Operating Systems 2 – CS3523

Programming Assignment 4 – Report

Suraj Telugu – CS20BTECH11050

Algorithm:

- **Test And Set (TAS):** This algorithm uses atomic_flag lock variable which is initalised to false and test_and_set() function from atomic_flag constructor serves as lock to ensure the mutual exclusion property of threads to enter critical section. lock.clear() sets the lock variable again to false and completing its execution in critical section.

- **Compare And Swap (CAS):** This algorithm uses atomic_compare_exchange_strong function in atomic library (works similar to atomic compare and swap) making it as a lock to ensure the mutual execution for threads to enter critical section.

- **Compare And Swap Bounded (CAS-Bounded):** This algorithm uses a waiting array and the same atomic_compare_exchange_strong function as mentioned above. This algorithm uses waiting array to ensure both mutual execution and bounded property satisfied for threads entering critical section.

Design and Implementation:

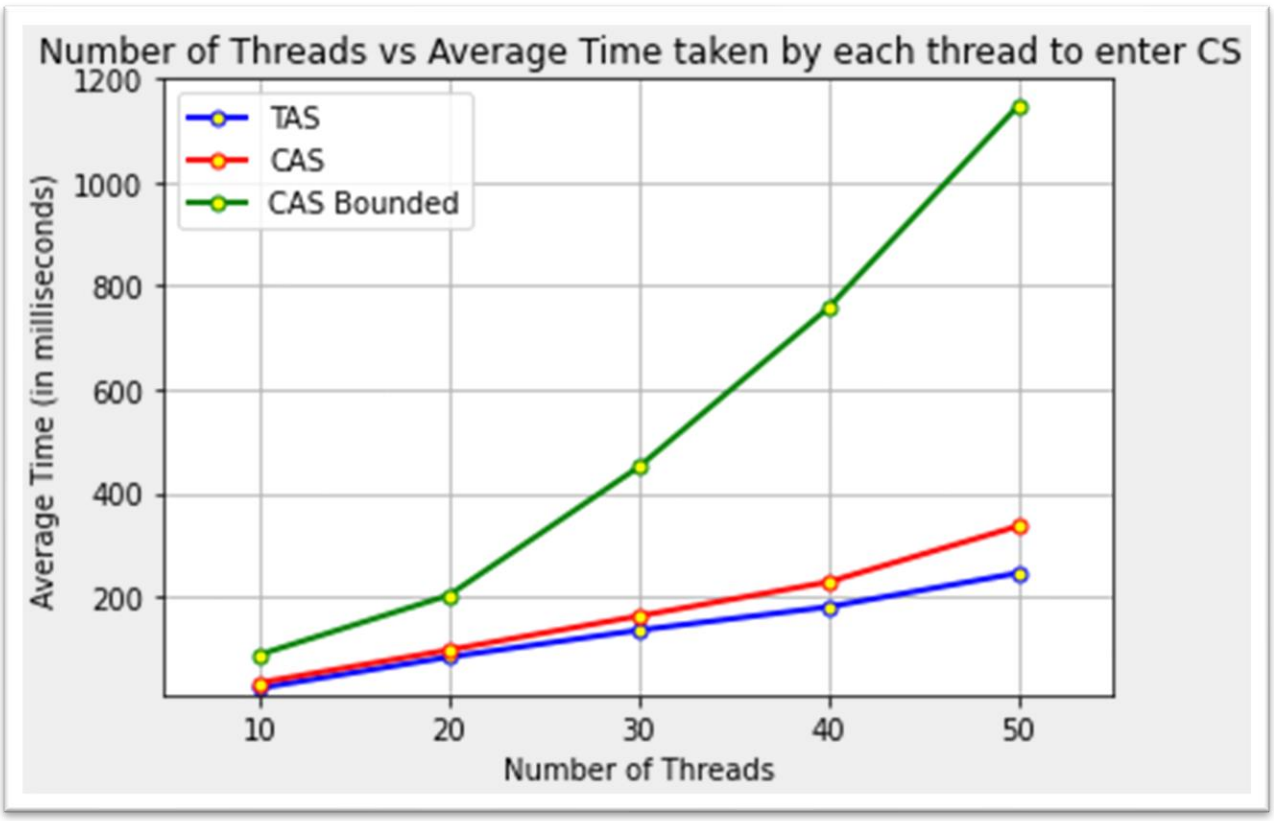
- All global variables (inputs and outputs) are defined. thread_id is defined as atomic_int so that it does not get raced between threads. atm_lock is defined as atomic_flag variable for TAS and atomic <bool> for CAS and CAS_Bounded it is initialised to false.

- The SysTime() gives present time with a precision of microseconds using ctime() function to get up to seconds precision and gettimeofday() function from sys/time.h library is used to get milliseconds and microseconds at that particular instant. Mutex lock is used so that no two processes race their SysTime() functions given wrong output.

- The PrintMessage() function prints the requested, entered, exited message into file as per print specifier (char c) the thread id, loop number (i), time are taken as input and printed accordingly as per description. Loop numbers ended with 1, 2, 3 are given 'st', 'nd', 'rd' specifically and remaining numbers are given 'th' at their end.

- In TestCS() function the Critical Section is implemented. First requested time is stored in a string before critical starts. The TAS, CAS, CAS Bounded are implemented as per above algorithms. Times are obtained using SysTime() function and printed into file.

Graph and Analysis:



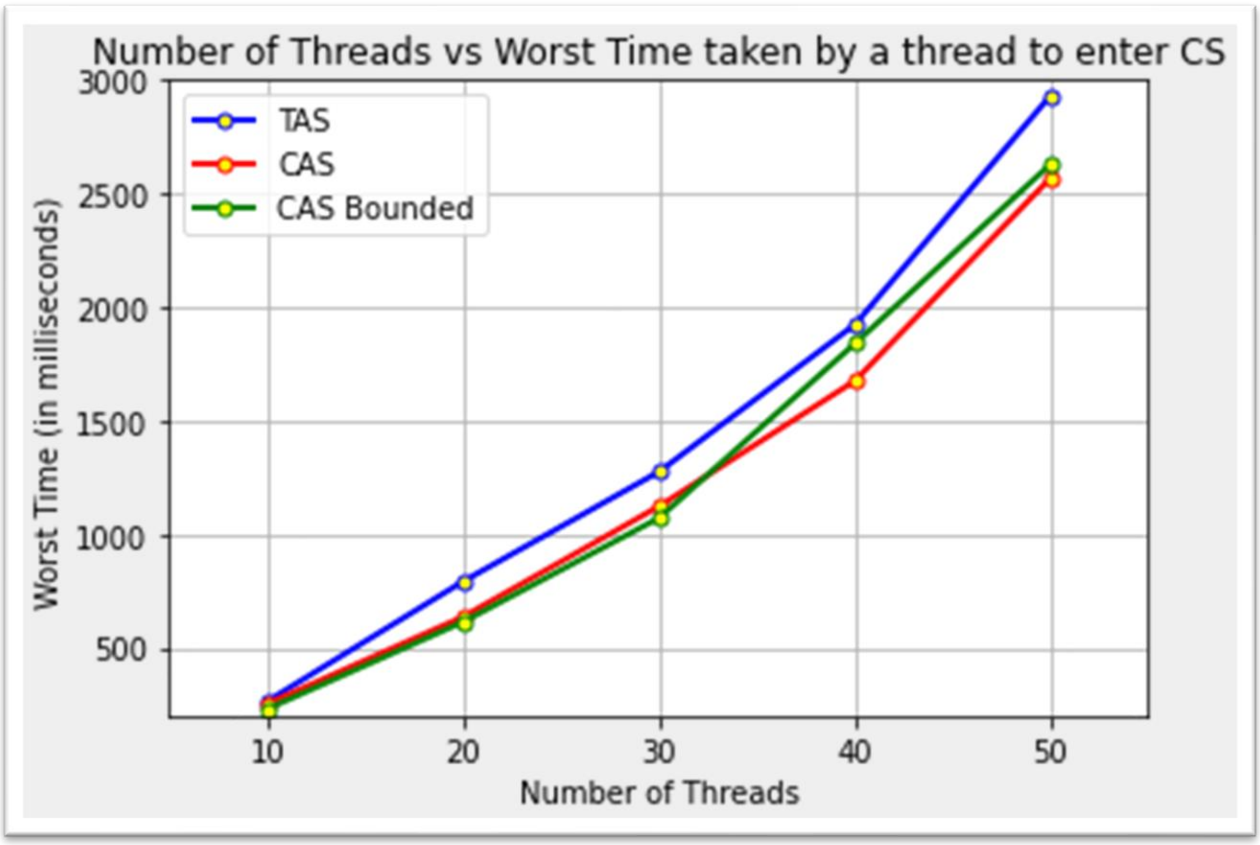
Graph Input:

Number of Threads (n) = 10 to 50 **Number of times each thread repeats (k)** = 10

Exponential Distribution Averages: λ1 = 5 λ2 = 20

Analysis and Conclusion:

- Average waiting time taken by each thread to enter the critical section increases as the number of threads increases for all the three algorithms.
- The Average time indicates the average starvation time of particular thread. It is nearly same for TAS and CAS but for CAS Bounded average time is larger than TAS and CAS.
- As the number of threads increases the difference between the average times of CAS Bounded and TAS and CAS keeps on increasing.
- The Average waiting time of CAS bounded is very high so its performance is low whereas TAS has less average time compared to CAS and CAS Bounded as number of threads increases hence its performance is better than both in terms of average waiting time.



Graph Input:

Number of Threads (n) = 10 to 50 **Number of times each thread repeats (k)** = 10

Exponential Distribution Averages: λ1 = 5 λ2 = 20

Analysis and Conclusion:

- Worst time (maximum waiting time) taken to enter the critical section increases as the number of threads increases for all the three algorithms.
- The Worst time indicates the maximum starvation time possible by that algorithm. It is least for CAS Bounded and highest for TAS for lower number of threads.
- As the number of threads increases the worst time for CAS Bounded increases and for higher number of threads it is least for CAS and highest for TAS.
- The worst time for TAS is larger compared to CAS and CAS Bounded as number of threads increases so its performance is least with respect to worst case time. For less number of threads CAS Bounded gives the best performance among the three algorithms, as the number of threads increases CAS Bounded performance decreases and for higher number of threads CAS algorithm gives best performance.

Conclusion:

- If the threads should have less average starvation even though if a particular thread has larger starvation, then TAS Algorithm gives better performance compared to CAS and CAS Bounded algorithms.

- If the threads should have less maximum starvation even though it has larger average starvation time then CAS Bounded Algorithm gives better performance compared to CAS and TAS for smaller number of threads and CAS algorithm gives better performance compared to CAS Bounded and TAS for higher number of threads.