Computación Concurrente

Equipo Tijeras

Práctica 3 - Spinlocks

López Balcazar Fernando - 316128182
Reyes Ramos Luz María - 318211073
Sánchez Castro Gustavo - 318213888

Fecha de entrega: 5 de octubre de 2023

1. Introducción:

Zeratun estaba muy contento pues ya pudo recuperar su tanque de gas y sus macetas, pero se dio cuenta que le roban mucho, por lo que decidio aprender computación concurrente para que no le pasara de nuevo y asi estar prevenido, ayudemos a zeratun a aprender computo concurrente y pueda por fin descansar.

2. Teoría:

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

   Sirve para suspender temporalmente la ejecución de un hilo o proceso. Se suele usar para ceder el control de los procesadores a otros hilos o procesos. Se suele utilizar en situaciones donde se desea evitar que un solo hilo o proceso acapare todos los recursos del sistema.

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

   Dentro de la biblioteca *atomic* de Java tenemos varias clases como pueden ser *AtomicInteger*, *AtomicLong* y *AtomicBoolean* que nos sirven para crear objetos que nos permitan facilitar la concurrencia e incluso para evitar candados.

   La forma de utilizarlos es creando ejemplares de dichas clases (instancias) las cuales contienen métodos que nos ayudan a evitar el problema de que dos hilos lean una variable y cuando la quieran modificar dicha variable ya haya cambiado, un ejemplo de esto es el ejemplo de que si tenemos un contador entero y hacemos *counter++* si dos hilos tienen acceso a la variable *counter* entonces pueden tener los problemas mencionados anteriormente. Una solución a esto es en vez de que la variable sea un entero que sea un *AtomicInteger* y en vez de utilizar *counter++* utilizar su método *addAndGet(1)* como en el ejemplo *counter.addAndGet(1)*.

3. Ventajas de usar atributos atomicos

   - Puedes ahorrarte de utilizar candados.
   - Evitas resultados inesperados en caso de que no hubieras considerado manejar la concurrencia.

4. Desventajas de usar atributos atómicos

- Estamos trabajando con objetos y no con datos primitivos.
- Mayor uso de memoria por lo mismo de que son objetos.
- Realizar operaciones más grandes es complicado ya que se necesitan muchos métodos en vez de hacer un simple cálculo y reasignarlo.
- Se escribe un poco más de código.

5. Da 2 ejemplos en donde se puedan aplicar este tipo de atributos.

   En el caso de contadores compartidos por hilos. También podemos utilizarlos para los switch de valores booleanos como el ejemplo de un apagador, podemos cambiar el valor sin preocuparnos de que alguno se anule o incluso que se contrarresten.

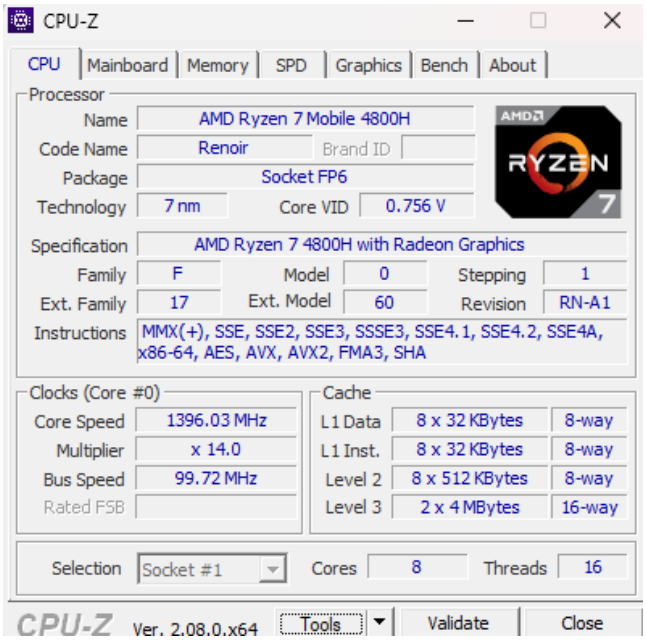
6. Algun uso que creas que se da en estructuras de datos concurrentes.

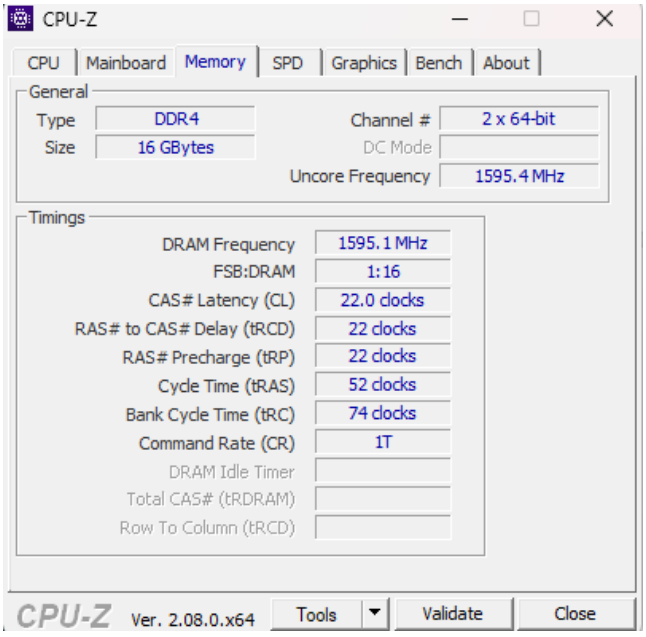
   Para evitar usar variable volátiles y ahorrarnos algunos candados, que aparte de hacer nuestro código más sencillo lo hace fácil de leer y entender.

3. Práctica:

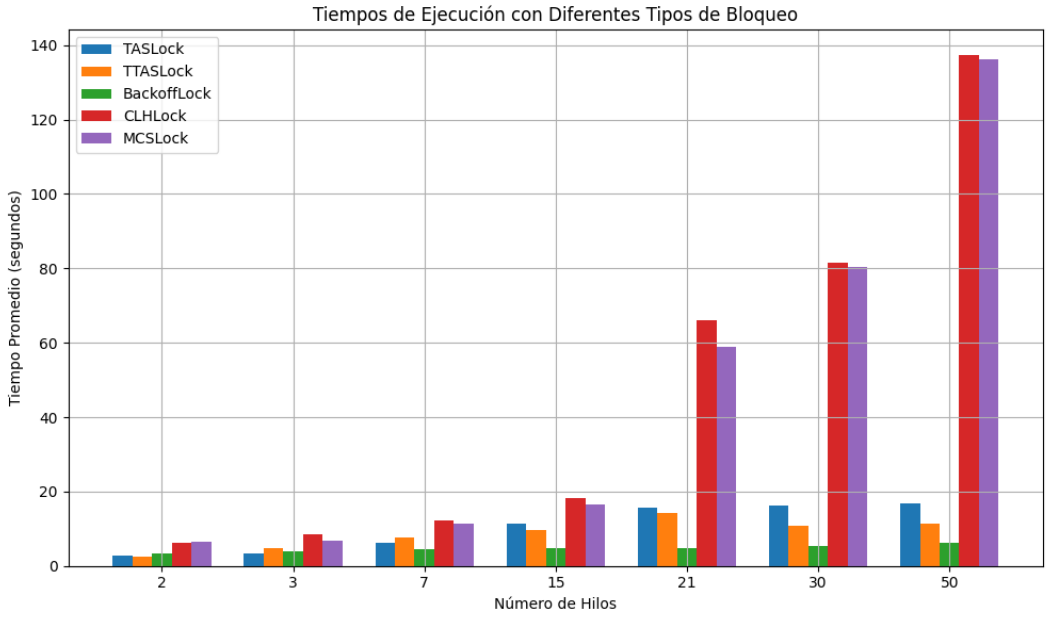
Para esta parte decidimos usar un graficador y un tabulador que programamos en phyton con lo cual quedaron de la siguiente manera :

- Gustavo :

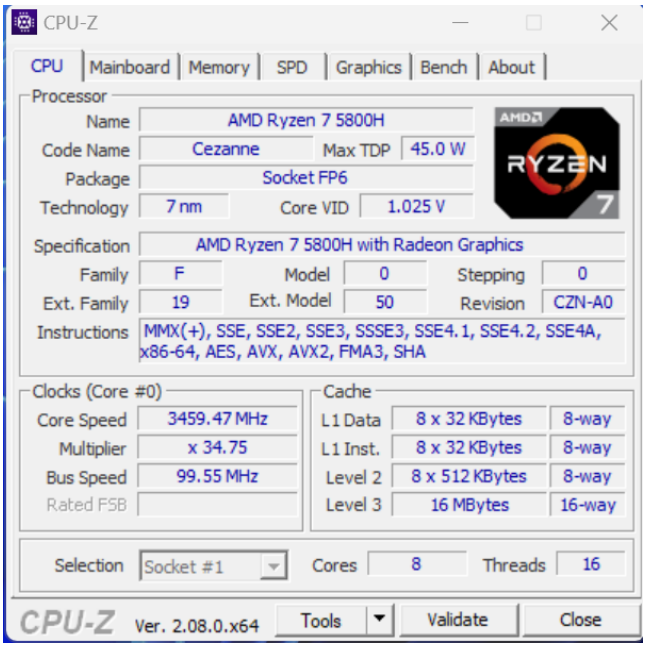

CPU-Z
CPU
Mainboard
Memory
SPD
Graphics
Bench
About
Processor
Name
AMD Ryzen 7 Mobile 4800H
Code Name
Renoir
Brand ID
Package
Socket FP6
Technology
7 nm
Core VID
0.756 V
Specification
AMD Ryzen 7 4800H with Radeon Graphics
Family
F
Model
0
Stepping
1
Ext. Family
17
Ext. Model
60
Revision
RN-A1
Instructions
MMX(+), SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, SSE4A, x86-64, AES, AVX, AVX2, FMA3, SHA
Clocks (Core #0)
Core Speed
1396.03 MHz
Multiplier
x 14.0
Bus Speed
99.72 MHz
Rated FSB
Cache
L1 Data
8 x 32 KBytes
8-way
L1 Inst.
8 x 32 KBytes
8-way
Level 2
8 x 512 KBytes
8-way
Level 3
2 x 4 MBytes
16-way
Selection
Socket #1
Cores
8
Threads
16
CPU-Z
Ver. 2.08.0.x64
Tools
Validate
Close

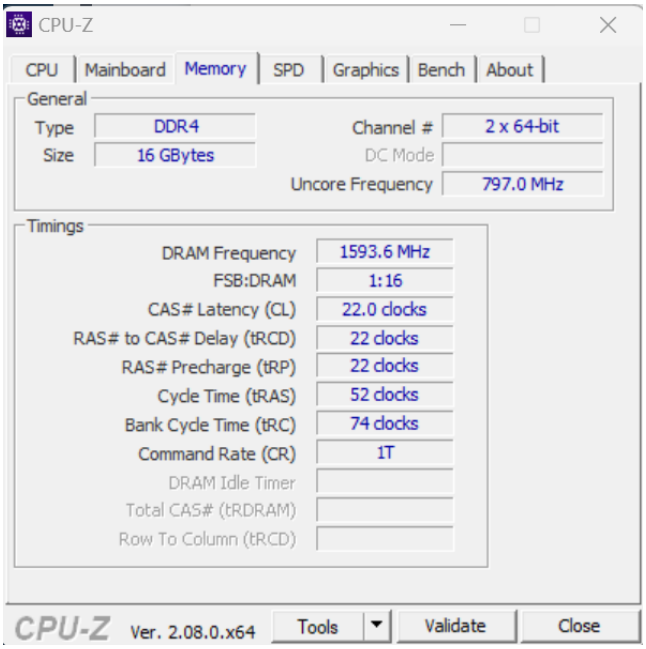

CPU-Z
CPU
Mainboard
Memory
SPD
Graphics
Bench
About
General
Type
DDR4
Size
16 GBytes
Channel #
2 x 64-bit
DC Mode
Uncore Frequency
1595.4 MHz
Timings
DRAM Frequency
1595.1 MHz
FSB:DRAM
1:16
CAS# Latency (CL)
22.0 clocks
RAS# to CAS# Delay (tRCD)
22 clocks
RAS# Precharge (tRP)
22 clocks
Cycle Time (tRAS)
52 clocks
Bank Cycle Time (tRC)
74 clocks
Command Rate (CR)
1T
DRAM Idle Timer
Total CAS# (tRDRAM)
Row To Column (tRCD)
CPU-Z
Ver. 2.08.0.x64
Tools
Validate
Close

Número de Hilos	TASLock	TTASLock	BackoffLock	CLHLock	MCSLock
2.0	2.899	2.493	3.247	6.28	6.58
3.0	3.385	4.861	3.885	8.546	6.691
7.0	6.114	7.772	4.412	12.113	11.239
15.0	11.49	9.59	4.862	18.344	16.524
21.0	15.692	14.297	4.831	66.0	58.915
30.0	16.37	10.838	5.267	81.6	80.4
50.0	16.876	11.471	6.21	137.4	136.2

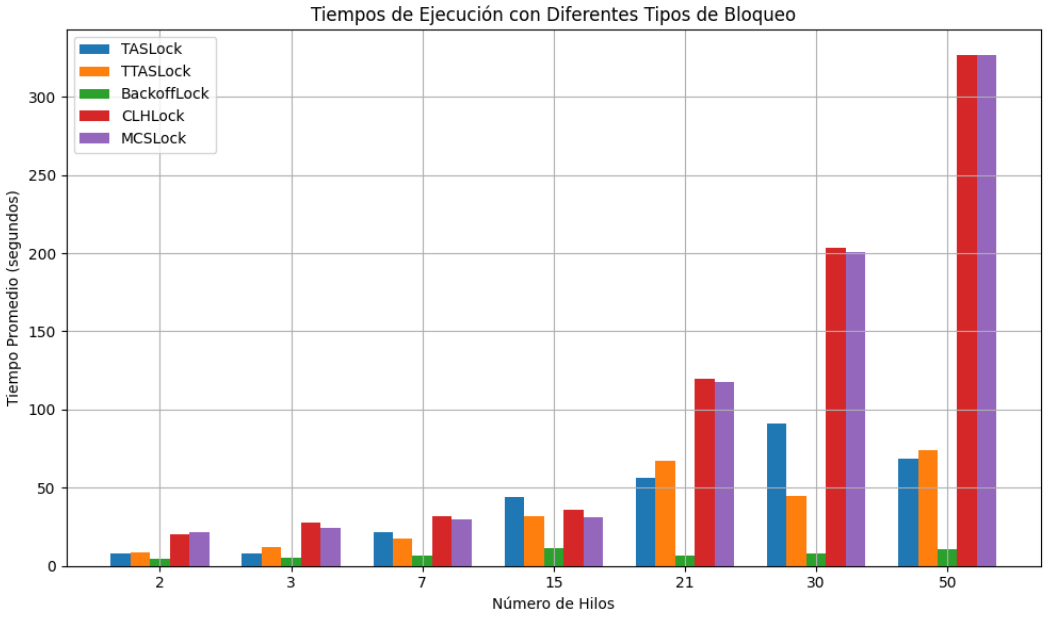


- Luz:

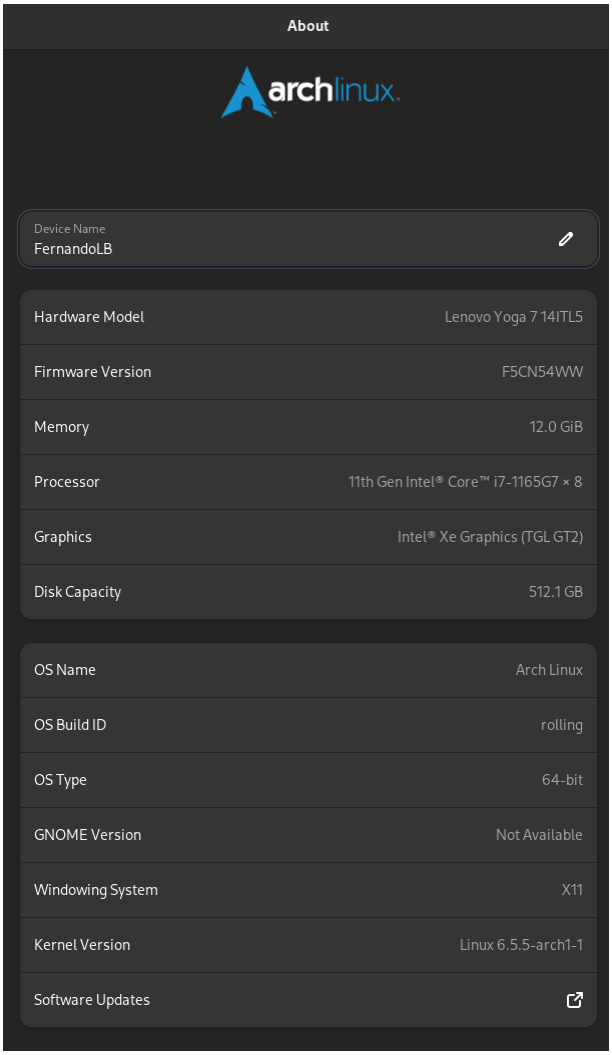

CPU-Z
CPU
Mainboard
Memory
SPD
Graphics
Bench
About
Processor
Name
AMD Ryzen 7 5800H
Code Name
Cezanne
Max TDP
45.0 W
Package
Socket FP6
Technology
7 nm
Core VID
1.025 V
Specification
AMD Ryzen 7 5800H with Radeon Graphics
Family
F
Model
0
Stepping
0
Ext. Family
19
Ext. Model
50
Revision
CZN-A0
Instructions
MMX(+), SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, SSE4A, x86-64, AES, AVX, AVX2, FMA3, SHA
Clocks (Core #0)
Core Speed
3459.47 MHz
Multiplier
x 34.75
Bus Speed
99.55 MHz
Rated FSB
Cache
L1 Data
8 x 32 KBytes
8-way
L1 Inst.
8 x 32 KBytes
8-way
Level 2
8 x 512 KBytes
8-way
Level 3
16 MBytes
16-way
Selection
Socket #1
Cores
8
Threads
16
CPU-Z
Ver. 2.08.0.x64
Tools
Validate
Close


CPU-Z
CPU
Mainboard
Memory
SPD
Graphics
Bench
About
General
Type
DDR4
Size
16 GBytes
Channel #
2 x 64-bit
DC Mode
Uncore Frequency
797.0 MHz
Timings
DRAM Frequency
1593.6 MHz
FSB:DRAM
1:16
CAS# Latency (CL)
22.0 clocks
RAS# to CAS# Delay (tRCD)
22 clocks
RAS# Precharge (tRP)
22 clocks
Cycle Time (tRAS)
52 clocks
Bank Cycle Time (tRC)
74 clocks
Command Rate (CR)
1T
DRAM Idle Timer
Total CAS# (tRDRAM)
Row To Column (tRCD)
CPU-Z
Ver. 2.08.0.x64
Tools
Validate
Close

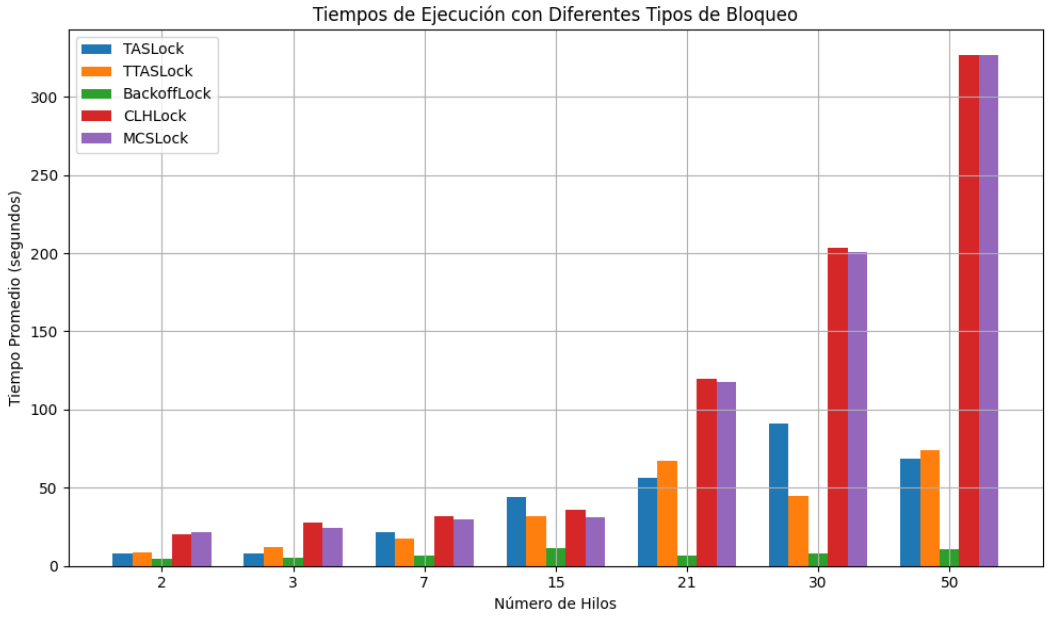
Número de Hilos	TASLock	TTASLock	BackoffLock	CLHLock	MCSLock
2.0	8.306	8.486	4.908	20.446	21.345
3.0	7.777	12.174	5.367	27.99	24.117
7.0	21.897	17.511	6.477	31.738	29.487
15.0	44.313	31.783	11.365	35.565	30.837
21.0	56.648	67.418	6.883	119.78	117.491
30.0	91.013	44.652	7.986	203.737	200.855
50.0	68.88	73.772	10.408	326.793	326.452



- Fernando:


About
archlinux.
Device Name
FernandoLB
Hardware Model
Lenovo Yoga 7 14ITL5
Firmware Version
F5CN54WW
Memory
12.0 GiB
Processor
11th Gen Intel® Core™ i7-1165G7 × 8
Graphics
Intel® Xe Graphics (TGL GT2)
Disk Capacity
512.1 GB
OS Name
Arch Linux
OS Build ID
rolling
OS Type
64-bit
GNOME Version
Not Available
Windowing System
X11
Kernel Version
Linux 6.5.5-arch1-1
Software Updates

Número de Hilos	TASLock	TTASLock	BackoffLock	CLHLock	MCSLock
2.0	6.324	4.33	2.994	9.141	9.279
3.0	9.242	6.453	3.808	11.856	10.355
7.0	11.032	8.589	5.329	14.423	15.869
15.0	10.502	7.636	7.636	65.677	62.72
21.0	9.015	12.239	7.941	80.883	110.825
30.0	18.574	10.956	9.665	93.745	97.667
50.0	12.798	6.806	9.548	166.19	168.446



Porque se obtienen esos tiempos:
Como podemos ver en las gráficas los que mas tarda son CLH y TLC se obtienen debido a la eficiencia de la gestión de bloqueos y la competencia por el recurso compartido además de que es muy importante el di positivo que estemos usando ya que algunos dispositivos pueden ejecutar con mayor potencia logrando menos tiempo en el caso de CLH s tardado debido a que usa una cola de nodos en la que los hilos se enfilan para adquirir el bloqueo en cambio los mas rapidos como TAS y TTAS no hace esto.
Compara el resultado:
Comparando los resultados podemos notar que son muy similares pero los resultados obtenido por Gustavo son mas cortos en la mayoría de los casos esto se debe a que puso su computadora en la opción Perfonmance tuning que es una opción en los dispositivos Huawei , esta opción lo que hace es acelerar el procesador para que de su máximo rendimiento.
En el caso de luz podemos notar que tuvo tiempos mas tardados incluso con un procesador mas actual que el de Gustavo , sus resultados se deben a que tenia su dispositivo en ahorro de batería lo que limita a su procesador.
Otro factor que puede afectar los tiempos de ejecución puede ser el sistema operativo , como podemos notar Fernando usa Kali Linux y tiene en ocasiones mejores tiempos .
Lo aprendido durante la practica y las impresiones recibidas al realizarla:
Con esta practica hemos aprendido mucho sobre los mecanismos de bloqueo que se ocupan en la computación concurrente además de como usar diferentes estrategias de bloqueo , igual como los tiempos de ejecución pueden variara demasiado dependiendo la implementación realizada y el equipo que estemos usando y como algunos mecanismos son más adecuados para escenarios con alta competencia por recursos, mientras que otros son más eficientes en situaciones con menos competencia y asi la elección del mecanismo de bloqueo correcto depende de la aplicación y sus requisitos de concurrencia.
Nos impresiono como la elección del mecanismo de bloqueo correcto puede tener una gran diferencia en el rendimiento y la eficiencia de una aplicación.

Referencias

- GeeksforGeeks. (2022). Java concurrency yield sleep and join methods. GeeksforGeeks. Recuperado de: https://www.google.com/amp/s/www.geeksforgeeks.org/java-concurrency-yield-sleep-and-join-methods/amp/
- AtomicInteger (Java Platform SE 8). (2023, 14 junio). Recuperado de: https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/AtomicInteger.html
- Jain, D., & Jain, D. (2023). An introduction to atomic variables in Java | Baeldung. Baeldung. Recuperado de: https://www.baeldung.com/java-atomic-variables