CS4532 Concurrent Programming

Take Home Lab 1

180114D - D.D.J.H. De Alwis

Design

1. Serial program

- The program takes input values for the case number and the number of tests needed to run to get the confidence level. Depending on the case its probability is assigned to insert, member, and delete.
- The linked list is populated with n random values using genUniqueRandNum() function.
- Then create an array of the size of number of operations and value assigned for operation type depending on the probability.
- Depending on the value of the array, the program performs the corresponding operation on the linked list.
- Tests are run for the number of times defined to get a confidence level and each iteration mean is calculated. Finally, the average of those tests was calculated, and give the mean and std.

2. Parallel program (based on Pthreads) with one mutex for the entire linked list

- The program implements the same input arguments, linked list population, and random operation selection as the serial program. Additionally the number of threads needed to give to the program.
- A mutex is initialized and a user-defined number of threads are created using the pthread_create function.
- These threads are joined with the main program.
- The program uses start and end variables to calculate the time which is initialized just before thread creation and calculate at the end of all threads.
- Total insertion, member, and delete operations and the total number of all operations are shared globally among all the threads.
- To each thread an equal number of operations has been given to perform.
- The critical parts of each operation are locked with the mutex separately.
- The thread checks whether each individual operation is finished before acquiring the mutex lock.
- Inside the mutex lock, the current count of each operation is checked against the total number of operations for each operation.
- Member, insert and delete operations are carried out within the lock.
- Once the thread operations are finished, the mutex is destroyed.

3. Parallel program (based on Pthreads) with read-write locks for the entire linked list

- The program implements the same input arguments, linked list population, and random operation selection as the serial program. Additionally the number of threads needed to give to the program.
- A read-write lock is initialized and a user-defined number of threads are created using the pthread_create function.
- These threads are joined with the main program.
- The program uses start and end variables to calculate the time which is initialized just before thread creation and calculated at the end of all threads.
- Total insertion, member, and delete operations and the total number of all operations are shared globally among all the threads.
- To each thread an equal number of operations has been given to perform.
- Inside the read lock, the Member function is locked with a read lock as it is not updated and only read. The count member variable is updated outside the lock.
- Inside the write lock, the Insert function and Delete function are locked with write locks as they are updated by all the threads. The count insert and delete variables are updated outside the lock.
- Once the thread operations are finished, the read-write lock is destroyed.

Finally a simple bash program is created to run and compile all the program with different type of input combinations to make the task easier.

Results

Case 1

n = 1000 and m = 10000 $m_{Member} = 0.99$, $m_{Insert} = 0.005$, $m_{Delete} = 0.005$

Implementation	No of threads							
	1		2		4		8	
	Average	Std	Average	Std	Average	Std	Average	Std
Serial	0.015305	0.001113						
One mutex for entire list	0.015896	0.001948	0.036868	0.002012	0.043816	0.003077	0.045228	0.002263
Read-Write lock	0.015270	0.000949	0.017442	0.000950	0.018437	0.001163	0.020042	0.001137

Case 2

n = 1000 and m = 10000 $m_{Member} = 0.90$, $m_{Insert} = 0.05$, $m_{Delete} = 0.05$

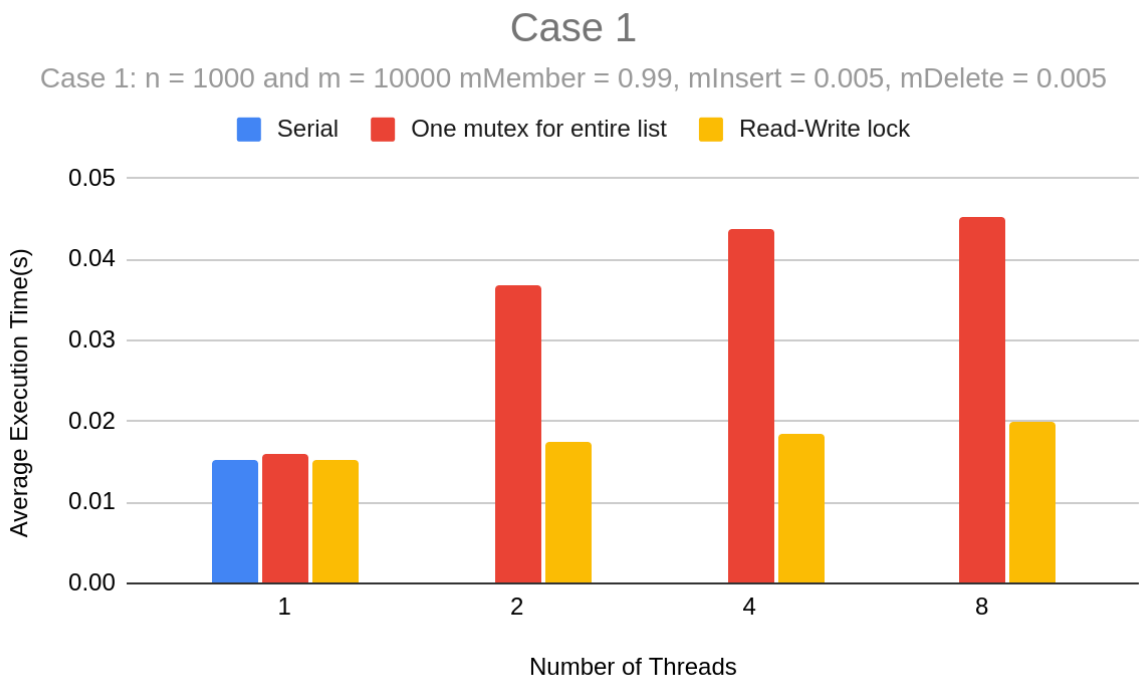
Implementation	No of threads							
	1		2		4		8	
	Average	Std	Average	Std	Average	Std	Average	Std
Serial	0.020640	0.001285						
One mutex for entire list	0.021672	0.001810	0.046676	0.002118	0.050677	0.002201	0.053110	0.001797
Read-Write lock	0.022344	0.001676	0.030776	0.001162	0.034522	0.001158	0.037590	0.001369

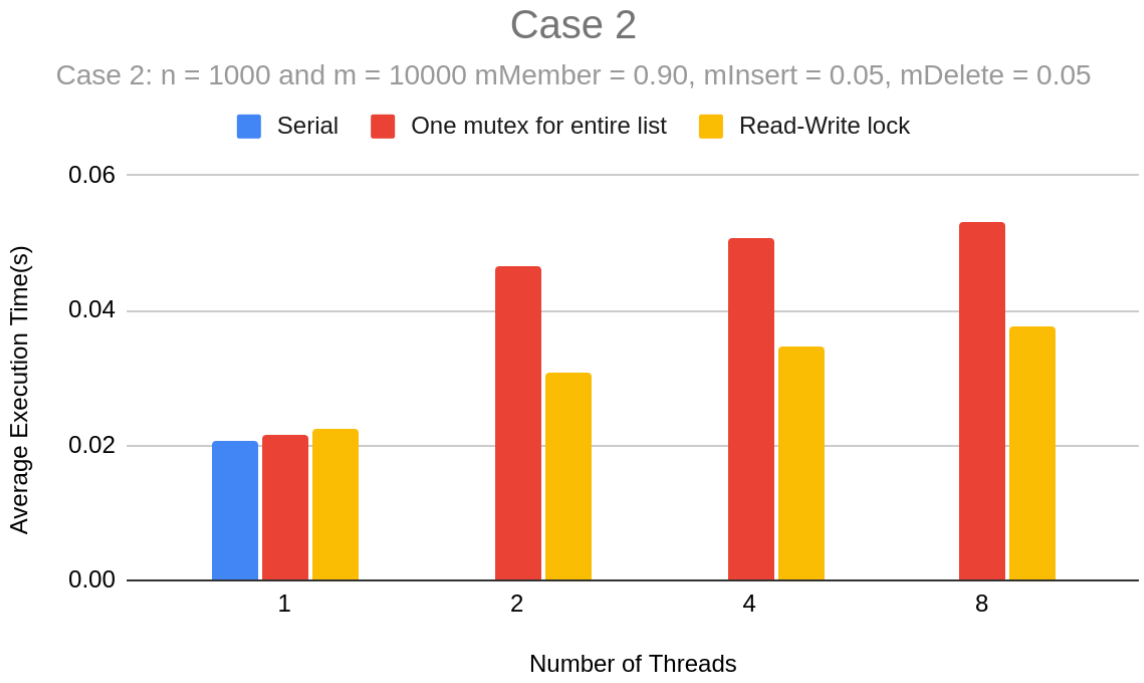
Case 3

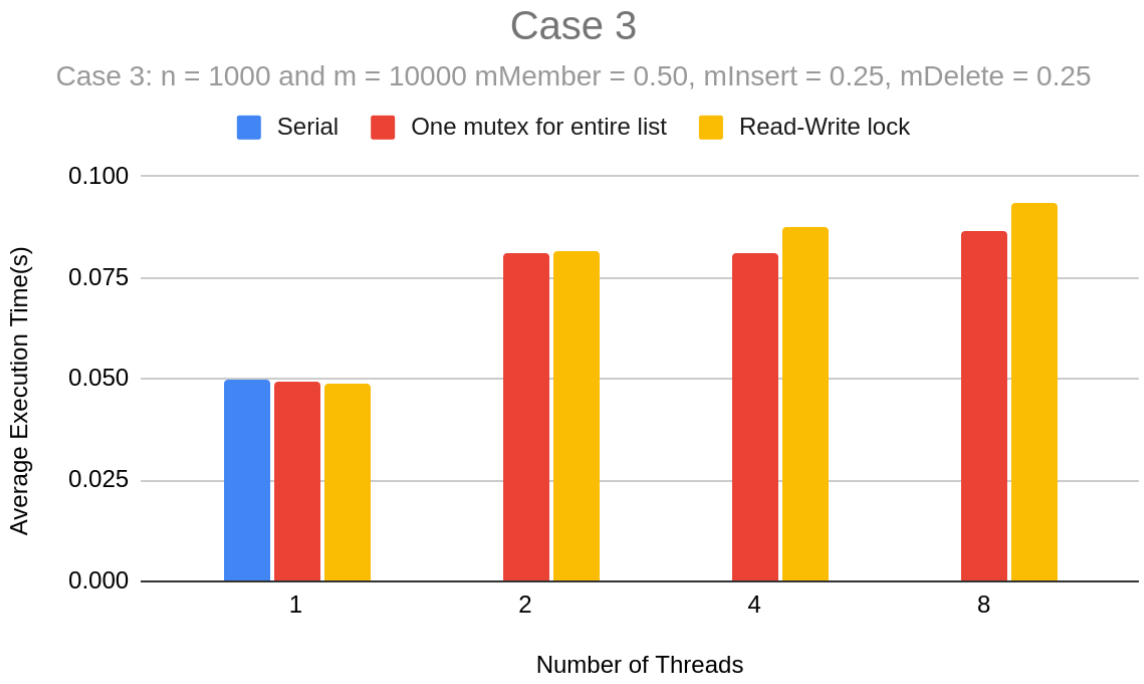
n = 1000 and m = 10000 $m_{Member} = 0.50$, $m_{Insert} = 0.25$, $m_{Delete} = 0.25$

Implementation	No of threads							
	1		2		4		8	
	Average	Std	Average	Std	Average	Std	Average	Std
Serial	0.049642	0.004489						
One mutex for entire list	0.049409	0.003728	0.080837	0.004370	0.080997	0.004418	0.086617	0.003754
Read-Write lock	0.048959	0.004103	0.081362	0.004801	0.087401	0.004553	0.093549	0.005168

Plots


Case 1
Case 1: n = 1000 and m = 10000 mMember = 0.99, mInsert = 0.005, mDelete = 0.005
Serial
One mutex for entire list
Read-Write lock
0.05
0.04
0.03
0.02
0.01
0.00
Average Execution Time(s)
1
2
4
8
Number of Threads


Case 2
Case 2: n = 1000 and m = 10000 mMember = 0.90, mInsert = 0.05, mDelete = 0.05
Serial
One mutex for entire list
Read-Write lock
0.06
0.04
0.02
0.00
Average Execution Time(s)
1
2
4
8
Number of Threads



Discussion

When comparing the performance of the serial program, parallel program with a single mutex, and parallel program with read-write locks, the following observations were made

In the single-threaded case, the serial program has better performance as the parallel approaches with mutex and read-write locks introduce an overhead with thread creation and scheduling. In this scenario, the mutex program has less overhead than the read-write lock program. Therefore, in the single-threaded case, the serial program has the best performance, followed by the mutex program and the worst performance is with the read-write lock program.

Case 01: In a scenario where 99% of the total operations are reads and the rest are writes, the read-write lock program has better performance than the mutex program. In the mutex program, a single lock is used for both reads and writes. As a result, even though the member function is read-only, all the threads are blocked with the mutex lock, moving the threads into a waiting state and incurring significant overhead. However, in the read-write lock program, since the member function is read-only, when a thread enters it, it is read-locked, allowing multiple threads to read simultaneously. Therefore, in this case, the read-write lock program shows a significant performance gain compared to the mutex program.

Case 02: When compared to the average values of Case 1, Case 2 has increased time consumption due to the decrease in the member operation count and the increase in delete and insert operations overhead. The mutex lock performance decreases with the increment of the number of threads as more threads are blocked when the number of threads increased.

Case 03: In a scenario where 50% of the operations are read operations (member function) and 50% of the operations are write operations (delete and insert), the time consumption is higher due to the increased write operations. The difference in time consumption between mutex and read-write locks is reduced since half of the operations are writes. This is because the parallelism of the read-write locks is reduced with the increment of write operations. The read-write lock program is much better in performance when read operations are higher compared to write operations.