId = "T-" + client.getPort();
    try (PrintWriter out = new PrintWriter(client.getOutputStream(), autoFlush:true)) {
        Random rnd = new Random();
        while (!raceFinished) {
            int advance = rnd.nextInt(bound:10) + 1;
            positions.compute(client, (k,v) -> (v == null ? 0 : v) + advance);
            int pos = positions.get(client);
            out.println("POS " + raceName + " " + tortugaId + " " + pos);

            if (pos >= meta && !raceFinished) {
                raceFinished = true;
                out.println("GANADOR " + raceName + " " + tortugaId);
                // envia mensaje de que ha ganado
                broadcastToClients("GANADOR " + raceName + " " + tortugaId);
                // envia mensaje al coordinador
                coordOut.println("GANADOR " + raceName + " " + tortugaId);
                break;
            }
            Thread.sleep(millis:500);
        }
    } catch (IOException | InterruptedException e) {
        System.err.println("[" + raceName + "] error tortuga " + tortugaId + ": " + e.getMessage());
    } finally {
        try { client.close(); } catch (IOException ignored) {}
    }
}
```

Por último tenemos a las tortugas (clientes), las cuales deciden a qué carreras quieren conectarse (ya sea que se conecten a 1 o 2 o a las 3) y reciben mensajes de cada servidor sobre su avance y resultados.

```java
private static void runClient(String host, int port) {
    try (Socket s = new Socket(host, port);
         BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));
         PrintWriter out = new PrintWriter(s.getOutputStream(), autoFlush:true)) {

        System.out.println("Conectado a " + host + ":" + port + " (localPort=" + s.getLocalPort() + ")");
        String line;
        while ((line = in.readLine()) != null) {
            System.out.println("[" + host + ":" + port + "] " + line);
        }

    } catch (IOException e) {
        System.err.println("Error tortuga en " + host + ":" + port + " -> " + e.getMessage());
    }
}
```

Resultados

Inicializamos el servidor coordinador y esperamos a que inicien los demás servidores y asu vez se conecten las tortugas correderas en cada uno de ellos para poder dar inicio en cada carrera, cuando estas finalizan se muestra al ganador de cada carrera y una pequeña lista (un resumen de los ganadores).

```
C:\Users\kirit\Downloads\ESCOM\DSD-R\Multiclientes-Multiservidor>java Coordinador
Coordinador arrancando en puerto 6000
Servidor registrado: RACE100 (port=5000, meta=100)
Servidor registrado: RACE50 (port=5001, meta=50)
Servidor registrado: RACE25 (port=5002, meta=25)
Coordinador: servidor LISTO para carrera RACE100 -> autorizando INICIAR
Servidor registrado: RACE50 (port=5001, meta=50)
Servidor registrado: RACE25 (port=5002, meta=25)
Coordinador: servidor LISTO para carrera RACE100 -> autorizando INICIAR
Coordinador: servidor LISTO para carrera RACE100 -> autorizando INICIAR
Coordinador: servidor LISTO para carrera RACE50 -> autorizando INICIAR
Coordinador: servidor LISTO para carrera RACE25 -> autorizando INICIAR
Conexion con servidor cerrada: RACE100
Conexion con servidor cerrada: RACE100
Conexion con servidor cerrada: RACE50
Conexion con servidor cerrada: RACE25
Servidor registrado: RACE100 (port=5000, meta=100)
Servidor registrado: RACE50 (port=5001, meta=50)
Servidor registrado: RACE25 (port=5002, meta=25)
Coordinador: servidor LISTO para carrera RACE100 -> autorizando INICIAR
Coordinador: servidor LISTO para carrera RACE50 -> autorizando INICIAR
Coordinador: servidor LISTO para carrera RACE25 -> autorizando INICIAR
Coordinador: recibio GANADOR para RACE25 -> T-47510
Coordinador: recibio GANADOR para RACE50 -> T-47508
Coordinador: recibio GANADOR para RACE100 -> T-47506
=== RESUMEN DE GANADORES ===
Carrera RACE25 -> Ganador: T-47510
Carrera RACE100 -> Ganador: T-47506
Carrera RACE100 -> Ganador: T-47506
Carrera RACE50 -> Ganador: T-47508
============================
```

Inicializamos el servidor para la carrera de 100 pasos y esperamos a que se conecten al menos dos tortugas para poder enviar mensaje al coordinador de que ya hay dos tortugas para poder iniciar la carrera, cuando esta da inicio se avisa a las tortugas y luego imprime cuando ya haya finalizado la carrera.

```
C:\Users\kirit\Downloads\ESCOM\DSD-R\Multiclientes-Multiservidor>java ServerRace RACE100 5000 100 localhost 6000 2 60000
RACE100 escuchando tortugas en puerto 5000
[RACE100] Tortuga conectada. Total: 1
[RACE100] Tortuga conectada. Total: 2
[RACE100] NOTIFICANDO LISTO al coordinador (clients=2)
[RACE100] esperando instruccion INICIAR del coordinador...
[RACE100] Coordinador autorizo INICIAR
[RACE100] INICIANDO carrera con 2 tortugas (meta=100)
[RACE100] Carrera finalizada localmente.
```

Inicializamos el servidor de la carrera de 50 pasos y se repite el mismo proceso.

```
C:\Users\kirit\Downloads\ESCOM\DSD-R\Multiclientes-Multiservidor>java ServerRace RACE50 5001 50 localhost 6000 2 60000
RACE50 escuchando tortugas en puerto 5001
[RACE50] Tortuga conectada. Total: 1
[RACE50] Tortuga conectada. Total: 2
[RACE50] NOTIFICANDO LISTO al coordinador (clients=2)
[RACE50] esperando instruccion INICIAR del coordinador...
[RACE50] Coordinador autorizo INICIAR
[RACE50] INICIANDO carrera con 2 tortugas (meta=50)
[RACE50] Carrera finalizada localmente.
```

Finalmente inicializamos el servidor de la carrera de 25 pasos y esperamos a que las tortugas se conecten para poder iniciar la carrera.

```
C:\Users\kirit\Downloads\ESCOM\DSD-R\Multiclientes-Multiservidor>java ServerRace RACE25 5002 25 localhost 6000 2 60000
RACE25 escuchando tortugas en puerto 5002
[RACE25] Tortuga conectada. Total: 1
[RACE25] Tortuga conectada. Total: 2
[RACE25] NOTIFICANDO LISTO al coordinador (clients=2)
[RACE25] esperando instruccion INICIAR del coordinador...
[RACE25] Coordinador autorizo INICIAR
[RACE25] INICIANDO carrera con 2 tortugas (meta=25)
[RACE25] Carrera finalizada localmente.
```

Al terminar de inicializar los servidores de las carreras procedemos a conectar a las tortugas a las carreras y a su vez nos indica el numero asignado de la tortuga, nos muestra la carrera en la que esta compitiendo y su numero de pasos, finalmente muestra al ganador.

```
C:\Users\kirit\Downloads\ESCOM\DSD-R\Multiclientes-Multiservidor>java ClienteTortuga localhost:5000
Conectado a localhost:5000 (localPort=47506)
Conectado a localhost:5000 (localPort=47506)
[localhost:5000] POS RACE100 T-47506 9
[localhost:5000] POS RACE100 T-47506 15
[localhost:5000] POS RACE100 T-47506 24
[localhost:5000] POS RACE100 T-47506 34
[localhost:5000] POS RACE100 T-47506 43
[localhost:5000] POS RACE100 T-47506 52
[localhost:5000] POS RACE100 T-47506 62
[localhost:5000] POS RACE100 T-47506 72
[localhost:5000] POS RACE100 T-47506 81
[localhost:5000] POS RACE100 T-47506 85
[localhost:5000] POS RACE100 T-47506 89
[localhost:5000] POS RACE100 T-47506 95
[localhost:5000] POS RACE100 T-47506 102
[localhost:5000] GANADOR RACE100 T-47506
```

```
C:\Users\kirit\Downloads\ESCOM\DSD-R\Multiclientes-Multiservidor>java ClienteTortuga localhost:5001 localhost:5002
5001 localhost:5002
Conectado a localhost:5002 (localPort=47507)
Conectado a localhost:5002 (localPort=47507)
Conectado a localhost:5001 (localPort=47508)
[localhost:5001] POS RACE50 T-47508 10
[localhost:5002] POS RACE25 T-47507 4
[localhost:5001] POS RACE50 T-47508 13
[localhost:5002] POS RACE25 T-47507 7
[localhost:5001] POS RACE50 T-47508 23
[localhost:5002] POS RACE25 T-47507 8
[localhost:5001] POS RACE50 T-47508 30
[localhost:5002] GANADOR RACE25 T-47510
[localhost:5001] POS RACE50 T-47508 35
[localhost:5001] POS RACE50 T-47508 44
[localhost:5001] POS RACE50 T-47508 53
[localhost:5001] GANADOR RACE50 T-47508
```

```
C:\Users\kirit\Downloads\ESCOM\DSD-R\Multiclientes-Multiservidor>java ClienteTortuga localhost:5000 localhost:5001 localhost:5002
Conectado a localhost:5002 (localPort=47510)
Conectado a localhost:5001 (localPort=47511)
Conectado a localhost:5000 (localPort=47509)
[localhost:5000] POS RACE100 T-47509 2
[localhost:5001] POS RACE50 T-47511 5
[localhost:5002] POS RACE25 T-47510 8
[localhost:5000] POS RACE100 T-47509 7
[localhost:5001] POS RACE50 T-47511 11
[localhost:5002] POS RACE25 T-47510 13
[localhost:5000] POS RACE100 T-47509 8
[localhost:5001] POS RACE50 T-47511 16
[localhost:5002] POS RACE25 T-47510 22
[localhost:5000] POS RACE100 T-47509 15
[localhost:5001] POS RACE50 T-47511 16
[localhost:5002] POS RACE25 T-47510 22
[localhost:5000] POS RACE100 T-47509 15
[localhost:5000] POS RACE100 T-47509 15
[localhost:5001] POS RACE50 T-47511 25
[localhost:5002] POS RACE25 T-47510 26
[localhost:5002] GANADOR RACE25 T-47510
[localhost:5002] GANADOR RACE25 T-47510
[localhost:5002] GANADOR RACE25 T-47510
[localhost:5000] POS RACE100 T-47509 25
[localhost:5001] POS RACE50 T-47511 34
[localhost:5000] POS RACE100 T-47509 29
[localhost:5001] POS RACE50 T-47511 38
[localhost:5000] POS RACE100 T-47509 30
[localhost:5001] POS RACE50 T-47511 42
[localhost:5001] GANADOR RACE50 T-47508
[localhost:5000] POS RACE100 T-47509 40
[localhost:5000] POS RACE100 T-47509 44
[localhost:5000] POS RACE100 T-47509 54
[localhost:5000] POS RACE100 T-47509 57
[localhost:5000] POS RACE100 T-47509 67
[localhost:5000] POS RACE100 T-47509 70
[localhost:5000] GANADOR RACE100 T-47506
```

Conclusiones

Similar a los hilos y procesos, crear varios servidores para atender a distintos clientes es una forma de facilitar las cosas y optimizar recursos para liberar espacio en un solo servidor, creo que es una muy buena forma de distribuir tareas en distintos procesos.

Sin embargo, creo que seguir usando mi ejemplo de las carreras de tortugas es algo muy bueno, y he estado entendiendo mejor estas cosas.

Enlace a la práctica en GitHub

https://github.com/Marlene0807/Sistemas-Distribuidos.git

Referencias

Building a Multi-Client Server using Java Multithreading | by Gaurangjotwani | Medium

Lifecycle and States of a Thread in Java | GeeksforGeeks

Comunicación entre un servidor y múltiples clientes

Lesson: Concurrency (The Java™ Tutorials > Essential Java Classes)