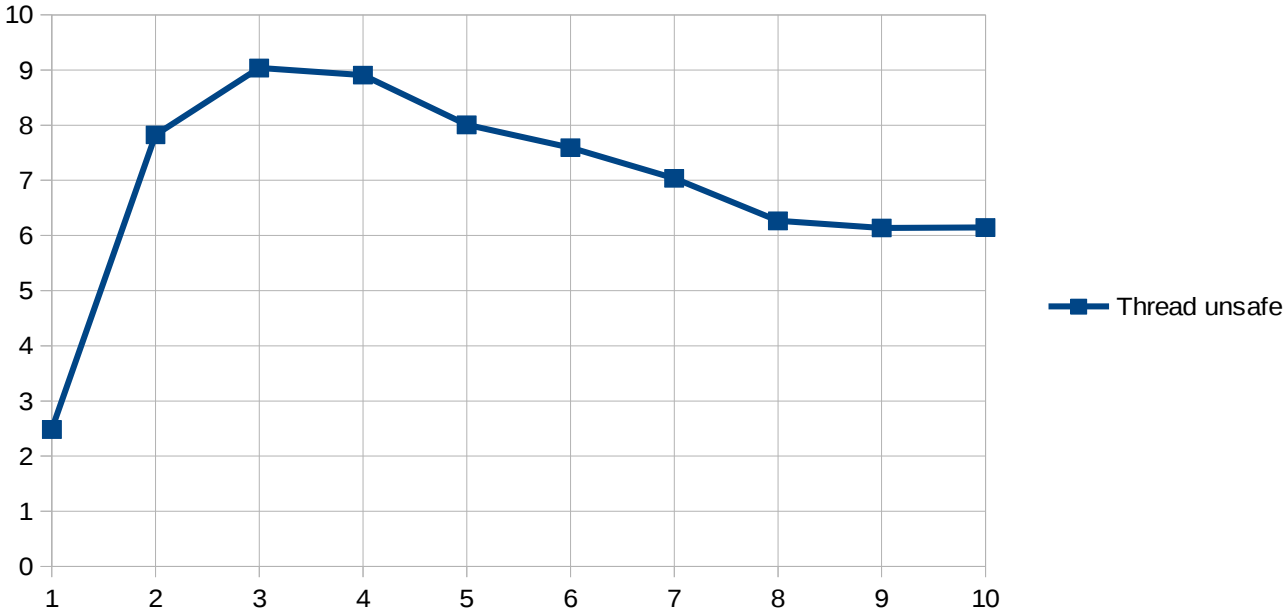
# CSE231 (Operating Systems) Assignment 4 : OS Impact Analysis

*Anshuman Suri (2014021)*

All of the counters were made to count up to $10^9$ (averaged over 50 iterations to account for other running programs) using 'x' threads, where 'x' was varied in [2,10]. The following are the results (data plots with their analysis).All tests were run on a machine with 8 virtual cores @ 2.8GHz (Turbo Boost upto 3.2GHz)

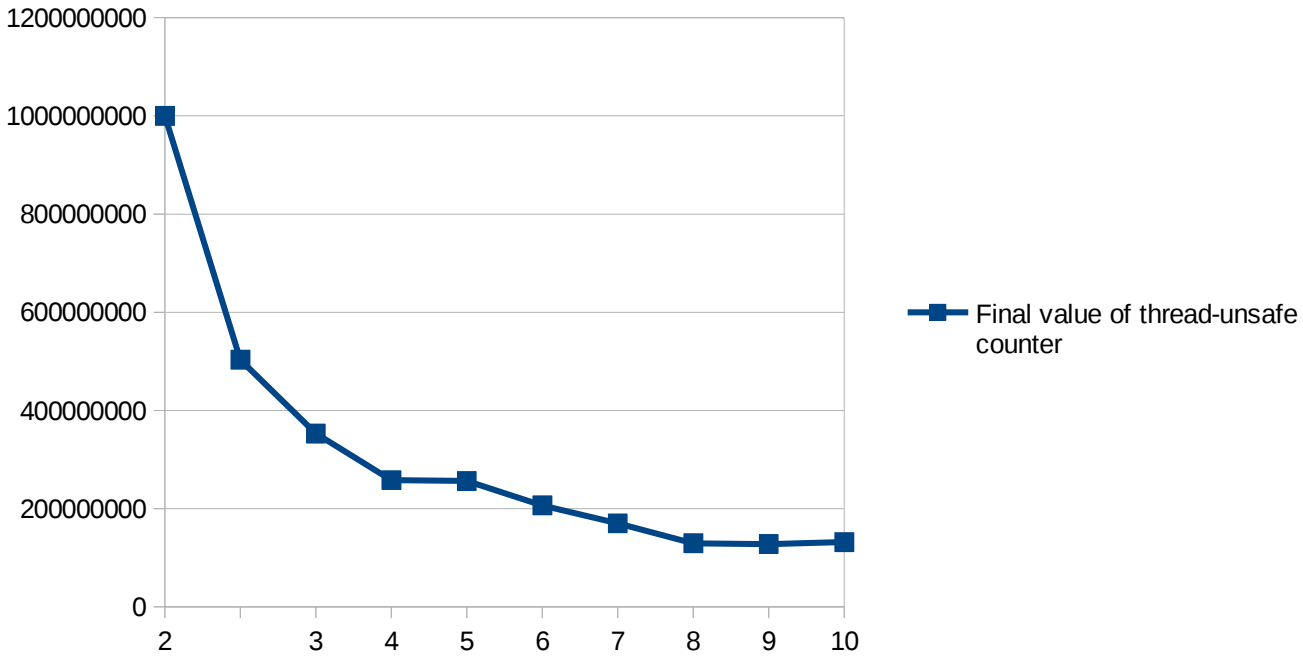
**Thread non-safe :**

The data plot for number of threads v/s time taken (in seconds) suggests that the time taken to count upto a given value using multiple threads first rises upto a maximum and then gradually comes down (stagnating around the number of cores of the machine running the steps).



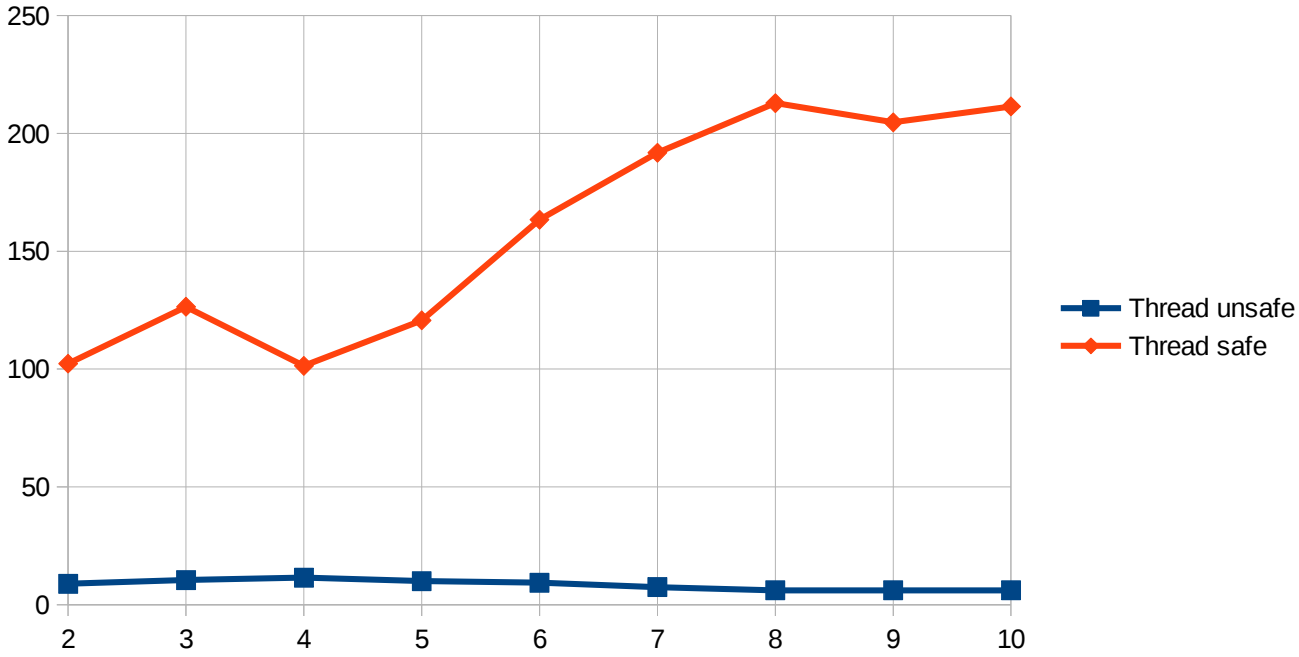

As indicated by the graph,the time taken for counting turns out to be surprisingly less than multi threaded counters! This is because the overhead of creating processes and switching between them is much more than the time taken to finish the thread.So, there is an initial rise of execution. The stagnation in running time after 8 threads is because of the running system's CPU having 8 cores.

The below data plot of the final value of the counter v/s the number of threads shows a steady decrease in the counter's value with the number of threads.The decrease in value in a counter without locks is because of the fact that the increment (++) operation isn't atomic and pre-emptive calls in between executing this instruction will lead to loss of data (in terms of increments). However, the final value is a lot less than the correct value ($10^9$). Hence, a process may fetch the value of the counter into a temporary register, increment the temporary register and just before it stores the value to the temporary register, be interrupted by the scheduler, scheduling another thread. In the worst case,this would happen for all the threads,leading to a minimum value of $(10^9)/N$ , where 'N' is the number of threads. In fact,the final value of the counter (average) comes out to be close to this value.Therefore , the performance increases with time (after a specific number of threads,i.e. once the overhead is covered) whereas the correctness decreases.




**Thread safe :**

The data plot for number of threads v/s time taken (in seconds) suggests that the time taken to count upto a given value using multiple threads more or less increases with the number of threads (achieving a minima ). As is visible in the plot, the time taken for a thread safe counter is much higher than that taken by a thread safe counter.




With increasing number of threads, the work to be done per thread keeps decreasing.However, because there are multiple threads running at the same time and the increment is thread-safe, only one thread ends up updating the common value,while the others wait for it.So, increasing the number of threads does not decrease the execution time (like it was for thread unsafe).There are two factors which determine the time taken for a thread safe counter : the overhead of creating threads and calling lock() and unlock() again and again, and the load on each thread. Increasing number of threads reduces the load on each thread (in terms of time complexity).However, increasing number of threads

means more threads are put to waiting state by the scheduler when the lock is owned by another thread. From 2 to 3 threads, the overhead factor dominates.From 3 to 4, the load balancing factor dominates. After that, the overhead balancing factor dominates. Stagnation around 8 threads is because the machine can run only 8 threads at a time( octa-core). Thus,we can conclude that for the given machine (or for any machine with a similar scheduling algorithm and similar number of cores), using 4 parallel threads is the optimal solution in terms of performance. (Correctness is ensured for any number of threads,as we have used locks).So, increasing the number of threads leads to a decrease in performance, and no change in the correctness ( correct for all values).