Universidad Nacional
Autónoma de México
Facultad de Ciencias

Cómputo Concurrente

2024-1

Práctica 1

Equipo: Dinamitabb

Miranda Mijangos, Mónica 317060524
Sánchez Velasco, Eduardo Leonel 420004035

31 de agosto de 2023

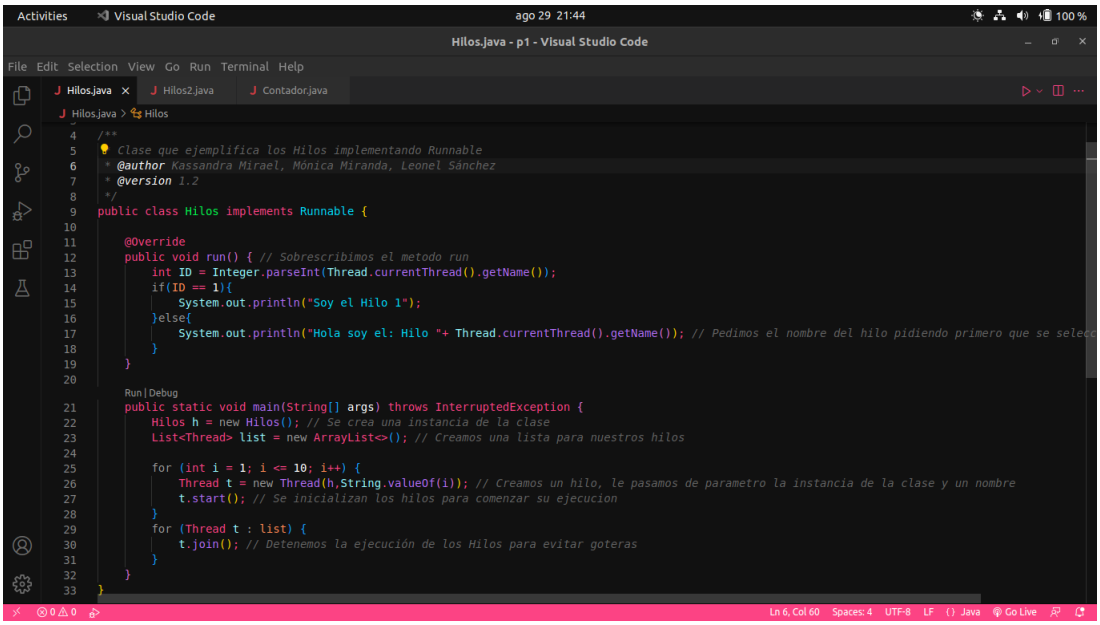

Figura 1: Código final: Introducción y jugando con Hilos

1. ¿Por qué se pone un *InterruptedException* en el método *main*? Porque los *join()* pueden arrojar esta excepción si el hilo que está esperando es interrumpido por otro.

2. ¿Para que sirve el método *Join*? Pone una barrera a los hilos, que espera hasta que terminen los demás para continuar con su ejecución.

3. ¿Qué pasa si no le hacemos *Join* a los hilos? Puede haber goteras, ya que algunos hilos se pueden quedar olvidados, sin terminar, en algún proceso aunque el programa haya finalizado.

4. Explica de manera consisa como usar Hilos extendiendo la clase *Thread*. Definímos una nueva clase que hereda de *Thread* y sobrescribimos el método *run()* que se ejecutará en cada instancia o hilo cuando se inicien con el método *start()*.

5. ¿Cuáles son las ventajas en implementar *Runnable* contra extender de *Thread*? Dado que en Java no tenemos extensión mutliple, a diferencia de extender sólo de la clase Thread, al implementar Runnable podremos implementar las interfaces que queramos, además de extender de alguna otra clase que necesitemos y reutilizan código fácilmente.

6. ¿Se puede predecir el orden en el que se imprimira el mensaje de la clase *Hilos*? No, dependerá del calendarizador y del orden en que se ejecuten.

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 declar *t1*? Permite que la instancia de la clase *"h"* sea accesible dentro de Runnable.

8. Explica como podriamos tener comportamientos diferentes implementando *Runnable*. Podríamos obtener el identificador del hilo o los hilos que queramos diferenciar para capturarlos y aplicarles un comportamiento diferente.

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 asi como su versión concurrente, ¿Ves algun cambio significativo? ¿Mejoro el tiempo o empeoro?
   R: Mejora bastante, casi el doble, pero solamente en el caso de matriz de 1000 x 1000. En los casos pequeños el secuencial es mas rápido.

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

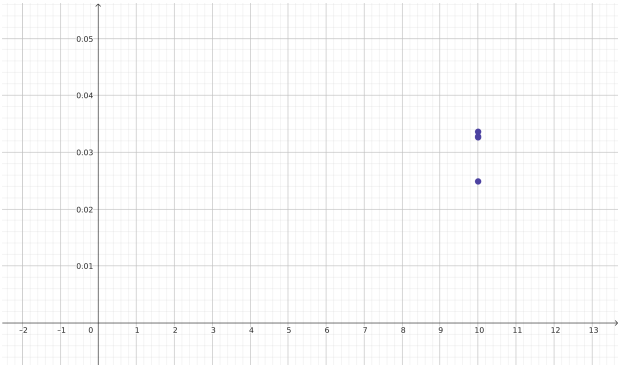

Figura 2: Resultados con la matriz de 10x10

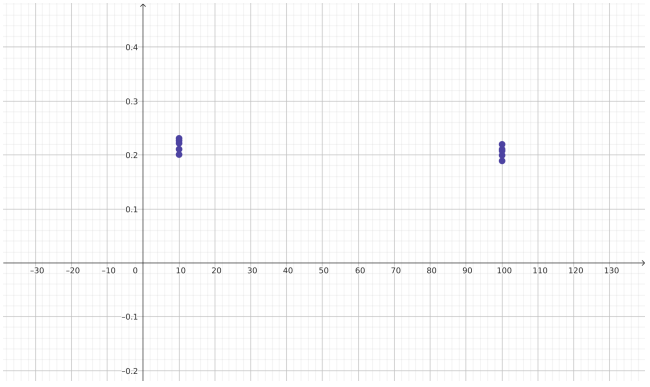

Figura 3: Resultados con la matriz de 100x100

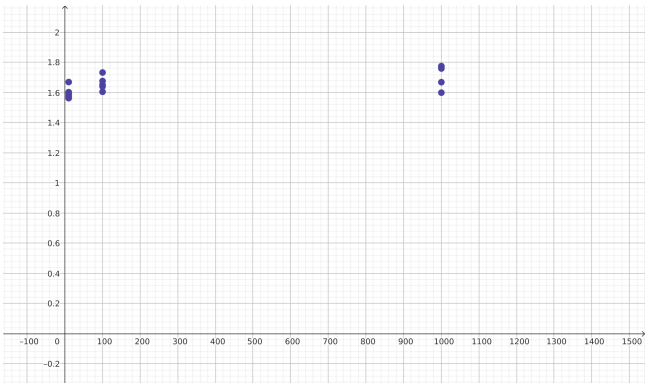

Figura 4: Resultados con la matriz de 1000x1000

Comentario: como podemos observar en los casos con matrices pequeñas el concurrente es menos eficiente incluso que el secuencial, pero al trabajar con matrices mas grandes el secuencial es mucho mas efectivo.

c) Finalmente pondremos a prueba la teoría , pues realizaremos la siguiente tabla con la info siguiente:

# Hilos	Aceleración Teórica	Aceleración Obtenida	% Código en paralelo
10 en 10x10	1.1794	0.035	0.169
10 en 100x100	1.1794	-	0.169
10 en 1000x1000	1.1794	-	0.169
100 en 10x10	1.2009	0.24	0.169
100 en 100x100	1.2009	0.22	0.169
100 en 1000x1000	1.2009	-	0.169
1000 en 10x10	1.203	1.6	0.169
1000 en 100x100	1.203	1.7	0.169
1000 en 1000x1000	1.203	1.67	0.169

d) En la columna de Aceleración Teórica, utilizaremos la Ley de Amdahl y la compareremos con la Aceleración obtenida y responde la siguientes preguntas.

e) ¿Se cumple siempre dicha ley?

f) ¿En que casos no se cumple?

g) ¿Por qué crees que pasa esto?

R (e,f,g): Como comentamos antes, en las matrices de tamaño pequeño no cumple con esta ley, ya que al ser pequeños, hacer los hilos tarda mas que hacerlo en secuencial, con los tamaños grandes es incluso mejor que la aceleración teórico, nos imaginamos que con tamaños incluso mas grandes esta aceleración se acercaría al teórico.

h) ¿Cuál seria el nivel máximo de mejora? **R:** El nivel máximo suponemos que seria muy cercano al teórico, entre mas grande crezca la matriz mas se acerca a este.

i) Para obtener una mejora aun mayor, en términos de Hardware, ¿Qué es mejor?, tener muchos mas núcleos con hilos o tener mayor frecuencia de reloj. Justifica.
R: En general, depende que es lo que se necesita, si estamos el proceso que vamos a realizar tiene poco porcentaje de código en paralelo, nos conviene mas tener mayor frecuencia de reloj, y esto fácilmente lo podemos ver con la ley de ahmdal, acercando a P (el porcentaje en paralelo) a 1, y mayor numero de hilo, la aceleración es muchísimo. En este caso en particular nos conviene mayor frecuencia de reloj por lo antes mencionado.

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

    1) Ejecutar *Contador.java* varias veces y ver que valores da, ¿Qué puedes decir de este resultado? El resultado varía.
    2) ¿Por qué lo da? Por las condiciones de carrera.
    3) Utiliza *Thread.sleep* para dormir el hilo que selecciones, con el fin de realentizar al Hilo, ¿Por qué crees que hariamos esto? Para facilitar la observación de problemas.
    4) Finalmente, investiga para que funciona *synchronized*.
    Nos asegura que solo un hilo puede ejecutar ese método o bloque a la vez. Esto evita que varios hilos accedan y modifiquen datos compartidos, lo que podría llevar a resultados inconsistentes.
    5) Descomenta el bloque de codigo y ejecutalo. ¿Qué sucede cuando el bloque tiene *synchronized*?. Tenemos menos variaciones en el resultado del contador.
    6) Descomenta el siguiente bloque de codigo y ejecutalo nuevamente, ¿Qué diferencia ves? El valor tiende a ser el mismo.
    7) ¿Por qué sucede esto? El *"límite"* del lock. Ya que en el método, todos los subprocesos tendrán que esperar mientras el método lo ejecuta un subproceso. Mientras que en el bloque "synchronized", solo se aplicará al objeto entre paréntesis.

11. Escribe lo aprendido con esta practica, asi como descubrimientos que obteniste mientras la realizaron.
    En esta práctica, aprendimos sobre hilos, las condiciones de carrera que estos pueden tener y cómo pueden afectar los resultados de un programa. Logramos simular concurrencia e identificar problemas potenciales, para así hacer uso de *synchronized* y evitarlos, logrando la sincronización segura de hilos.

1 Referencias

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

Leyva Castillo, L. A. (2023). Parte 1 [Archivo de video]. Ayudantía Lab (24/08/23). Google Drive. URL: `https://drive.google.com/file/d/1Dd__2dJyiHN6C1Op1qAFA-Uu2MPGeImg/view?usp=drive_link` (Consultado el: 29 Agosto 2023).

Synchronized methods. Synchronized Methods (The JavaTM Tutorials & Essential Java Classes & Concurrency). Disponible en: `https://docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html` (Consultado el: 30 Agosto 2023).

Intrinsic locks and synchronization. Intrinsic Locks and Synchronization (The JavaTM Tutorials & Essential Java Classes & Concurrency). Disponible en: `https://docs.oracle.com/javase/tutorial/essential/concurrency/locksync.html` (Consultado el: 31 August 2023).