Taller de Sistemas Operativos

Implementación y Análisis de Threads
"Taller 02"

Taller de Sistemas Operativos
Escuela de Ingeniería Informática
Cristian Valencia Avila
cristian.valenciaa@alumnos.uv.cl

***Resumen**: Se implementa un programa de dos módulos cuyo propósito es llenar un arreglo de números enteros para luego sumarlos, con la condición de que ambas tareas sean realizadas de forma paralela por medio de la implementación de hilos(Threads-safe).*

1. Introducción

Ejecutar un programa que realice de forma paralela ciertas tareas, se puede lograr implementando un método llamado Threads, los cuales consisten en una unidad básica de ejecución de un SO. Cabe mencionar que cualquier programa el cual se ejecute consta de un thread mínimo.

En el Sistema Operativo los hilos son una secuencia de tareas que ocurren de manera simultánea y concurrente, para ser más preciso el SO reparte la CPU entre los threads que se ejecutan en todo momento, ya que un programa puede poseer varios threads, de manera que simplemente se "adueña" de la CPU y ejecuta el siguiente thread. Cada vez que ocurre esto el SO se encarga de guardar las características (registros y pila) que posee cada thread para que al momento de conmutar el mismo thread pueda restaurar su contexto inicial, esto nos da paso a llevar a cabo distintas funciones simultáneamente.

Se debe tener en cuenta que cada thread trabaja como si tuviese un microprocesador para el solo, es decir, se ejecuta de forma absolutamente independiente. Los threads que comparten recursos se conocen como proceso, esta característica posibilita que cualquiera de los thread pueda modificar dichos recursos, para lo cual se necesita algún tipo de sistema de sincronización. Estos procesos no terminan su ejecución mientras que uno de sus threads esté activo.

De manera general, un thread se define como una tarea que puede ser ejecutada al mismo momento que otra tarea. En la figura 1 se puede apreciar un thread en ejecución.

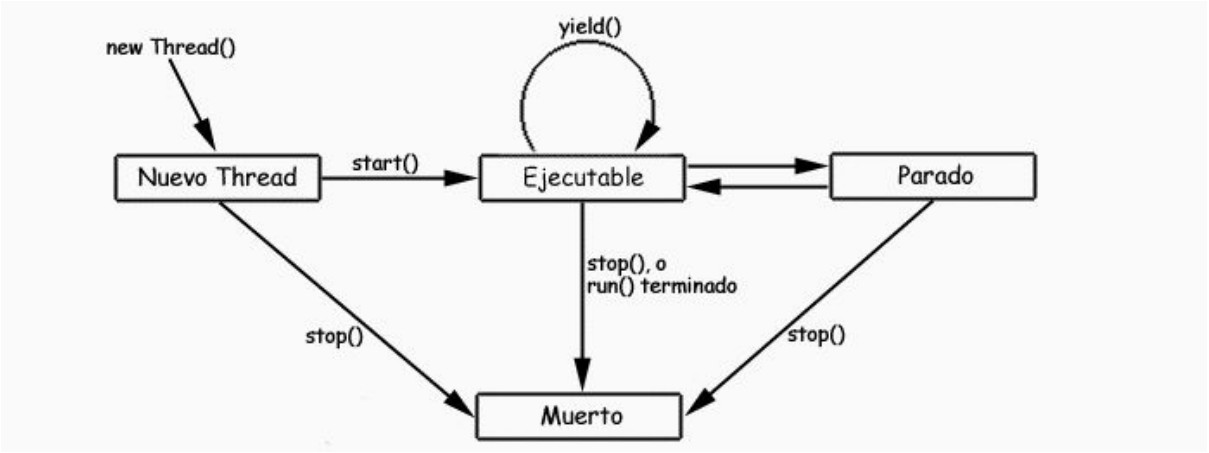


Figura 1: Esquema de un thread en ejecución.

Para el desarrollo de este taller se requiere utilizar el lenguaje de programación C++ 2014 o superior, para la creación de los thread, permitiendo efectuar de manera simultánea, en este caso, de dos tareas explicadas más detalladamente en la siguiente sección.

2. Descripción del Problema

2.1 Planteamiento del Problema

Se solicita la implementación de un programa, el cual consta de dos módulos principales. Uno se encarga de llenar un arreglo y de forma paralela, el otro módulo se encarga de sumar el contenido del arreglo mencionado. En la Figura 2 se puede ver de manera general el proceso por el cual pasan ambos módulos.

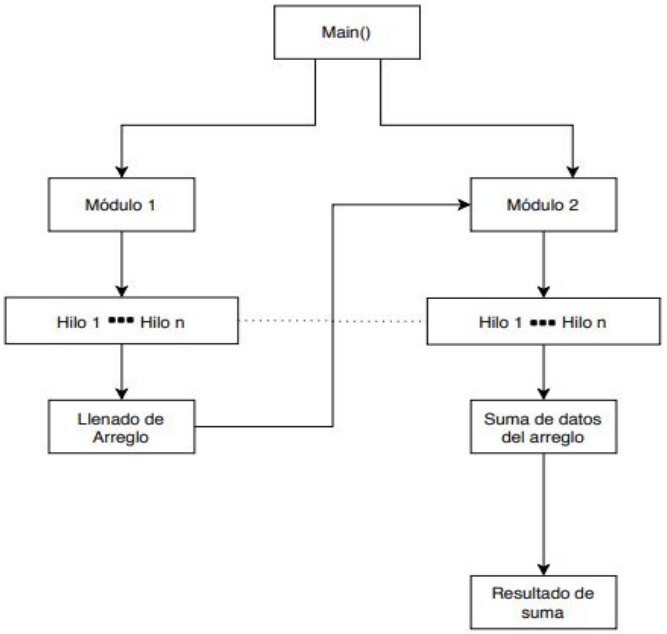
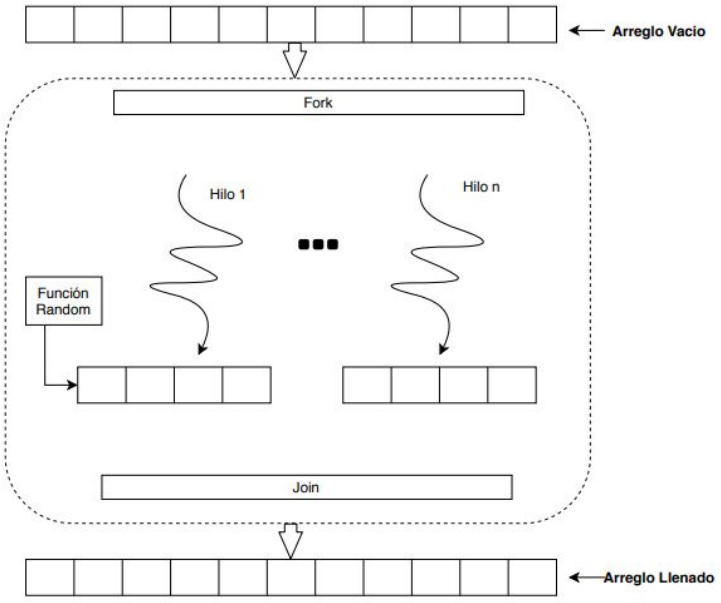


Figura 2: Vista general del problema

2.1.1 Módulo de llenado de arreglos

Este módulo se encarga de llenar los arreglos vacíos creados por el Main(), a través de alguna función que entregue números de forma randómica siendo del tipo `uint32_t`, donde cada hilo guardará los valores entregados por dicha función hasta el momento de realizar el llenado(Ver figura 3).
Como recomendación para generar los números aleatorios se requiere usar funciones que sean *thread safe* para observar una mejora de desempeño en el programa.

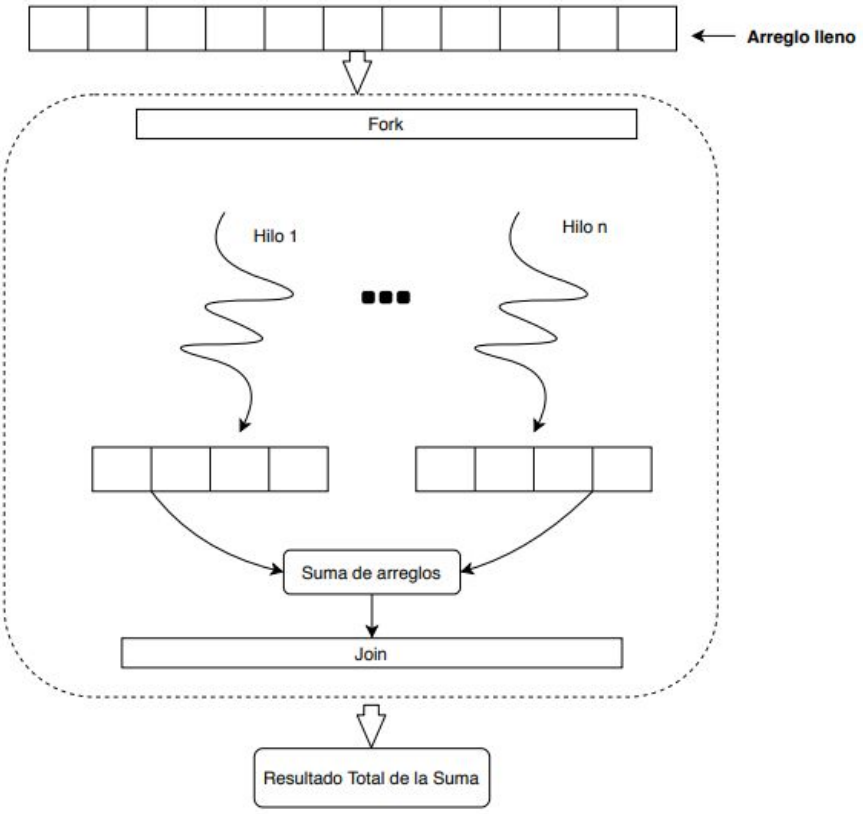
Figura 3: Esquema llenado de arreglo



2.1.2 Módulo de suma de arreglos

En este módulo se encarga de sumar los valores que contiene el arreglo llenado anteriormente, para el cual los hilos creados en el primer módulo realizan una división del arreglo para proseguir a realizar la suma de valores contenidos en el arreglo(ver Figura 4).

Figura 4: Esquema de Suma del arreglo.



2.2.1 Forma de uso

Para la ejecución del código se realiza de la siguiente manera:

./sumArray -N <nro> -t <nro> -l <nro> -L <nro> [-h]

Los parámetros utilizados se pueden ver en la tabla 1.

-N	Tamaño del arreglo.
-t	Número de Threads.
-l	Límite inferior rango aleatorio
-L	Lìmite superior rango aleatorio.
-h	Muestra la ayuda de uso.

Tabla 1: Descripción de los parámetros.

2.1.2 Ejemplo de la forma de uso

1) Creación de un arreglo de 100 posiciones, con 6 threads. Los números enteros aleatorios están en el rango [20,40], por lo que su línea de comando sería:

   ```
   ./ sumArray -N 100 -t 6 -l 20 -L 40
   ```

2) Muestra de la ayuda:

- ./sumArray -h
- ./sumArray

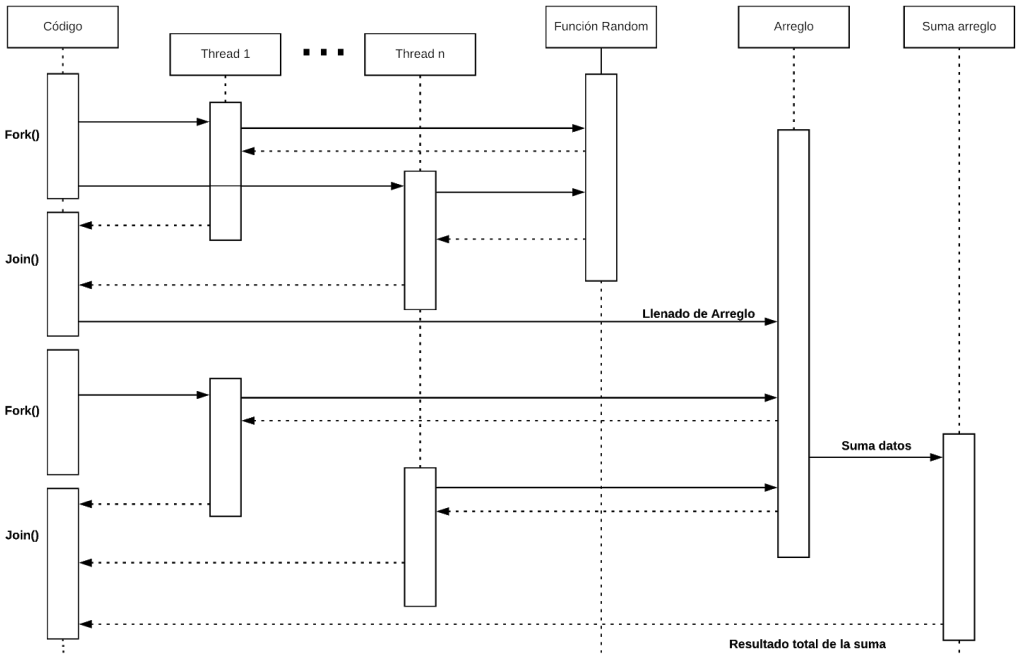
2.2 Tareas requeridas

Se requiere visualizar el comportamiento del tiempo de ejecución que toma cada módulo, esto dependiendo del tamaño de cada problema y de la cantidad de threads que se utilizaron, para esto se deben realizar pruebas de desempeño que generen cierto datos que nos permita esta visualización.

3. Diseño de Solución

El modelado de la solución se puede ver representado en la Figura 5, a través de un diagrama de secuencia se expresa el modelado de los procesos de ambos módulos, llenado y suma de los datos del arreglo.

Figura 5: Diagrama de Secuencia del proceso de los hilos.



4. Pruebas de Desempeño

Para comprender el comportamiento del código implementado y con ello visualizar el desempeño de este mismo, refiriéndose al tiempo en ejecución que lleva la realización de cada módulo. En primer lugar se evalúa el algoritmo implementado bajo distintas características para un arreglo de tamaño N, para esto se realiza una comparativa de los módulos con dos tipos de llenado: *Secuencial y Paralelo*, calculando su tiempo de ejecución y con ello su desempeño.

4.1 Comparativa de llenado de forma secuencial y paralela

```
cristian@DESKTOP-VS0OV9G:/mnt/c/Users/Cristian/Desktop/U2-master/taller02$ ./sumArray -N 10000000 -t 1 -l 1 -L 1000
=======LLenado y Suma de numeros de un Arreglo=======
================================================
Tiempo de LLenado secuencial: 347[ms]
Tiempo de LLenado con threads: 341.889[ms]
SpeedUp de suma: 1.01495
```

Figura 6: Muestra de llenado secuencial y paralelo con un thread..

En la Figura 6 se puede apreciar que el SpeedUp tiene este valor, ya que el algoritmo en su sección de llenado secuencial y con threads es prácticamente igual.

Algo cabe destacar que a medida que el número de hilos aumenta el SpeedUp aumentará de manera proporcional, ya que la función al ser de tipo *thread-safe*, los thread pueden acceder de manera simultánea, implicando este aumento de desempeño. En la Figura 7 y 8 se aprecian ejemplos con distintas cantidades de hilos.

```
cristian@DESKTOP-VS0OV9G:/mnt/c/Users/Cristian/Desktop/U2-master/taller02$ ./sumArray -N 143000000 -t 10 -l 1 -L 12100
=======LLenado y Suma de numeros de un Arreglo=======
================================================
Tiempo de LLenado secuencial: 5019[ms]
Tiempo de LLenado con threads: 1963.12[ms]
SpeedUp de suma: 2.55665
```

Figura 7: Ejemplo de muestra de tiempo de los llenados.

```
cristian@DESKTOP-VS0OV9G:/mnt/c/Users/Cristian/Desktop/U2-master/taller02$ ./sumArray -N 54305000 -t 10 -l 1 -L 8800
=======LLenado y Suma de numeros de un Arreglo=======
================================================
Tiempo de LLenado secuencial: 1878[ms]
Tiempo de LLenado con threads: 768.447[ms]
SpeedUp de suma: 2.44389
```

Figura 8: Ejemplo 2 de muestra de tiempo de llenados.

4.2 Comparativa de suma de forma secuencial y paralela

Para la sección del apartado de la suma se ejecuta de igual manera que en la sección anterior con un arreglo de tamaño N, los cuales fueron creados en la sección de llenado, para esta se realizan pruebas con distintos

parámetros para visualizar la diferencia de desempeño entre las sumas de manera secuencial y paralela. En la Figura 9, 10 se aprecian ejemplos con distinto número de threads.

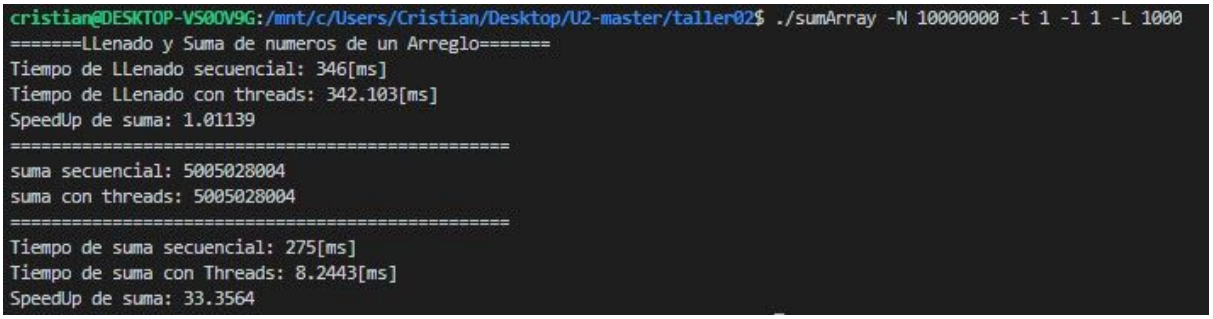


Figura 9: Ejemplo Muestra de datos de suma secuencial y paralela.

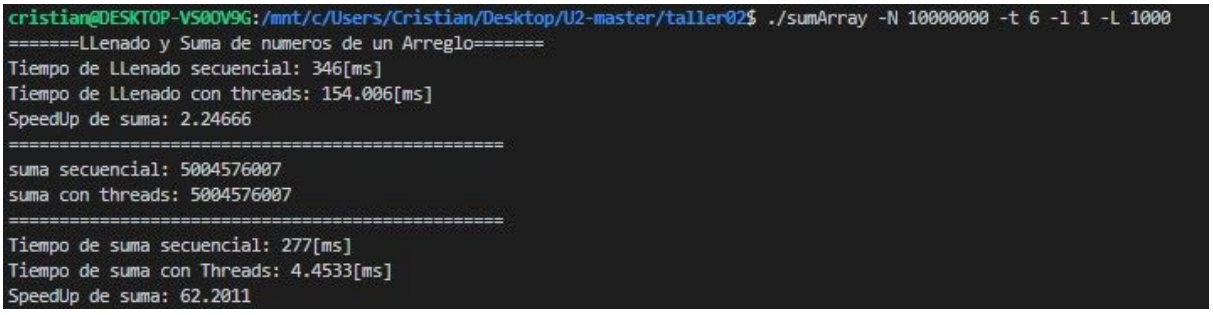


Figura 10: Ejemplo 2 Muestra de datos de suma secuencial y paralela.

5. Resultados

Al igual que en la anterior sección como se mostró en los ejemplos, de la misma manera se realizó una serie de pruebas para comprender el comportamiento del algoritmo con respecto al tiempo tomado en la ejecución de este mismo. Para este testeo se mantuvieron los mismo parámetros cambiando su número de threads para obtener un gráfico de *tiempo vs número de threads*, como se puede ver en la Figura 11.

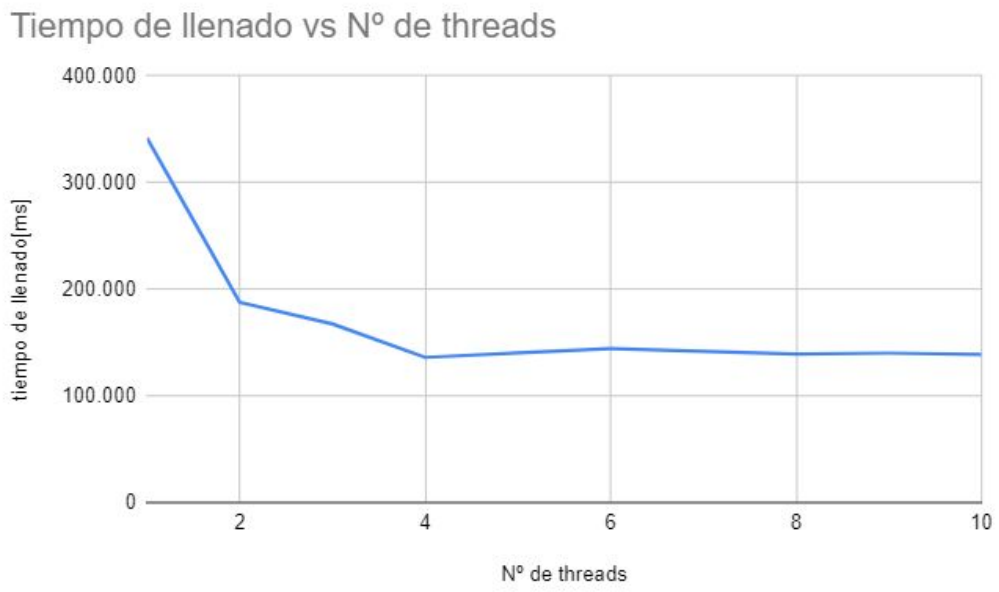


Figura 11: Gráfico Tiempo de llenado vs nº de threads.

Como se puede apreciar en la Figura 11, hay una clara vista de al momento de realizar el llenado del arreglo dinámico, el tiempo disminuye significativamente a medida que se aumenta el número de threads.

Para el caso del tiempo de suma de los arreglos creados, se realizó la misma serie de pruebas, con lo que se llegó al siguiente gráfico de *Tiempo de Suma vs número de Threads*, se puede ver la Figura 12.

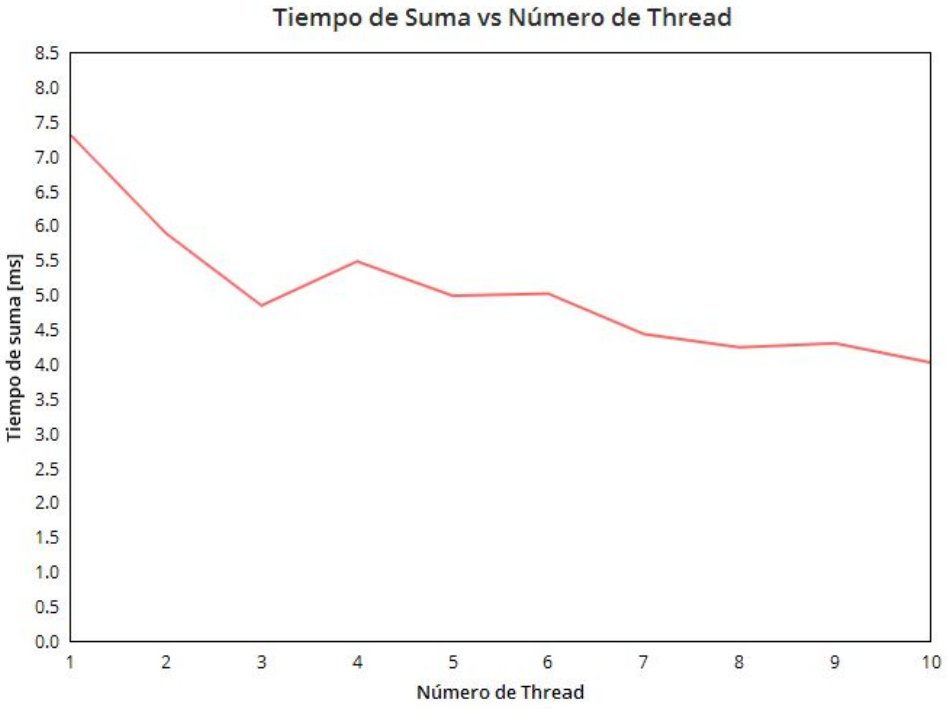


Figura 12: Gráfico de Tiempo de Suma vs nº de threads

Como se aprecia en el gráfico desde cierto punto el número de threads provoca una “ineficiencia”, por lo que ya no se vuelve tan viable como se espera.

6. Conclusión

Como se hace referencia en un inicio y los objetivos que estaban propuestos para el taller presentado, se logró realizar de manera correcta la creación de ambos módulos, llenado y suma de datos de un arreglo dinámico, donde se logró visualizar el desempeño con respecto al tiempo tomado de la ejecución de cada proceso, además de comprender el funcionamiento de los hilos por medio de distintas pruebas.

Como comentario cabe destacar que el desempeño del módulo de suma se puede ver afectado dependiendo de los números random con los que se llene el arreglo, también no siempre se dará el caso de que el desempeño con threads sea mejor que de manera secuencial.