Universidad Nacional Autónoma de México
Facultad de Ciencias

Computación Concurrente

Práctica 6

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

Monitores y Consenso

Descripción de los programas

- `CASConsensus.java:`

  Programa que implementa un protocolo de consenso con `compareAndSet()` y `get()` (código de la profesora).

- `CASConsensusT.java:`

  Programa que implementa un protocolo de consenso con `getAndSet()` y `get()`.

- `ConsensusProtocol.java:`

  Interfaz para protocolos de consenso (código de la profesora).

- `CountDownLatch.java:`

  Programa que implementa una versión de `CountDownLatch` (código de la profesora).

- `ExecConsenRounds.java:`

  Programa que ejecuta varias rondas de consenso, usa `CASConsensus.java` (código de la profesora).

- `ExecConsenRoundsH.java:`

  Programa que ejecuta varias rondas de consenso, usa `CASConsensus.java` e intenta que cada hilo gane el consenso eventualmente.

- `ExecConsenRoundsT.java:`

  Programa que ejecuta varias rondas de consenso, usa `CASConsensusT.java`.

- `ExecReadersWriters.java:`

  Programa que ejecuta y prueba `FifoReadWriteLock.java` (código de la profesora).

- `ExecReadersWritersN.java:`

  Programa que ejecuta y prueba `FifoReadWriteLockN.java`.

- `FifoReadWriteLock.java:`

  Programa para el candado FIFO de lectores y escritores que cumple unas propiedades (código de la profesora).

- `FifoReadWriteLockN.java:`

  Programa para el candado FIFO de lectores y escritores que cumple otras propiedades.

Reporte de ejercicios

1. **Modifica el *compareAndSet()* por un *testAndSet()* en el programa *CASConsensus*. Argumenta si sigue resolviendo el consenso para 4 hilos, agrega una captura de pantalla de la ejecución que lo sustente. Justifica tu respuesta en no más de dos renglones.**

   El código de este ejercicio se encuentra en `CASConsensusT.java` y `ExecConsenRoundsT.java`. Usamos `getAndSet()` en vez de `testAndSet()`, debido a que `AtomicInteger` no tiene el metodo `testAndSet()`

   Tenemos este fragmento de una ejecución (hicimos varias ejecuciones para que nos saliera una así y solo es un fragmento ya que las ejecuciones imprimen mucho texto)

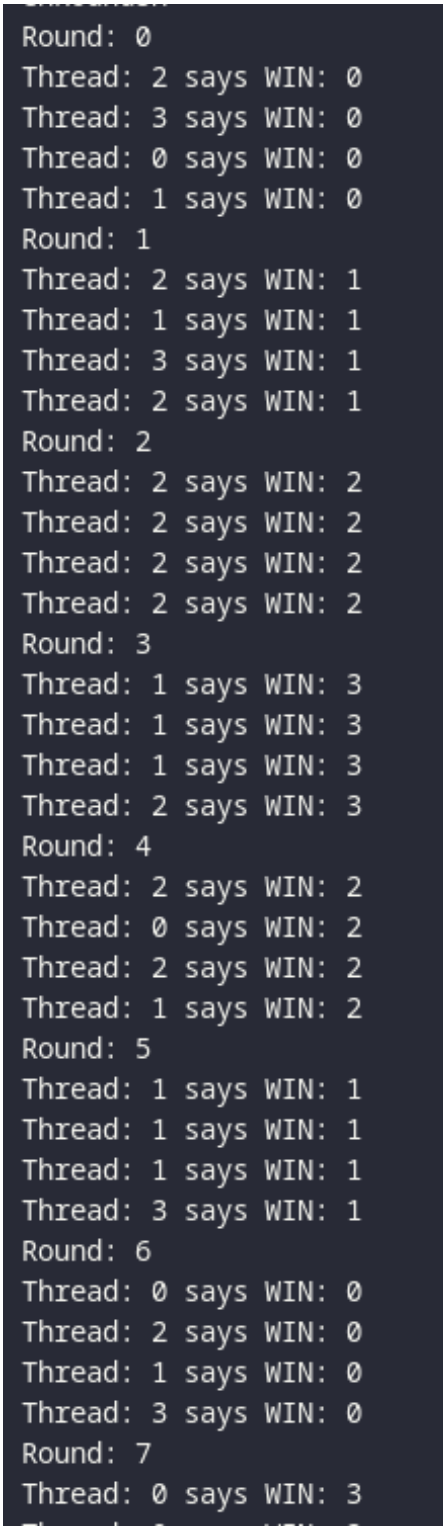
```
Round: 6
Thread: 2 says WIN: 2
Thread: 2 says WIN: 2
Thread: 2 says WIN: 2
Thread: 2 says WIN: 2
Round: 7
Thread: 3 says WIN: 3
Thread: 1 says WIN: 1
Thread: 2 says WIN: 2
Thread: 0 says WIN: 0
Round: 8
Thread: 1 says WIN: 1
Thread: 3 says WIN: 3
Thread: 2 says WIN: 2
Thread: 0 says WIN: 0
Round: 9
Thread: 1 says WIN: 1
Thread: 1 says WIN: 1
Thread: 1 says WIN: 1
Thread: 1 says WIN: 1

 Winners: [2, 2, 1, 2, 1, 1, 2, 0, 0, 1]
camilo@wowi:~/CC/septimoSemestre/Con/Computacion-Concurrente$
```

   Notemos que en la ronda 7, los hilos dijeron que ganaron diferentes hilos, esto es debido a que `testAndSet()` y `getAndSet()` tienen numero de consenso igual a 2 (lo vimos en clase), por lo cual como nuestra ejecución uso 4 hilos, entonces usar `getAndSet()` no nos garantiza resolver el consenso para los cuatro hilos, ya que no existe una manera de notar quien fue el hilo en ganar el consenso (el que realizo la acción primero).

2. **La implementación en *ExecConsenRounds* permite que solo unos hilos ganen el consenso. Implementa una *técnica de ayuda* similar al de la *Construcción universal wait-free* para que todos los hilos ganen el consenso eventualmente en alguna ronda. *Hint: Unos hilos ayudan a otros.***

El código de este ejercicio se encuentra en `ExecConsenRoundsH.java`. Nuestra solución logra que cada hilo gane al menos una vez el consenso en alguna ronda (en particular en las primeras c rondas cada hilo debería ganar una vez el consenso, con c la cantidad de hilos). Tenemos este fragmento de una ejecución (solo es un fragmento ya que las ejecuciones imprimen mucho texto) donde podemos notar que en las cuatro primeras rondas cada hilo gana una.



3. **Considera el monitor *FifoReadWriteLock*, actualmente cumple con:**

   (a) **Si un lector está en su sección crítica, ningún escritor puede entrar a su propia sección crítica**

   (b) **Si un escritor está en su sección crítica, ningún escritor o lector puede entrar a su propia sección crítica**

   Implementa el monitor modificando la propiedad (b) de forma que ahora se cumpla:

   (a) **Si un lector está en su sección crítica, ningún escritor puede entrar a su propia sección crítica**

   (b) **Si un escritor está en su sección crítica, ningún lector puede entrar a su propia sección crítica**

   No es necesario que el protocolo sea justo.

   El código de este ejercicio se encuentra en `FifoReadWriteLockN.java` y `ExecReadersWritersN.java`.

   Tenemos este fragmento de una ejecución (hicimos varias ejecuciones para que nos saliera una así, se notaran 2 lectores a la vez, y solo es un fragmento ya que las ejecuciones imprimen mucho texto).

```
Escritor entra y hay 1 escritores
Escritor entra y hay 2 escritores
Escritor sale y hay 0 escritores
Escritor sale y hay 1 escritores
Escritor entra y hay 1 escritores
Escritor sale y hay 0 escritores
Lector entra y hay 1 lectores
Lector entra y hay 2 lectores
Lector sale y hay 0 lectores
Lector sale y hay 1 lectores
Escritor entra y hay 1 escritores
Escritor sale y hay 0 escritores
Lector entra y hay 1 lectores
Lector sale y hay 0 lectores
Escritor entra y hay 1 escritores
Escritor sale y hay 0 escritores
Lector entra y hay 1 lectores
Lector sale y hay 0 lectores
Escritor entra y hay 1 escritores
Escritor entra y hay 2 escritores
Escritor sale y hay 1 escritores
Escritor sale y hay 0 escritores
Escritor entra y hay 1 escritores
Escritor sale y hay 0 escritores
camilo@wowi:~/CC/septimoSemestre/Con/Computacion-Concurrente$
```

4. **¿Cómo proporcionarías justicia a tu implementación del inciso anterior? No es necesario que lo implementes.**

   Una posible solucion es llevar un registro de la cantidad de lecturas y escrituras completadas, de esta manera, podemos dar prioridad a quienes tengan menor cantidad. Tambien podemos pensar en una cola FIFO, aunque, pueden surgir complicaciones debido a que los lectores pueden entrar a la SC mientras otro lector esta dentro, pero los escritores no pueden entrar si hay un lector dentro, esto hace mas dificil cumplir con justicia, y una posible solución podria ser que nadie pueda entrar a la SC, mientras alguien más ente en la SC. Otra solución podria ser asociar una marca de tiempo a cada solicitud y dar prioridad a las que tengan mas antiguedad.

Referencias

[1] Clases de laboratorio, teoría y ayudantías