Operating Systems - II: CS3523

Implementing TAS, CAS & Bounded Waiting CAS Mutual Exclusion Algorithms

Assignment Report

Sagar Jain - CS17BTECH11034

February 18, 2019

Contents

Salient Features of Program Design

To implement TAS & CAS, we need their operation to be atomic. In plain CPP we can atmost guarantee the atomic update of a single variable(a single instruction), however we cannot be sure about interleaving of instructions while using a manual implementation of test_and_set(using *mutex* would beat the purpose of the assignment). This is the reason why I have chosen to use the funcitons from ***atomic*** header ***test_and_set***, ***compare_exchange_weak***
The following are the key points involved in designing each of the three programs.

1. For TAS, as mentioned, I have made use of :
   locker.test_and_set(memory_order_acquire) to enter the critical section and ***locker.clear(memory_order_release)*** to exit the critical section. Here locker is of type ***atomic_flag*** which can be updated/read atomically by the aforementioned functions.

2. For CAS, as mentioned, I have made use of:
   locker.compare_exchange_weak(expected,1) to enter the critical section and ***locker.operator=(0)*** to exit the critical section. Here locker is of type ***atomic_int*** which can be updated atomically by the aforementioned functions, ***expected*** is a varibale with value 0.

3. For bounded CAS, the implementation is as follows:

   (a) There is a global array ***waiting*** to store the waiting status of every thread.

   (b) The variable ***key*** is updated by calls to compare and exchange.

   (c) To exit the critical section the running thread then looks for the next waiting thread and sets it value in waiting array to false. This is the thread that runs next.

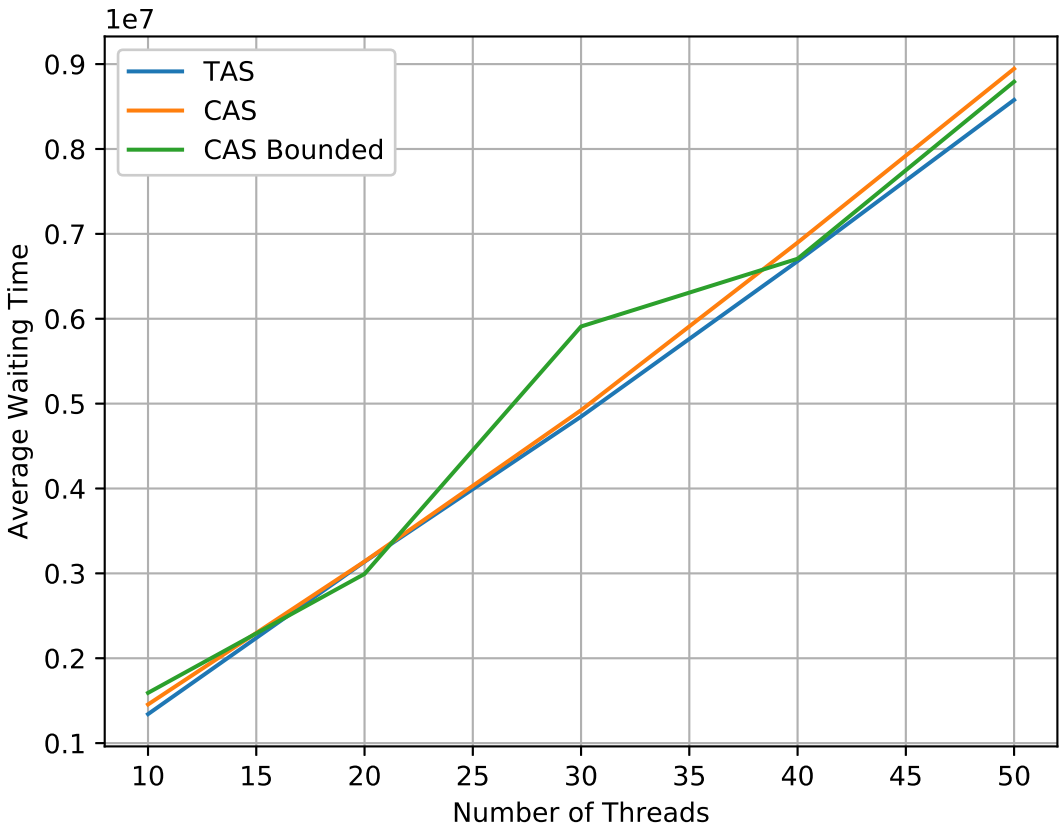
4. To measure the waiting times, I have used a global waiting matrix which is updated whenever a thread enters the critical section.

5. For time measurements I have used ***chronos***.

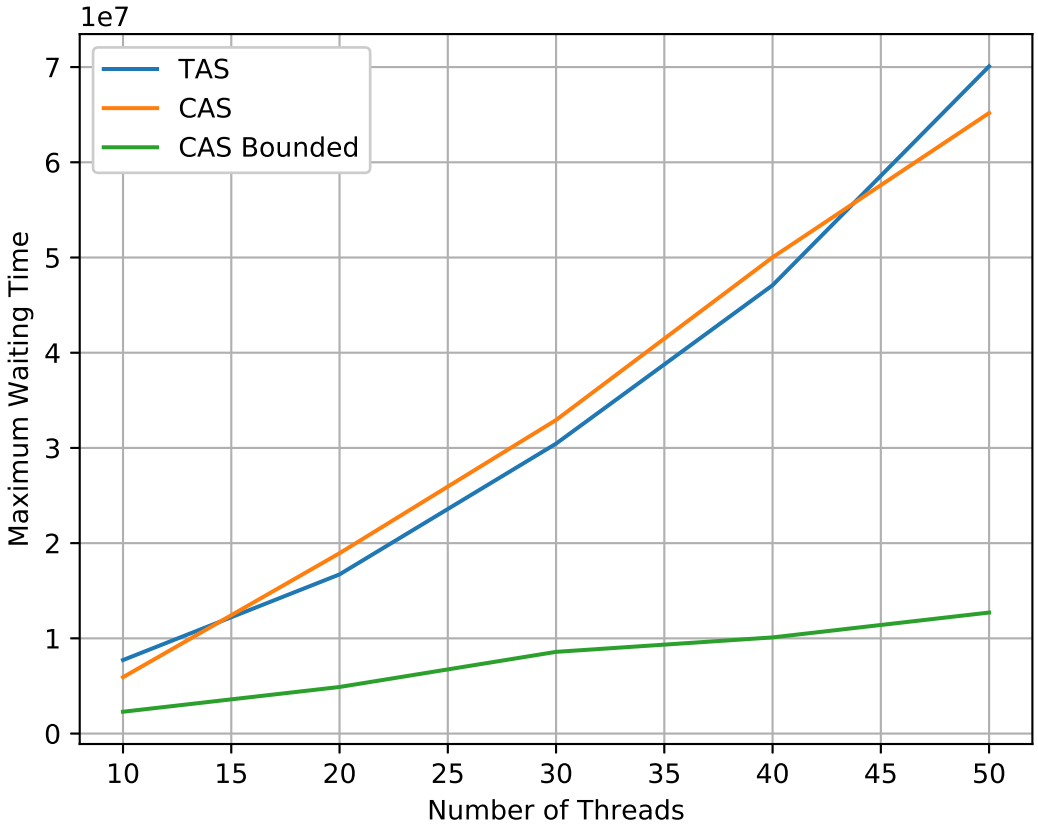
6. For writing to files I have used ***fprintf*** as it does not have concurrency issues.

Results & Graphs

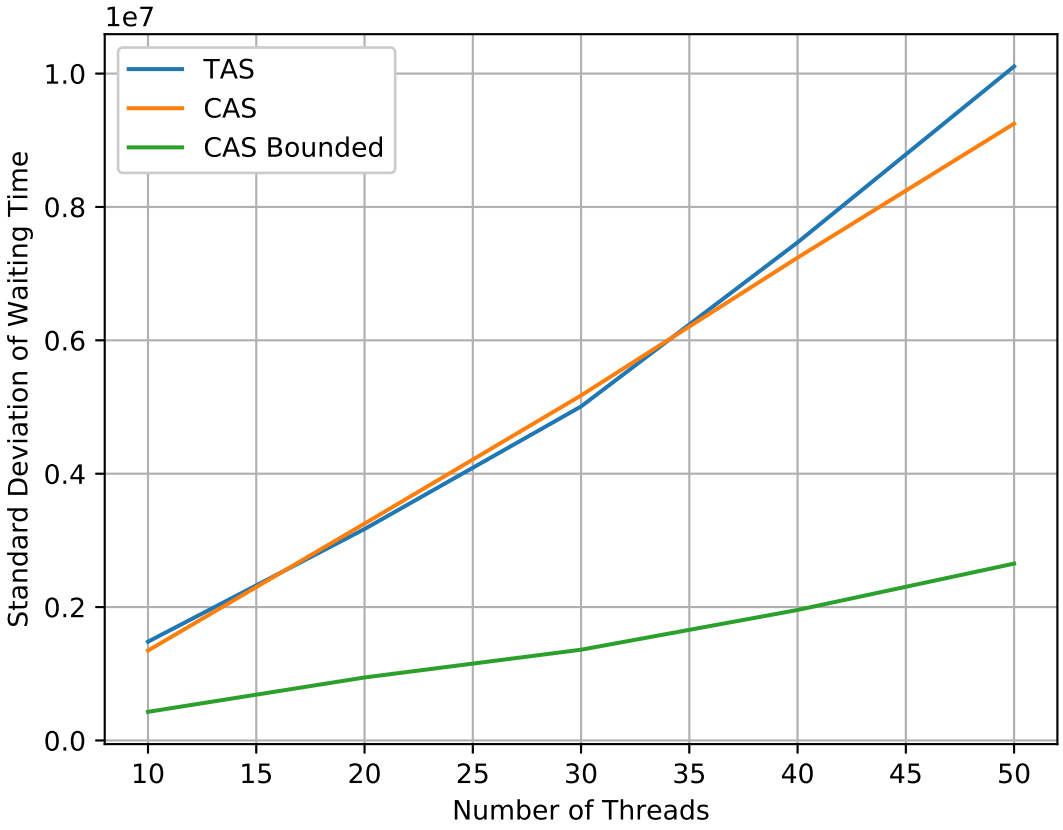
We have three graphs which have been made by running the three solutions for mutual exclusion with varying number of proceses. Each of the following graphs gives insight into an aspect of the solutions.



Maximum Waiting Time vs Number of Threads



Standard Deviation of Waiting Time vs Number of Threads



Explaination of Results

The following is the interpretation of each of the graphs presented above:

1. The average waiting time of threads gives th following insights:
   (a) The average waiting time increases for all the three with increase in the number of threads, this is obvious since each thread would end up waiting for more number of threads.
   (b) The average waiting times for all of the three solutions is similar, albeit slightly higher for CAS and Bounded CAS, this can be attributed to the fact that they involve more instructions when compared to TAS.
2. The maximum waiting time is also not surprising.
   (a) The maximum waiting time increases with the number of threads since now every thread waits for more number of other threads to exit their critical section. This is observed in all the three solutions.
   (b) The maximum waiting time for CAS Bounded is much less compared to TAS and ordinary CAS, this is expected since we explicitly try to avoid starvation in bounded CAS.
3. The standard deviation of waiting time also is as expected, since we have avoided starvation in Bounded CAS, the values for waiting time do not shoot up for any thread and this keeps the variance down, but the other two do not have this provision, which results in some threads having very large waiting times, this leads to a big value of variance.