Prácticas Concurrencia y Distribución (20/21)

Arno Formella, Anália García Lourenço, Noelia García Hervella

semana 1 marzo – 5 marzo

Práctica 3: Gestión de hilos para tareas concurrentes

Objetivos: Gestión de hilos: interrupciones y variables propias

Material adicional:

- https://docs.oracle.com/javase/10/docs/api/java/lang/Thread.html
- https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/ThreadLocal.html
- https://docs.oracle.com/javase/tutorial/essential/concurrency/interrupt.html

1. Variables locales del hilo e interrupciones

   Este problema continúa explorando dos temas importantes en la administración de hilos:

   a) el alcance de las variables locales y
   b) la interrupción de hilos.

   Para hacer eso, sigue los siguientes pasos:

   a) **Configuración del problema:** crea una clase de hilo vacío (por el momento) (`MyThread`) que extenda `Thread` (o implemente `Runnable`). También crea una clase principal (por ejemplo, `MyProblem`) donde dentro de su método *main* crea un *array* (o `ArrayList`) para almacenar las referencias a objetos `MyThread`.
   b) **Variables locales del hilo:** En la clase `MyThread`, crea una variable privada de tipo `Integer` (con nombre como `mySum`). ¿Qué sucede con respecto a este atributo privado cuando hay multiples instancias de hilos ejecutandose? A continuación, sobreescribe el método `run()` para sumar los números de 0 a algún número *N* y guardarlo dentro de `mySum`. Una estructura muy esquemática para la clase del hilo sería:

```
class MyThread ... {
  private Integer mySum;
    //...
  @Override
  public void run() {
    // for loop {
       // print "started", threadID, mySum
       // sleep
       // increment mySum
    // }
    // print "finished",  threadID, mySum
  }
}
```

Explica el resultado, ¿El programa se comporta como lo esperabas?

c) **Variables locales en un subproceso:** en el API de Java, `ThreadLocal<>` es un método que se puede usar para mantener variables locales dentro de hilos. Reescribe la clase anterior creando una variable local con el mecanismo `ThreadLocal<Integer>` (consulta la documentación de Java). Demuestra que con este mecanismo los hilos solo suman sus propias variables locales. Ten en cuenta que al utilizar `ThreadLocal<>` tendrás que usar métodos como `get()` y `set()`.

d) **Interrumpir un hilo desde `main`:** cambia la tarea del hilo de la parte *b)* (de arriba) para calcular la constante π (pi). Para ello, existen muchas fórmulas iterativas, pero una muy sencilla de implementar es la siguiente:

$$\pi = 4 \cdot \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots\right)$$

```
for(int i=1; i<100000; i+=2) {
  if(negative) pi -= 1.0/i;
  else pi += 1.0/i;
  negative = !negative;
}
pi*=4.0;
```

A continuación, escribe el código apropiado en el programa principal para que interrumpa los hilos (empieza con un único hilo trabajador) después de que el hilo *main* duerma por un tiempo aleatorio. Para realizar y captar la interrupción consulta la documentación de Java, especialmente los métodos `interrupt()` y `interrupted()` de la clase `Thread`. ¿Qué comportamiento observas? ¿Se interrumpe inmediatamente el hilo? ¿Cómo podrías cuantificar tu respuesta, por ejemplo medir el tiempo entre iniciar la interrupción en el hilo principal y captarla en el hilo interrumpido?