Operating Systems Programming Assignment 3

CS21BTECH11011

February 24, 2023

1 Code Objective

1. Implementing three programs that perform TAS, CAS and Bounded waiting with CAS mutual exclusion algorithms.

2. Comparing the average and worst case waiting times for the three implementations

2 Code Design

2.1 Input and Output design

1. inputs are taken from a file named input.txt

2. inputs are n , k , lambda1 and lambda2 are obtained from the input.txt file by using C++ file handling

3. The output file contains 3 parts named out_tas.txt , out_cas.txt and finally out_cas_bound.txt for the respective mutual algorthims

4. the output file contains the data related to the log came from the respective algorthims

2.2 Thread design

1. n Threads are created using pthread create function and using for loops .

2. Threads are joined back using pthread join function and for loops

3. The lock variable is made to be atomic for performing atomic operations.

4. All the input variables as declared globally, so that they are accessible to every thread.

5. The values with exponential distribution of lambda1 , and lambda2 are obtained using the inbuilt exponential distribution functions.

2.3 TAS algorithm

1. TAS full form is test and set algorithm
2. Here I used while(locker.test_and_set() which is an inbuilt defined for an atomic_flag variable which is nothing but locker)
3. So here each process will be in a while loop which we defined above until the locker is available
4. after any process coming from the while loop the process indirectly goes into critical situation
5. we calculate the waiting time after just entering the critical section and pushed it into the vector defined globally named meantime
6. after coming from the critical section I am setting the locker free so any process in that while can come from while loop
7. Now in the current process the remainder section will start and eventually completes
8. The respective information whatever we got from here will be stored into an output file for our reference

2.4 CAS algorithm

1. CAS full form is compare and swap algorithm
2. here I have defined an atomic_bool variable and named it to locker for future reference in globally
3. and also a local bool variable named k which is set to false
4. then used an inbuilt function like !atomic_compare_exchange_strong(&locker, &k, true) for the while loop condition and inside while loop k = false
5. so what happens is when a process arrives it will be in that while loop and when the locker is set to flase one of the process which are in while loop will go into critical section and set the locker to true
6. we calculate the waiting time after just entering the critical section and pushed it into a vector defined globally under the name meantime which will be used for various purposes
7. then we do required calculation and exit critical section
8. Then we release the lock by changing the locker variable to false
9. Then remainder section for the current process will be eventually started and will be exites after some time
10. The respective information whatever we got from here is stored into an output file and later used for our reference

2.5 Bounded waiting CAS algorithm

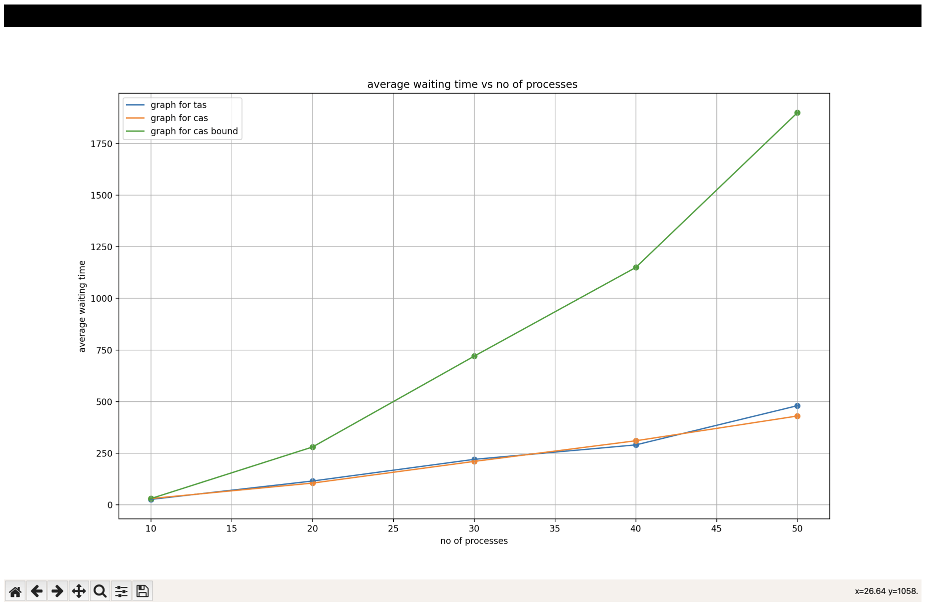
1. like the name suggets the full form is bounded waiting time with compare and set algorithm
2. The main idea used here is taken directly based on test book
3. The bounded_waiting_compare_and_swap algorithm here has been implemented as specified in the textbook using the waiting arrays and a key variable which are defined globally
4. For the loop part, we again use the atomic_compare_exchange_strong() function like above
5. after this go into critical section
6. in the critical section we change the waiting[tid - 1] to false
7. just after entering calculating the waiting time and pushing in into a global vector named meantime which will be used for various calculations
8. and in the exit section we use a while loop to searching in a circular way and searches the first process that is in the while loop of above (not this while loop) with waiting[j] == true as next one to enter the critical section like discussed from book

2.6 Waiting Times and Max Waiting Time Designs:

1. store all waiting time into a globally declared vector named meantime
2. the we calculate the averege waiting time by finding average of all those values
3. simillarly calculated the maximum waiting time also based on that vector values
4. printing the respective values into the console

3 Graph1 Analysis: Average waiting time vs No of processes

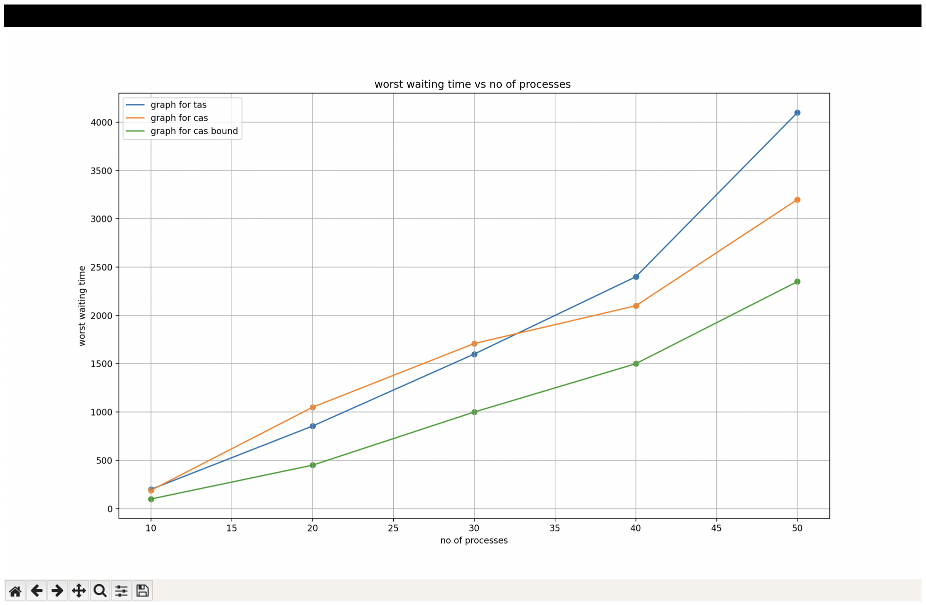
1. For small values of ‘n’ , the average waiting time for CAS,TAS and Bounded CAS are similar.
2. here for all 3 plots in the graph asno of threads increases the average waiting time is increasing which is expected becaused more no of processes want to access the critical region now
3. so waiting time increases
4. the plot is shown below and legend is in the figure itself



5. from the graph we can clearly see that average waiting time for TAS and CAS algorithms are same
6. and for Bounded CAS it is higher than the other 2 algo
7. based on graphs we can say like for lesser average waiting time it is better to take CAS or TAS over bounded CAS

4 Graph2 Analysis: Worst waiting time vs No of processes

1. We see that the graph is increasing as the number of threads increases for all TAS,CAS,Bounded CAS , which is also same as the first experiment.
2. below is plot



3. We can see that again that the worst waiting time for TAS and CAS algorithms is almost similar , with CAS being slightly lower than that of TAS.
4. For comparision between TAS and CAS For lower values of ‘n’ TAS showed a better performance(having less worst time taken value). As ‘n’ increased CAS had less worst time compared to TAS values.
5. but overall Bounded CAS performed best in all values of n
6. from here we can say that it’s better to take Bounded CAS if we want to have less worst waiting time