**Assignment Report**

Roll number : - cs21resch11012

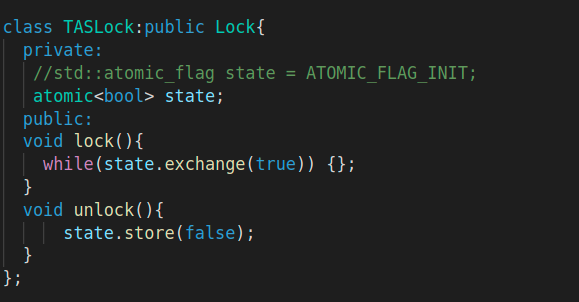
**Program Design:**

* I have declared an abstract class Lock for all the implementation. The Lock class have two pure virtual function lock( ) and unlock ( ). The functions are pure virtual in nature implies that whatever the other class inherit the Lock class as their base class will have to define these two functions.
* In all the locks implementation, for the purpose of time calculation I have used four global variables i.e totalEntryWaitingTime and totalExitWaitingTime for calculating average time taken by thrrads to obtain the locks, and worstcaseentry and worstcaseexit for recording worst case time taken by thread to enter into critical section and exit from critical section.
* Rest of the program design follows, as described in the assignment.

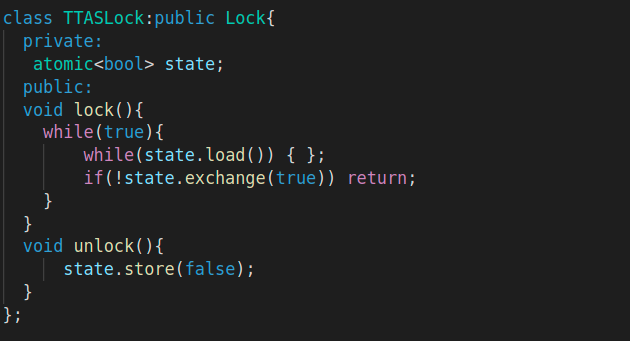
**Algorithms:**

The algorithms for all the implementation has already been discussed in the book. I have tried to implement the same algorithm that has been discussed in the book.

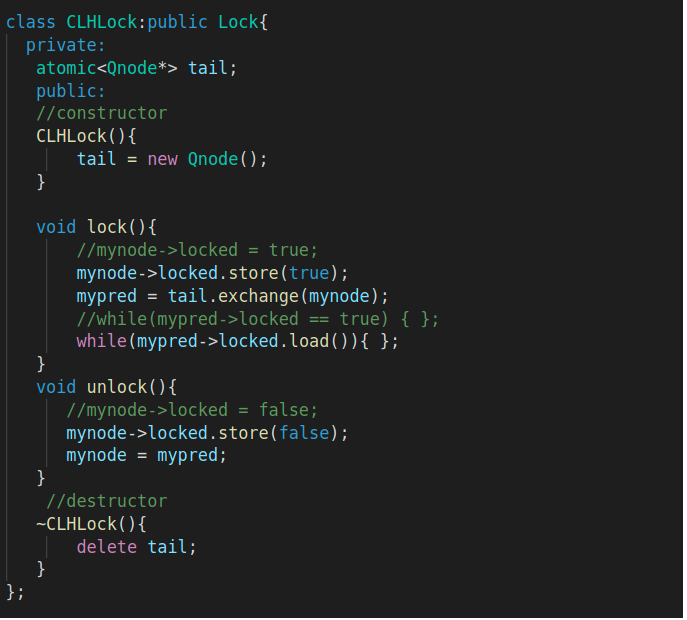
TAS Lock:-



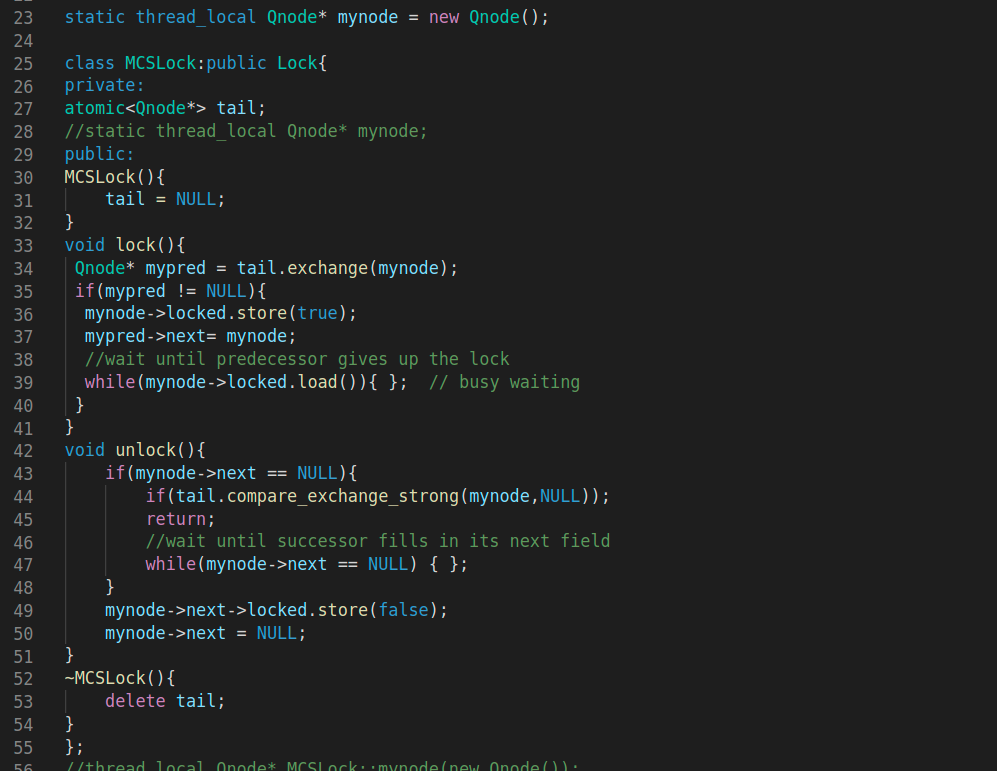
TTAS Lock:-



CLH Lock:-



MCS Lock:-



**Performance Comparision and Analysis:**

For all the Comparision, the value of k is fixed to 10, and value of λ1, λ2 as 1 and 2(milliseconds). The number of threads varies from n = 2 to n = 64, in the power of two. The reason I used milliseconds as opposed to seconds, is because for having large value of n there is very much considerable delay in execution to get finished i.e n = 64. For each input points, I have run my implementation 5 times, and taken average.

1) Average Critical section entry time taken by threads.

Here, I am getting the observation that, of all the locks, threads using TTAS lock take less time to enter into the CS. Ideally, MCS should have taken the lesser time of all. But strangely, here I am getting TTAS performing best among all. CLH and MCS is almost taking similar time, which is understandable as the system where I am implementing all these locks have cache-coherent architecture. Theoritically, TTAS shouldn’t perform better than CLH and MCS because of heavy traffic a thread creates in the BUS, when it leaves the critical section. So, here it is an anomaly. The reason I think this is happening is that, OS is fast enough to schedule next thread among all threads competing , so it is not creating that much memory contention effect.

2) Average Critical section exit time taken by threads.

TAS is taking most time to exit from CS, followed by TTAS. CLH and MCS are taking almost similar time, CLH is performing slightly better than MCS. Theoretically, It is exactly what should happen.

3) Worst Case Critical section entry time taken by threads

TAS and TTAS worst case entry time is significantly higher than CLH and MCS. The reason is that ,in case of TAS and TTAS, once a thread when comes out from critical section it invalidates all the cache lines. So now it creates heavy amount of bus traffic, where every thread is trying to acquire the lock. Whereas in case of CLH and MCS, the thread when it comes out of critical section, it only makes its node as false, so the next thread in the queue which were spinning on its local cache copy, only cache copy of that thread gets invalidated, and it can now acquire the lock.

4) Worst Case Critical section exit time taken by threads

TAS is taking worst-case most time to exit from CS, followed by TTAS. CLH and MCS are taking almost similar time, CLH is performing slightly better than MCS. Theoretically, It is exactly what should happen. There is no anomaly here.

**Correctness:-**

To check, whether the lock implantation is right or not, I have confirmed from the output files that msg2 and msg3 , for a thread is printing consecutively.