Project Report

Team:

G. Vishal Siva Kumar - CS18BTECH11013
Krishna Prashanth - CS18BTECH11045
P.V.Asish - CS18BTECH11037

References:

Granuke and Thakkar's lock

Graunke, Gary, and Shreekant Thakkar. "Synchronization algorithms for shared-memory multiprocessors." *Computer* 23.6 (1990): 60-69.

M Lock

Magnusson, Peter, Anders Landin, and Erik Hagersten. "Queue locks on cache coherent multiprocessors." *Proceedings of 8th International Parallel Processing Symposium*. IEEE, 1994.

Multi resource lock

Zhang, Deli, Brendan Lynch, and Damian Dechev. "Fast and scalable queue-based resource allocation lock on shared-memory multiprocessors." *International Conference On Principles Of Distributed Systems*. Springer, Cham, 2013.

GT Lock:

The GT Lock gives a classic queue-based lock where each thread waits on it's previous node in the queue. Each node just gets the id of the tail of the queue and spins on it till it is flipped. The id of the process that last tried to acquire lock is also recorded in the lock. Each thread has a different meaning(or representation) for what means locked.

M Lock:

The M Lock tries to extend the CLH lock by minimizing at least one or global accesses in total. The core idea remains the same. The modification is made so that if the thread finds that there is no queue then it doesn’t need to exchange a flag.

Multi-Resource Lock:

In Multi-resource lock, each thread wants to access multiple resources but to make it simple for comparison and implementation we set threads to access a single resource. We used bitset to find the request for resources for each thread.
For a given set of resources, we use bitset as each bit is uniquely mapped to each resource. A thread when required resources, sets the bit corresponding bit in the bitset to 1. The multi resource lock handles the case of accessing multiple resources by a thread by waiting till all the resources required by the thread are available. If not, the thread waits in the queue. This all-or-nothing atomic acquisition makes this algorithm a bit slow but manages to provide all the required resources by the threads.

Implementation:

All the lock implementations are queue based spin locks. We implemented the locks in C++ language by using provided atomic constructs like “Compare and swap” which is “compare_exchage_weak” and other provided constructs. In GT lock, there is no explicit queue to be found. It has a virtual queue data structure.

M lock also is implemented using a virtual queue. Whereas a Multi-resource lock has an explicit array of structures whose size is the number of threads. We implemented GT lock with a bit of variation by not using hardware locks but using C++ provided atomic constructs.

In Multi-resource lock, we used std::bitset<32> to represent resource requests. To demonstrate mutual exclusion, for every thread we have given that it requires all the 32 resources set. Specifically, we have set all the bits in the resource request bitset to 1.

Times Comparison:

GT Lock:

Number of threads	Average		Worst	
	Acquire	Release	Acquire	Release
2	65.989	0.7367	216.701	1.429
4	723.308	0.7164	1964.22	1.433
8	1557.81	0.1921	3662.65	0.754
16	110548	0.1633	159204	0.824
32	410409.75	0.1659	524483	0.631
64	1234474	0.2342	1551970	1.425

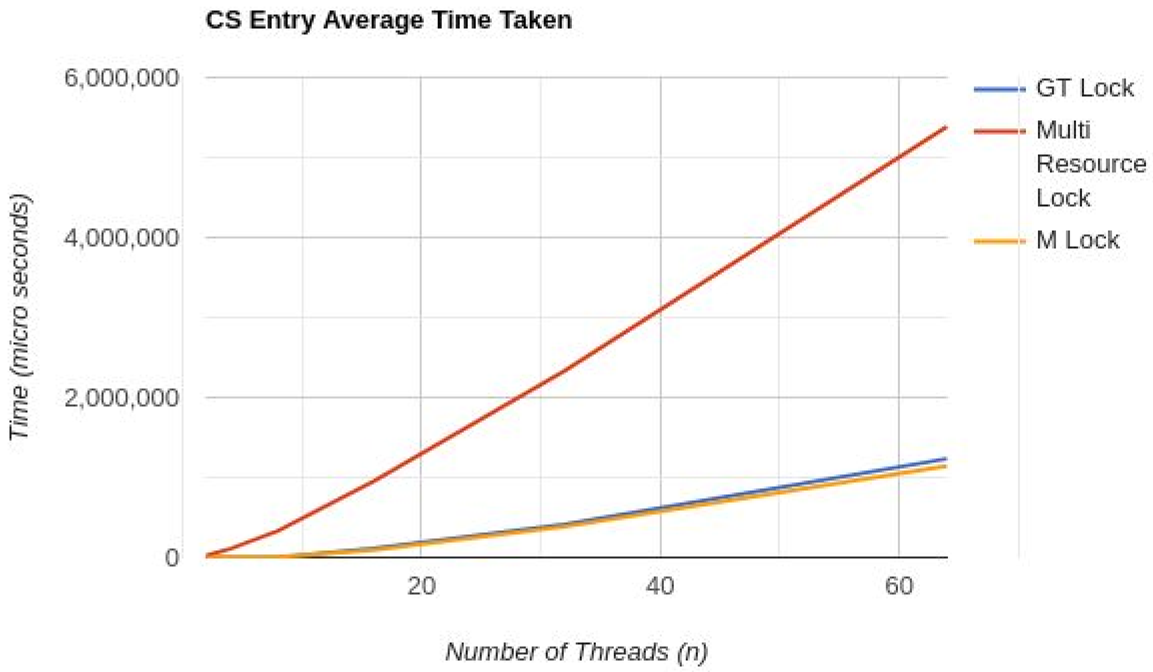
M Lock:

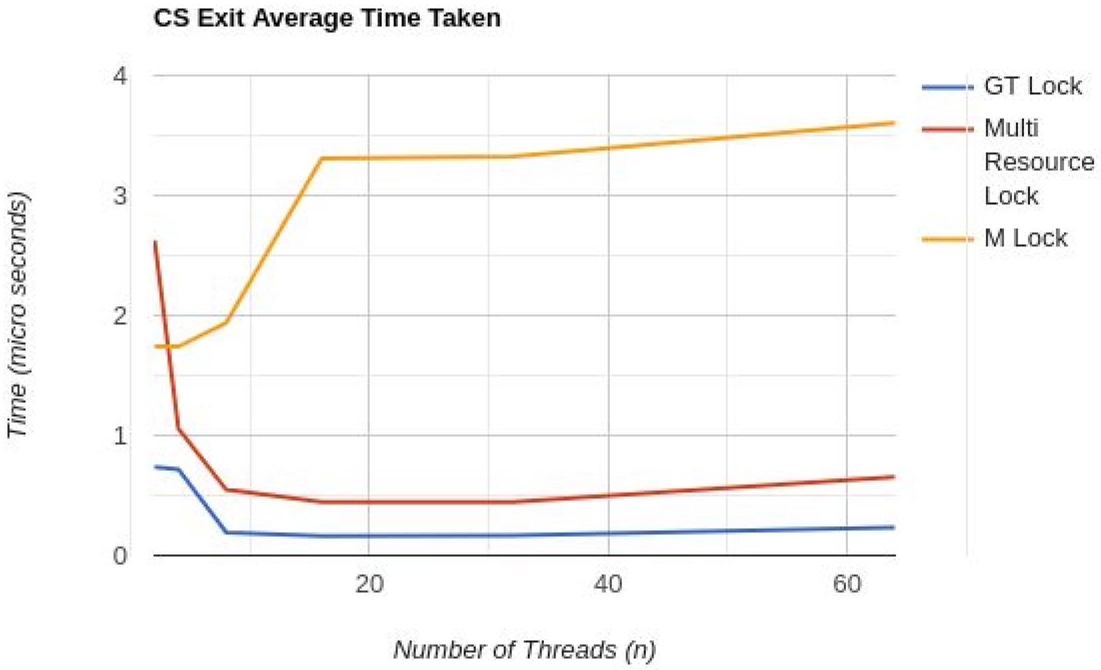
Number of threads	Average		Worst	
	Acquire	Release	Acquire	Release
2	11.611	1.742	61.93	10.269
4	162.021	1.7408	263.65	3.352
8	4313.57	1.9384	9986.21	9.137
16	93625.7	3.31	147745	28.84
32	386368.5	3.326	567947	37.66
64	1146360	3.606	1531360	37.631

Multiresource lock:

Number of threads	Average		Worst	
	Acquire	Release	Acquire	Release
2	23923.86	2.62739	247580.12	4.116
4	109758.6	1.0554	744174	2.388
8	334708.6	0.5482	1764670	1.966
16	953809.2	0.4474	4043510	1.969
32	2334848	0.4439	8623710	2.086
64	5385600	0.6534	18156200	14.07

Graphs:


CS Entry Average Time Taken


CS Exit Average Time Taken

Observations:

We ran all our lock algorithms on an Intel i5 8300H processor with 4 cores and 2 physical threads for each core. We can observe that GT lock and M lock are performing way better than Multi resource lock as in multi resource lock, a thread waits until all the resources required are available. But as we are using a single resource there’s only one CS. Every thread checks for availability over all the bitset. So, it takes a little bit more time.

Correctness :

To show the correctness of our program, we can check the log file that is generated after running the program, Since all our locks are queue based and follow FIFO order, we can see that the threads enter CS in the order they make a request to enter CS, and also we can show mutual exclusion by checking no thread can enter CS before a thread in CS exits.

Space complexity:

M Lock: **O**(L + n)

where L is the number of lock objects and n is the number of threads accessing them.

GT Lock: **O**(L*n)

because for each lock access, we need a new memory allocation for each thread accessing them.

Multi resource Lock: **O**(L*n*k)

where L is the number of locks, n is the number of threads and k is the number of resources.

Re-use of variables:

Re-use of variables can happen in M Lock but not in the remaining two locks.

Note:

- When writing output to a file since the write statements are not atomic, some overlap may happen. And also some statements may repeat if the output buffer is not cleared before another thread starts writing. Generally “endl” clears the output buffer. Since these are not atomic, we can’t guarantee that.

- We use “substr” method to get the required part of the timestamp from the whole timestamp. “substr” may result in a segmentation fault which is rare. But this is not the problem in the algorithm.