CS4532 - Concurrent Programming

Take Home Lab 1 - Report

170214B - H.M.M.V.B Herath

170094E - D.H.M.S.N Dassanayaka

Design

-

- Created three separate files for each mode of execution (link_list_serial.c/ link_list_mutex.c/ link_list_rd_lock.c).
- Designed three separate functions in each file for every basic operation i.e., Member(), Insert(), Delete().
- Users can run each file and give inputs as below.

Mode	Serial Program	Concurrent with Mutex	Concurrent with read_write lock
Input Arguments	• m_member_fraction • m_insert_fraction • m_delete_fraction	• m_member_frac • m_insert_frac • m_delete_frac	• m_member_frac • m_insert_frac • m_delete_frac
mMember	m_member_fraction * m	m_member_frac * m	(int)(m_member_frac * m)
mInsert	m_insert_fraction * m	m_insert_frac * m	(int)(m_insert_frac * m)
mDelete	m_delete_fraction * m	m_delete_frac * m	(int)(m_delete_frac * m)

Results

Case 1:

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

Implementation	No of threads							
	1		2		4		8	
	Average	Std	Average	Std	Average	Std	Average	Std
Serial	0.011255	0.002552						
One mutex for entire list	0.000923	0.00213	0.000987	0.002305	0.001237	0.002840	0.001306	0.002219
Read-Write lock	0.337419	0.226030	0.179651	0.116619	0.113460	0.077317	0.076442	0.046490

Case 2:

n = 1,000 and m = 10,000, m_{Member} = 0.90, m_{Insert} = 0.05, m_{Delete} = 0.05

Implementation	No of threads							
	1		2		4		8	
	Average	Std	Average	Std	Average	Std	Average	Std
Serial	0.016306	0.003151						
One mutex for entire list	0.000973	0.002717	0.001083	0.002277	0.001197	0.002443	0.001364	0.002097
Read-Write lock	0.317853	0.214331	0.188910	0.123286	0.128161	0.084832	0.101881	0.065404

Case 3:

n = 1,000 and m = 10,000, m_{Member} = 0.50, m_{Insert} = 0.25, m_{Delete} = 0.25

Implementation	No of threads							
	1		2		4		8	
	Average	Std	Average	Std	Average	Std	Average	Std
Serial	0.024886	0.002567						
One mutex for entire list	0.001262	0.005052	0.001352	0.003623	0.001419	0.003421	0.001623	0.002421
Read-Write lock	0.278051	0.162753	0.258822	0.157050	0.278325	0.201267	0.243522	0.155711

(Assumption - Assume that 500 sample size is normally distributed)

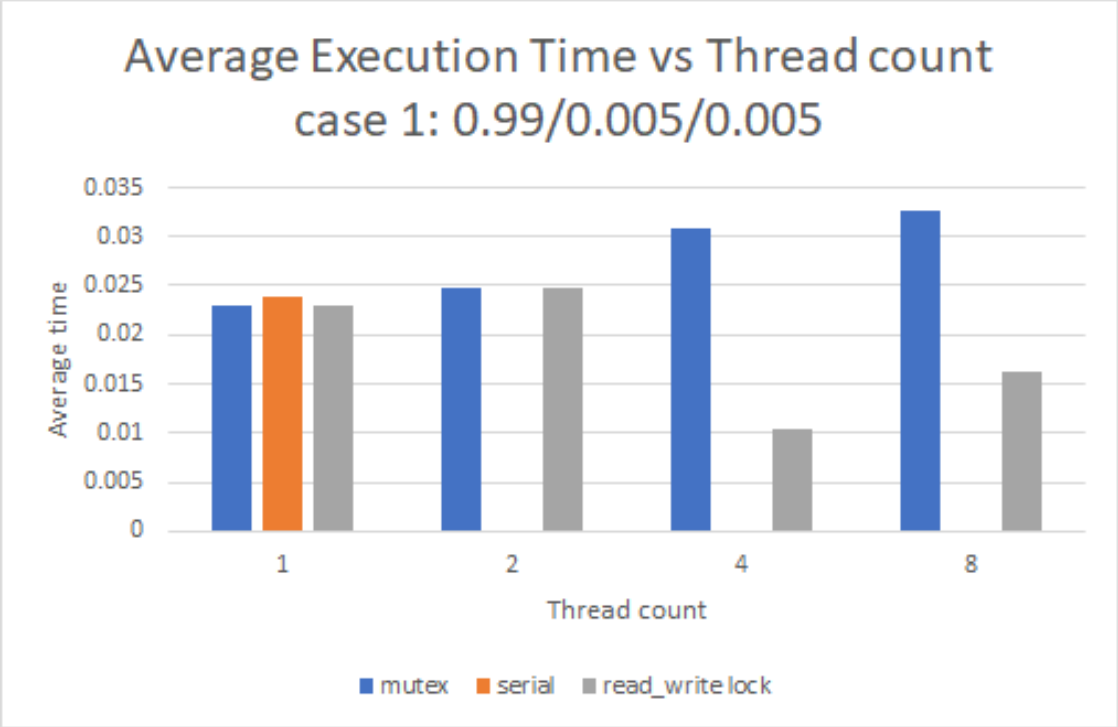
Computer(s) Used

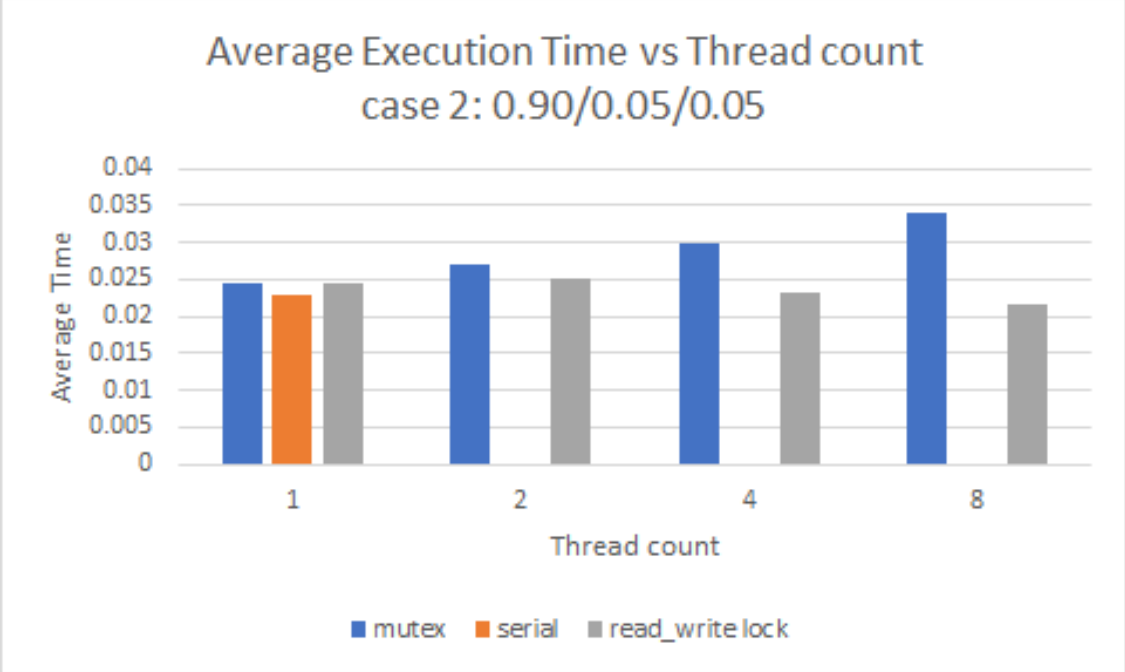
- Cores - 4
- Ram - 16Gb
- Clock speed - 1.50 GHz
- Cache size
  - L1 - 320 KB
  - L2 - 2.0 MB
  - L3 - 8.0 MB
- CPU Model - Intel(R) Core(TM) i7-1065G7

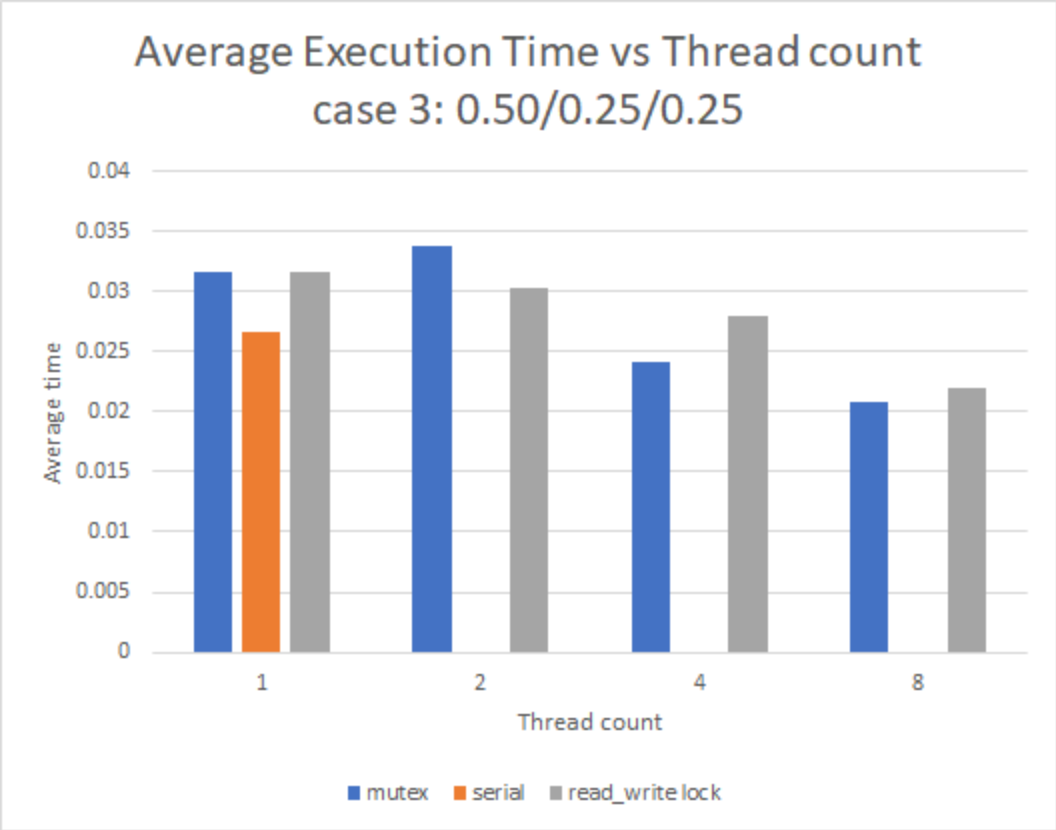
Applications/Simulators/Emulators/Libraries

- C programming language (gcc.exe (Rev5, Built by MSYS2 project) 10.3.0) - pthread concurrent library

Plots for the average execution time against the number of threads







Observations and Evaluation

- When comparing serial programs with the relevant parallel programs with 1 threads, serial programs show better execution times since overheads like thread creation, scheduling, thread termination etc. are added to the parallel programs.

- In Case 1 read-write locks have performed better.
- Evaluation - 99% of the operations in this case are read operations and there are only a few insert and delete operations. Read-write locks allow multiple threads to obtain the lock for read at the same time. But mutex allows only one thread at a time to read the link list. As it serializes the process and makes other threads wait, mutex does not give the expected parallelism in the case of majority members.
  Therefore read-write locks perform better in this case.

- When compared to case 1, all values of average execution time have increased in case 2, and when compared to case 2, all values of average execution time have increased in case 3.

- Evaluation - write operations have more atomic instructions than the read operations. Since in case 2 and 3 read operations have been decreased and the insert, delete operations have increased, the overhead and the average execution time have been increased.

- In both Case 1 and Case 2, average execution time for the mutex has increased with the number of threads.
- Evaluation - when there are multiple threads and we use mutex, though we use multiple threads they are blocked to access the linked list when a one thread is reading or writing.

- Case 3 - The difference between average execution time of the mutex and the read-write locks has decreased.
- Evaluation - Since there are equal amounts of read and write operations, both mutex and read write locks are working in the same way (sequentially). Therefore both mutex and read write locks take similar execution times.

Conclusion

When the amount of read operations are higher, read-write locks perform better with multiple threads. If write locks are higher, both mutex and read-write locks behave in the same way. Therefore we get higher execution time even though we use multiple threads.