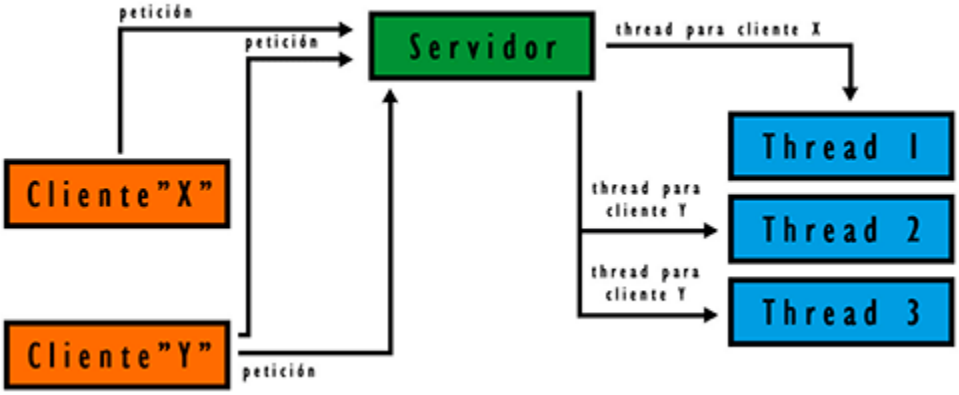
INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Desarrollo de Sistemas Distribuidos

Práctica 3 ‘Multiclientes - Servidor’



Docente:

Carreto Arellano Chadwick

Grupo:

7CM6

Alumnos:

Rodríguez Hernández Marlene Guadalupe

Fecha: martes 7, septiembre 2025

Antecedentes

Como hemos visto en las prácticas anteriores un thread o hilo es un flujo de control secuencial dentro de un programa y sabemos que Java proporciona el Multi Threading que permite varias ejecuciones de tareas en un programa, ejecutándose concurrentemente llevando a cabo tareas distintas, por lo que, Java se vuelve eficiente al manipular los recursos del sistema.

La arquitectura multicliente-servidor es un modelo de comunicación en el que varios clientes pueden conectarse a un servidor al mismo tiempo, entendiendo a su vez los siguientes conceptos:

- Java Threads
    - son flujos de control secuencial
    - aquí cada tortuga se ejecuta en su propio hilo para simular movimiento concurrente.
- Sockets
    - Comunicación entre cliente-servidor, así mismo, envía las posiciones y mensajes de estado
- ConcurrentHashMap y sincronización
    - los usamos para mantener el estado de la carrera seguro entre los hilos, en este caso entre las Tortugas
- ExecutorService
    - nos permite manejar múltiples clientes (Tortugas) sin bloquear el servidor
- Servidor
    - es un programa que se ejecuta en una máquina y espera conexiones de varios clientes
    - generalmente, usa ServerSocket para escuchar conexiones en un puerto específico
- Cliente
    - es un programa que se conecta al servidor utilizando Socket
    - envía y recibe respuestas del servidor
- Multicliente
    - el servidor debe ser capaz de manejar múltiples clientes simultáneamente, para esto suelen usarse los hilos (Threads) o los ejecutores (Executors) para gestionar las conexiones de manera concurrente

Al tener en cuenta estos conceptos, podemos analizar el flujo de comunicación, estructurando de la siguiente forma:

- Servidor activo
    - el ServerSocket escucha en un puerto

- Cliente solicita conexión
    - un cliente inicia un Socket y solicita conexión al puerto del servidor
- Servidor acepta la conexión
    - el servidor acepta la conexión usando accept(), el cual retorna un nuevo Socket para comunicarse con el cliente
- Comunicación
    - se establece un canal de comunicación a través de flujos de entrada y salida (InputStream y OutputStream)
- Manejo de múltiples clientes
    - cada cliente se asigna a un nuevo hilo (Thread) o usa un ExecutorService para ejecutarse en paralelo

Sabiendo esto, no podemos olvidar los conceptos clave a utilizar para el multicliente:

- Hilos, cada conexión de cliente se maneja en un hilo separado (Thread)
- Sincronización, si hay recursos compartidos estos pueden requerir sincronización con synchronized o Locks
- Seguridad, siempre se validan los datos recibidos y maneja las excepciones de manera adecuada
- Optimización, ya que para muchas conexiones se recomienda usar ThreadPoolExecutor

Planteamiento del problema

Retomando todos estos conceptos, debemos recordar que en la práctica anterior implementamos los sockets con la estructura cliente/servidor, resolviendo así la problemática de la carrera de tortugas en la cual cada tortuga (cliente) debe conectarse a la carrera (servidor) simular que cada una avanza a una velocidad distinta y la carrera finaliza cuando alguna logra llegar a la meta (paso número 100), avanzando en distintas velocidades (simulando pasos del 1 al 9).

Ahora para la implementación de Multicliente-Servidor pretendo hacer que las tortugas se conecten en diferentes momentos para que así el servidor maneje múltiples clientes al mismo tiempo sin olvidar que cada tortuga pueda recibir actualizaciones en tiempo real y el servidor pueda reaccionar sin bloquear la ejecución.

Propuesta de solución

En lugar de esperar a que las cinco tortugas se conecten antes de empezar la carrera vamos a crear un hilo para cada tortuga permitiendo así que cada una se conecte en distinto momento, pero tomaremos en cuenta un máximo de un minuto para que inicie la carrera con las tortugas que estén (al menos deben ser 2), de esta forma si alguna

tortuga se conecta después de haber iniciado la carrera pues llevará una desventaja en tiempo de pasos pero eso no la excluye de ganar.

Desarrollo de la solución

Teniendo en cuenta el desarrollo de las prácticas anteriores creamos el archivo Server.java y Tortuga.java

A continuación explicare primero un poco del archivo Tortuga.java, en la siguiente imagen en la clase Tortuga definimos el puerto al cual se va a conectar cada una de las tortugas

```java
public class Tortuga {
    private static final String HOST = "localhost";
    private static final int PUERTO = 5500;
```

Una vez ejecutado el código y corriendo el servidor, las tortugas se conectaran al servidor principal enviando un mensaje de que se han conectado con un numero en especifico

```java
public static void main(String[] args) {
    try (Socket socket = new Socket(HOST, PUERTO);
        BufferedReader input = new BufferedReader(new InputStreamReader(socket.getInputStream()))) {
            System.out.println("Conectado al servidor como Tortuga #" + socket.getPort());

            String mensaje;
            while ((mensaje = input.readLine()) != null) {
                System.out.println(mensaje);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
}
```

En las siguientes imagenes explico un poco sobre el código del Server.java, definimos el puerto que vamos a usar, y la meta para que termine la carrera que son 100 pasos, al inicializar el servidor mostrará servidor iniciado y conforme se vayan conectando las tortugas mostrará tortuga x conectada, asimismo se mostrará la forma en que avanzan cada una de las tortugas en la terminal que cada una de ellas se encuentre usando

```java
public class Servidor {
    private static final int PUERTO = 5500;
    private static final int META = 100;
    private static final int MIN_TORTUGAS = 2;
    private static final int TIEMPO_ESPERA = 60; // 1 minuto

    private static List<PrintWriter> clientes = new ArrayList<>();
    private static Map<Integer, Integer> posiciones = new ConcurrentHashMap<>();
    private static boolean carreraIniciada = false;
    private static boolean carreraTerminada = false;
    private static int contadorTortugas = 0;
```

Una vez que el servidor inicia, espera las conexiones de las tortugas; cada tortuga se conecta como un cliente y comienza a recibir actualizaciones de su posición así como de la posición de las demás que se vayan uniendo

La carrera inicia automáticamente tras un tiempo de un minuto con la condición de que haya mínimo dos tortugas para correr en la carrera

Cada tortuga avanza de forma aleatoria y, al llegar a la meta, se declara ganador avisando a las demás tortugas que ya ha ganado para que estas se detengan y a su vez el servidor indique que la carrera ha finalizado y muestra quién ha ganado

Resultados

Podemos ver que al utilizar el ExecutorService para gestionar los hilos del servidor y ConcurrentHashMap para almacenar las posiciones de cada tortuga de forma segura. Cada tortuga se conecta y recibe actualizaciones periódicas hasta que una llegue a la meta (100 pasos)

```
C:\Users\kirit\Downloads\ESCOM\DSD-R\Multiclientes-Servidor>javac Servidor.java Tortu

ga.java
ga.java
ga.java
ga.java
ga.java
or>java Servidor
Servidor de la carrera de Tortugas iniciado en el puerto 5500
Carrera iniciada con 2tortugas
Carrera finalizada. Gano la tortuga #2
AC
```

```
C:\Users\kirit\Downloads\ESCOM\DSD-R\Multiclientes-Servidor>java Tortuga
Conectado al servidor como Tortuga #5500
La carrera ha terminado!
Tortuga #2avanzo a 3
Tortuga #1avanzo a 3
Tortuga #2avanzo a 5
Tortuga #1avanzo a 5
Tortuga #2avanzo a 7
Tortuga #1avanzo a 8
Tortuga #2avanzo a 8
Tortuga #1avanzo a 10
Tortuga #2avanzo a 9
Tortuga #1avanzo a 11
Tortuga #2avanzo a 10
Tortuga #1avanzo a 14
Tortuga #2avanzo a 13
Tortuga #1avanzo a 16
Tortuga #2avanzo a 14
Tortuga #1avanzo a 18
Tortuga #2avanzo a 17
Tortuga #1avanzo a 20
Tortuga #2avanzo a 19
Tortuga #1avanzo a 22
Tortuga #3avanzo a 3
Tortuga #2avanzo a 22
Tortuga #1avanzo a 23
Tortuga #3avanzo a 4
Tortuga #2avanzo a 25
Tortuga #1avanzo a 24
Tortuga #3avanzo a 5
Tortuga #2avanzo a 28
Tortuga #1avanzo a 26
Tortuga #3avanzo a 7
Tortuga #2avanzo a 29
Tortuga #1avanzo a 27
Tortuga #3avanzo a 9
Tortuga #2avanzo a 31
Tortuga #1avanzo a 30
```

```
Tortuga #3avanzo a 10
Tortuga #2avanzo a 32
Tortuga #1avanzo a 33
Tortuga #3avanzo a 12
Tortuga #2avanzo a 33
Tortuga #1avanzo a 34
Tortuga #3avanzo a 14
Tortuga #2avanzo a 36
Tortuga #1avanzo a 37
Tortuga #3avanzo a 16
Tortuga #2avanzo a 39
Tortuga #1avanzo a 38
Tortuga #3avanzo a 19
Tortuga #2avanzo a 41
Tortuga #1avanzo a 41
Tortuga #3avanzo a 20
Tortuga #2avanzo a 42
Tortuga #1avanzo a 42
Tortuga #3avanzo a 23
Tortuga #2avanzo a 45
Tortuga #1avanzo a 43
Tortuga #3avanzo a 24
Tortuga #2avanzo a 47
Tortuga #1avanzo a 44
Tortuga #3avanzo a 27
Tortuga #2avanzo a 49
Tortuga #1avanzo a 47
Tortuga #3avanzo a 30
Tortuga #2avanzo a 50
Tortuga #1avanzo a 49
Tortuga #3avanzo a 31
Tortuga #2avanzo a 52
Tortuga #1avanzo a 50
Tortuga #3avanzo a 33
Tortuga #2avanzo a 54
Tortuga #1avanzo a 51
```

```
Tortuga #3avanzo a 34
Tortuga #2avanzo a 57
Tortuga #1avanzo a 52
Tortuga #3avanzo a 35
Tortuga #2avanzo a 58
Tortuga #1avanzo a 53
Tortuga #3avanzo a 36
Tortuga #2avanzo a 61
Tortuga #1avanzo a 56
Tortuga #3avanzo a 37
Tortuga #2avanzo a 63
Tortuga #1avanzo a 58
Tortuga #3avanzo a 39
Tortuga #2avanzo a 64
Tortuga #1avanzo a 59
Tortuga #3avanzo a 41
Tortuga #2avanzo a 66
Tortuga #1avanzo a 61
Tortuga #3avanzo a 44
Tortuga #2avanzo a 68
Tortuga #1avanzo a 63
Tortuga #3avanzo a 45
Tortuga #2avanzo a 70
Tortuga #1avanzo a 65
Tortuga #3avanzo a 48
Tortuga #2avanzo a 73
Tortuga #1avanzo a 68
Tortuga #3avanzo a 50
Tortuga #2avanzo a 76
Tortuga #1avanzo a 69
Tortuga #3avanzo a 53
Tortuga #2avanzo a 78
Tortuga #1avanzo a 71
Tortuga #3avanzo a 55
Tortuga #2avanzo a 79
Tortuga #1avanzo a 72
Tortuga #3avanzo a 58
```

```
Tortuga #2avanzo a 80
Tortuga #1avanzo a 75
Tortuga #3avanzo a 59
Tortuga #2avanzo a 81
Tortuga #1avanzo a 77
Tortuga #3avanzo a 62
Tortuga #2avanzo a 83
Tortuga #1avanzo a 79
Tortuga #3avanzo a 63
Tortuga #2avanzo a 84
Tortuga #1avanzo a 81
Tortuga #3avanzo a 65
Tortuga #2avanzo a 87
Tortuga #1avanzo a 82
Tortuga #3avanzo a 67
Tortuga #2avanzo a 89
Tortuga #1avanzo a 84
Tortuga #3avanzo a 70
Tortuga #2avanzo a 92
Tortuga #1avanzo a 87
Tortuga #3avanzo a 71
Tortuga #2avanzo a 93
Tortuga #1avanzo a 88
Tortuga #3avanzo a 72
Tortuga #2avanzo a 94
Tortuga #1avanzo a 91
Tortuga #3avanzo a 74
Tortuga #2avanzo a 97
Tortuga #1avanzo a 91
Tortuga #3avanzo a 74
Tortuga #2avanzo a 97
Tortuga #3avanzo a 74
Tortuga #2avanzo a 97
Tortuga #1avanzo a 93
Tortuga #3avanzo a 76
Tortuga #2avanzo a 97
Tortuga #1avanzo a 93
```

Conclusiones

Estas prácticas las he ido trabajando y mejorando con cada implementación, así he comprendiendo cada vez mejor cómo funcionan los procesos e hilos con la carrera de tortugas, después implementamos la parte del servidor como la carrera y después he implementado los sockets con clientes servidor interpretando todavía la carrera de las tortugas pero a si mismo he ido implementando unas pocas mejoras para ir avanzando el código e irlo mejorando hasta llegar a esta implementación de multiclientes servidor, interpretando que hay muchos clientes (tortugas), en este caso hay al menos dos tortugas conectadas al mismo servidor, lo vemos como la misma carrera, para que se unan todas a una sola carrera

Enlace a la práctica en GitHub

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

Referencias

Implementación de un Servidor HTTP Multihilo en Java para Gestión Concurrente de Solicitudes - ACADEMIA SANROQUE

Chat multi-hilos cliente/servidor escrito en Java siguiendo el patrón de diseño MVC - Parte II

Ejemplo de Sockets en Java: chat básico entre cliente y servidor - Parzibyte's blog

Microsoft PowerPoint - CursoJava1-PrincipiosJava