Prácticas Concurrencia y Distribución (22/23)

Arno Formella, Alba Nogueira Rodríguez, David Ruano Ordás

semana 27 marzo – 31 marzo

Práctica 5: Sincronización de hilos

Objetivos: Diferentes formas de sincronización de hilos sobre una variable compartida.

1. El enfoque de esta práctica es desarrollar un contador concurrente y conpartido donde sincronizamos el acceso de un grupo de hilos a tal acumulador. Sigue los siguientes pasos:

   a) **Clase Counter:** crea una clase `Counter` que tenga un método `increment()` que incremente el valor del contador en una unidad y un método para obtener el valor actual del contador. El valor inicial del contador debe ser 0.

   b) **Clase MyTask:** crea una clase `MyTask` que extienda `Thread` o implemente `Runnable` y que se pueda construir recibiendo como parámetro un objeto de tipo `Counter`. Esta tarea debe dormir—dentro de un bucle `for` de cierto rango– un tiempo aleatorio entre 0 y 100 milisegundos y a continuación invocar el método `increment()` del contador.

   c) **Método principal:** crea un método principal que construya varios objetos de la clase `MyTask` que comparten un sólo objeto de tipo `Counter`, ejecuta cada tarea en un hilo y espera a que todos los hilos hayan finalizado. Cuando esto ocurra, imprime el valor actual del contador. Ejecuta el programa varias veces y con distintos números de hilos (por ejemplo: 1, 2, 4, 8, 16, 32, ..., 1024). Explica el comportamiento de la salida.

   d) **Bloques sincronizados:** Modifica la clase `MyTask` para que el incremento del contador se haga desde un bloque sincronizado (el bloque con la palabra reservada `synchronized` debe actuar sobre el objeto contador compartido). ¿Cómo cambia esto el comportamiento del programa principal? Familízate con las dos formas sintácticas que ofrece Java para tal fin (bloque síncronizado o método sincronizado).

   e) **Utilizando Java AtomicInteger/LongAdder:** Haz las modificaciones necesarias en el código anterior para usar un `AtomicInteger` y/o `LongAdder`. Para realizar la operación del incremento, usa un método adecuado del gran

conjunto disponible para estos objetos (consulta la documentación de la API de Java). Explica lo que observas.

2. Primer uso de `lock` (candado) para realizar la exclusión mutua de otra manera (comparado con el uso de `synchronized`).

   a) **Java Locks:** En lugar de bloques sincronizados, la API de Java tiene un conjunto de objetos de alto nivel para su uso en programas concurrentes. Iremos a utilizar el paquete `java.util.concurrent.locks`, que proporciona un mecanismo de bloqueo similar que proporciona la semántica de `synchronized`: es decir, se puede reemplazar el bloque sincronizado con un uso de un `lock`, que se adquiere antes de modificar el contador y que se libera una vez haberlo hecho.

   b) **Análisis/Medidas:** Modifica tu programa del apartado uno adecuadamente y ejecuta el código del contador para todos los métodos de sincronización empleados variando la cantidad de hilos utilizados. ¿Puedes notar una diferencia en el rendimiento (puedes quitar las esperas dentro del bucle for y aumentar el rango para obtener mejores medidas)? Explica tus resultados.