Universidad Nacional Autónoma de México

Facultad de ciencias

Computación Concurrente 2024-1

Profesora:

Gilde Valeria Rodríguez Jiménez

Ayudantes:

Gibrán Aguilar Zuñiga

Luis Angel Leyva Castillo

Rogelio Alcantar Arenas

Equipo:

Concorriente

Integrantes:

DJLP

ADLG

TEORÍA

1. ¿Para qué sirve el método Yield? (yield())

   Respuesta:

   Su función principal es sugerir al programador de hilos que otro hilo con el mismo estado de prioridad pueda ejecutarse antes de que el hilo actual continúe su ejecución. Sin embargo, es importante comprender que yield() no garantiza que el hilo actual cederá su tiempo de CPU; simplemente sugiere que lo haga.

2. ¿Qué es un atributo atómico? (En la biblioteca atomic de Java)

   Respuesta:

   Un "atributo atómico" se refiere a una variable que se puede modificar atómicamente, lo que significa que las operaciones en esa variable se realizan de forma indivisible, sin ser interrumpidas por otras operaciones. Esto es importante en entornos de programación paralela donde varios subprocesos de ejecución pueden acceder y modificar simultáneamente una variable, ya que garantiza que las operaciones en la variable se realicen de forma coherente y sin conflictos.

3. Ventajas de usar atributos atómicos

   Respuesta:

   Atomicidad: Las operaciones en atributos atómicos se realizan de manera indivisible, lo que significa que son inmunes a condiciones de carrera.

   Sincronización eficiente: A diferencia de los bloques sincronizados tradicionales, que pueden ser costosos en términos de rendimiento, las operaciones en atributos atómicos son más eficientes.

   Rendimiento mejorado: En comparación con el uso de bloqueos explícitos o sincronización, el uso de atributos atómicos puede ofrecer un mejor rendimiento en escenarios de acceso concurrente.

   Mayor simplicidad de código: Las clases en `java.util.concurrent.atomic` proporcionan métodos simples y legibles para realizar operaciones comunes en atributos atómicos, como incremento, decremento y actualización condicional. Esto hace que el código sea más claro y menos propenso a errores en comparación con la implementación manual de sincronización.

   Evita problemas de bloqueo y deadlocks: El uso de atributos atómicos elimina la necesidad de adquirir y liberar bloqueos explícitos, lo que reduce la posibilidad de errores de programación relacionados con la sincronización, como los deadlocks.

   Facilita el diseño concurrente: Al utilizar atributos atómicos, es más fácil diseñar aplicaciones multihilo que sean seguras y eficientes, ya que se pueden compartir datos de manera segura entre hilos sin preocuparse por problemas de sincronización complejos.

4. Desventajas de usar atributos atómicos

Respuesta:

Complejidad conceptual: Para programadores menos experimentados, el uso de atributos atómicos puede introducir una mayor complejidad conceptual en el código debido a la necesidad de comprender cómo funcionan internamente y cuándo deben utilizarse.

Limitaciones en la complejidad de las operaciones: Los atributos atómicos son adecuados para operaciones simples como incremento, decremento y actualizaciones condicionales.

Overhead de memoria: Aunque las operaciones en atributos atómicos son eficientes en cuanto a rendimiento en la mayoría de los casos, todavía existe un cierto overhead de memoria asociado con la creación y mantenimiento de objetos de atributos atómicos, lo que puede ser relevante en aplicaciones con limitaciones de memoria.

Limitaciones en la escalabilidad: En situaciones de concurrencia extrema con un gran número de hilos compitiendo por acceso a atributos atómicos, puede haber limitaciones en la escalabilidad, ya que todos los hilos deben competir por los mismos recursos atómicos.

5. Da 2 ejemplos en donde se puedan aplicar este tipo de atributos (no pongan en resolver la tarea plox)

Respuesta:

Control de Acceso Concurrente a una Base de Datos en Memoria
Contador de Visitas en un Sitio Web

6. Algún uso que creas que se da en estructuras de datos concurrentes.

Respuesta:

Las colas concurrentes, Las listas concurrentes y los diccionarios concurrentes.

PRÁCTICA

Zeratun le marcó a su amix el Poncho para que le ayudará en la parte de estructuras de datos Concurrentes, este tiene los pseudocódigos pero se le olvido como programar en Java, ayudales a implementar el código explicando que hace cada línea.

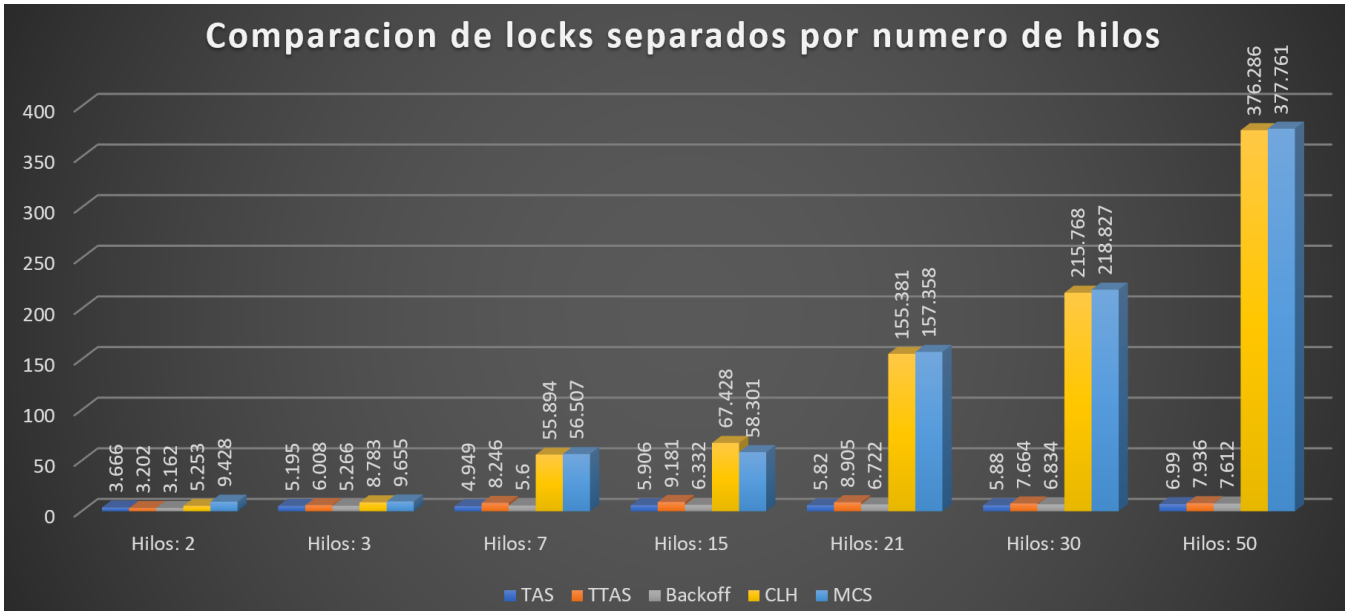
Los comentarios explicando que hace cada implementación de Lock se encuentran en sus respectivos archivos .java que vienen en el zip "Concorriente.zip".

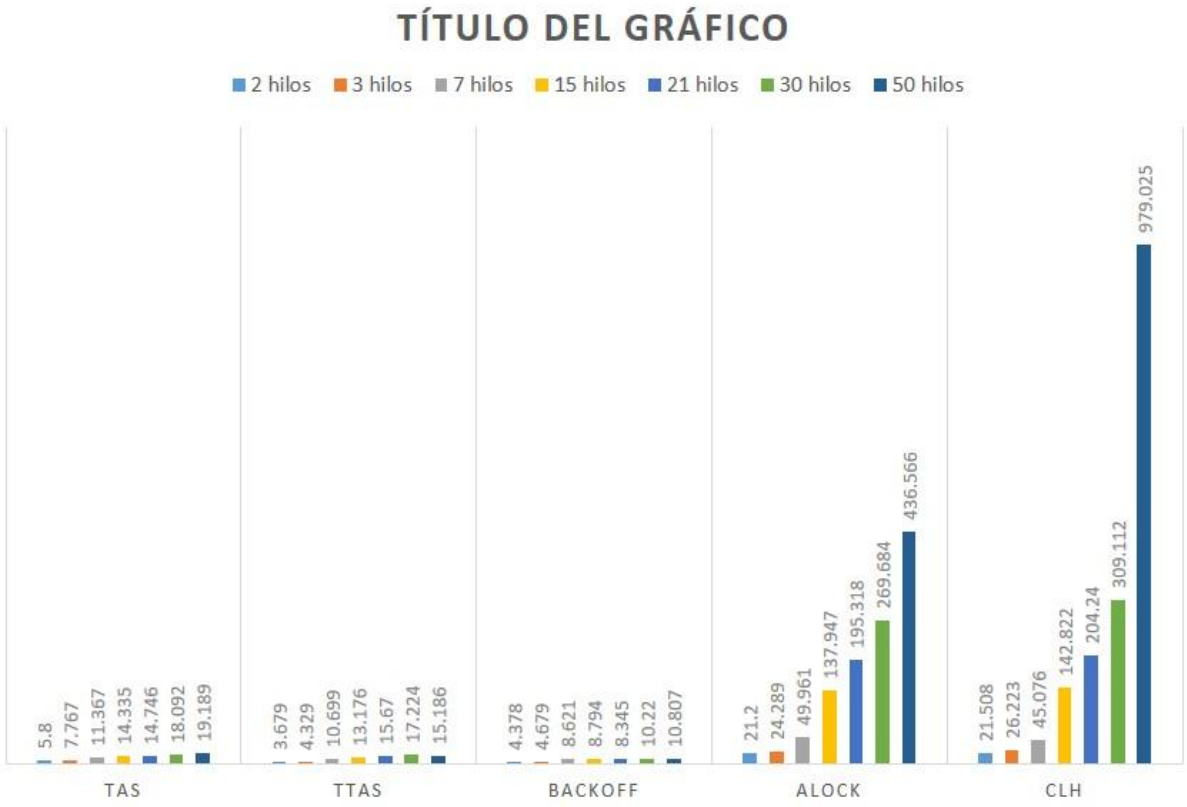
Pruebas de tiempo de ejecución en tablas usando distintos número de Hilos (2,3,7,15,21,30,50):

	TAS	TTAS	Backoff	CLH	MCS	Alock
Hilos: 2	3.666	3.202	3.162	9.428	7.583	5.253
Hilos: 3	5.195	6.008	5.266	9.655	9.476	8.783
Hilos: 7	4.949	8.246	5.6	56.507	55.114	55.894
Hilos: 15	5.906	9.181	6.332	58.301	111.191	67.428
Hilos: 21	5.82	8.905	6.722	157.358	157.25	155.381
Hilos: 30	5.88	7.664	6.834	218.827	217.862	215.768
Hilos: 50	6.99	7.936	7.612	377.761	376.459	376.286

	2 hilos	3 hilos	7 hilos	15 hilos	21 hilos	30 hilos	50 hilos
TAS	5.8	7.767	11.367	14.335	14.746	18.092	19.189
TTAS	3.679	4.329	10.699	13.176	15.67	17.224	15.186
BackOff	4.378	4.679	8.621	8.794	8.345	10.22	10.807
aLock	21.2	24.289	49.961	137.947	195.318	269.684	436.566
CLH	21.508	26.223	45.076	142.822	204.24	309.112	979.025

Gráficas:





Nota: En ocasiones tanto en la M1 y M2 se quedaban procesando y procesando, incluso llegamos a dejar el test haciéndose por 2 horas y no daba un resultado. Aquí en la M2 no se logró generar el test para el caso de mcsLock por lo que dichos valores no fueron agregados en la gráfica y tabla referentes a la máquina M2.

Specs de la máquina M1 donde se generaron la primer tabla y gráfica:
• CPU: Ryzen 3 3200G a 3.60GHz.
• RAM: 8 GB.
• Sistema: 64 bits, procesador x64.

Specs de la máquina M2 donde se generaron la primer tabla y gráfica:
• CPU: Intel(R) Core(TM) i7-8550U a 1.99GHz
• RAM: 8 GB
• Sistema operativo de 64 bits, procesador x64

Una vez hecho esto, explica el porqué se obtienen esos tiempos, compara el resultado con el de tus compañeros, y si la diferencia es muy grande explica el porqué.

Hay una diferencia relativamente considerable entre los primeros resultados obtenidos y los segundos resultados. Por ejemplo al correr TASLock en la M1 con 2 hilos tardó 3.666s y con 50 tardó 6.99s mientras que en la M2 con 2 hilos tardó 5.8s y con 50 tardó 19.189s. Al correr CLHLock en la M1 con 2 hilos tardó 9.428s y con 50 tardó 377.761s mientras que en la M2 con 2 hilos tardó 21.508s y con 50 tardó 979.025s y mas o menos esa es la diferencia en segundos que encontramos al correr los locks en la M1 y M2.

Creemos que las diferencias pueden ser atribuidas al procesador y la RAM de las pc's ya que como vimos en prácticas anteriores la cantidad de núcleos y velocidad del reloj del procesador son factores a considerar a la hora de correr muchos o pocos hilos. Recordando que si utilizamos muchos hilos es mejor una mayor cantidad de núcleos por otro lado si tenemos corriendo pocos hilos es mejor una mayor velocidad de reloj para realizar alguna tarea en específico.

Anota lo aprendido durante la práctica así como las impresiones recibidas al realizarla.

En esta práctica aprendimos el concepto de los spin o giros que dan los hilos para esperar hasta que un lock sea liberado o adquirido, lo que nos ofrecen las respectivas clases de enteros y booleanos atomicos, además de las variaciones interesantes de Locks y un panorama general del potencial que tienen nuestras pc's y cuales son sus limitantes ante ciertos escenarios relacionados con hilos y procesos.

Referencias

- [HS08] Maurice Herlihy and Nir Shavit. The Art of Multiprocessor Programming. Morgan Kauf-mann Publishers Inc., San Francisco, CA, USA, 2008.