CS4532 - Concurrent Programming

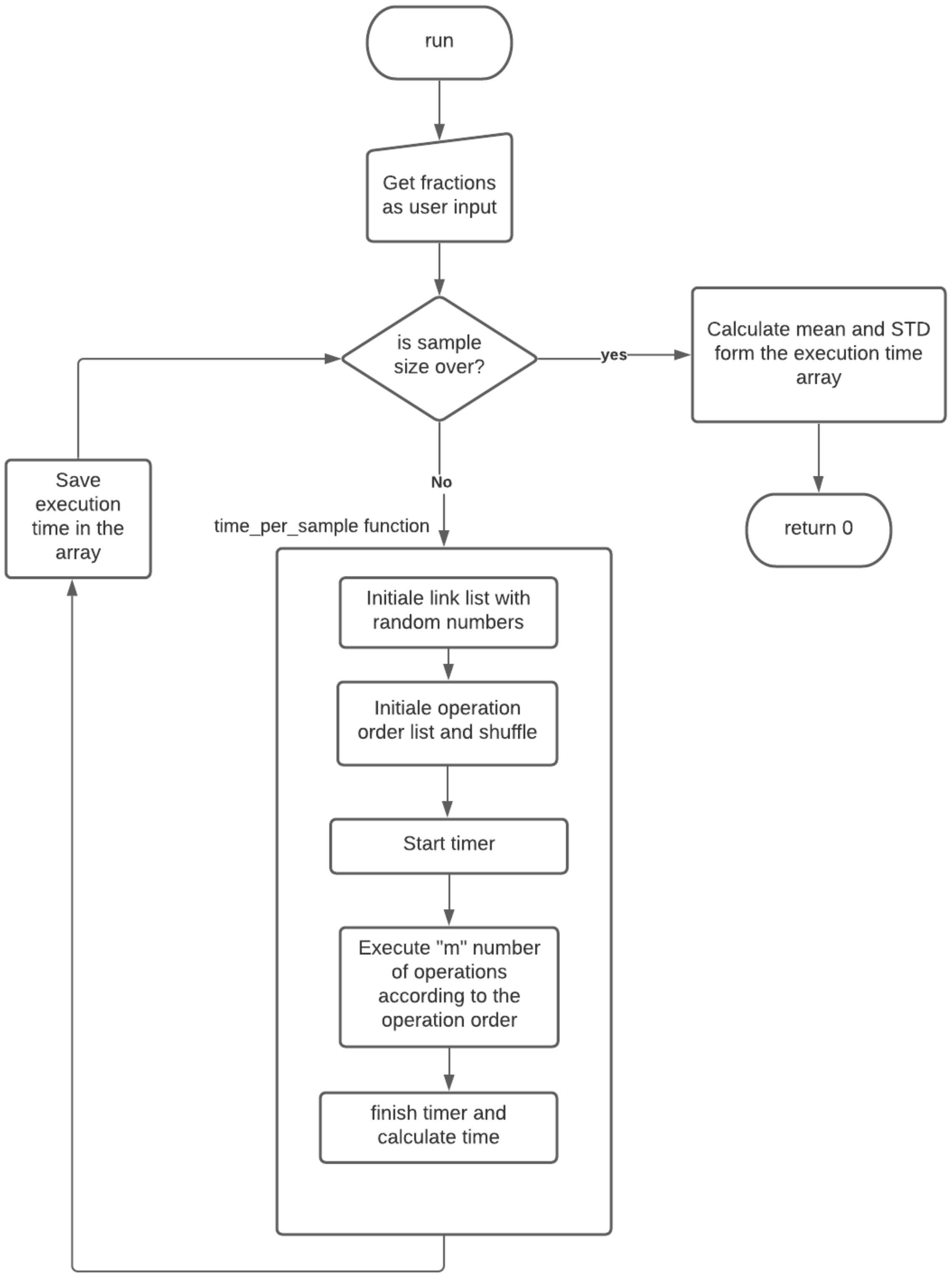
Take Home Lab 1 - Report

170406P - R.A.K.C. Nilanga

170521M - R.M.H.N.Rathnayake

Design

First of all, We create a function for the basic operation in the link list and put those functions in separate files(member.c/insert.c/delete.c). After that, create 3 different files for each mode of execution(link_list_serial.c/ link_list_mutex.c/ link_list_rd_lock.c).


run
Get fractions as user input
is sample size over?
yes
Calculate mean and STD form the execution time array
return 0
No
time_per_sample function
Initiale link list with random numbers
Initiale operation order list and shuffle
Start timer
Execute "m" number of operations according to the operation order
finish timer and calculate time
Save execution time in the array

Above diagram describes the main schelton of our program. In the concurrent part, we add mutex and read write lock right before the basic operation runs. After that unlock those locks right after the basic operation terminates.(As our lecture slides)

Describing Experimental/Simulation Setup

Type of Simulation

This is an experiment to show how the serial program, concurrent program with mutex and read-write lock in pthread work. We run our program for different combinations of execution methods, different fractions of basic operations and thread count.Theoretically, there are advantages and disadvantages in the mutex and read and write locks with thread count and fractions of brasic operations.

Serial Program	**Concurrent with Mutex**	**Concurrent with read_write lock**
Just only one thread (main thread/ only one core)	Can have multiple thread (can use more cores)	Can have multiple thread (can use more cores)
No additional overhead (no post protocol and pre protocol)	Has additional overhead when generate threads	Has additional overhead when generating threads.
	Lock the overall link list when one thread acquires it.	There are two locks. ● Read lock - Block all writing threads and Allow to all reading threads. So this is fast when we have lot of reading parts ● Write lock - Block all writing and reading threads.

Number of Experiments/Simulations

In this experiment we run m = 100000 (for n=1000 size link list) operations per sample.As a description of assignment we have to make sure our execution time means lie within 95% confidence interval and 5% accuracy. So, We run all experiments for 500 sample sizes. Then calculate sample size using that mean and standard deviation. But all sample sizes lie within 2 and 100. So we can conclude that these mean and standard deviations have more than 5% accuracy.

Sampling method :- n = $\dfrac{(100*Z*S)^2}{(r*\overline{X})^2}$

Z=1.96, S = defer each experiment, r =5, X = defer each each experiment

Case 1: n = 1,000 and m = 10,000, m_{Member} = 0.99, m_{Insert} = 0.005, m_{Delete} = 0.00

Implementation	1	2	4	8
Serial	n= 9.150191885			
One mutex per list	n=12.24425961	n=4.027695176	n=3.309568911	n=2.280767452
Read-write	n=24.18665177	n=12.600114	n=14.73733363	n=30.85017269

Case 2: n = 1,000 and m = 10,000, m_{Member} = 0.90, m_{Insert} = 0.05, m_{Delete} = 0.05

Implementation	1	2	4	8
Serial	n= 7.342144775			
One mutex per list	n=48.47492795	n=4.637227199	n=2.132655129	n=2.211581631
Read-write	n=5.850527839	n=10.34108132	n=14.73733363	n=25.22112739

Case 3: n = 1,000 and m = 10,000, m_{Member} = 0.50, m_{Insert} = 0.25, m_{Delete} = 0.25

Implementation	1	2	4	8
Serial	n= 3.158099967			
One mutex per list	n=2.642443972	n=4.637227199	n=2.132655129	n=2.303768961
Read-write	n=2.779666322	n=4.537376972	n=3.024181265	n=3.565184962

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

Random number generations :-
For random number generation, we use the current time first number as seed. That makes sure every time we generate completely different random numbers.

1. **Computer(s) Used**

- Cores - 6
- Ram - 8Gb
- Clock speed - 2.60GHz
- Cache size- L1- 384KB and L2- 1.5MB and L3- 12 MB
- CPU Model - Intel(R) Core(TM) i7-9750H

Applications/Simulators/Emulators/Libraries

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

Results

Sample Size = 500 (for all experiments)

Case 1: n = 1,000 and m = 10,000, $m_{Member} = 0.99$, $m_{Insert} = 0.005$, $m_{Delete} = 0.005$

Implementation	1		2		4		8	
	Average	Std	Average	Std	Average	Std	Average	Std
Serial	0.013270	0.001024						
One mutex per list	0.013656	0.001219	0.017130	0.000877	0.019436	0.000902	0.020272	0.000781
Read-write	0.014706	0.001845	0.010712	0.000970	0.008026	0.000786	0.006126	0.000868

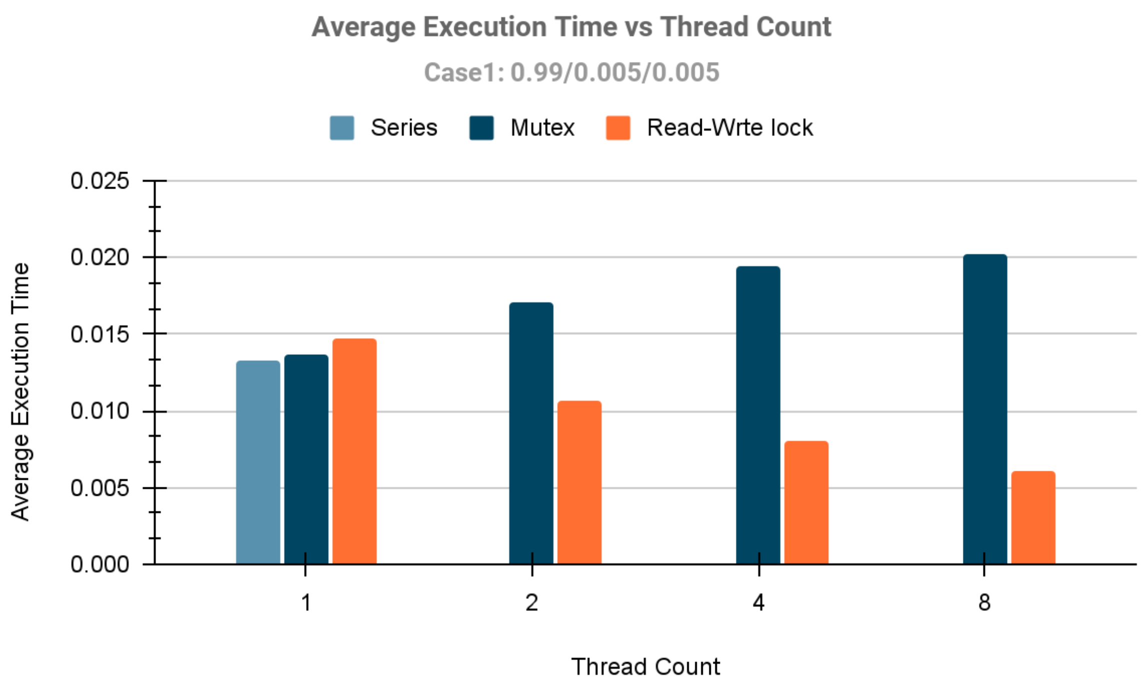
Case 2: n = 1,000 and m = 10,000, $m_{Member} = 0.90$, $m_{Insert} = 0.05$, $m_{Delete} = 0.05$

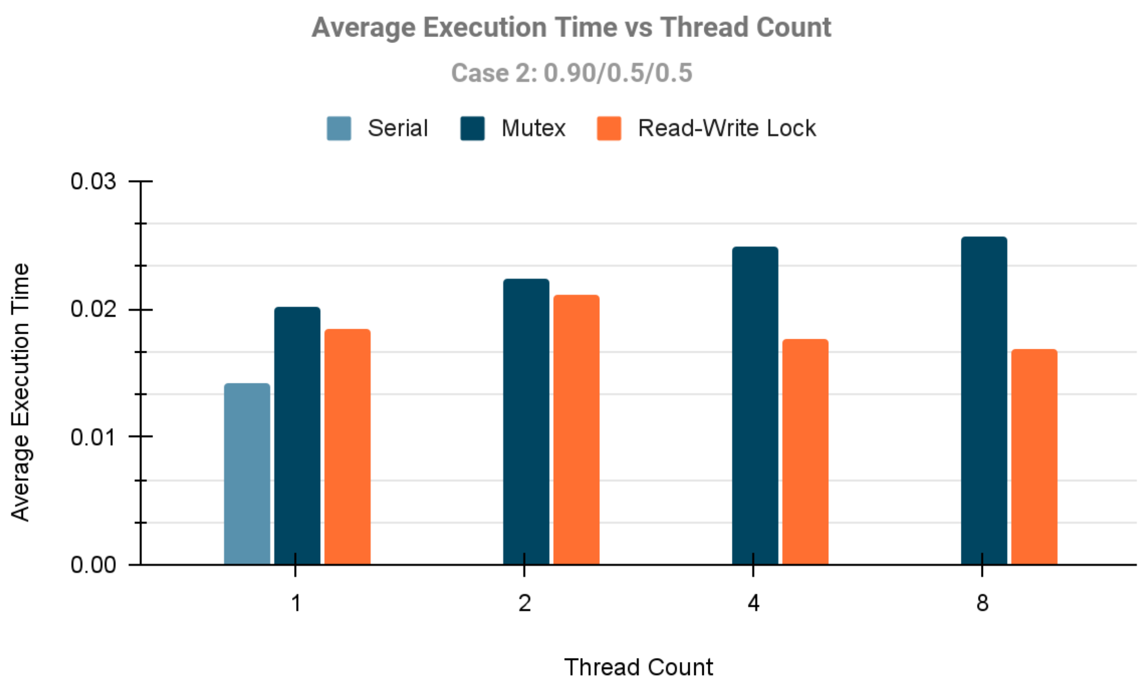
Implementation	1		2		4		8	
	Average	Std	Average	Std	Average	Std	Average	Std
Serial	0.014192	0.000981						
One mutex per list	0.020252	0.003597	0.022354	0.001228	0.024910	0.000928	0.025674	0.000974
Read-write	0.018524	0.001143	0.021174	0.001737	0.017632	0.001254	0.016860	0.002160

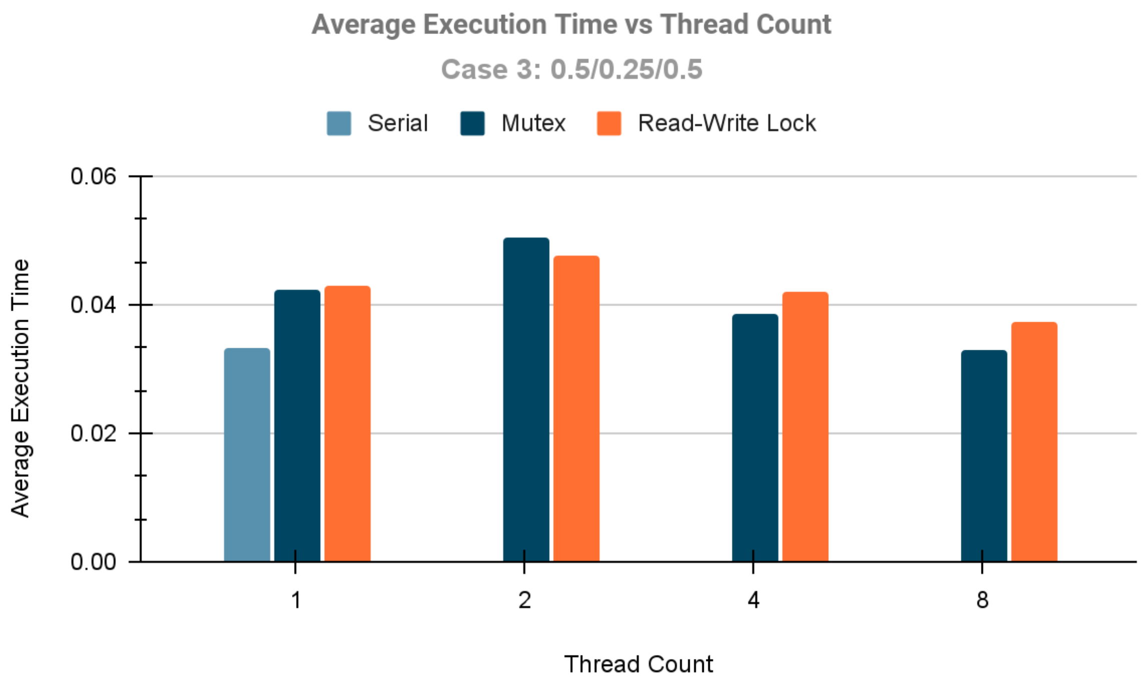
Case 3: n = 1,000 and m = 10,000, $m_{Member} = 0.50$, $m_{Insert} = 0.25$, $m_{Delete} = 0.25$

Implementation	1		2		4		8	
	Average	Std	Average	Std	Average	Std	Average	Std
Serial	0.033264	0.001508						
One mutex per list	0.042442	0.001760	0.050524	0.009757	0.038448	0.002481	0.033058	0.001280
Read-write	0.043074	0.001832	0.047516	0.002582	0.042130	0.001869	0.037328	0.001798

<u>Plots for the average execution time against the number of threads</u>







Observations and Evaluation

- When comparing the serial program and the parallel programs with 1 thread, the serial program has better execution time as the parallel program has some costs (overheads) such as thread creation, scheduling, thread termination, pre protocol, post protocol etc.
- In Case 1 read-write locks have performed better when comparing parallel programs.
  Evaluation - Most of the operations in this case are read operations and there are very few write operations(insert and delete). Read-write locks allow multiple threads to obtain the lock for reading at the same time and read the linked list simultaneously. But mutex allows only one thread at a time to access the linked list for both read and write operations. As it serializes the list and makes other threads to wait, mutex does not give the expected parallelism in the case of majority members. Therefore read-write locks perform better in this case.
- Compared to case 1, all values of average execution time have increased in case 2, and compared to case 2, all values of average execution time have increased in case 3.
  Evaluation - As the read operations (member) have decreased and the write operations (insert, delete) have increased, the overhead and the average execution time will be increased because write operations have more atomic instructions than read operations.
- In both Case 1 and Case 2, average execution time for the mutex has increased with the number of threads.
  Evaluation - The reason is when there are multiple threads in the mutex based linked list program, even if we have multiple threads, they are blocked and waiting to read the linked list and therefore the average execution time is increased. (Program behaves sequentially with mutexes)
- Case 3 - The difference between average execution time of the mutex and the read-write locks has decreased.

Evaluation - In this case 50% of operations are read operations and 50% of operations are write operations. Therefore the threads can be blocked as both mutex and read-write blocks performed in the same way for write operations.

Conclusion

The read-write lock has better performance when the read operations(Member) is higher compared to the write operation(insert and delete) with multiple threads. If the write operations are higher, no matter if we use mutex or read-write locks the program behaves sequentially and we will get a higher execution time even if we use multiple threads.