Operating System Homework #1

Platform

I used VS Code and the remote WSL1 extension to run the program

CPU type	Memory size	Operating system	System type	Kernel version	Machine type
Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz Cores：6 Logical processor：12	16.0GB	Microsoft Windows 10 Home 10.0.19042 N/A Build 19042	64-bit operating system, x64-based processor	Windows 10 Home 20H2	Virtual Machine

The Measurement Results

Hint:

1. The unit used for the result analysis is Microsecond（μs）
    - 1 s = 1000000 μs
    - 1 μs = 1.0E-6 s
2. The generating array is a set of integers not exceeding 50
3. Program execution content:
    - Find the frequency of user input values in the array（I always enter a value of 10）
    - The user can decide the length of the array and the number of processes/threads
4. Source code：
    - OSHW1A.c：Single Process
    - OSHW1B.c：Multi-Process
    - OSHW1C.c：Multi-Thread
    - OSHW1D.c：Multi-Thread with mutex

Single-process

Terminal Command：

- gcc OSHW1A.c -lpthread -Wall -std=c99 -o OSHW1A
- taskset -c 0,1,2,3,4,5,6,7 ./OSHW1A(bind the 8 logical cores of the CPU)

Array Length	Used Time（μs）
256	109
3840	87
57600	95
864000	94

Multi-process

Terminal Command：

- gcc OSHW1B.c -lpthread -Wall -std=c99 -o OSHW1B
- taskset -c 0,1,2,3,4,5,6,7 ./OSHW1B(bind the 8 logical cores of the CPU)

For multi-process (Number of Processes : 2)

Array Length	Used Time（μs）
256	11469
3840	11356
57600	11773
864000	16965

For multi-process (Number of Processes : 4)

Array Length	Used Time（μs）
256	15427
3840	12428
57600	12990
864000	18438

For multi-process (Number of Processes : 8)

Array Length	Used Time（μs）
256	10323
3840	10610
57600	13076
864000	14931

Multi-thread

Terminal Command:

- gcc OSHW1C.c -lpthread -Wall -std=c99 -o OSHW1C
- taskset -c 0,1,2,3,4,5,6,7 ./OSHW1C (bind the 8 logical cores of the CPU)

For multi-thread (Number of Threads : 2)

Array Length	Used Time (**μs**)
256	3192
3840	3292
57600	3429
864000	8719

For multi-thread (Number of Threads : 4)

Array Length	Used Time (**μs**)
256	3390
3840	3556
57600	3623
864000	9018

For multi-thread (Number of Threads : 8)

Array Length	Used Time (**μs**)
256	3946
3840	4178
57600	4177
864000	9767

Multi-thread(with mutex)

Terminal Command:

- gcc OSHW1D.c -lpthread -Wall -std=c99 -o OSHW1D
- taskset -c 0,1,2,3,4,5,6,7 ./OSHW1D (bind the 8 logical cores of the CPU)

For multi-thread (Number of Threads : 2)

Array Length	Used Time (**μs**)
256	1130
3840	1207
57600	1525
864000	10154

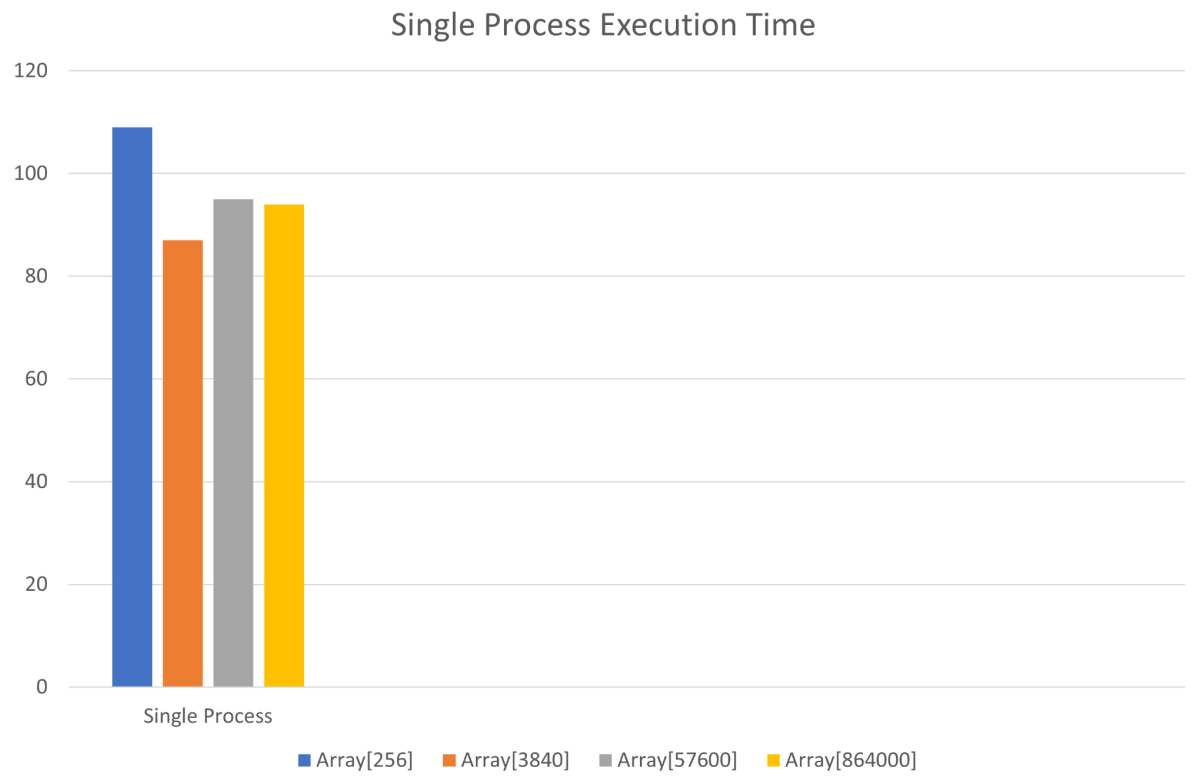
For multi-thread (Number of Threads : 4)

Array Length	Used Time (**μs**)
256	1727
3840	1849
57600	1944
864000	10235

For multi-thread (Number of Threads : 8)

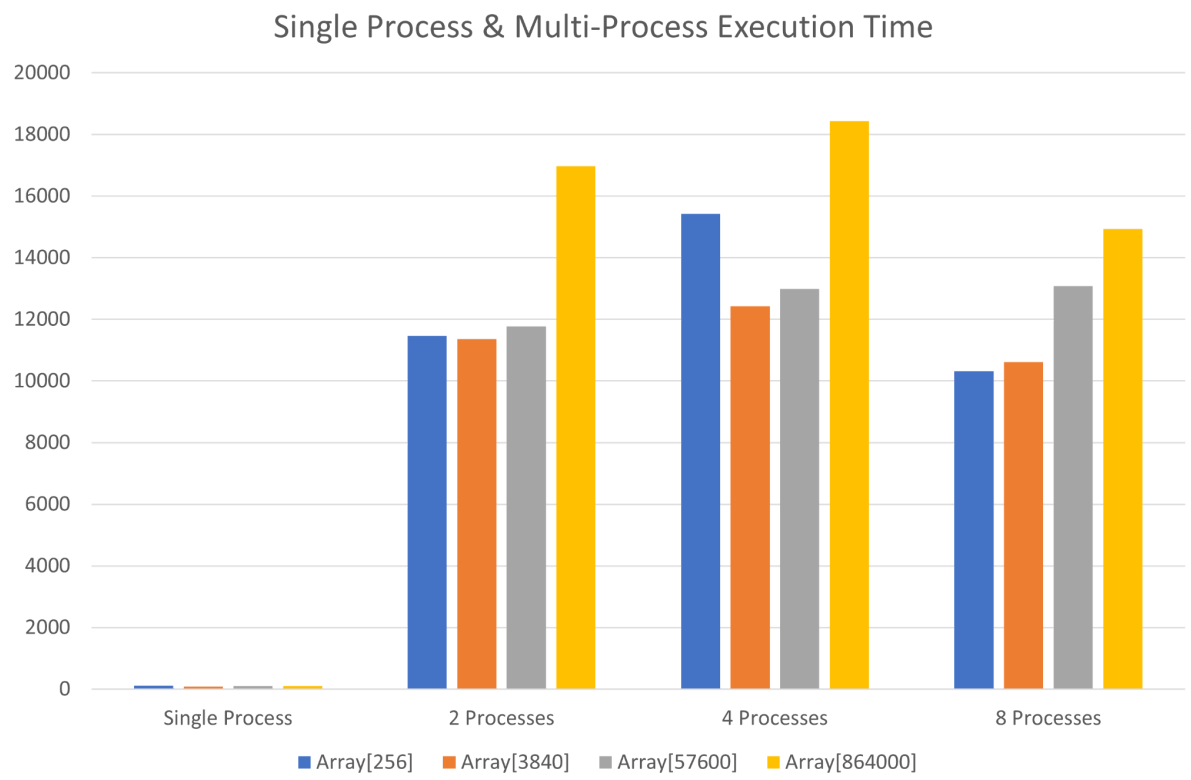
Array Length	Used Time (**μs**)
256	2563
3840	2914
57600	3149
864000	10660

Analysis



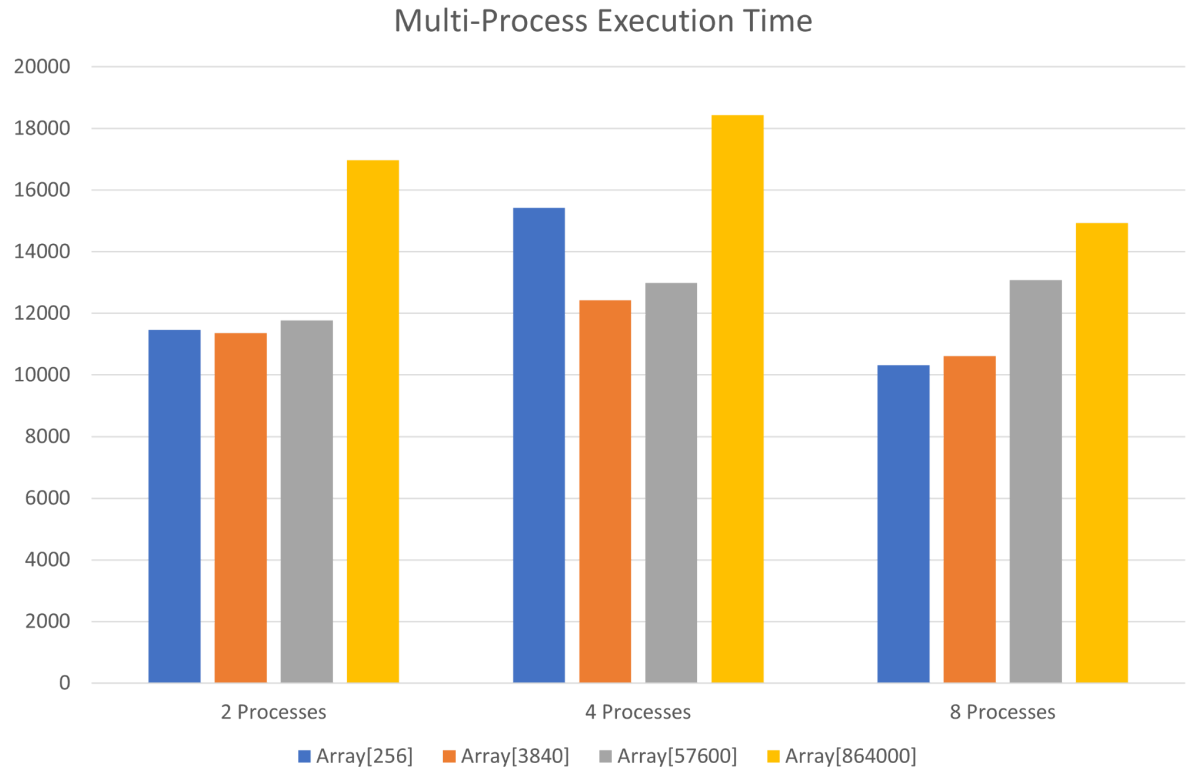
For Single Process:

- Strangely, the shortest length array has the longest runtime
    - There may be some other influencing factors, to be discussed



Single process v.s. Multi-process:

- For arrays of length 256,3849,57600,864000, the execution time for the single process is much less than for multi-processes
    - When the amount of data is small and the calculation is simple, the process of context switching can have a negative impact on system performance
        - Within some scheduling scheme, one process must be switched out of the CPU so another process can run
        - Switching from one process to another requires a certain amount of time for doing the administration



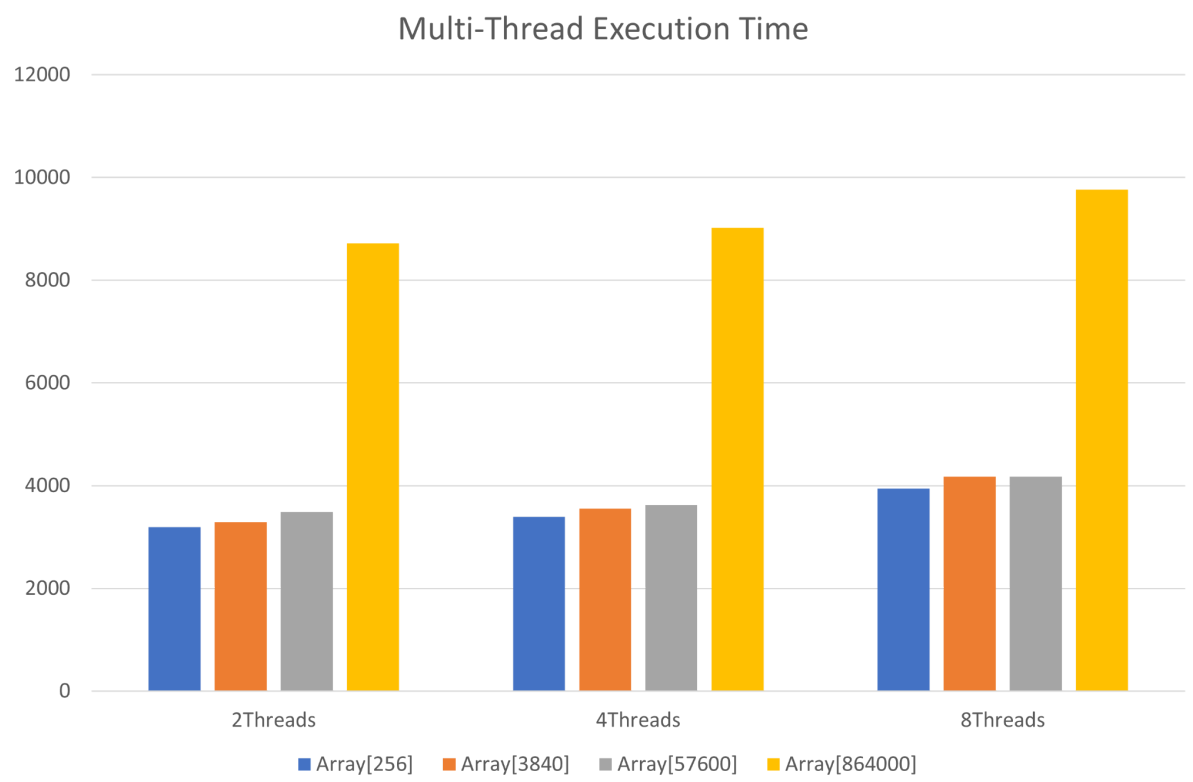
For multi-process:

Interpretation of this chart:

- The execution time of 4 processes is more than 2 processes and 8 processes in general
- The array of length 57600 runs longest when the number of processes is 4 and is much higher than when the number of processes is 8
- When the number of processes is 8, shorter arrays take less time to run

Conclusion:

- The higher the number of processes, the shorter the execution time is not necessarily
- When the amount of data is small, the performance of 8 processes is good
- Best overall performance when the number of process is 8
    - Since the cpu has 8 logical cores, ideally 8 process programs run on separate cores, so the best performance is achieved when the number of processes is 8
- When the amount of data becomes large and the calculation becomes complex, the advantage is obvious when there are many processes



For multi-thread:

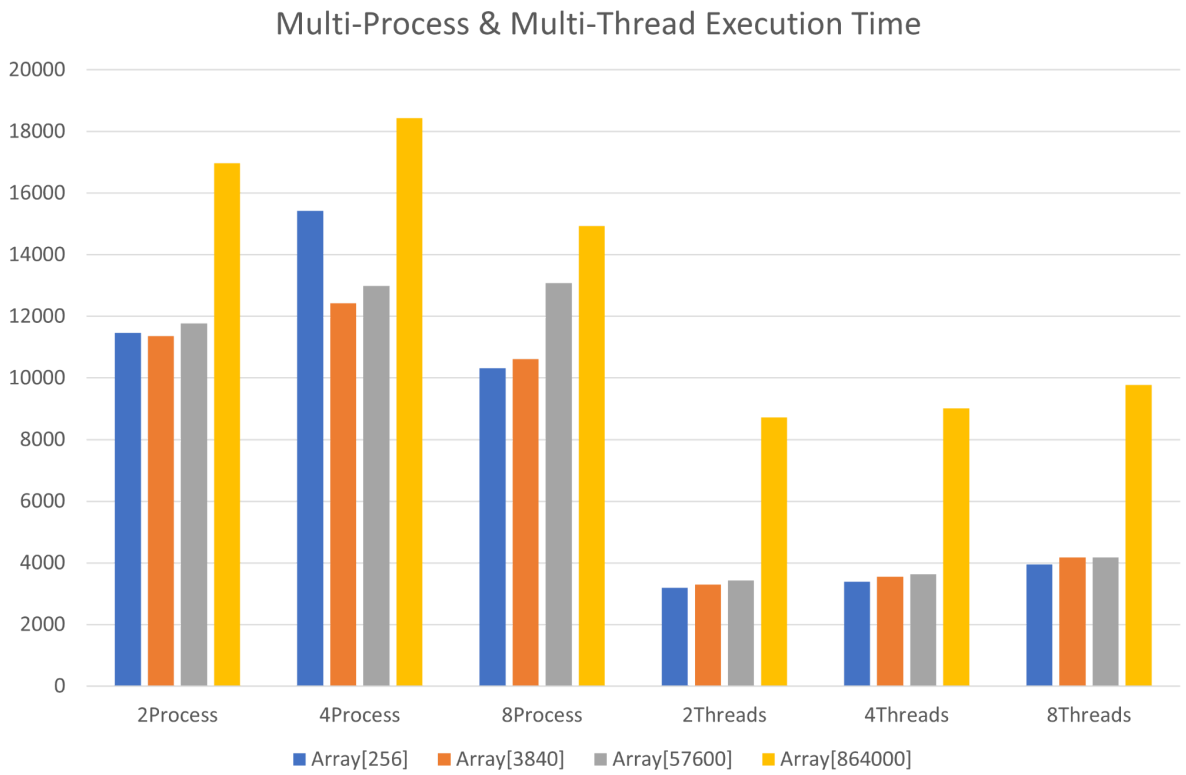
Interpretation of this chart:

- When the number of threads is the same, if the length of array is 864000(very large), the execution time will be much longer than the short length array
- When the number of threads increases from 2 to 4, the execution time of arrays of length 256,3840 and 57600 does not change much
- When the number of threads is the same, the execution time does not necessarily get longer as the amount of data increases when the amount of data does not vary greatly from one profile to another
- When the number of threads is the same, if the amount of data differs significantly from one profile to another, the execution time difference will also be significant

Conclusion:

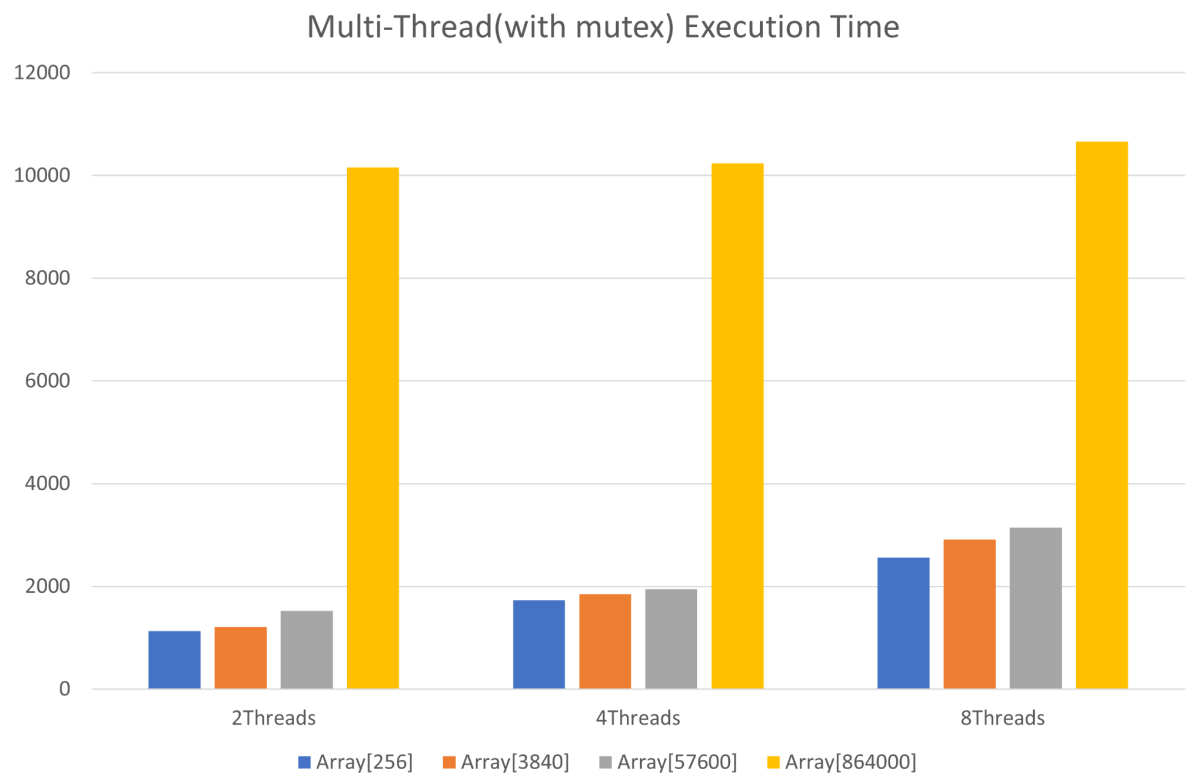
- According to the statistics, the more threads there are, the shorter the runtime is not
    - It takes time for a thread to request computational resources from the operating system
    - Switching between threads takes time
- In general, the execution time increases with the volume of data, and this pattern becomes more obvious especially when the volume of data varies greatly

- When the amount of data is small and the calculation is simple, but the number of threads is large, it is time consuming



Multi-process v.s. Multi-thread:

- Multi-thread performance is clearly much better than multi-progress
  - Threads under a Process share resources such as memory, Global Variable, etc., while different Processes do not
  - Since each Process requires some resources to work, Multi-process will consume more resources than Multi-thread
  - Multiple processes require resource requests and memory copies, which consumes some time



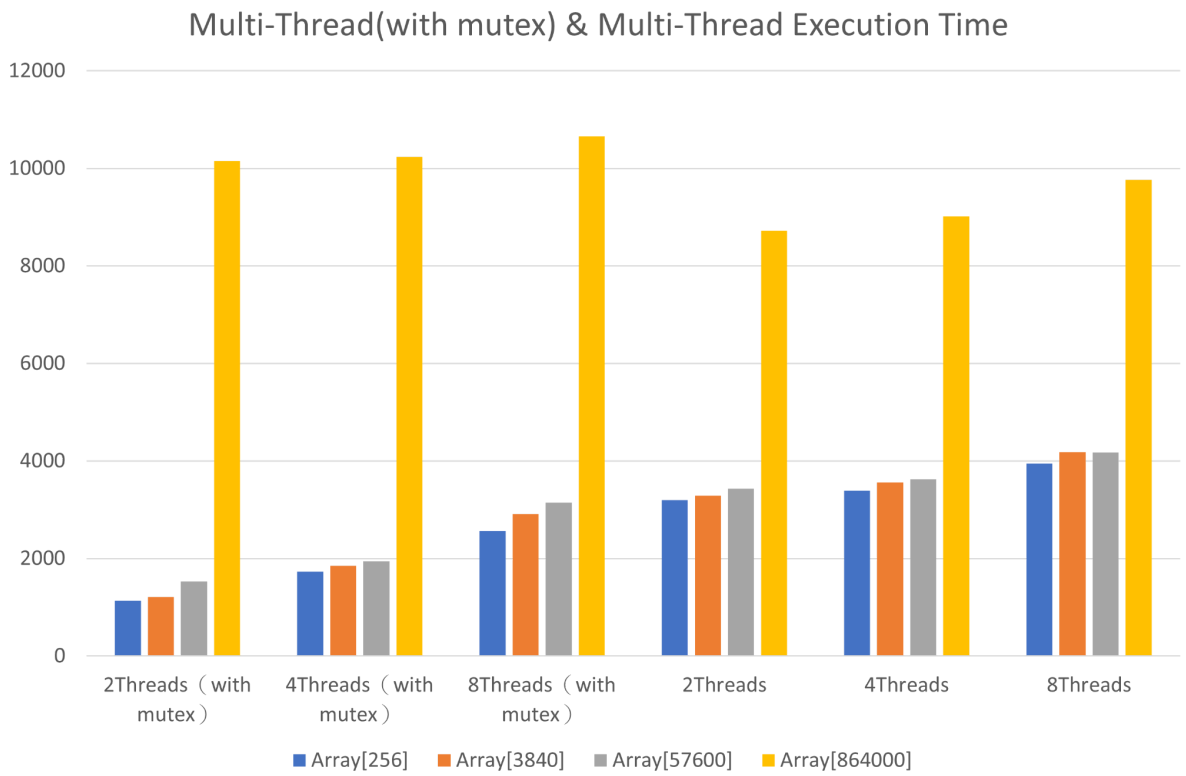
For multi-thread(with mutex):

Interpretation of this chart:

- The higher the number of threads, the longer the execution time
- The more the amount of data, the longer the execution time

Conclusion:

- The more threads use Mutex, the more time it takes to go from Lock to Unlock, so the more threads there are, the more time this part takes



Multi-thread(with mutex) v.s. Multi-thread:

- When the amount of data is small, the execution time of threads with mutex is significantly less
- When the amount of data is big, the threads with mutex will take longer time to execute than the threads without mutex

Conclusion

1. Within some scheduling scheme, one process must be switched out of the CPU so another process can run, switching from one process to another requires a certain amount of time for doing the administration, so the process of context switching can have a negative impact on system performance.
2. For multi-process, best overall performance when the number of process is 8 since the cpu has 8 logical cores, ideally 8 process programs run on separate cores. It is reasonable to say that 8 threads should also perform the best, but the actual result is not so, this part is open to discussion, ,there may be some other influencing factors exist.
3. Multi-thread performance is clearly much better than multi-progress because threads under a process share resources such as memory, global variable, etc., while different processes do not and each process requires some resources to work, which consumes some time.
4. The more threads use Mutex, the more time it takes to go from Lock to Unlock, so the more threads there are, the more time this part takes.
5. If the thread needs to use mutex, it should be concerned about the amount of the data, if the amount of data is too large, the performance of using mutex is not better.

6. I think the use of WSL may cause some errors, so the actual data does not seem to fit the theory.