Universidad Nacional Autónoma de México
Facultad de Ciencias

Computación Concurrente

Práctica 3

Bonilla Reyes Dafne - 319089660
García Ponce Jose Camilo - 319210536
Juárez Ubaldo Juan Aurelio - 421095568

Exclusión Mutua: Candados Clásicos y el Modelo de Memoria de Java (JMM)

Descripción de los programas

- `Bakery.java:`

  Programa que implementa Bakery usando objetos atómicos (código de la profesora).

- `BakeryFor4.java:`

  Programa que implementa un candado Bakery para 4 hilos.

- `CounterNaive.java:`

  Programa que implementa un contador (código de la profesora).

- `DoublePeterson.java:`

  Programa que implementa un candado Peterson de 4 hilos.

- `Ejercicio1.java:`

  Programa que simula las 400 tareas del ejercicio 1.

- `ExecuteBakery.java:`

  Programa que simula tareas usando `Bakery.java` (código de la profesora).

- `ExecuteBakeryFor4.java:`

  Programa que simula las 400 tareas del ejercicio 2.

- `LockPeterson.java:`

  Programa que ejecuta 1000 tareas con `Peterson.java` (código de la profesora).

- `Nodo.java:`

  Programa que simula un nodo usando objetos atómicos (código de la profesora).

- `Peterson.java:`

  Programa que implementa un candado Peterson de 2 hilos. (código de la profesora).

Reporte de ejercicios

1. **El algoritmo de candado Peterson solo funciona para 2 hilos, utiliza el algoritmo de candado Peterson para implementar un algoritmo de candado que funcione para 4 hilos.** ***Hint: Apóyate del programa Peterson***

   (a) Para probar que tu candado funciona, crea una implementación en donde utilices un *ExecutorService* para ejecutar 400 tareas, cada tarea debe aumentar en uno un objeto Contador. Utiliza tu candado para 4 hilos para tener consistencia en tu Contador.

   Este ejercicio se implementó en los archivos `DoublePeterson.java` y `Ejercicio1.java`. Notemos que a veces el número de contador es menor a 400, ya que el candado Peterson (*Peterson.java*) usa módulo 2, por lo que, si los hilos 19,21 o 20,22 (o cuales quiera dos hilos con id par o los dos con id impar) entran al mismo tiempo, el candado los ve como el mismo hilo, ya que $19\,\%2 = 1$ y $21\,\%2 = 1$ y $20\,\%2 = 0$ y $22\,\%2 = 0$, por lo tanto, entran a la sección crítica al mismo tiempo, lo cual genera que no se cumpla la exclusión mutua entre estos dos hilos, no es común que pase esto, pero podría suceder. Este problema surge en el `lock3` de `DoublePeterson.java`, ya que en `lock1` y `lock2` nunca entran dos hilos con id impares o dos hilos con id pares.

   (b) De alguna forma obtén el número de veces que los hilos aumentan el contador. ¿Cada uno realiza exactamente 100 tareas o hay algunos que realizan más?

   Este ejercicio se implementó en el archivo `Ejercicio1.java`. Notemos que hay hilos que realizan más, no todos realizan el mismo número de tareas.

   ```
   El hilo 21 aumento el contador a 396
   El hilo 20 aumento el contador a 397
   El hilo 19 aumento el contador a 398
   El hilo 21 aumento el contador a 399
   El hilo 19 aumento el contador 139 veces
   El hilo 20 aumento el contador 115 veces
   El hilo 21 aumento el contador 120 veces
   El hilo 22 aumento el contador 26 veces
   400
   ```

   (c) ¿Consideras que la implementación cumple con Justicia? Justifica tu respuesta.

   En la tarea 2, vimos que `Double Peterson` no cumple con justicia. Como usamos el pseudocódigo de la tarea 2 como base, no se debería cumplir justicia. Ejemplo de ejecución usado en la tarea 2 en donde no se cumple justicia:

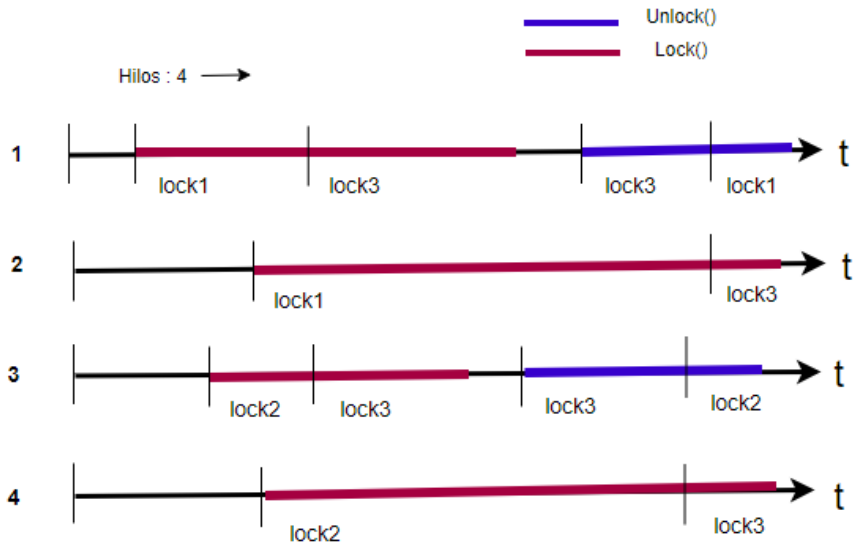

Figura 1: Violación de la propiedad de justicia

2. Con base en la implementación de Bakery vista en la clase teoría, implementa Bakery para 4 hilos. *Hint: Crea los arreglos flag y label de tamaño 4, si consideras necesario utiliza campos volatile*

(a) Con ayuda de un *ExecutorService* ejecuta 400 tareas, cada tarea debe aumentar en uno un objeto Contador. Utiliza tu candado para 4 hilos para tener consistencia en tu Contador.
La implementacion de bakery se encuentra en BakeryFor4.java y el ejecutor de tareas en ExecutorFor4.java

(b) De alguna forma obtén el número de veces que los hilos aumentan el contador. ¿Cada uno realiza exactamente 100 tareas o hay algunos que realizan más?
Usamos un hashmap concurrente que guarda cada vez que un hilo incrementa el contador:

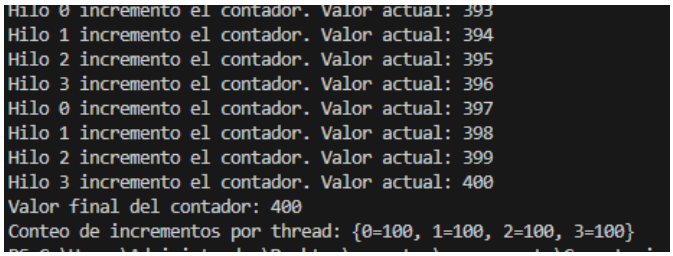

Figura 2: Veces que los hilos aumentan el contador

Podemos observar que el valor final del contador es 400 y que cada hilo aumenta el contador 100 veces.

(c) ¿Consideras que la implementación cumple con Justicia? Justifica tu respuesta.
Si observamos nuestra implementación, se cumple que cada hilo entra a la sección critica en el orden en el que llegaron, además con la imagen del inciso anterior, podemos observar que cada

hilo aumenta el contador el mismo numero de veces, por lo que nos aseguramos de que nuestra implementación cumple con justicia

3. Revisa el programa de Bakery que se les compartió como ejemplo, ejecútalo varias veces, revisa el código y contesta:

(a) Revisa para qué sirve el campo *AtomicReference*, y qué hacen los métodos *compareAndSet()* y *get()*.

El campo *AtomicReference* es un objeto referencia que se puede escribir y leer atómicamente. Que sea atómico significa que si varios hilos intentan cambiar el valor al mismo tiempo no termina con inconsistencias.

El método *compareAndSet()* define automáticamente el valor al valor dado si el valor actual es igual al valor esperado. Los parámetros son el valor a poner y el valor esperado. Por otro lado, el método *get()* regresa el valor actual del objeto *AtomicReference*.[1]

(b) Describe como funciona en a lo más 6 líneas de computadora.

El candado usa una lista de nodos encadenados. En el método `lock()` el hilo crea un nuevo nodo y lo intenta agregar a la cola y luego solo sale de `lock()` hasta que el hilo que agregó el nodo anterior haya terminado su trabajo. En *unlock()* solo se actualiza la bandera para que el siguiente nodo pueda entrar a la sección crítica. Las operaciones de agregar a la cola son seguras, ya que se usan operaciones atómicas.

(c) ¿Si *next* no es un *AtomicReference* sigue funcionando? *Hint: Recuerda la Cola concurrente sin candados que implementaste en la Práctica 2.*

No, ya que varios hilos podrían intentar agregar y, por lo tanto, la referencia de quien es el siguiente en la cola puede ser inconsistente, lo cual no sería bueno en una ejecución concurrente, ya que se podrían crear condiciones de carrera e inconsistencias.

(d) ¿Consideras que mantiene la lógica de la implementación vista en clase/tu implementación del ejercicio anterior? Justifica por qué.

El visto en la clase tiene que los hilos toman una etiqueta para saber en qué turno o momento van a pasar, de manera similar, en la implementación de la profesora, los hilos ponen su nodo en la pila para poder pasar. Notemos que en las dos formas los primeros que llegan son los primeros que pasan.

Referencias

[1] AtomicReference (Java Platform SE 8). (2024, 5 junio). https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/AtomicReference.html