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

Práctica 1

1.- Introducción y jugando con Hilos

Modifica el código de Hilos.java, de tal manera que se haga lo siguiente:

- Genera una lista de Hilos.
- Agrega 10 hilos a esta mediante un for e inicializandolos.
- Hazle join a cada hilo con un for o foreach
- Agrega una captura de pantalla a tu reporte de cómo quedó tu código al final.

```java
public class Hilos implements Runnable
{
    public static void main(String[] args) throws InterruptedException
    {
        Hilos h = new Hilos();//Se crea una instancia de la clase

        List<Thread> list = new ArrayList<>();

        for (int i=1; i<11; i++)
        {
            Thread t = new Thread(h,"Hilo "+i);
            list.add(t);
            t.start();
        }

        for (Thread t: list)
        {t.join();}

    }

    @Override
    public void run()
    { //Sobrescribimos el metodo run
        int a = 10;
        int b = 12;
        long ID = (Thread.currentThread().getId());
        if(ID == 1)
            System.out.println("Soy el hilo 1");
        else
            System.out.println("Hola soy el: "+ Thread.currentThread().getName());//Pedimos el nombre del hilo pidiendo primero que se seleccione
    }

}
```

Ejecuciones del código

```
C:\Users\ADLG\Concurr  C:\Users\ADLG\Concurr
Hola soy el: Hilo 1    Hola soy el: Hilo 2
Hola soy el: Hilo 3    Hola soy el: Hilo 4
Hola soy el: Hilo 4    Hola soy el: Hilo 1
Hola soy el: Hilo 2    Hola soy el: Hilo 6
Hola soy el: Hilo 5    Hola soy el: Hilo 5
Hola soy el: Hilo 6    Hola soy el: Hilo 3
Hola soy el: Hilo 7    Hola soy el: Hilo 9
Hola soy el: Hilo 8    Hola soy el: Hilo 8
Hola soy el: Hilo 10   Hola soy el: Hilo 7
Hola soy el: Hilo 9    Hola soy el: Hilo 10

C:\Users\ADLG\Concurr  C:\Users\ADLG\Concurr
Hola soy el: Hilo 4    Hola soy el: Hilo 1
Hola soy el: Hilo 5    Hola soy el: Hilo 2
Hola soy el: Hilo 2    Hola soy el: Hilo 4
Hola soy el: Hilo 1    Hola soy el: Hilo 3
Hola soy el: Hilo 3    Hola soy el: Hilo 6
Hola soy el: Hilo 7    Hola soy el: Hilo 5
Hola soy el: Hilo 6    Hola soy el: Hilo 7
Hola soy el: Hilo 10   Hola soy el: Hilo 8
Hola soy el: Hilo 9    Hola soy el: Hilo 10
Hola soy el: Hilo 8    Hola soy el: Hilo 9
```

El archivo con este código se encuentra en "Hilos.java".

2.- Multiplicación de Matrices Secuencial

Realiza un programa que realice la multiplicación de matrices.
Considera que serán matrices de tamaño n x n
De entrada serán 2 matrices de tamaño n x n y el retorno será la matriz resultante.

El programa que resuelve este punto se encuentra en "MatrixMultiplicacionSecuencial.java".

3.- Multiplicación de Matrices Concurrente

Realiza un programa que realice la multiplicación de matrices con hilos.
Considera que serán matrices de tamaño n x n
De entrada serán 2 matrices de tamaño n x n y el retorno será la matriz resultante.

El programa que resuelve este punto se encuentra en "MatricesConcurrente.java".

4.- Contador Compartido: Synchronized y Durmiendo Hilos

En el archivo Contador.java, nos encontramos con lo que viene siendo un contador, donde varios hilos acceden a él para ir actualizando su valor, en esta actividad el objetivo es que el valor del contador sea 20000 y 30000.

Lo primero que deben hacer es ejecutar el programa varias veces y ver que valores da, **¿Qué puedes decir de este resultado?, ¿Por qué lo da?.**

Da valores variados, por lo general, valores mayores a 10,000. El valor que regresa el programa se debe a que 2 hilos están llevando a cabo un ciclo for, el cual aumenta en 1 el valor que tienen.
Tanto el hilo 1 como el hilo 2 inician uno seguido del otro, después se hace un join de cada hilo respectivamente, sin embargo, como los dos hilos inician casi al mismo tiempo, el valor de retorno se modifica en cada iteración y es por ello que el valor de retorno supera el valor final establecido, que es: 10,000.

(En resumen es como si los dos hilos quisieran llegar a 10,000, cada hilo suma sin considerar lo que otro hilo ya sumó).

Utiliza lo que viene siendo Thread.sleep(<Tiempo de mimir>) para dormir el hilo que selecciones, esto con el fin de realentizar al Hilo, **¿Por qué crees que haríamos esto?.**

Con el fin de tratar de sincronizar hilos, hacer que un hilo realice una acción antes que otro para que no haya influencias en los valores o recursos con los que trabajen.

Finalmente, investiga para qué funciona synchronized (Pues lo utilizaremos durante el curso), descomenta el bloque de código y vuelvelo a ejecutar. Dada tu investigación, **¿Qué sucede cuando el bloque tiene synchronized?.**

Synchronized
Ayuda a reducir la concurrencia cuando se comparte el mismo recurso entre múltiples subprocesos o procesos. Bloquea los recursos en un subproceso para que ningún otro subproceso pueda acceder a él a la vez.

Cuando se tiene synchronized lo que ocurre es parecido a lo que sucedería si durmiéramos a algún hilo. Ahora el valor que regresa el programa es: 20,000.

Descomenta el siguiente bloque de código (comentando ahora este) y ejecútalo nuevamente, **¿Qué diferencia ves? ¿Por qué sucede esto?.**

Ocurre relativamente lo mismo pero en este caso se sincroniza (por así decirlo) ese bloque de código, por lo que no más de un hilo podrá acceder al código dentro de dicho bloque. Y ocurre precisamente por la palabra reservada sinchronized.
Genera algún ejemplo que se te venga a la mente donde se use sinchronized.

Al final, lo que entregaras de esta sección:
• El contador obteniendo 20000 (o aproximado) usando Thread.sleep().
• El contador obteniendo 30000 (o aproximado) usando Thread.sleep().

El programa “Contador.java” resuelve estos puntos.

• El ejemplo usando synchronized que crearon.

El programa “SynEj.java.” resuelve este punto.

TEORÍA

1. ¿Por qué se pone un InterruptedException en el método main?
La excepción InterruptedException se utiliza para manejar situaciones en las que un hilo está esperando (mediante sleep(), wait() o join()) y otro hilo lo interrumpe. Esto puede suceder por varias razones, como cuando se desea detener un hilo en ejecución o cuando se produce una señal para que el hilo despierte de su estado de espera.

2. ¿Para qué sirve el método Join?
Para poner una especie de barrera y mantener un orden en la ejecución de hilos. Esto es útil cuando tienes varios hilos y necesitas coordinar su ejecución de manera que un hilo no avance demasiado rápido y cause problemas de sincronización.

3. ¿Qué pasa si no le hacemos Join a los hilos?
Podría ocurrir que un hilo A se siguiera procesando, y si otro hilo, un hilo B necesitará la información del primer hilo que se sigue procesando, el hilo B tendría que esperar hasta que el hilo A termine de procesarse. Es decir, corremos el riesgo de tener problemas de sincronización con los hilos.

4. Explica de manera concisa cómo usar Hilos extendiendo la clase Thread.
1. Primero, en la clase hay que extender de Thread.
2. Hacer un @Override en el método run para definir el comportamiento de los hilos.
3. Y claro, definir el método main en el que haremos instancia de la clase, en este caso (en la práctica) fue la clase Hilos, además de añadir la excepción “InterruptedException”.
4. Ahí en el main podremos definir los hilos con los que vayamos a trabajar y cada hilo podrá hacer uso de los métodos start y join.

5. ¿Cuáles son las ventajas en implementar Runnable contra extender de Thread?
Que al implementar de Runnable se pueden implementar las interfaces que se necesiten además de poder extender de alguna otra clase (brinda una mayor flexibilidad en el diseño del programa, mantenimiento más sencillo,).

6. ¿Se puede predecir el orden en el que se imprimirá el mensaje de la clase Hilos?
Se puede predecir (por una forma de decirlo) manipulando qué hilos se ejecutan primero y cuáles después, pero predecir como tal, no.

7. En el archivo Hilos2.java, ¿Qué pasa si sacamos la instancia de la clase "h" de t1, es decir, poner h por ejemplo, antes de declarar t1?
Al parecer nada, sigue imprimiendo lo mismo instanciando antes de t1 o después en Runnable() ya que con ello se puede usar el método imprimeAlgo de la clase Hilos2.

8. Explica cómo podríamos tener comportamientos diferentes implementando Runnable.
Se podrían tener comportamientos diferentes al implementar o definir lo que harán los hilos en el método run.

9. Para la sección 3 realiza lo siguiente:
a. Se incluyen una carpeta con 3 tipos de matrices, 10x10, 100x100 y 1000x1000, para cada prueba ejecutala y obtene el tiempo obtenido, tanto su versión secuencial así como su versión concurrente, ¿Ves algún cambio significativo? ¿Mejoró el tiempo o empeoró?

***Prueba 10x10:**
Tiempo Concurrente(10 hilos): 25 Milisegundos.
Tiempo Secuencial: 39 Milisegundos.
El tiempo como podemos ver en esta prueba fue muy insignificante, pero aunque sea muy poca la diferencia pues si hubo una mejora de la prueba concurrente en comparación a la secuencial.

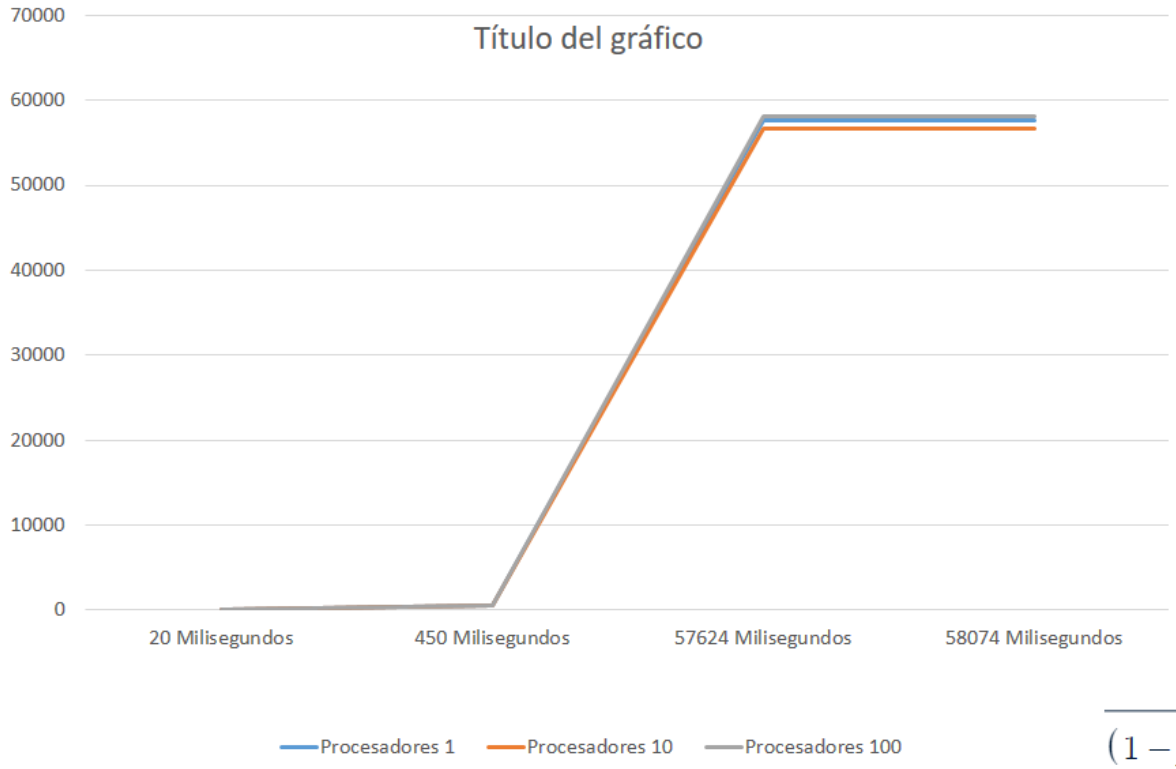
Prueba 100x100
Tiempo Concurrente(100 hilos): 479 Milisegundos.
Tiempo Secuencial: 578 Milisegundos.
El tiempo una vez más es muy poca la diferencia, pero en esta ocasión ya se ve más notoria la mejora que tiene la prueba concurrente contra la secuencial mejorando aún más el tiempo de ejecución.

Prueba 1000x1000
Tiempo Concurrente(1000 hilos): 58074 Milisegundos.
Tiempo Secuencial: 57624 Milisegundos.

En esta prueba la prueba secuencial salió victoriosa realizando un menor tiempo que la prueba concurrente por lo que el tiempo empeoró respecto a las otras dos pruebas.

Los programas que utilizamos para estos puntos son:
“MatrixMultiplicacionSecuencialLeer.java”
“MatrixMultiplicacionSecuencial.java”
“MatricesConcurrente.java”

b. Genera una gráfica con tus resultados obtenidos, usando 1,10 y 100 hilos.



$$\frac{1}{(1-p)+\left(\frac{p}{n}\right)}$$

c. Finalmente pondremos a prueba la teoría , pues realizaremos la siguiente tabla con la info siguiente:
d. En la columna de Aceleración Teórica, utilizaremos la Ley de Amdahl y la compararemos con la Aceleración obtenida y responde las siguientes preguntas.

# Hilos	Aceleración Teórica	Aceleración Obtenida	% Código en paralelo
1	1.098	1	32.81%
10	1.1098	1.4044	32.81%
100	1.1176	1.4637	32.81%

e. ¿Se cumple siempre dicha ley?
No siempre se cumple que hay mejoras.

f. ¿En qué casos no se cumple?
Cuando el número de tareas crece y el número de procesadores crece a la misma cantidad de tareas la ley nos dice que hay aunque sea una mínima mejora pero en realidad como vimos en la práctica hay ocasiones donde ni siquiera hay una mejora. Sin mencionar la parte de las optimizaciones de código o la carga de trabajo de cada hilo.

g. ¿Por qué crees que pasa esto?
Porque los procesadores al tener todos una tarea quizá el tiempo que toman unos en hacer la tarea sea mucho más lento que el que les toma a otros y eso retrasa mucho el resultado final ya que son muchos hilos y pues este factor puede aumentar entre más hilos hay.

h. ¿Cuál sería el nivel máximo de mejora?
Justo donde se encuentra el punto de equilibrio donde añadir más hilos al mismo trabajo no logra aportar mayor rendimiento.

i. Para obtener una mejora aún mayor, en términos de Hardware, ¿Qué es mejor?, tener muchos más núcleos con hilos o tener mayor frecuencia de reloj. Justifica.
En teoría para lo que estamos haciendo debería ser mejor tener más núcleos ya que estamos trabajando con varios hilos y estamos tratando con el tema de la concurrencia y el cómputo paralelo.

10. Responde TODAS las preguntas de la sección 4 de la parte práctica (7 preguntas). LISTO.

11. Escribe lo aprendido con esta práctica, así como descubrimientos que obtuviste mientras la realizaron.

Aprendimos cómo usar Synchronized, sleep(), join(), usar hilos extendiendo de Runnable, multiplicación de matrices con hilos y un poco más sobre la ley de Amdahl.

Referencias

InterruptedException (Java Platform SE 7). (2020, 24 junio). https://docs.oracle.com/javase%2F7%2Fdocs%2Fapi%2F%2F/java/lang/InterruptedException.html

3.5.1 Amdahl's Law (Sun Studio 12: C User's Guide). (s. f.). https://docs.oracle.com/cd/E19205-01/819-5265/bjaem/index.html#:~:text=Amdahls%20Law%20can%20be%20misleading,The%20following%20example%20demonstrates%20this.

GeeksforGeeks. (2023). Computer Organization Amdahl s Law and its proof. GeeksforGeeks. https://www.geeksforgeeks.org/computer-organization-amdahls-law-and-its-proof/