Instituto Politécnico Nacional

Escuela Superior de Cómputo

DESARROLLO DE SISTEMAS DISTRIBUIDOS

Práctica 1

***Profesor**: Chadwick Carreto Arellano*

***Grupo**: 4CM11*

***Alumno**: González Hipólito Miguel Ángel*

Fecha de entrega: 25/Febrero/2025

Contents

1 Introducción

Un *subproceso* es un hilo de ejecución de un programa. Cada thread tiene una prioridad, los de mayor prioridad son ejecutados primero que los de baja prioridad y cada uno puede ser marcado como un Daemon. Cuando se ejecuta el código en algún thread, este crea un nuevo objeto Thread e inicialmente tiene la misma prioridad del thread donde se creó.

En la programación moderna, el manejo de múltiples procesos o tareas simultáneas es fundamental para optimizar el rendimiento y la eficiencia de las aplicaciones.

2 Desarrollo

En esta práctica, implementamos la ejecución de hilos secuenciales y concurrentes en Java utilizando la clase Thread. Para ello, creamos una clase principal Hilos que contiene dos métodos y dos clases que implementan la interfaz threads para diferenciar entre el thread 1 y el thread 2:

ejecutaHiloConcurrente() → Ejecuta dos hilos en paralelo.

ejecutaHiloSecuencial() → Ejecuta dos hilos de forma ordenada, uno después del otro.

2.1 Implementación de Hilos Concurrentes

El método ejecutaHiloConcurrente() crea dos hilos (Hilo1 y Hilo2) y los inicia con start(). Luego, utiliza join() para esperar a que ambos finalicen antes de continuar.

```java
public void ejecutaHiloConcurrente() {
        Hilo1 h1 = new Hilo1();
        Hilo2 h2 = new Hilo2();

        h1.start();
        h2.start();

        try {
            h1.join();
            h2.join();
            System.err.println("Hilos ejecutados concurrentemente");
        } catch (InterruptedException e) {
            System.out.println("Error en la ejecución de los hilos");
        }
    }
```

Ilustración 1

2.2 Implementación de Hilos Secuenciales

El método ejecutaHiloSecuencial() inicia Hilo2, espera a que termine (join()), y luego inicia Hilo1, asegurando una ejecución en orden.

```java
public void ejecutaHiloSecuencial(){
    Hilo1 h1 = new Hilo1();
    Hilo2 h2 = new Hilo2();

    h2.start();
    try {
        h2.join();
    } catch (InterruptedException e) {
        System.out.println("Error en la ejecución de Hilo1");
    }

    h1.start();
    try {
        h1.join();
    } catch (InterruptedException e) {
        System.out.println("Error en la ejecución de Hilo2");
    }
    System.err.println("Hilos ejecutados secuencialmente");
}
```

Ilustración 2

3 Ejecución del programa

En la Ilutración 3 se aprecia que la ejecución de los hilos de forma concurrente se aprecia como se imprime el texto de ambos hilos de forma desordenada, indicándonos que se están ejecutando al mismo tiempo pero dado que ambos no pueden imprimir al mismo tiempo, en algunas líneas se imprime primero el hilo 2 y posteriormente el hilo 1 o viceversa. Por otro lado se aprecia que en la ejecución secuencal el hilo 2 siempre se ejecutará antes que el hilo 1, permitiéndonos tener un mejor control sobre la ejecución de un programa.

```
Hilo 2: 0
Hilo 1: 0
Hilo 2: 1
Hilo 1: 1
Hilo 1: 2
Hilo 2: 2
Hilo 1: 3
Hilo 1: 4
Hilo 2: 3
Hilo 1 terminado
Hilo 2: 4
Hilo 2 terminado
Hilos ejecutados concurrentemente


Hilo 2: 0
Hilo 2: 1
Hilo 2: 2
Hilo 2: 3
Hilo 2: 4
Hilo 2 terminado
Hilo 1: 0
Hilo 1: 1
Hilo 1: 2
Hilo 1: 3
Hilo 1: 4
Hilo 1 terminado
Hilos ejecutados secuencialmente
PS C:\Users\Migue\Documents\GitHub\Desarrollo-de-sistemas-distribuidos>
```

Ilustración 3

4 Conclusiones

En esta práctica, exploramos la ejecución de hilos en Java utilizando dos enfoques: concurrente y secuencial. A partir de la implementación y ejecución del código, se pueden extraer las siguientes conclusiones:

1. La concurrencia mejora el rendimiento:

   Al ejecutar los hilos en paralelo, el sistema aprovecha mejor los recursos del procesador.

   Esto es útil cuando se tienen múltiples tareas independientes que pueden ejecutarse simultáneamente sin depender una de la otra.

2. La ejecución secuencial garantiza el orden:

   Cuando se requiere que una tarea finalice antes de iniciar otra, la ejecución secuencial es la mejor opción.

   Esto es importante en procesos donde los datos de una tarea dependen de la finalización de otra.