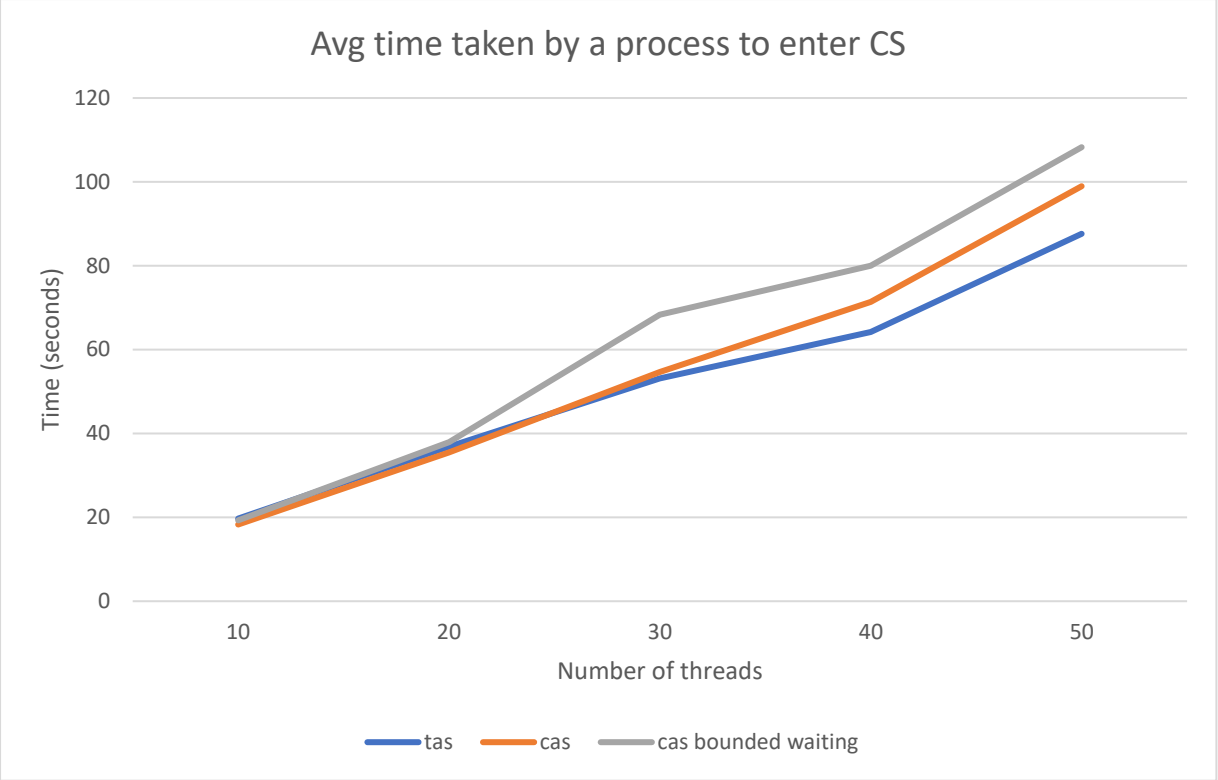
For **Test and set with mutual exclusion**, I have used the inbuilt function test_and_set() provided by c++ atomic library in the entry-sec() function and set the lock back to 0 in exit-sec() function. And testCS() function calls entry-sec() and exit-sec() in entry section and exit section of the code respectively. Lock is initialised to 0 at the start. And the first thread that invokes test_and_set() will set the lock to 1 and it will enter into its CS and since lock is 1, no other process can enter CS. After that process completes execution, it will set lock back to 0 in its exit section. And other process can enter CS now. Thus, mutual exclusion is maintained within threads.

For **compare and swap with mutual exclusion** also I have done the same thing but instead I used compare_exchange_weak() function which is available in atomic library in c++. This also ensures mutual exclusion within threads.

For **compare and swap with mutual exclusion and bounded waiting**, I have created a bool waiting array which is initialised with false, and lock is initialised to 0. For mutual exclusion, thread i can enter its CS only if waiting[i] is false and key equals to 0. And key becomes 0 only when compare_exchange_weak() is executed. First process executes compare_exchange_weak() and key becomes 0, while the other threads have to wait. And waiting[i] becomes false in the exit section when a thread completes its execution in CS, and thus mutual exclusion is maintained.

For bounded waiting, after a thread completes execution of CS, it scans the waiting array in a cyclic order and allows the first process which has waiting[j] = true to enter the CS. Thus, any process waiting to enter CS will enter within n-1 turns.



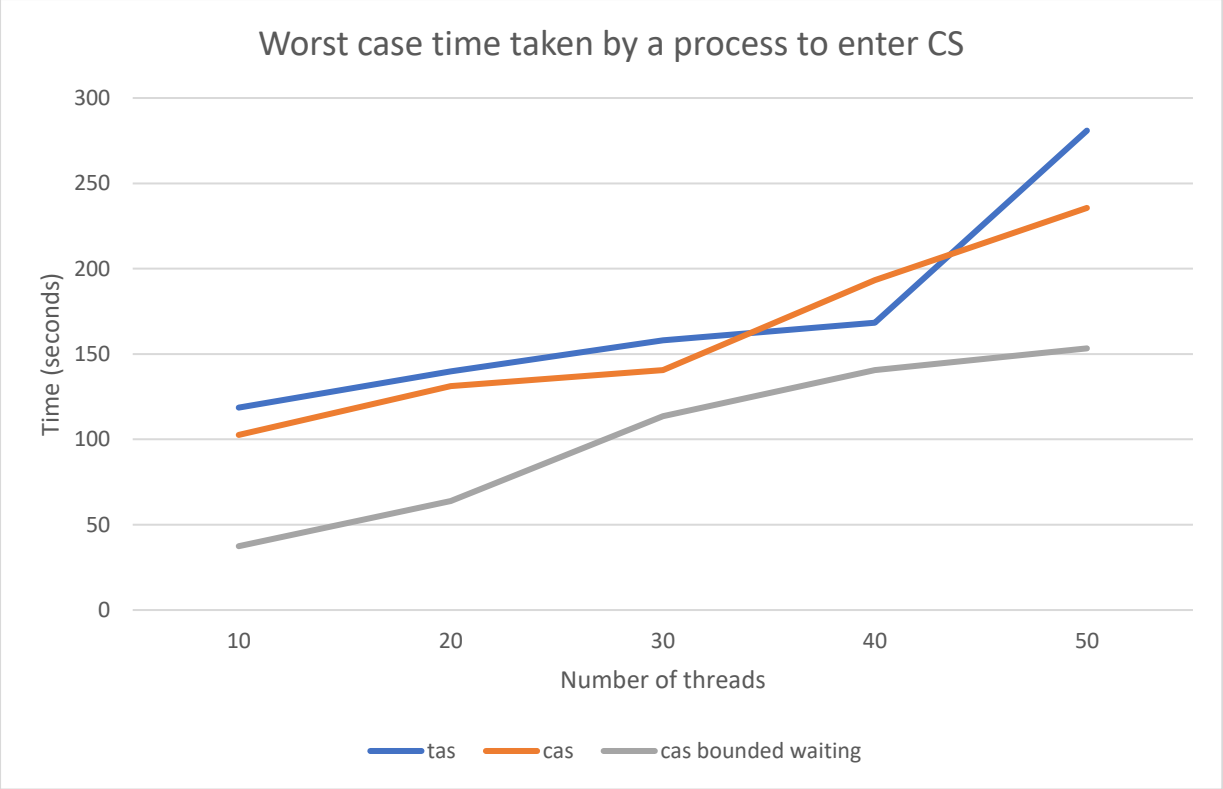

The x-axis varies the number of threads.

The y-axis shows average time taken to enter the CS by each thread.

Here I have taken k = 10, lambda1 = 3 and lambda2 = 6

We can see that for n = 10 and 20 the average time taken in all the three are almost same, and with increase in number of threads, cas bounded waiting has more value in comparison with cas and tas.

Cas bounded waiting has more average time taken as it happens in a cyclic manner, and this causes some extra time.




The x-axis varies the number of threads.

The y-axis shows the worst-case time taken by a process to enter the CS in a simulation.

Here I have taken k = 10, lambda1 = 3 and lambda2 = 6

In this we can see that the worst-case time is the least for cas bounded waiting as it ensures all the threads to execute in a cyclic way, but this does not happen for cas and tas.

Here also cas and tas have approximately equal worst-case waiting time.