Report

G Harsha Vardhan Reddy

CS21BTECH11017

The 'ProgAssgn3-CS21BTECH11017.zip' folder contains 3 src files and these .cpp files take 4 integers input $n,\ k,\ \lambda_1,\ \lambda_2$ from the '.txt' file named 'inp-params.txt'.

These 3 programs are divided into 5 functions

1. Main function - takes input and creates threads in accordance with input, prints the time at which the several threads requests, enters and exits critical section after completion of all threads in a .txt file named "'*#name*'-me-output.txt" where '*#name'* is tas,cas,bounded_cas according to the src_program.

2. testCS function - takes threadID as input and according to input it makes request to critical section and enter the critical section, sleeps for a time t1, exits and sleeps for time t2 where t1, t2 are exponentially distributed w.r.t mean $\lambda_1\ and\ \lambda_2$ respectively.

3. getSysTime function - gives the time at which it is called as a string in format

$hrs : mins : sec\ .\ millisec$

4. 1) for tas: lock function - it doesn't take any input and calls a pre-defined function exchange() and according to it, it returns or spins over there.

   2) for cas: lock function- it doesn't take any input and calls a pre-defined function compare_exchange_strong and according to it, it returns or spins over there.

   3) for bounded_cas: lock(id) function- it takes id as input and calls a pre-defined function compare_exchange_strong and according to it, it returns or spins over there

5 1) for tas: unlock function - it doesn't take any input and changes flag value to false.

   2)for cas: unlock function - it doesn't take any input and changes flag value to false

   3) for bounded_cas: lock(id) function- it takes id as input and it either changes the succeeding threads waiting to false or flag to false.

Process:

The main function takes input from 'inp-params.txt' and corresponding to that it creates n threads and each thread calls testCS function and stores their thread ids.

Each thread in the testCS function requests k times to enter to the critical section, Pushes down the time at which they request for CS to a vector vi, enters CS and sleeps for time t1,pushes down the time at which they entered CS into vi, exits from CS, pushes down the time at which they exits to vi and sleeps for time t2 before another request.

To request for CS it calls lock function which is as follows

1) Tas: it calls a function exchange() and changes flag value according to it and if the lag becomes true then the corresponding thread enters CS.
2) Cas: it calls a function compare_strong_exchange(), makes expected to false and changes flag value according to it and if the flag becomes true then the corresponding thread enters CS.
3) Bounded_cas: it takes input threadID and sets the corresponding waiting to true and calls compare_strong_exchange(), makes expected to false and changes the flag value according to it and if either the waiting is false or the flag is true it enters the critical section making waiting as false.

And after sleeping for t1 seconds it calls unlock function which is as follows

1) Tas: it stores flag value to false and returns.
2) Cas: it stores flag value to false and returns.
3) Bounde_cas: it takes thread id as input and sees whether any thread is waiting to enter CS consecutively according to thread id if there exists a thread which is waiting to enter CS then it makes the waiting of the corresponding thread to false and else makes flag false and returns.

After completion of all requests the testCS stores vi in an array arr.

The main function waits till the completion of all threads and writes the values stored in arr to the file named "'*#name*'-me-output.txt" where '*#name'* is tas,cas,bounded_cas according to the src_program.

Performance:

Time complexity = O(n*k/$min(No\ of\ cores\ available,\ k)$) $ie.$ O(n*k).

Space complexity = O(n*k).

File handling : takes input from "inp-params.txt" and creates a file called "'*#name*'-me-output.txt" where '*#name'* is tas,cas,bounded_cas according to the src_program.

Corresponding to input 'k' (no of threads) the program(algorithm) uses CPU

cores to make it run parallelly.

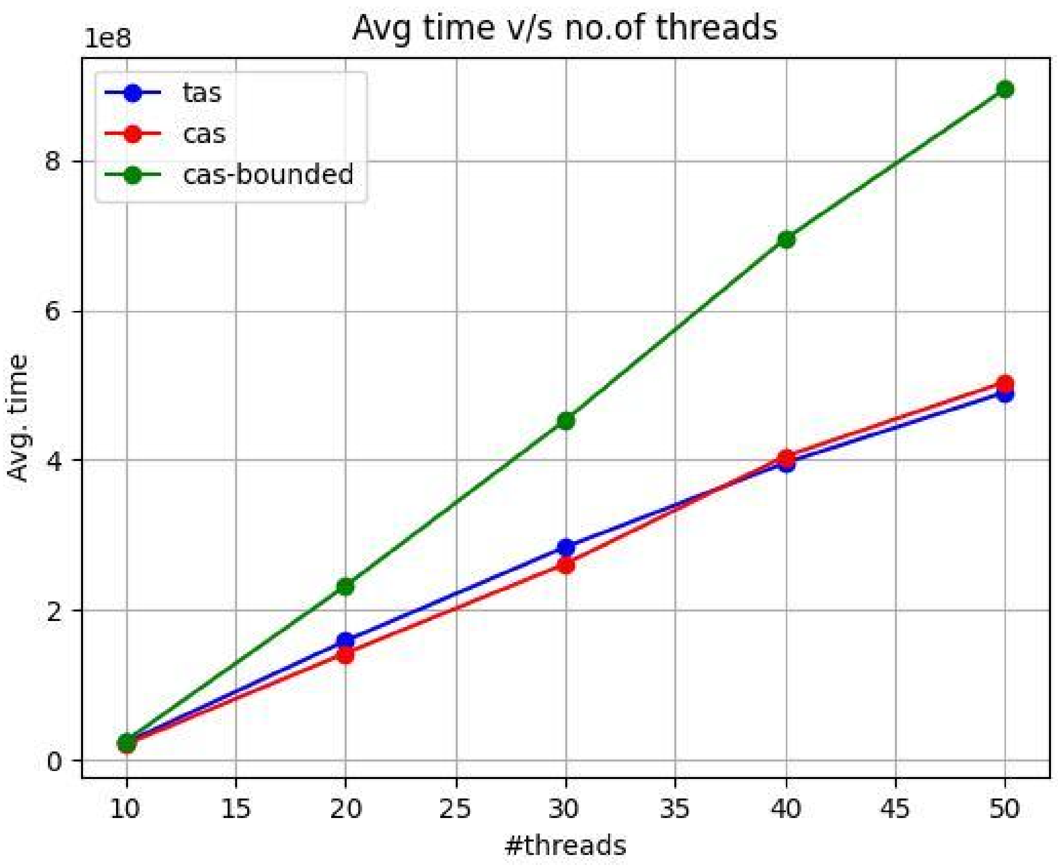
Output:

The program creates a .txt file named "'*#name*'-me-output.txt" where '*#name'* is tas,cas,bounded_cas according to the src_program.

It contains the time at which the particular thread requests, enters and exits the critical section respectively.

Plots:

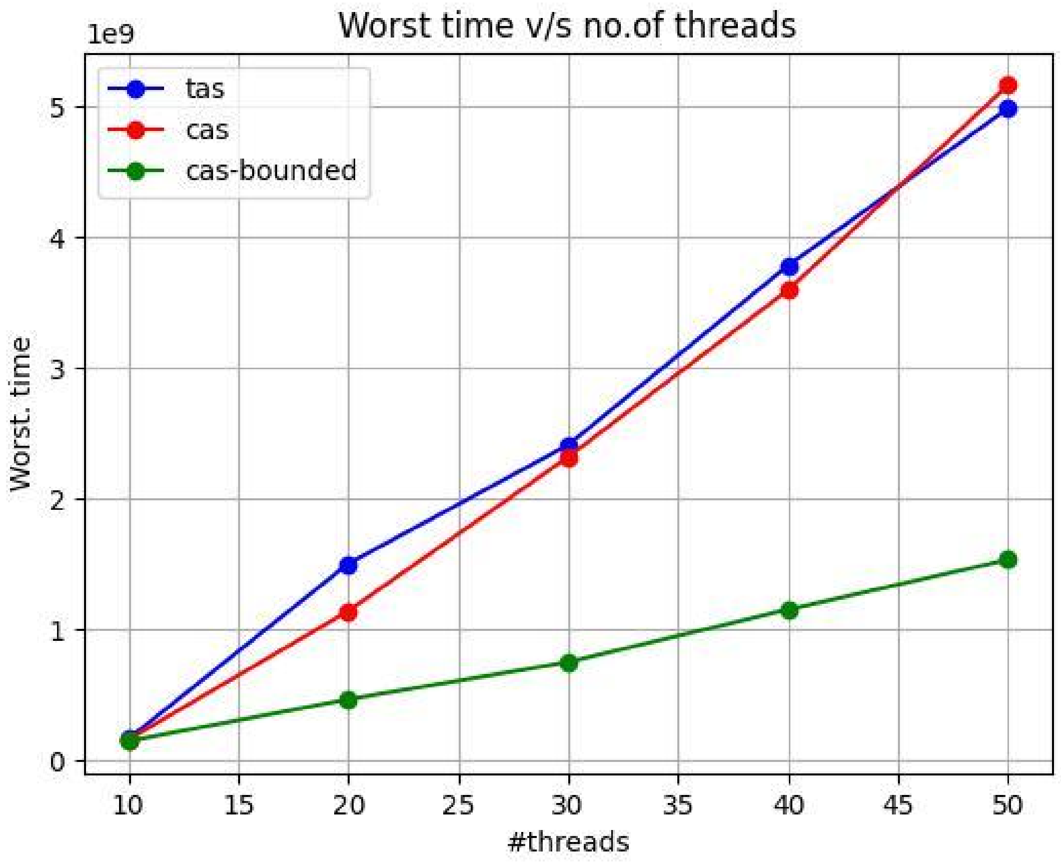
1. Average time taken(in $10^{-1} s$) to enter CS v/s No.of threads keeping $k = 10,\ \lambda_1 = 5ms,\ \lambda_2 = 20ms$



1) As increase in no.of threads increases the competition for entering critical section increases and therefore the worst time taken to enter CS increases.
2) As the computational overhead is more in case of bounded_cas (since we have to ensure nor thread is starving ie.. to ensure all the threads are entering CS), the average time taken to enter CS is more when compared to cas and tas

.

2. Worst time taken(in s) to enter CS v/s No.of threads keeping



1) As increase in no.of threads increases the competition for entering critical section increases and therefore the average time taken to enter CS increases.
2) As ‘cas_bounded’ ensures that no thread starves ie.. every thread enters CS in every cycle in a finite time. So the worst time of waiting for CS is less compared to ‘cas’ and ‘tas’ where starvation is possible(ie.. In a cycle threadI may not be able to enter CS as there is no perfect procedure for a thread to enter CS)