Programming Assignment-4 Report
CS3523: Operating Systems-2
Implementing TAS, CAS, and Bounded Waiting CAS Mutual Exclusion Algorithms

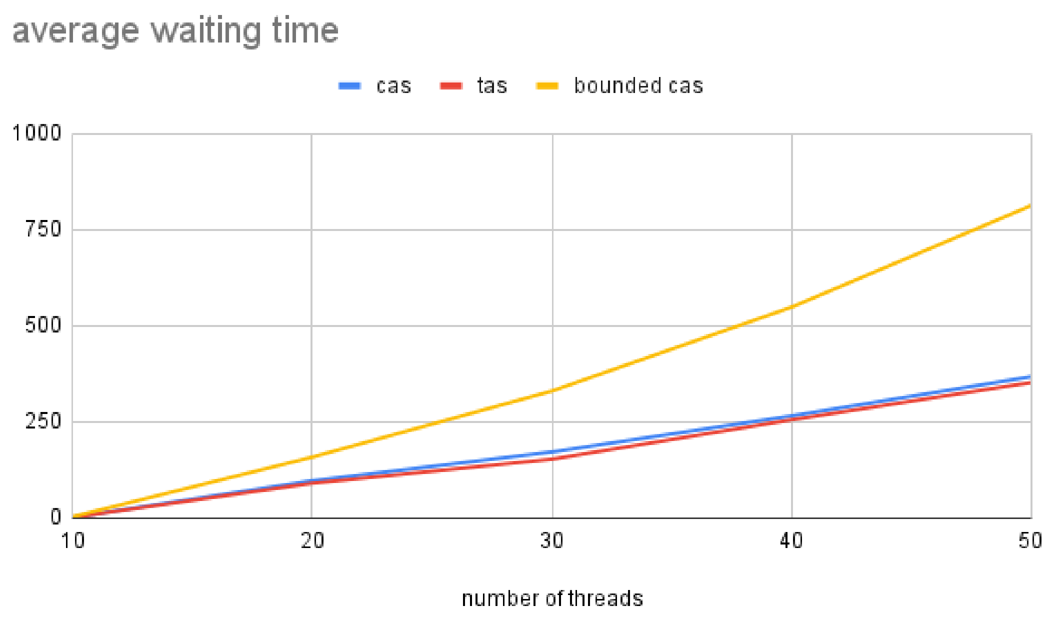
Name: Vikhyath Sai Kothamasu
Roll Number: CS20BTECH11056

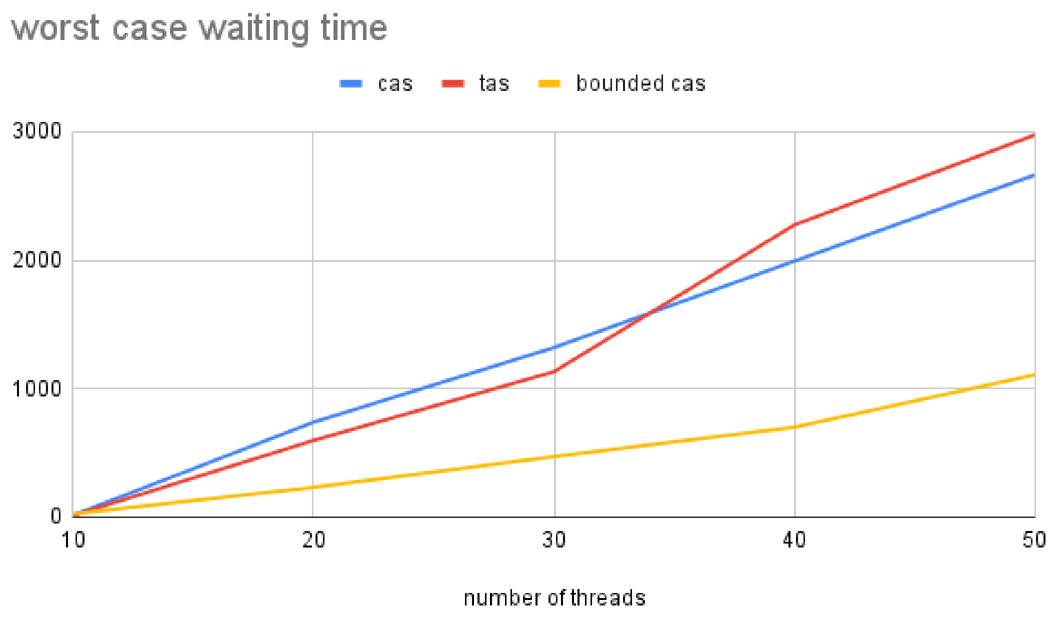
Goal: The goal of this assignment is to implement TAS, CAS, and Bounded Waiting with CAS mutual exclusion (ME) algorithms studied in the class. Implement these algorithms in C++.

Low-Level Design: The main thread first reads the text file “inp-params.txt” which contains the necessary inputs for the code to run. It contains the number of threads to be created, the number of times each thread will enter the critical section, two averages of exponential distributions. Now that we know the number of threads, we create the required number of threads, each with a unique id. For the mutual exclusion part, we have created a testCS function so this function is passed to each thread while initializing. Once these threads have been created and are in the testCS function, the main thread waits for all the threads to terminate and then prints important information such as the average waiting time to enter the critical section and the maximum waiting time to enter the critical section.
Globally, we initialize a random number generator to generate random numbers for the exponential distribution. These exponential distributions have an average of lambda1 and lambda2 respectively. These exponential distributions are used to obtain t1 and t2 randomly which are used to simulate processing time in the critical section and remainder section respectively. We also define 2 functions namely current_time() which returns the current local time in a specified format and suffix() which returns the appropriate suffix for a number.
When a thread enters the testCS function, it enters a loop that takes care of each thread entering the critical section multiple times. Each time it loops, it basically requests to enter its critical section and thus it waits to obtain the lock. So, in the entry section, the threads wait until they obtain the lock. Depending on the algorithm, the entry section will be a bit different from each other as they try to obtain the lock in their own way. Once, a thread obtains the lock, we calculate the time it took to obtain the same and add it to the total waiting time and compare it with the maximum waiting time. Once a thread obtains the lock and enters the critical section, we use usleep to suspend the execution of the thread for a certain amount of time. This is used to simulate processing time (some complicated time-consuming tasks) inside the critical section. Once the thread becomes active again, we free the lock so that another thread can now enter the critical section without any issues. The thread is again put to sleep to simulate processing time in the remainder section.

For printing the logs, I have used fprintf because when cout was used, it caused mixing of logs like 2 lines were printed together and some lines were empty.

Graphs: The comparison metrics used were average waiting time and worst-case waiting time. For comparison, the number of threads was kept varying from 10 to 50 while other parameters were kept constant. Also, for each input, the program was run multiple times, and plotted values are the average of all. The constant parameters were k = 10, lambda1 = 0.5, lambda2 = 0.2.





Analysis of graphs:

1. The average waiting time for cas and tas algorithms are similar while that of bounded cas is much higher as compared to the other two because it doesn't leave processes to starve due to which other processes might have to wait a bit longer.
2. The worst-case waiting time for cas and tas algorithms are similar again while that of bounded cas is significantly lower because it tries to avoid starvation and thus there isn't a single process that will have to wait for a very long time.