Operating Systems 2
SPRING SEMESTER 2018
Programming Assignment 2

INSTRUCTOR: DR. SATHYA PERI
Mayank Hooda -cs16btech11022@iith.ac.in

1. Design of the Program :

We use three different techniques to implement the critical section , using a lock.

Critical Section :

A segment of code which we want only a single thread to access at a given time is called a 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 wherein a thread exits from the critical section and opens the lock for other threads to enter the CS.

Waiting Time :

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

2. Implementation of the Program :

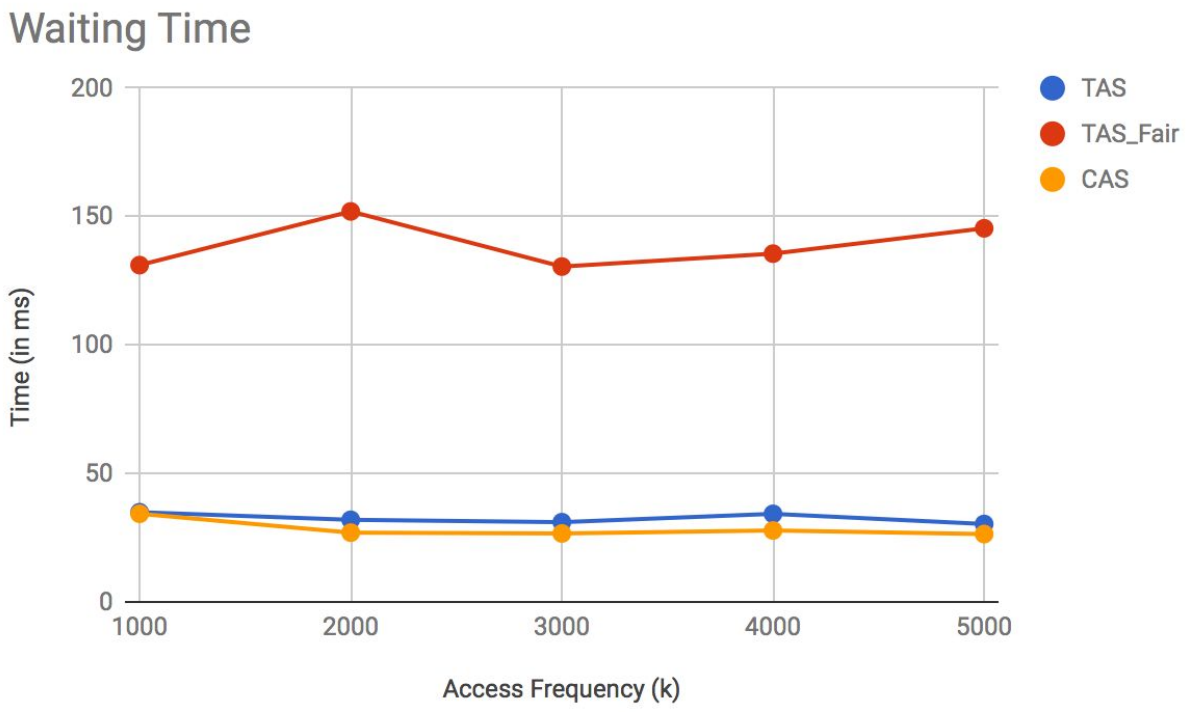
- All the three programs follow a similar design , except the implementation of the critical section. The main thread first reads the input from “inp.txt”. These are stored in global variables to allow access to all the threads.
- Accordingly, two arrays: one containing thread creation attributes *attrs*, and one containing thread ids, *workers* are created.
- We use *default_random_engine eng1,eng2* to generate random numbers with *uniform distribution* .
- They generate numbers according to uniform distributions *cs* and *rm*.
- The main thread then creates n threads, each of which executes testCS function and is passed an id typecasted into *void** . It acts as a identifiers for the threads.
- 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).
- 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.
- TAS-bounded uses additional local variable j, and a global array waiting to allow for fairness - i.e. each thread should get a chance to enter the critical section. It reduces the randomness in selecting the next thread to send in the critical section by

selecting the thread which has thread_id linearly greater than the thread in critical section , namely ***j***.

- 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 .

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.



Graph Plotted for following parameters :

N = 10

1000<=k <=5000

CSseed = 10

RemSeed = 20

The graph shows that the according to waiting times , the **efficiency** of the algorithms is :

CAS > TAS > TAS_Fair

TAS_Fair gives the fairest chance for threads to enter the Critical region but because of the various operations involved it goes on to become the algo with the highest waiting time.