Hyun Ho Kim 1001967176

REPORT

Multi-Threading

1) A brief executive summary/ overview

- By using multiple threads, I attempted to complete the task concurrently. Each thread had a specific portion of the task to finish, and each thread changed the value of a global variable while performing its function.

2) A description of your effort to include which libraries, if any, you used, and the reasons you chose your timing methods.

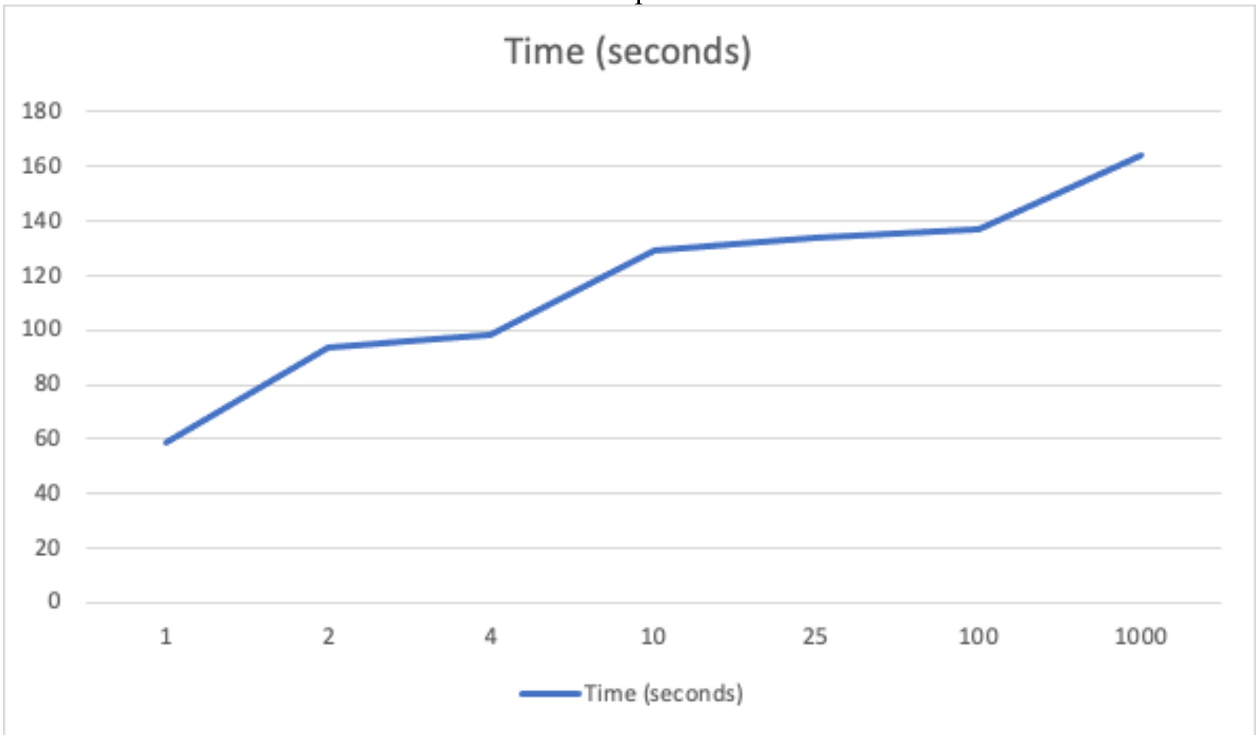
- I included <pthread.h> to create, join, and lock threads, and used for loops to create multiple threads that were all joined. I also used mutexes to avoid race conditions. To measure execution time, I included <time.h> and <math.h>, with time.h providing a way to measure time and math.h converting it to seconds.

3) Results in both graph and table form.

Table

Number of Threads	Time (seconds)
1	58.618904
2	93.627381
4	98.427697
10	129.090026
25	133.820565
100	136.675241
1000	164.127088

Graph



4) A discussion of any anomalies you found in the resulting data.
- Theoretically, having more workers should reduce the time required to complete the task. However, the data showed that as the number of threads increased, more time was required. This outcome was likely due to differences in CPU performance. If the number of threads exceeds the number of available CPU cores, the threads must take turns executing, which can reduce performance.

5) A conclusion explaining the optimal number of threads.
- Since the Codespace has a 2-core CPU, two threads should be optimal. However, the result showed that using only one thread took the shortest time.