Universidad Nacional Autónoma de México
Facultad de Ciencias

Computación Concurrente

Práctica 4

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

Spinlocks y algunas primitivas

Descripción de los programas

Todos los archivos son los vistos durante el laboratorio

- `ALock.java`

  Archivo para el candado ALock

- *Backoff.java*

  Archivo para el retraso"del candado BackoffLock

- *BackoffLock.java*

  Archivo para el candado ALock

- *CLHLock.java*

  Archivo para el candado CLHLock

- *CounterAtomic.java*

  Archivo para el contador atómico

- *MCSLock*.java

  Archivo para el candado MCSLock

- *RunSpin*.java

  Archivo para probar los candados

- *TASLock*.java

  Archivo para el candado TASLock

- *TTASLock*.java

  Archivo para el candado TTASLock

Reporte de ejercicios

1. **Utiliza el programa *RunSpin* para probar cada uno de los spinlocks, cada integrante en el equipo debe completar la siguiente tabla: (max-1 es el número máximo de hilos en tu compu menos uno)**

   - Tabla 1: Dafne Bonilla Reyes

   Computadora con 12 hilos

SpinLocks/Counter	TAS	TTAS	Backoff	MCS	ALock	CLHLock	Reentrant	Counter Atomic
Tiempo de ejecución 400 tareas con 4 hilos	10.0352 ms	12.7792 ms	32.0328 ms	15.0224 ms	10.5697 ms	12.6017 ms	12.06529 ms	13.6090 ms
Tiempo de ejecución 1000 tareas con 4 hilos	14.012901ms	15.487101ms	42.8792 ms	16.2454 ms	15.9262 ms	16.1296 ms	16.0233 ms	15.2936 ms
Tiempo de ejecución 1000 tareas con max-1 hilos	19.8951 ms	17.4799 ms	44.262ms	20.47659 ms	24.8615 ms	19.3078 ms	19.6463 ms	18.3145 ms

   - Tabla 2: José Camilo García Ponce

   Computadora con 64 hilos

SpinLocks/Counter	TAS	TTAS	Backoff	MCS	ALock	CLHLock	Reentrant	Counter Atomic
Tiempo de ejecución 400 tareas con 4 hilos	10.679798ms	10.365542ms	35.289291ms	11.093975ms	11.32798 ms	10.928635ms	11.840986ms	11.886322ms
Tiempo de ejecución 1000 tareas con 4 hilos	13.810328ms	13.431944ms	37.787839ms	14.398173ms	14.604256ms	13.962797ms	14.47318 ms	14.360762ms
Tiempo de ejecución 1000 tareas con max-1 hilos	28.480711ms	27.770457ms	56.324268ms	28.835939ms	35.758615ms	28.283222ms	28.928981ms	28.361407ms

   - Tabla 3: Juan Aurelio Juárez Ubaldo. **NOTA:** Ya que mi computadora solo tiene 4 hilos, no tiene caso probar max-1 hilos.

Computadora con 4 hilos

Tabla comparativa cuando la tarea es muy breve								
Spinlocks/Counter	TAS	TTAS	Backoff	MCS	Alock	CLHLock	Reentrant	Counter Atomic
Tiempo de ejecución: 400 tareas con **4** hilos	19.27ms	19.52ms	31.43ms	29.89ms	20.30ms	27.36ms	19.66ms	19.14ms
Tiempo de ejecución: 1000 tareas con **4** hilos	30.45ms	31.93ms	47.49ms	38.56ms	45.54ms	34.85ms	28.37ms	26.02ms
Tiempo de ejecución: 1000 tareas con **max-1** hilos	32.64ms	31.65ms	61.27ms	38.21ms	43.77ms	36.75ms	30.25ms	28.67ms

2. Modifica el método *task(lock)* para añadir un Sleep de 2000ms, y completa la siguiente tabla:

- Tabla 1: Dafne Bonilla Reyes

con Sleep de 2000ms

SpinLocks/ Counter	TAS	TTAS	Backoff	MCS	ALock	CLHLock	Reentrant
Tiempo de ejecución 200 tareas con 4 hilos	401604.5708ms	401483.498799ms	401492.98929999996 ms		401680.59249999997 ms		402000.3381ms
Tiempo de ejecución 200 tareas con max-1 hilos	402000.3381ms	402000.3381ms	401697.93919999996 ms		401697.93919999996 ms		401783.7179ms

- Tabla 2: José Camilo García Ponce

con Sleep de 2000ms

SpinLocks/ Counter	TAS	TTAS	Backoff	MCS	ALock	CLHLock	Reentrant
Tiempo de ejecución 200 tareas con 4 hilos	400041.004217ms	400039.250635ms	400039.777909ms		400040.7537639ms		400049.952413ms
Tiempo de ejecución 200 tareas con max-1 hilos	400037.831969ms	400049.808331ms	400047.100746ms		400043.331119ms		400044.159284ms

Con los candados MCS y CLHLock se intentó ejecutar el código durante 10 minutos, pero no lograron terminar, creemos que esto es debido al sleep, ya que pudimos notar que el contador si aumentaba a 1 pero no volvió a aumentar, tal vez poner un hilo a dormir afecta a la manera en que los hilos saben que ya es su turno en el candado, esto revisando la misma memoria del hilo (el caso de CLHLock) o la de otro hilo (el caso de MCS) (esto puede ver con la forma en la que se maneja la memoria, el cache y cómo esto interactúa con la máquina virtual de Java, pero no tenemos suficientes conocimientos para saber la razón exacta del porqué no funcionaron estos candados), ya que esos dos candado si cumplen con justicia y son FIFO.

3. ¿Cuál es la implementación más rápida del inciso 1? ¿A qué crees se debe?

Considerado las tablas de cada integrante para esta práctica, notamos una mejor implementación en los algoritmos **TAS** y **TTAS**. Son aquellos con los que a mayor cantidad de hilos a manejar, notamos un tiempo de ejecución mejor que los demás, aunque entre ellos varían de acuerdo a cada equipo de cómputo y sus capacidades.

- **TAS** es sencillo y directo; sin embargo, puede sufrir de congestión de memoria debido a la contención en la variable compartida que controla el bloqueo.
- **TTAS** mejora la eficiencia de TAS, ya que al verificar si el bloqueo ya está adquirido antes de intentar adquirirlo, se reduce el número de accesos a la variable compartida.

4. ¿Cuál es la implementación más rápida del inciso 2? ¿A qué crees se debe?

Como notamos en los valores de la tabla, los tiempos son demasiado cercanos entonces cualquier implementación podría ser útil, ya que los tiempos para cada algoritmo fueron muy similares. Esto se puede deber a que el *sleep* tiene mayor peso para cada `increment` (400000 ms en total de puro *sleep*), por lo que podríamos usar el mejor del inciso anterior, pero hay que notar hubo dos candados que no lograron terminar, entonces todos los demás hilos obtuvieron tiempo muy parecidos.

5. Dados tus resultados. ¿En qué escenario conviene (más rápido) más utilizar un Contador No bloqueante atómico que un Contador sin consistencia con un Spinlock o Lock?

Dados los resultados notamos que por lo general usar un contador sin consistencia con un candado es más rápido (por ejemplo el candado TTAS), pero hay que notar que las diferencias son muy pequeñas, entonces si no se realizan muchas tareas conviene usar cualquiera de las dos opciones, pero si son demasiadas tareas tal vez convenga más usar un contador sin consistencia con un candado.

6. ¿Cuáles son las implementaciones que satisfacen la propiedad de Justicia? (Por el momento ignora *ReentrantLock*). Argumenta cómo lo hacen.

- `TAS:`

  Esta implementación no cumple con justicia, debido a que como los hilos esperan en un `while`, no existe algo que nos garantice que tenga un funcionamiento FIFO, al usar el `while` dependerá del primer hilo que note el cambio.

- `TTAS:`

  Esta implementación no cumple con justicia (es similar a TAS), debido a que como los hilos esperan en dos `while`, entonces no existe algo que nos garantice que tenga un funcionamiento FIFO, al usar el `while` dependerá del primer hilo que note el cambio y esto pasa dos veces por los dos `while`.

- `Backoff:`

  Esta implementación no cumple con justicia, debido a que como cuando un hilo intenta obtener el candado, pero falla se le "aplica" un `backoff`, por lo tanto, puede pasar que otro hilo obtenga el candado mientras que el hilo que primero intentó obtenerlo sigue en el `backoff`, por lo tanto, no garantiza que su funcionamiento sea FIFO.

- `MCS:`

  Esta implementación si cumple con justicia, debido a que funciona de una manera que respeta FIFO, ya que los hilos se van "formando" a esperar su turno mediante el uso de una cola y solo revisando la memoria del hilo que va formando antes para ver si ya es su turno, haciendo

que se respete FIFO y, por lo tanto, respetando el orden en que los hilos piden él candando, generando justicia.

- `ALock:`

  Esta implementación si cumple con justicia (de igual manera que MCS), debido a que funciona de una manera que respeta FIFO, los hilos se van "formando" como si estuvieran en una cola, pero usando slots donde cada hilo tendrá una bandera y va a estar girando hasta que su bandera está en falso (lo que significa que ya es su turno) y cuando terminas de usar el candado pones la bandera del siguiente hilo como true, es como si se hiciera una cola, pero sin usar una cola como tal, es algo difícil de entender y explicar, pero sabemos que cumple FIFO, entonces los hilos respetan el orden en el piden el candado para obtener y así generar justicia.

- `CLHLock:`

  Esta implementación si cumple con justicia (de igual manera que MCS), debido a que funciona de una manera que respeta FIFO, ya que los hilos se van "formando" a esperar su turno mediante el uso de una cola y solo revisando la memoria del hilo mismo para ver si ya es su turno (si el hilo formado adelante ya termino), haciendo que se respete FIFO, entonces los hilos respetan el orden en el cual llegaron a pedir el candado y generan justicia.

- `Counter Atomic:`

  Esta implementación sí cumple con justicia, debido a que como es un contador atómico, entonces su operación de incremento es atómica y la realiza el primer hilo que la llama, no tenemos que bloquear o esperar a que otros hilos terminen, por lo tanto, si cumple con FIFO y justicia.

Referencias

[1] Clase de laboratorio del 3/10/24