CS4532 Concurrent Programming Take-Home Lab 1

170007T : W.A.S.D.Abeygunawardhana

170294R: S.I.Karunarathne

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

Implementation	**No of threads**							
	1		**2**		**4**		**8**	
	Average	**Std**	**Average**	**Std**	**Average**	**Std**	**Average**	**Std**
Serial	0.00143 27027	0.0003 903104						
One mutex for entire list	0.00265 57377	0.0006 526550	0.00301 26582	0.0004 512875	0.00444 11765	0.0008 354524	0.00483 11475	0.0007 491132
Read-Write lock	0.00412 30769	0.0022 933851	0.00243 66972	0.0016 344596	0.00265 26718	0.0009 959401	0.00305 67010	0.0012 826492

- **Case 2:** *n* = 1,000 and *m* = 10,000, *mMember* = 0.90, *mInsert* = 0.05, *mDelete* = 0.05

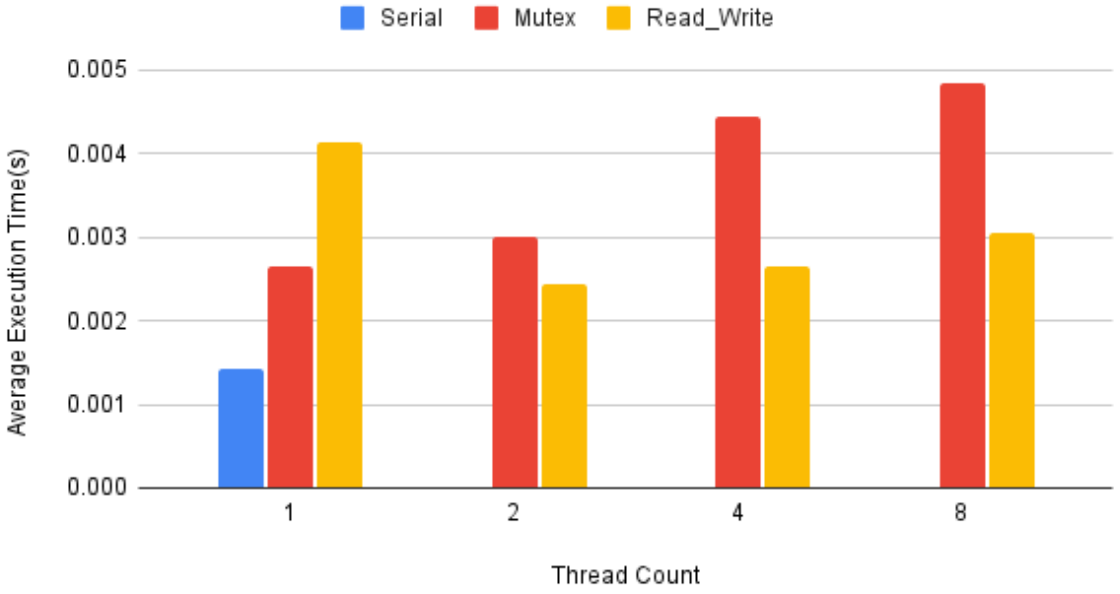
Implementation	**No of threads**							
	1		**2**		**4**		**8**	
	Average	**Std**	**Average**	**Std**	**Average**	**Std**	**Average**	**Std**
Serial	0.01471 42857	0.00390 89005						
One mutex for entire list	0.01692 33871	0.01092 23759	0.01166 66667	0.00115 47005	0.01240 00000	0.00148 15100	0.01144 44444	0.00072 64832

Read-Write lock	0.0154222222	0.0110007650	0.0059845475	0.0004533353	0.0056969697	0.0006786796	0.0054540636	0.0006993115

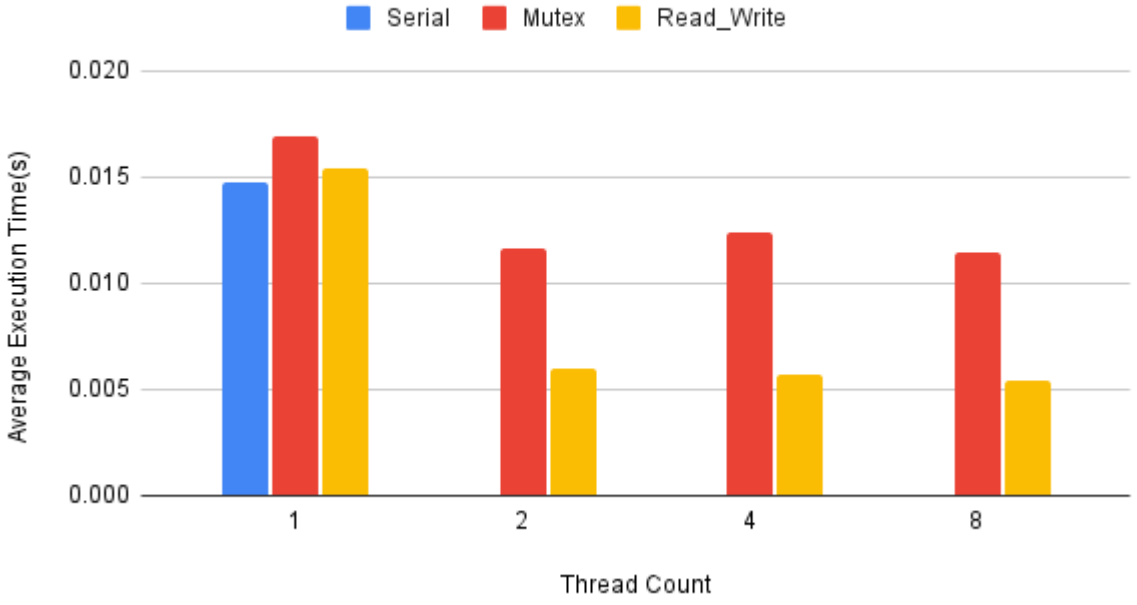
- **Case 3:** n = 1,000 and m = 10,000, $mMember$ = 0.50, $mInsert$ = 0.25, $mDelete$ = 0.25

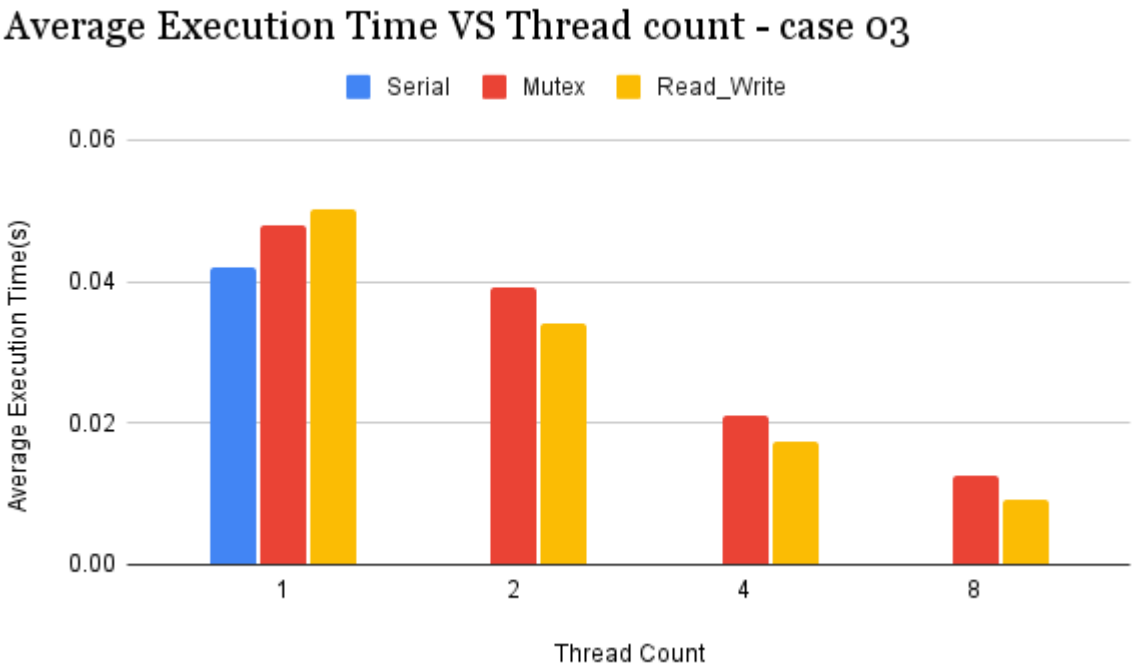
Implementation	**No of threads**							
	1		**2**		**4**		**8**	
	Average	**Std**	**Average**	**Std**	**Average**	**Std**	**Average**	**Std**
Serial	0.0419365079	0.0158016435						
One mutex for entire list	0.0477931034	0.0228698623	0.0391666667	0.0203117372	0.0210000000	0.0027080128	0.0126666667	0.0008164966
Read-Write lock	0.0503141593	0.0196041600	0.0340000000	0.0078102497	0.0175000000	0.0030000000	0.0091666667	0.0009831921

Average Execution Time VS Thread count -Case 01



Average Execution Time Vs Thread Count -case 02





Graph Comparison

- In all three cases, the serial method outperforms the other two methods in terms of average execution time in a single threaded environment. The reason for that is there is overhead in the single mutex method and the read write lock method because of thread creation, thread scheduling and management of the mutexes or the locks.
- When we consider the overall average execution time, it is increasing in the order of Case-01, Case-02 and Case-03 respectively. In these programs the insert and delete percentage is increasing in the respective order which is led to an overhead, as a result the performance of the Case-01, Case-02 and Case-03 reduced respectively.
- In case 1, we can see the read-write lock method performs better than the single mutex method. Here, 99% of the total operations are read-only operations (member). In the single mutex method, there is only one mutex for both read and write operations (all member, insert, delete operations). Therefore, when one thread is in the critical section, all the other threads have to wait. This results in a massive overhead. However,in the read-write method, it uses two locks, one for read and write operations separately. Therefore, multiple read operations can take place together while blocking the threads only when there is a requirement to perform write operations. This reduces the overhead and increases the performance of the linked list. That is the reason for having a better performance of the read write lock method than the single mutex method for the cases with less insert and delete operations (case 1 and 2).

- In case 2, percentage for deletion and insertion are increased compared to case 1. Similar to case one, the read write lock method outperforms the single mutex method for the same reason explained above. In addition, the average execution time of the mutex method and the read write lock method are decreasing with the increase of thread count, due to achieving parallelism.
- In case 3, 50% of the operations are write operations and we can clearly see how the performance of read write locks have decreased compared to other cases. A significant change in the pattern of average execution time with the increase of number threads between the two considered mutual exclusion methods can be seen when compared to case 1 and case 2. That is, there is a weaker performance of the read write lock method. When we compare the gap between the mutex and read-write, it is lower than that of the other two cases. This is because the read write locks tend to perform better when read operations are significantly larger than the write operations, but here we have more equal amounts of read and write operations.

Machine Specification

- ➔ CPU : Intel i5 - 7600K
- ➔ No. of cores : 4
- ➔ Logical processors : 4
- ➔ Clock speed : 3.8GHz
- ➔ Memory : 16GB
- ➔ Operating system : Windows 10 Pro
- ➔ Tools used : VS Code
- ➔ Compile : GCC