OPERATING SYSTEMS 2

INSTRUCTOR: DR. SATHYA PERI
Hitesh Sehrawat - ma17btech11004@iith.ac.in

Theory

Critical Section : **Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section.**

Entry Section : A part of code which is available to all the threads where the threads contend to enter into the Critical Section. Bounded wait guarantees that no thread waits forever in the entry section and gets a fair chance to enter Critical section.

Exit Section : A segment of code where in a thread exits from the critical section and opens the lock for other threads to enter the Critical Section.

Waiting Time : The time that a thread spends in the entry section waiting to get into the critical section in called waiting time.

Implementation

Important Libraries Used :

- **<pthread.h> : Used for creation and work assignment of the threads.**
- **<atomic> : Used for making the lock atomic so that only one thread can edit it at a time**

● **<random> : Used for the random number generator for seeding the sleep interval.**

● **<unistd.h> : Used for calculating the current system time.**

Execution :

● **Each program achieves process synchronization.**

● **All the three programs follow a similar design , except the implementation of the critical section. The main thread first reads the input from "inp.txt". These are stored in global variables to allow access to all the threads.**

● **The lock variable(std::atomic_flag locki = ATOMIC_FLAG_INIT) is an atomic variable used by the standard library functions for mutual exclusion.**

● **TAS and CAS use the standard library functions "atomic_flag_test_and_set", and "atomic_compare_and_exchange " to implement critical section.**

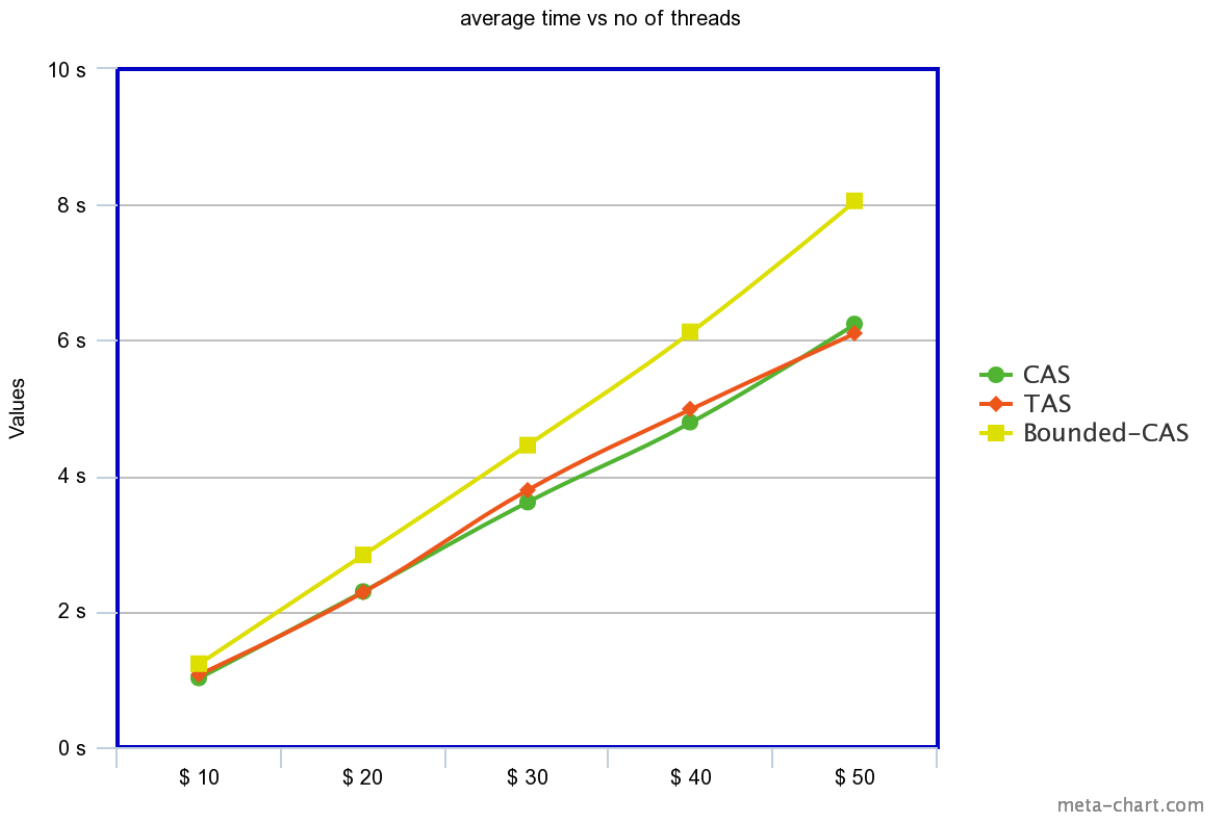
● **We use default_random_engine "eng" to generate random numbers with uniform distribution .**

● **Each thread, in testCS function, simulates Entry section, Critical Section and Remainder section. Each thread also measures the time at which it entered all of these sections and also measures the waiting time (time spent in entry section before entering the critical section).**

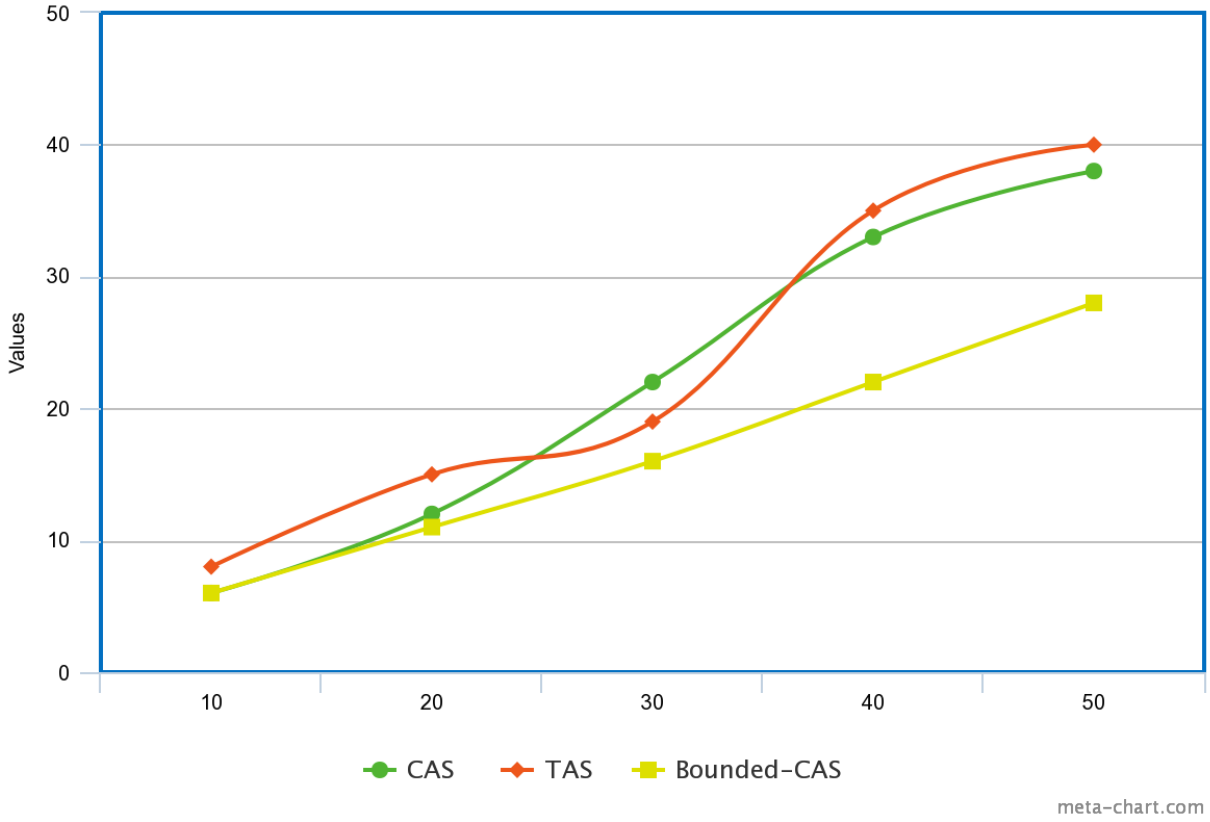
● **We use usleep() to make the thread sleep for some while to simulate the thread in the Critical Section as well as the remainder section doing some hefty work .**

Graphs

Average waiting time



Worst waiting time



Conclusion

In terms of efficiency

CAS > TAS > Bounded CAS

But in worst case time

Bounded CAS > CAS > TAS