REPORT

Theory :

CLH lock:

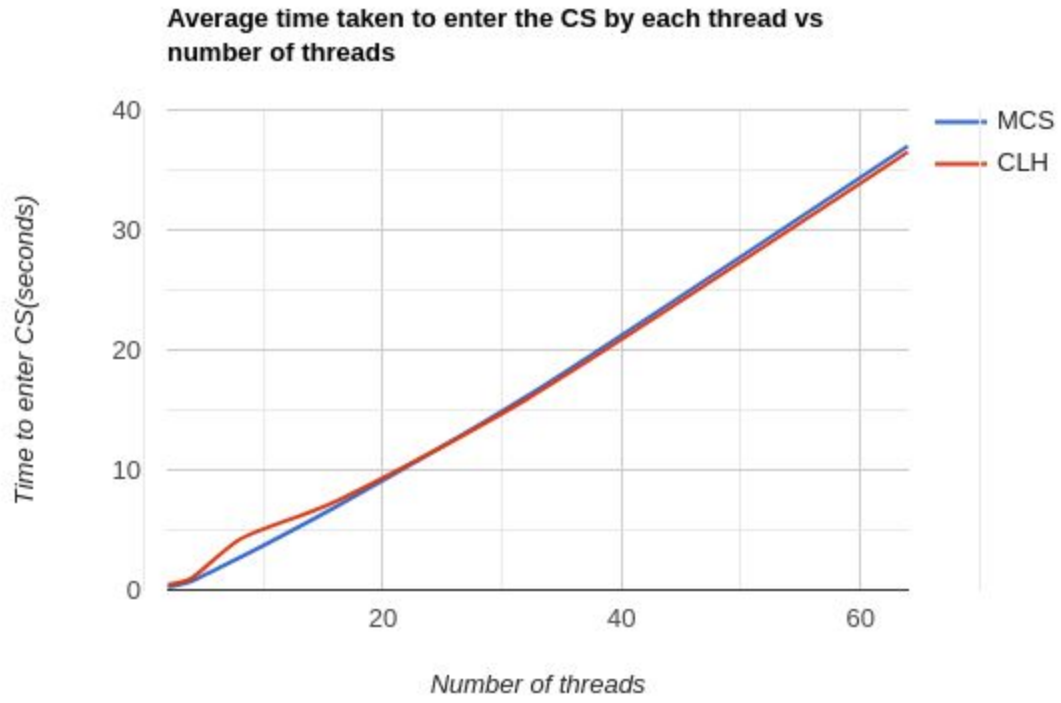
Initially the tail field refers to a QNode whose locked field is false. Suppose a thread A wants to acquire lock followed by thread B. Thread A then applies getAndSet() to the tail field to insert its QNode at the tail of the queue, simultaneously acquiring a reference to its predecessor's QNode. Next, B does the same to insert its QNode at the tail of the queue. A then releases the lock by setting its node's locked field to false. It then recycles the QNode referenced by pred for future lock accesses. If there are L locks, and each thread accesses at most one lock at a time, then the CLHLock class needs only O(L + n) space.

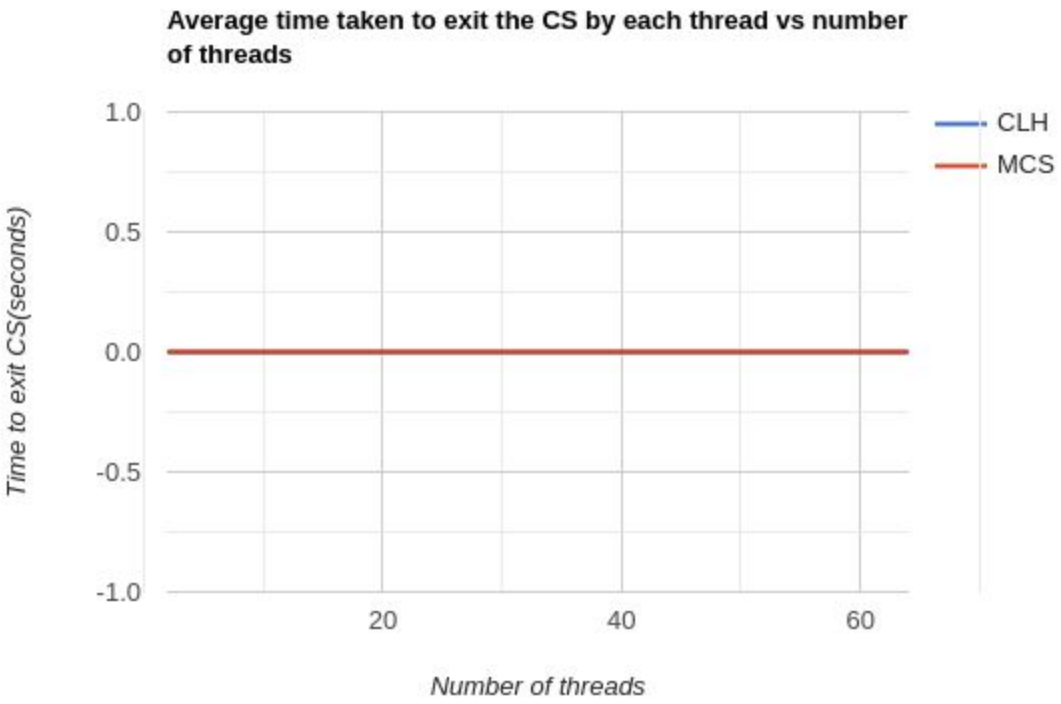
MCS Lock:

Initially the tail is null.Suppose a thread A wants to acquire lock followed by thread B followed by thread C.

To acquire the lock, thread A places its own QNode at the tail of the list and since it has no predecessor it enters the critical section. Thread B enqueues its own QNode at the tail of the list and modifies its predecessor's QNode to refer back to its own. Thread B then spins on its locked field waiting until A, its predecessor, sets this field from true to false. Thread C repeats this sequence. To release the lock, A follows its next field to its successor B and sets B's locked field to false. It can now reuse its QNode.

Observations:


Average time taken to enter the CS by each thread vs number of threads
Time to enter CS(seconds)
40
30
20
10
0
20
40
60
Number of threads
MCS
CLH


Average time taken to exit the CS by each thread vs number of threads
Time to exit CS(seconds)
1.0
0.5
0.0
-0.5
-1.0
20
40
60
Number of threads
CLH
MCS

It can be seen that initially MCS is faster than CLH.(for threads n<20 approx). Then they both are almost equal after n = 20 and no clear winner can be said. This can be explained by the below reasons.

The above picture shows details of my CPU.It is a 6 core,12 thread CPU.

Initially MCS is faster because it is better suited to cache-less NUMA architectures because each thread controls the location on which it spins. The disadvantage of the CLH lock algorithm is that it performs poorly on cache-less NUMA architectures. Each thread spins waiting for its predecessor's node's locked field to become false. If this memory location is remote, then performance suffers. Mine being a 12 thread cpu, MCS was faster initially.

But the drawback of the MCSLock algorithm is that releasing a lock requires spinning. Another is that it requires more reads, writes, and compareAndSet() calls than the CLHLock algorithm. So as the number of threads increases, this spinning to release the lock becomes an overhead.

There isn't considerable difference between reqCSexit and actCSexit, so the avgCSexit time boiled down to zero or very negligible amount of time in all the cases.