CS4532 Concurrent Programming -Lab 02

140019E, 140033P

Step 3:

Case 1

n = 1,000 and *m* = 10,000, *mMember* = 0.99, *mInsert* = 0.005, *mDelete* = 0.005

Implementation	Number of threads							
	1		2		4		8	
	Average	**Std**	**Average**	**Std**	**Average**	**Std**	**Average**	**Std**
Serial	0.02368	0.00168						
One mutex for entire list	0.01957	0.0092	0.02134	0.01497	0.02231	0.01595	0.02292	0.01917
Read-Write lock	0.01966	0.00886	0.01831	0.00918	0.01669	0.00873	0.01788	0.00855

Case 2

n = 1,000 and *m* = 10,000, *mMember* = 0.90, *mInsert* = 0.05, *mDelete* = 0.05

Implementation	Number of threads							
	1		2		4		8	
	Average	**Std**	**Average**	**Std**	**Average**	**Std**	**Average**	**Std**
Serial	0.01357	0.00190						
One mutex for entire list	0.00986	0.01013	0.01303	0.01632	0.013	0.01813	0.01308	0.01796
Read-Write lock	0.00999	0.01024	0.00855	0.01474	0.0087	0.01605	0.012	0.01624

Case 3

n = 1,000 and *m* = 10,000, *mMember* = 0.50, *mInsert* = 0.25, *mDelete* = 0.25

Implementation	Number of threads							
	1		2		4		8	
	Average	**Std**	**Average**	**Std**	**Average**	**Std**	**Average**	**Std**
Serial	0.1178	0.00344						
One mutex for entire list	0.04235	0.01329	0.0648	0.01656	0.8517	0.02054	0.10037	0.02069
Read-Write lock	0.04193	0.01331	0.06607	0.01630	0.8944	0.02255	0.10404	0.02360

For this assignment, we are required to calculate the number of samples in order to get the accuracy of ±5% and 95% confidence level. The Sample size is determined by an equation

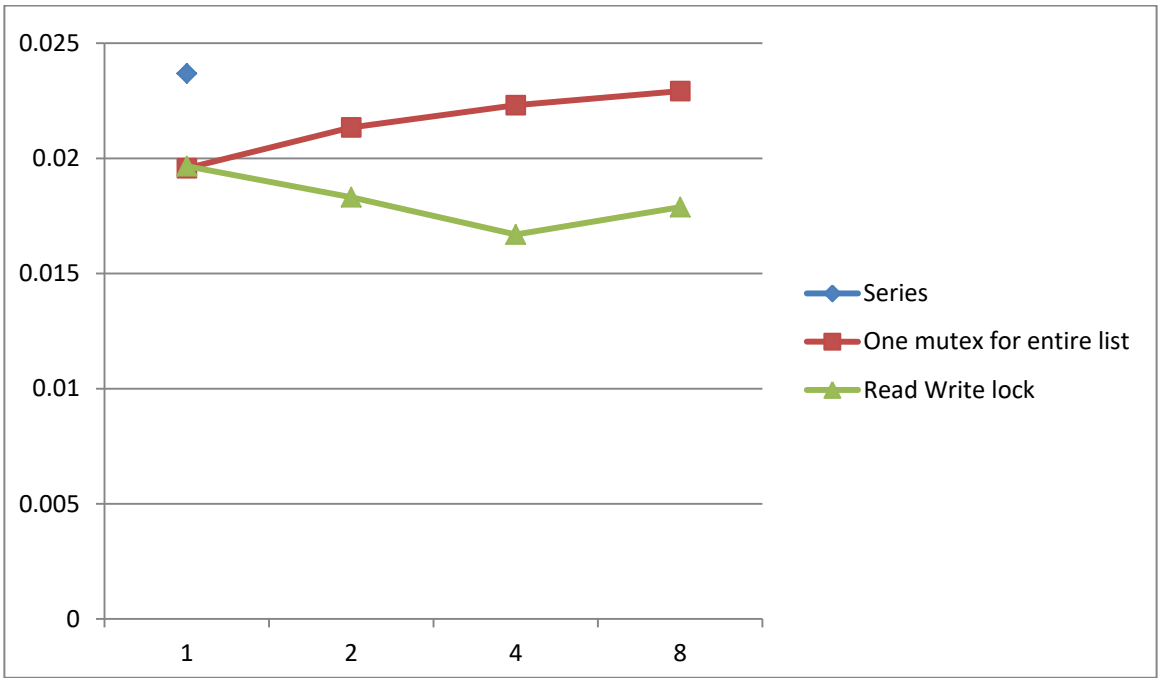
In order to calculate a prior standard deviation and mean of the distribution, we used a sufficiently large sample size of 200. Required number of samples with an accuracy of ±5% and 95% confidence level are calculated and shown in the following table.

Step 4 :

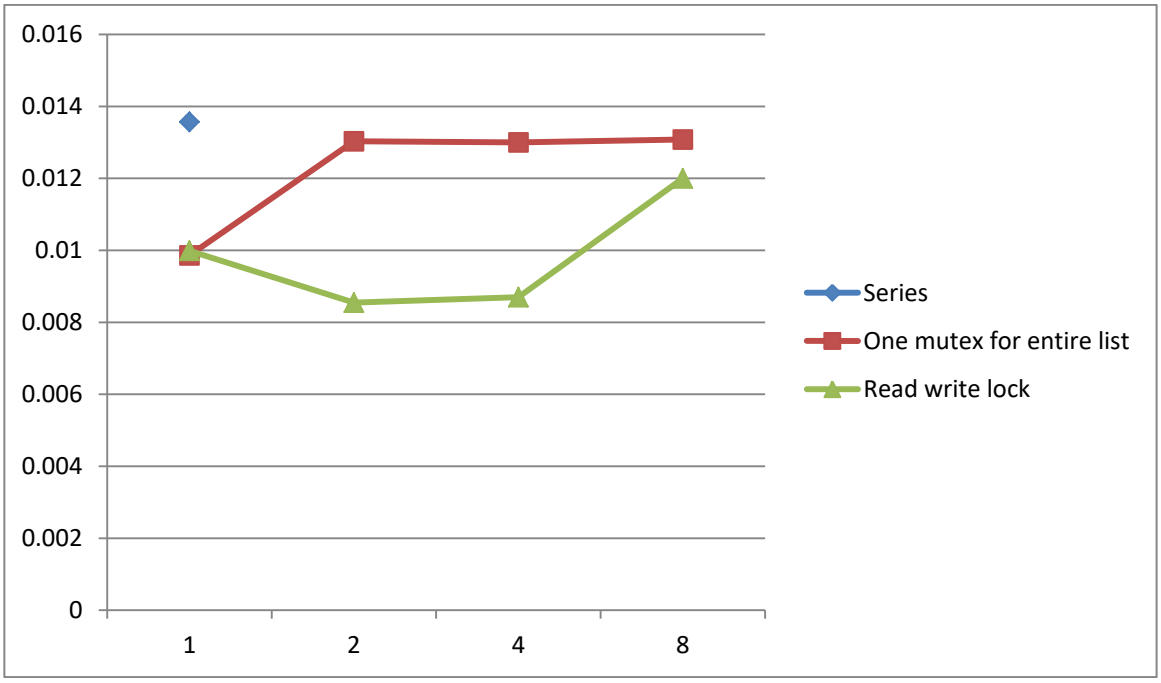
Specifications of the machine

CPU	Intel Pentium N3540 / 2.16 GHz
Max Turbo Speed	2.66 GHz
Number of Cores	Quad-Core
Cache	L2 - 2 MB
64-bit Computing	Yes
Features	Intel Burst Technology

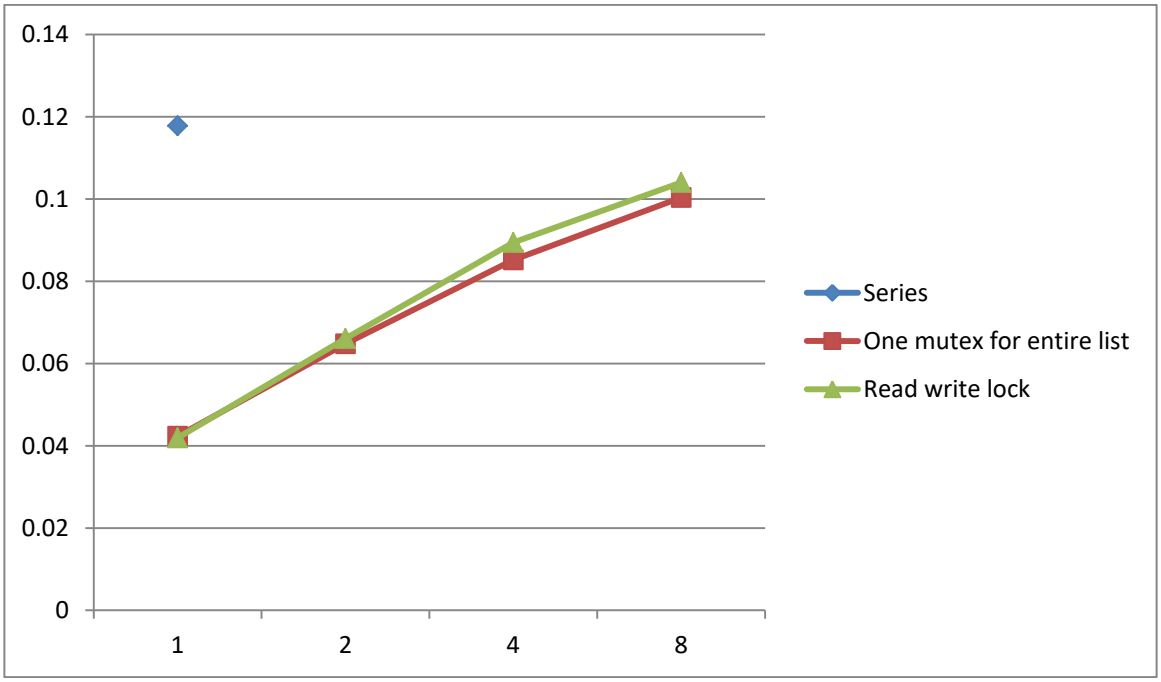
Case 1



Case 2


0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
1
2
4
8
Series
One mutex for entire list
Read write lock

Case 3


0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
1
2
4
8
Series
One mutex for entire list
Read write lock

Step 5:

In the implementation of operations on linked list, operations are shuffled. The orders of member, insert and delete operations are randomized. This randomization also will impact the time because when read operations are not consecutive (i.e. read operations followed by insert and member operations), many read operations can't be carried out parallel. So, the advantage of read-write locks over other will not be fully utilized.

In the case of single thread series case gives the better performance. Even though our observations are bit off, this should be the theoretical scenario. Advantages from the parallel coding techniques become useless since there is only one thread. Therefore the resources use by paralleling techniques will be a waste in this case. So the average time in the series case is lower than other scenarios.

In the mutex scenario only one operation is allowed in the critical section at a time. But the read write block, when one thread doing read operation another thread with read requirements also allowed. Because of this read write block gives better performance than the one mutex for entire list scenario. Case 2 represents this point clearly.

When we observe the graph we can see that, Case 1 average time < case 2 average time < case 3 average time. Since we are increasing the insert, delete operations it takes more time. That explains this time increase. In case 3, with only 50% read operations, the performance of read-write locks is less than mutex.

One mutex for entire list is does not give better performances in each case. This is due to the acquiring and releasing the lock. Since only one mutex for the whole list, threads experience a starvation. Increased thread count gives more starvation time. This is the reason behind the bad performance of one mutex.