OPERATING SYSTEMS

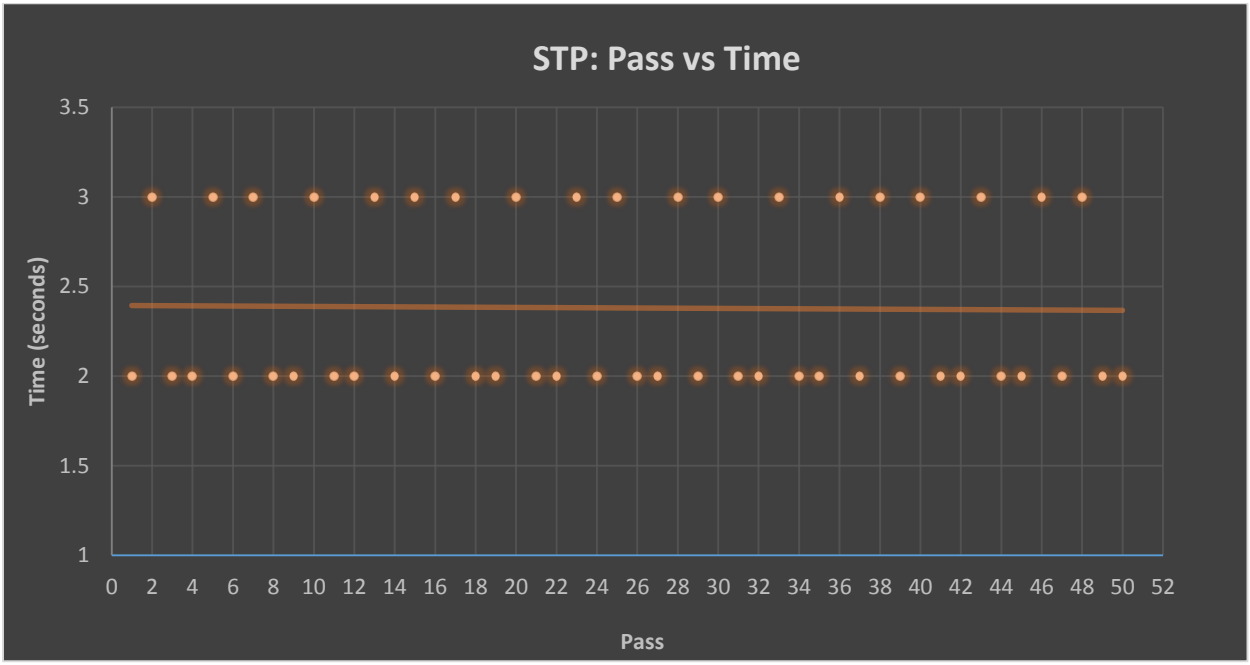
Assignment – IMPACT ANALYSIS

Gursimran Singh __ 2014041

Single Threaded Program:

- Source: 14041_stp.c
- Counter Used: Counter 1 __ Non Thread Safe
- Threads Created: 1
- Times Repeated: 50
- Average Time Taken: 2.38 seconds

The following graph plots Time taken per Pass vs. Number of Pass and shows the average Time taken:



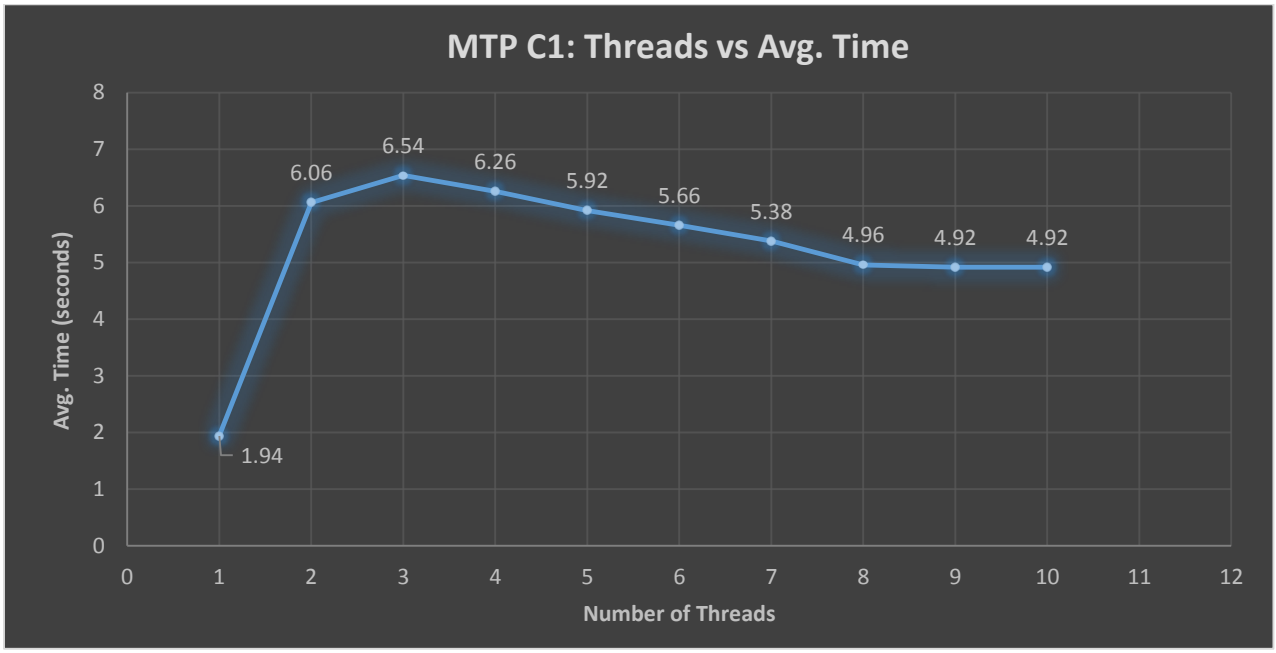
Observations:

- Correctness:
  - Single Threaded program produces correct output every time (as expected) since concurrency is not a problem here.
- Performance:
  - Single Threaded program also performs nicely since it needs not to acquire any locks to manipulate the data.

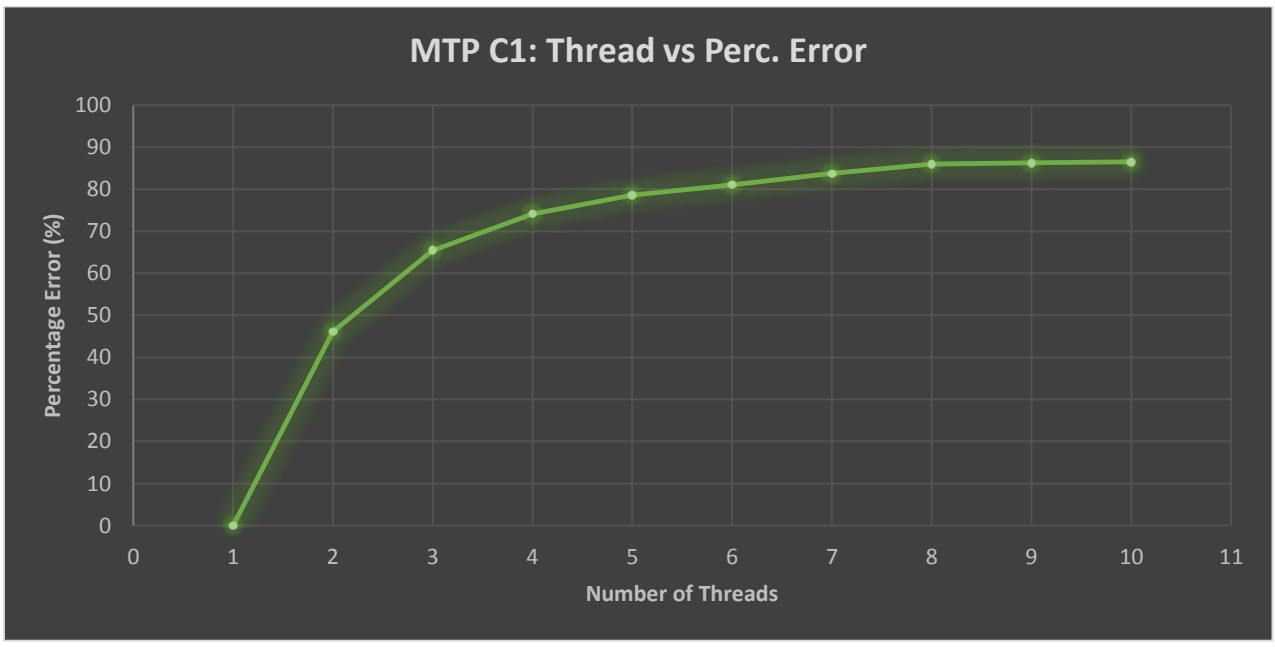
Multi-Threaded Program __ Counter 1:

- Source: 14041_mtp_c1.c
- Counter Used: Counter 1 __ Non Thread Safe
- Threads Created: Varied from 1 to 10
- Times Repeated for all No. of Threads: 50

The following graph plots Number of Threads vs. Average Time Taken to execute the program 50 times:



The following graph plots Number of Threads vs. Percentage Error in counting the value:



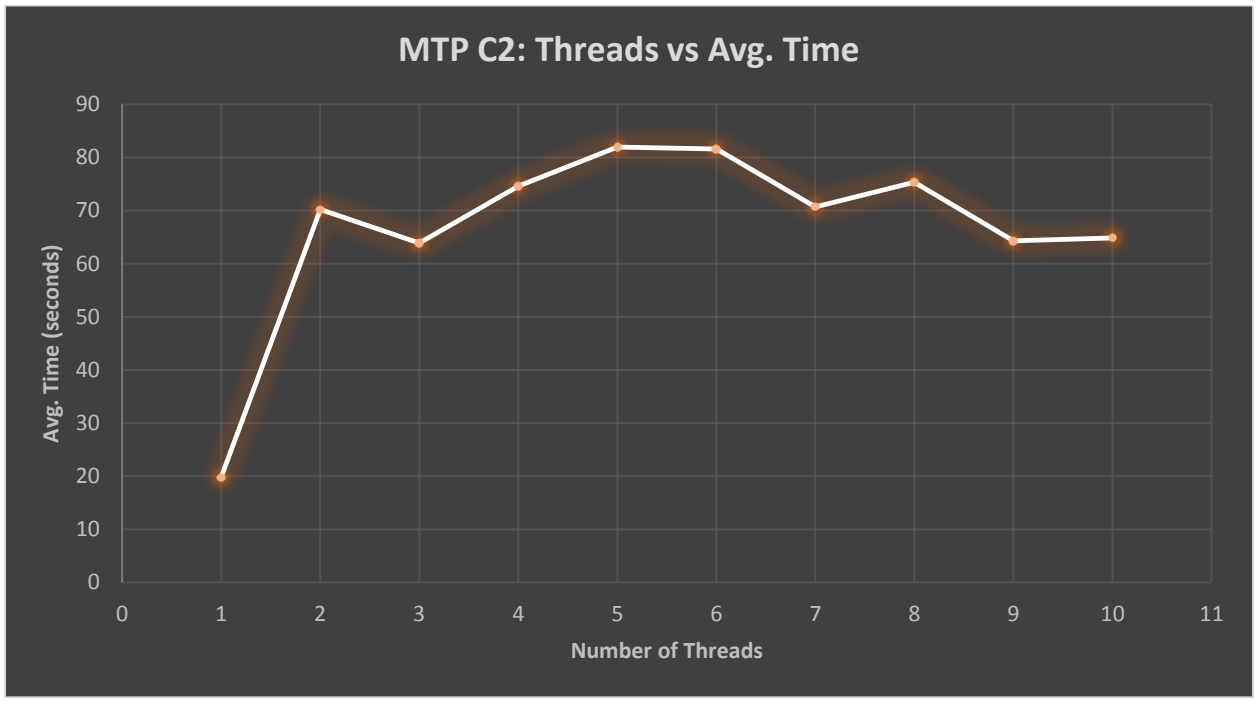
Observations:

- Performance vs. Number of Threads:
  - On an average Performance decreases as the Number of Threads increases.
  - But peculiarly Time taken decreases to an almost constant value when number of Threads increase further. But this attainment of a peak value may only be due to random scheduling.
- Correctness vs. Number of Threads:
  - MTP_C1 produces correct output when number of threads is 1 because then it mimics STP_C1 but surprisingly on an average takes lesser time than STP_C1.
    MTP_C1 (1 thr.) = 1.96 sec STP_C1 = 2.38 sec
  - As the number of threads increases from 2 to 10 MTP_C1 always produces the wrong output and this error keeps on increasing (at a decreasing rate) with the number of threads as is clearly shown by Graph II (MTP C1: Thread vs. Perc. Error).

Multi-Threaded Program __ Counter 2:

- Source: 14041_mtp_c2.c
- Counter Used: Counter 2 __ Thread Safe (Mutex Lock)
- Threads Created: Varied from 1 to 10
- Times Repeated for all No. of Threads: 50

The following graph plots Number of Threads vs. Average Time Taken to execute the program 50 times:



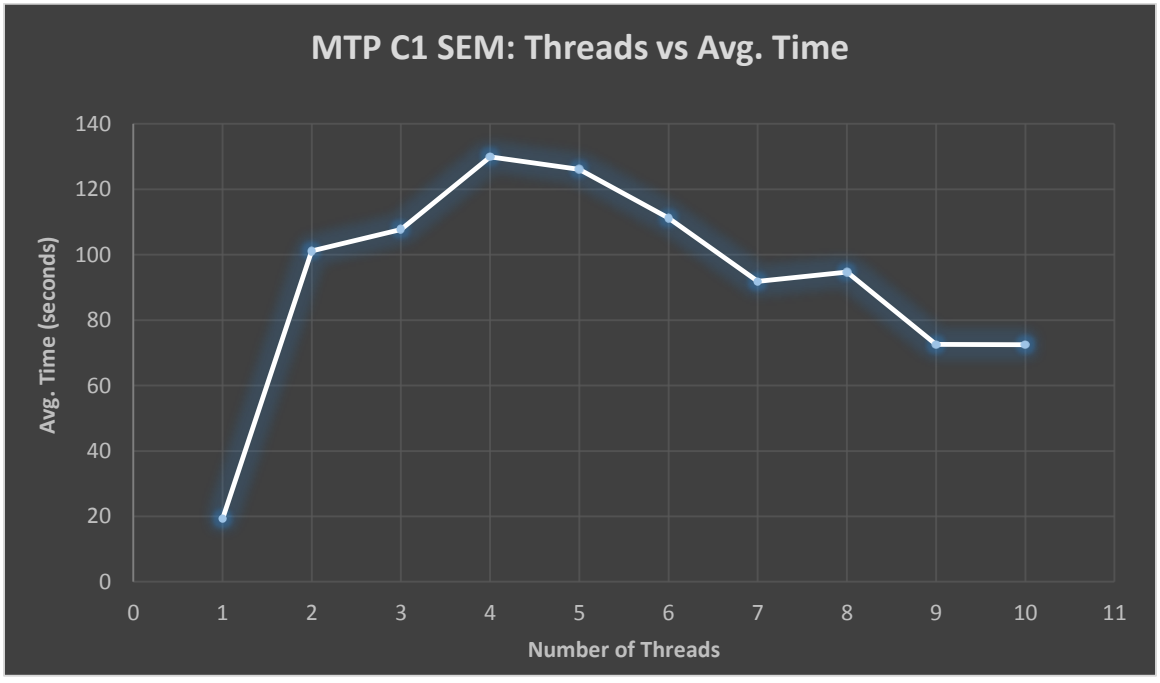
Observations:

- Performance vs. Number of Threads:
  - MTP_C2 takes a lot more time to execute than MTP_C1.
  - As the number of threads increase on an average the performance suffers. But again there are some outstanders in this pattern too. Sometimes performance graphs twitches sue to random scheduling but still it can be said that the decay in performance becomes almost constant as the number of threads increase from 2 to 10.
  - The best performance is when only 1 thread is created as it mimics STP_C1 and MTP_C1 with 1 thread but we can see the overhead of managing the lock is too big.
    STP_C1 = 2.38 sec MTP_C2 (1 thr.) = 19.76 sec
- Correctness:
  - MTP_C2 always produces the correct output as expected no matter the number of threads.

Multi-Threaded Program __ Counter 1 __ Semaphores:

- Source: 14041_mtp_c1_sem.c
- Counter Used: Counter 1 __ Thread Safe (Semaphores)
- Threads Created: Varied from 1 to 10
- Times Repeated for all No. of Threads: 50

The following graph plots Number of Threads vs. Average Time Taken to execute the program 50 times:



Observations:

- Performance vs. Number of Threads:
  - MTP_C1_SEM runs the slowest of all the three implementations.
  - MTP_C1_SEM with only 1 thread performs exactly the same as MTP_C2 with 1 thread taking 19.36 seconds.
  - But as the number of threads increase the Time consumption of MTP_C1_SEM increases suddenly to a peak of almost 129.96 seconds and then again drops to achieve an almost constant value of about 70 seconds.
- Correctness:
  - MTP_C1_SEM always produces the correct output as expected no matter the number of Threads.

FINAL THOUGHTS:

- Single Threaded Program always produced the correct result and performed better than all else.
- In a multithreaded environment considering concurrency is very important otherwise extremely incorrect outputs may be produced as in case of MTP_C1 reaching an error percentage of 87% with only 10 threads.
- Between Mutex Locks and Semaphores, Mutex clearly outperformed Semaphores with an average time difference of approx. 26.022 seconds.
  The reason for this may be:
  Mutex is a lock, therefore only one thread may acquire the lock at one time. Whenever a mutex is unlocked other thread is free to acquire the thread as soon as it is scheduled (providing no other thread acquires the lock first).
  Whereas in case of Semaphores, whenever a post occurs a signal is sent and the ISR is responsible for waking up one of the sleeping threads. This is slower than the thread just being rescheduled and checking for mutex availability.