ENCE360 Assignment

Jakib Isherwood

October 2023

1 Introduction

My testing methodology involved three sets of test cases: easy, mixed, and hard. I tested each file on a Linux lab computer, with nothing but Visual Studio Code open. I also commented out or removed each instance of `printf` that wasn't printing the integral result itself.

I used the commands: `time ./FILENAME < TESTFILENAME.txt`

1.1 Test Cases

1. Easy test cases:
   - -1 1 9999999 1
   - 0 3.14 9999999 0
   - 0 5 9999999 2
   - -10 10 9999 0
   - -10 10 9999 1
   - -10 10 9999 2

   These are small ranges with a small to medium number of steps.

2. Mixed test cases:
   - -1 1 9999999 1
   - 0 3.14 9999999 0
   - 0 5 9999999 2
   - -100 100 1000000000 0
   - -100 100 1000000000 1
   - -100 100 1000000000 2

   This mixes 3 from the easy test cases, with some more difficult tests which have many steps and a very large range.

3. Hard test cases:

   - -100 100 1000000000 0
   - -100 100 1000000000 1
   - -100 100 1000000000 2
   - -10 10 999999999 0
   - -10 10 999999999 1
   - -10 10 999999999 2

   Large or medium ranges with a very large number of steps.

2 Results

2.1 Real time performance differences (in seconds)

	Serial	Process	Thread	ProcessThread
Easy	0.548	0.226	0.172	0.104
Mixed	50.35	20.17	15.781	8.588
Hard	98.846	20.668	30.266	15.272

Table 1: Real time performance differences based on text input complexity.

Figures 1 and 2 show that the Serial implementation is rather linear in its speed given more complex or difficult input. This is expected as it has to process each integral one at a time, with no way of splitting up computation. Process, Thread, and ProcessThread are all significantly faster as they all have at least one way of speeding up the process. Process forks children so it's able to work on multiple calculations concurrently. There is a maximum 6 children per process in my implementation, and each test file has 6 test cases which should allow Process to be as fast as it could possibly be as there are exactly enough children to do all calculations simultaneously. Process is roughly 5 times faster than Serial. It is likely to not be 6 times faster precisely due to the overhead incurred by forking.[1] In addition to this, it needs to wait until the final child has finished to exit, so it will always be as fast as its slowest child. Thread is even faster than Process for everything but Hard. This is due to threads being very light-weight and fast due to having little resources attached to them as all they need are registers, the stack, and the state at the given time on creation.[2], [3] A reason that Process might be running faster than Thread for complex text input is due to the time it takes to compute, it could be offsetting the initial overhead imposed by forking, and Thread might be suffering due to having many mutexes and having to deal with concurrency and lock contention for the level of computations being made.[4] It does have a similar problem to Process in the sense of the join operation needing to wait for its slowest thread, but as

threads tend to be for smaller computations, it is significantly less expensive for performance. ProcessThread is simply the fastest by far. It has the most overhead as it is a combination of both Process and Thread, but it also gets the best of both implementations; combining multiple children with multi threading within each child to maximize speed. It is *almost* slower than Process for Hard, but this is due to the same problem that Thread has for difficult and long computation.

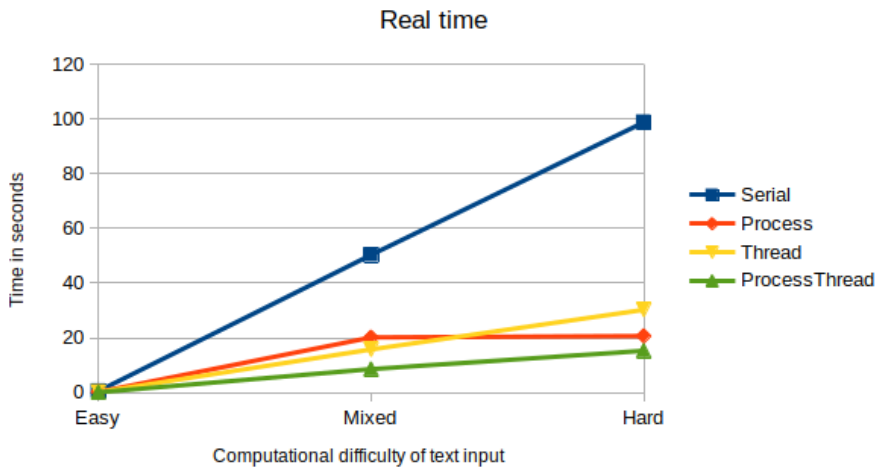

Figure 1: Graph of real time speed difference based on text input complexity

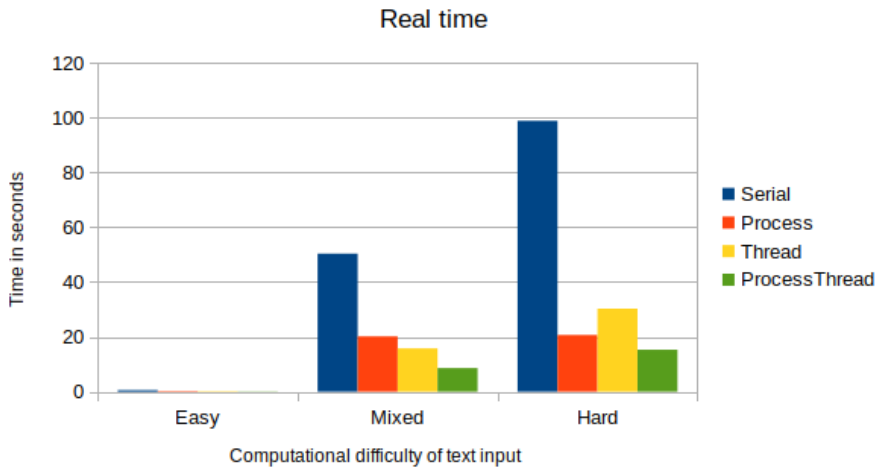

Figure 2: Bar chart of real time speed difference based on text input complexity

2.2 User time performance differences (in seconds)

	Serial	Process	Thread	ProcessThread
Easy	0.54	0.554	0.572	0.556
Mixed	50.209	51.225	52.123	52.566
Hard	98.784	102.026	102.84	113.828

Table 2: User time performance differences based on text input complexity.

This shows that they all take up roughly the same amount of time in User space doing calculations and such. Thread and Process do take slightly longer than Serial due to their own overhead but the only prominent exception is ProcessThread for Hard. This is due to it having the combined overhead of both Process and Thread. Their overhead involves creation and termination of processes and threads, context switching, and synchronizing the mutexes.

2.3 System time performance differences (in seconds)

	Serial	Process	Thread	ProcessThread
Hard	0.005	0.014	0.017	0.0392

Table 3: System time performance differences based on text input complexity.

System time tends to be very inconsistent, but these were collated over 5 executions of each file with the Hard text input. This simply shows that you spend more time in the Kernel when you have more overhead. ProcessThread spends the most time here by far due to the combination of waiting for threads to finish inside of the children, then waiting for the children as well.

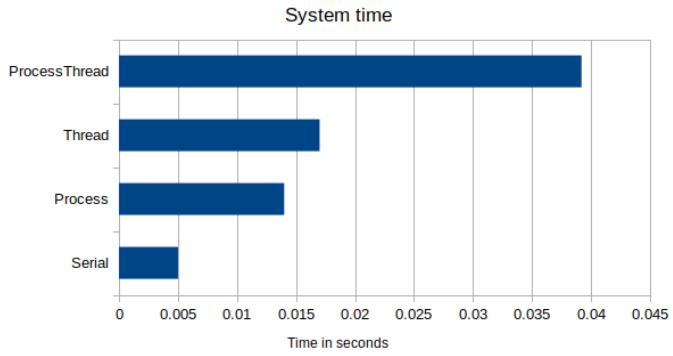

Figure 3: Graph of system time speed difference based on text input complexity

3 Conclusion

In conclusion, Serial is the slowest. It is the naive approach where you sequentially deal with each equation, brute forcing it. The differences between Thread and Process are less pronounced; Process is better for hard tasks where the heavyweight nature of Process is less apparent due to the sheer amount of time needing to calculate whereas constant synchronization issues might severely slow down Thread. ProcessThread is the fastest, while it does have some issues where it can be slowed down due to the nature of threading with values that need to be synchronized, it still has the best of both implementations and I would not expect it to ever be slower than Process outside of incredibly niche examples where threading could somehow be slowing it down via poor implementation.

References

[1] Baeldung. "Process vs thread in linux." Accessed: 2023-10-17. (2023), [Online]. Available: `https://www.baeldung.com/linux/process-vs-thread`.

[2] U. of Canterbury. "L2 threads." Accessed: 2023-10-17. (2023), [Online]. Available: `https://learn.canterbury.ac.nz/pluginfile.php/6558797/mod_resource/content/15/L2%20Threads.pdf`.

[3] IBM. "Benefits of threads." Accessed: 2023-10-17. (2023), [Online]. Available: `https://www.ibm.com/docs/en/aix/7.3?topic=programming-benefits-threads`.

[4] J. Preshing. "Locks aren't slow; lock contention is." Accessed: 2023-10-17. (2011), [Online]. Available: `https://preshing.com/20111118/locks-arent-slow-lock-contention-is/`.