Operating Systems II : CS3523
Spring 2019

Programming Assignment 2 : Implementing TAS, CAS and Bounded Waiting CAS . Mutual Exclusion Algorithms

Instructor : Dr. Sathya Peri
Author : Puneet Mangla (CS17BTECH11029)

Design of Program :

We implemented **TAS (test and set) , CAS (compare and swap) and CAS-Bounded** mutual exclusion algorithms.

Entry Section

This is the section of code is available to all threads .To enter the critical section threads have to go through this section. If one thread is already in critical section then all other threads wait in entry section. **Bounding waiting** ensures that no threads waits in the entry section forever.

Critical Section

This section of code is available to only one thread at a time . It contains accessing / reading the shared variables between thread.

Exit Section

After a thread completes the critical section it goes to exit section where it notifies all threads (by unlocking lock) that it is out of the CS and any one of the waiting threads can go into the critical section.

Remainder Section

This section comes after exit section and all threads can access this section . There are no concurrency conflicts in this section.

Comparison metrics :

1. **Worst case waiting time :** the worst case time taken by a threads to enter the CS in a simulation. This shows if threads are starving.
2. **Average waiting time :** the average time taken by a thread to enter the CS.

Algorithm :

1. The main thread reads the "`inp-params.txt`" and stores all the parameters as global for all theads to be accessible.
2. `N` threads are created and their `id` are stored in array `pthread_t tid[n]` and pointer to appropriate function is passed in `pthread_create`.
3. `default_random_engine` is used to generate random numbers from `exponential_distribution` to stimulate critical and remainder section . Parameters for both critical and remainder section were $1/\lambda_1$ and $1/\lambda_2$ respectively.
4. `void* testTAS/CAS/CASBounded` fucntions stimulate critical and remainder section by sleeping thread for random seconds generated from above mentioned distributions.
5. `lock` is of type atomic for perfroming atomic operations on lock.
6. TAS used `lock.test_and_set()` fucntion for implementing ME whereas CAS and CAS bounded use `lock.comapre_exchange_strong()` for implementing ME.
7. CAS Bounded has an extra shared array `waiting` that keeps track of which thread is waiting. It ensures bounded waiting

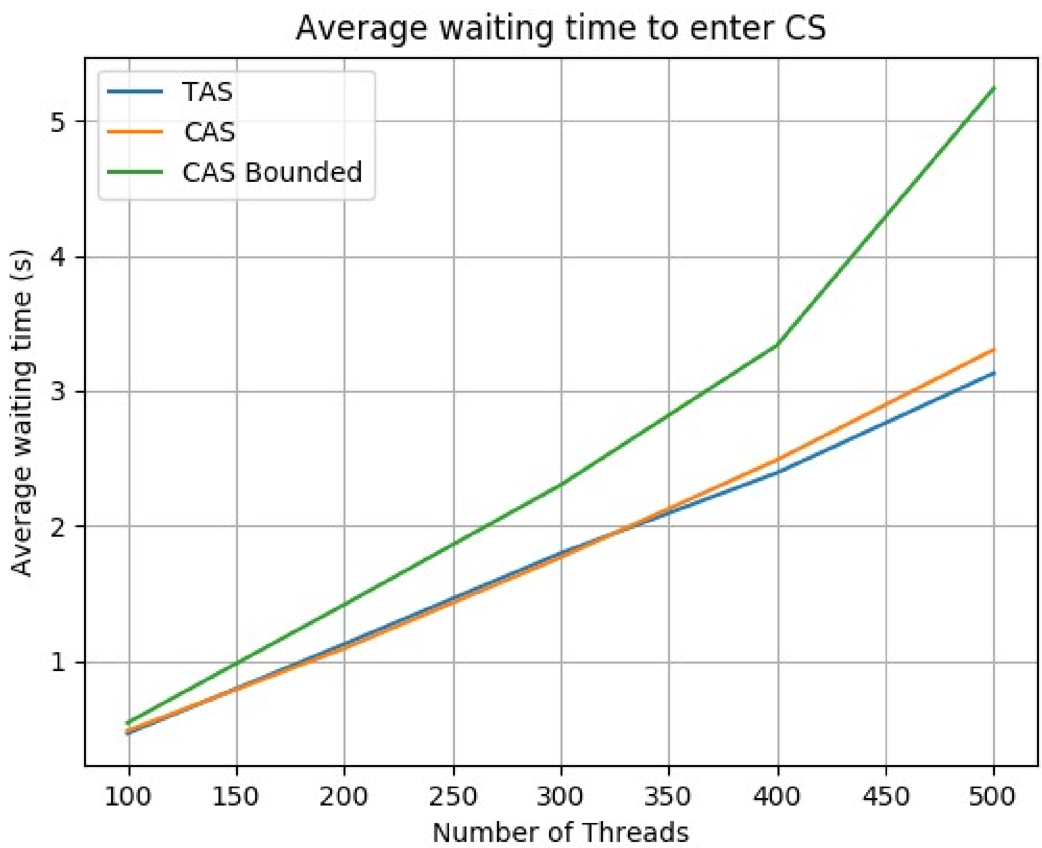
time for each thread. `waiting` array is kept atomic for dealing with concurrency issues.

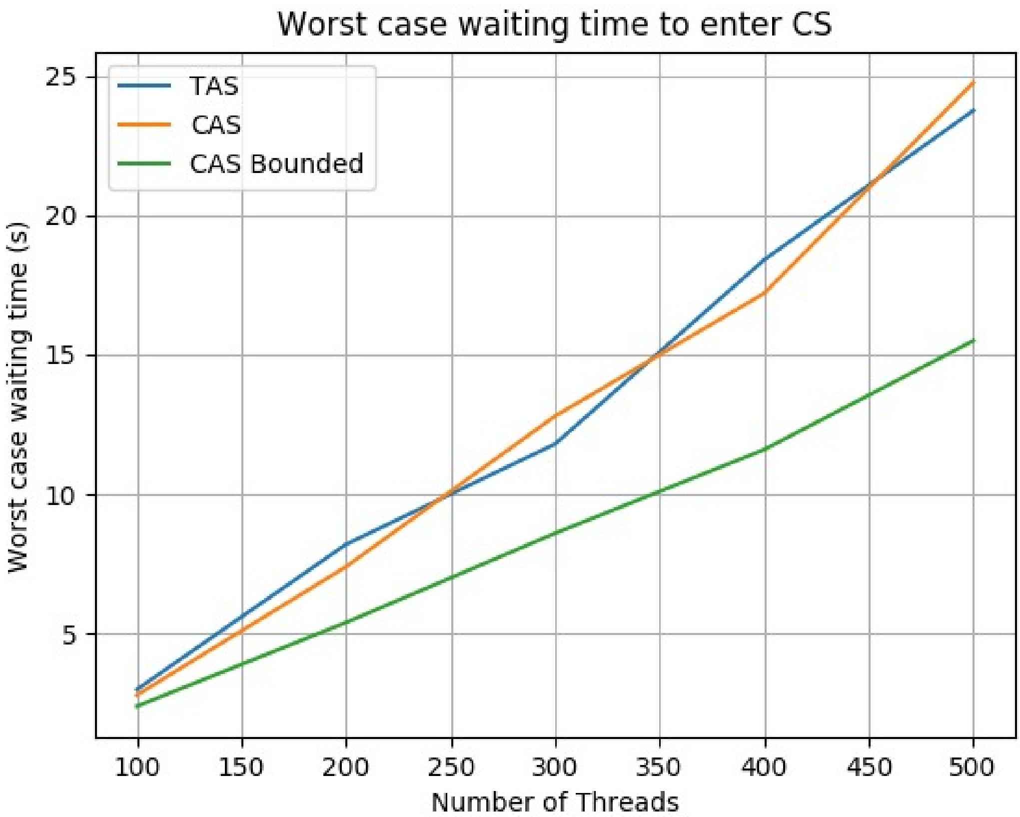
8. For printing logs `fprintf/printf` are used instead of `cout` as `cout` streams causing mixing of logs.

Graphs :

For comparison of different algorithms we varied the number of threads from 10 to 50 while keeping other parameters same. Avergaing and worst waiting time are used as comparison metrics.

$$\mathbf{k = 10}$$
$$\boldsymbol{\lambda_1 = 0.1}$$
$$\boldsymbol{\lambda_2 = 0.14}$$





As evident from graphs we can see that the average waiting time of TAS and CAS are very close . CAS average time is slightly greater because of more operations it take compared to TAS.

CAS bounded performs better in terms of worst waiting time as it gives thread almost equal chance to enter CS and thus avoids starvation. Side effects of this can be seen in high average waiting time .

Conclusion :

In terms of avergaing waiting time TAS performs best among all the three ME algorithms whereas CAS-bounded beforms worst .

In terms of worst case waiting time CAS-bounded performs best among all ME algorithms whereas CAS performs worst.