CS 4532 – Concurrent Programming

Lab 1

P. Sarveswarasarma – 180572D
T.Sathulakajan – 180573G

Step 1

The entire program was written in C independently in three files: one for the serial program, one for the mutex program, and one for the read-write lock program. The README.txt file contains instructions on how to run our program.

The linked list is randomly filled with 1000 distinct numbers between 0 and 216-1.

The serial program has been carried out in the order depicted below.

A specific order has been followed in the serial program as shown below.

1. Insert operation (Finish given number of Member operations)
2. Delete operation (Finish given number of Delete operations)
3. Member operation (Finish given number of Member operations)

When randomly choosing the operation type, the actual time taken for only the operations cannot be measured. In that regard, the time taken to select the operation randomly too is calculated. Thus, this order of operations was followed.

With specified numbers of threads as mentioned in the lab description, mutex and read-write lock programs were executed for parallel computations.

Step 3

Case 1 :

n = 1,000 and m = 10,000, m_{Member} ***= 0.99,*** m_{Indert} ***= 0.005,*** m_{Delete} ***= 0.005***

Implementation	Number of threads							
	1		2		4		8	
	Average	Std	Average	Std	Average	Std	Average	Std

Serial	16.64	9.8775						
One mutex for entire list	7.18	7.231	12.13	7.492	12.76	5.927	22.51	9.7604
Read-Write lock	9.33	7.590	15.02	9.902	14.23	6.2746	16.06	9.910

Case 2 :

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

Implementation	Number of threads							
	1		2		4		8	
	Average	Std	Average	Std	Average	Std	Average	Std
Serial	29.53	13.74						
One mutex for entire list	20.97	9.8043	17.22	4.829	27.29	9.650	26.87	9.659
Read-Write lock	21.96	9.779	27.62	13.718	16.19	9.902	6.75	6.659

Case 3 :

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

Implementation	Number of threads							
	1		2		4		8	
	Average	Std	Average	Std	Average	Std	Average	Std
Serial	36.15	9.469						
One mutex for entire list	57.01	16.979	81.37	17.81	72.43	15.602	49.18	9.683
Read-Write lock	47.5	13.166	66.95	18.525	52.2	16.31	50.08	16.355

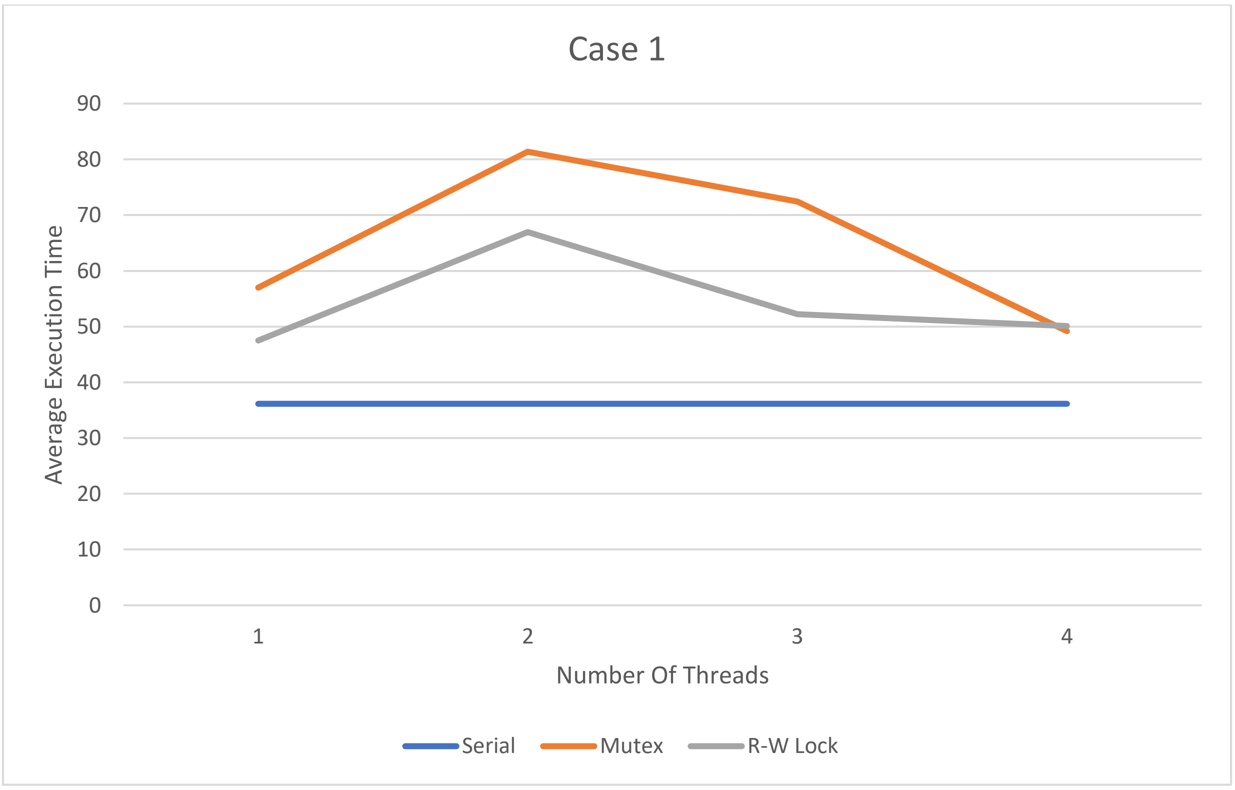
Specifications of the machine used

Operating System	Windows 11 64-bit
Architecture	x64
Cores	4/8 (Core/ Thread)
Model	11th Gen Intel(R) Core (TM) i5-11300H @ 3.10GHz 2.61 GHz
Memory	8GB DDR4 3200 MHz

Step 4

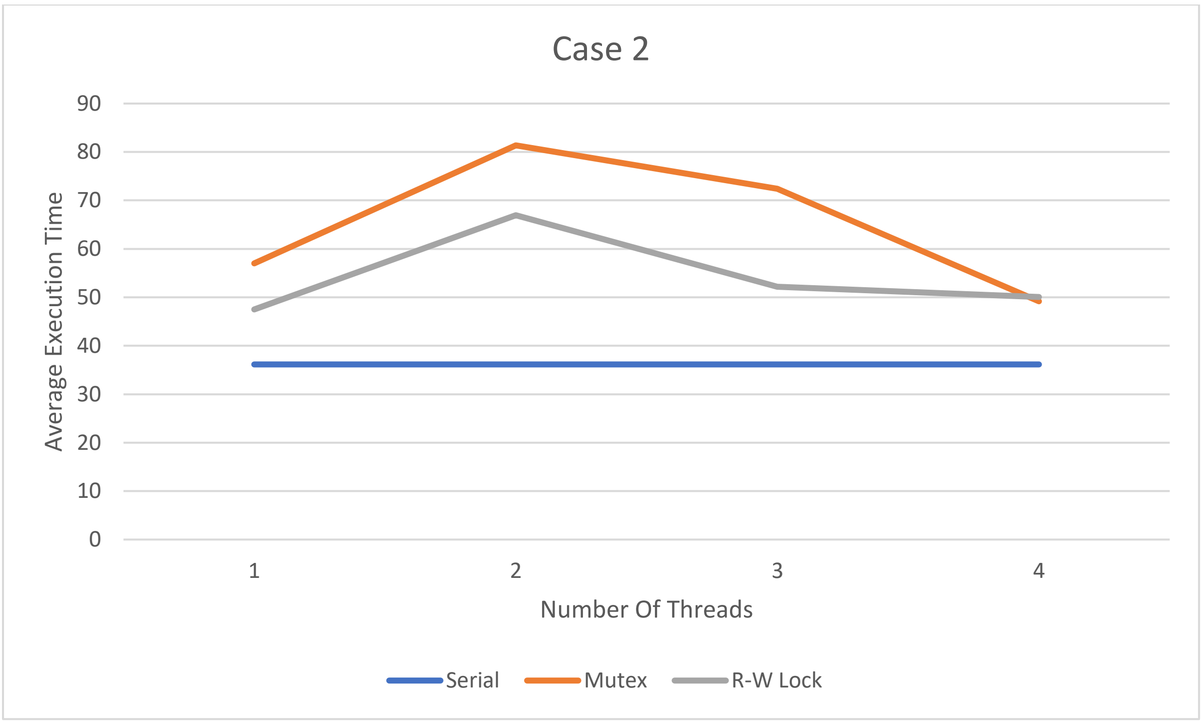
Case 1 :

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



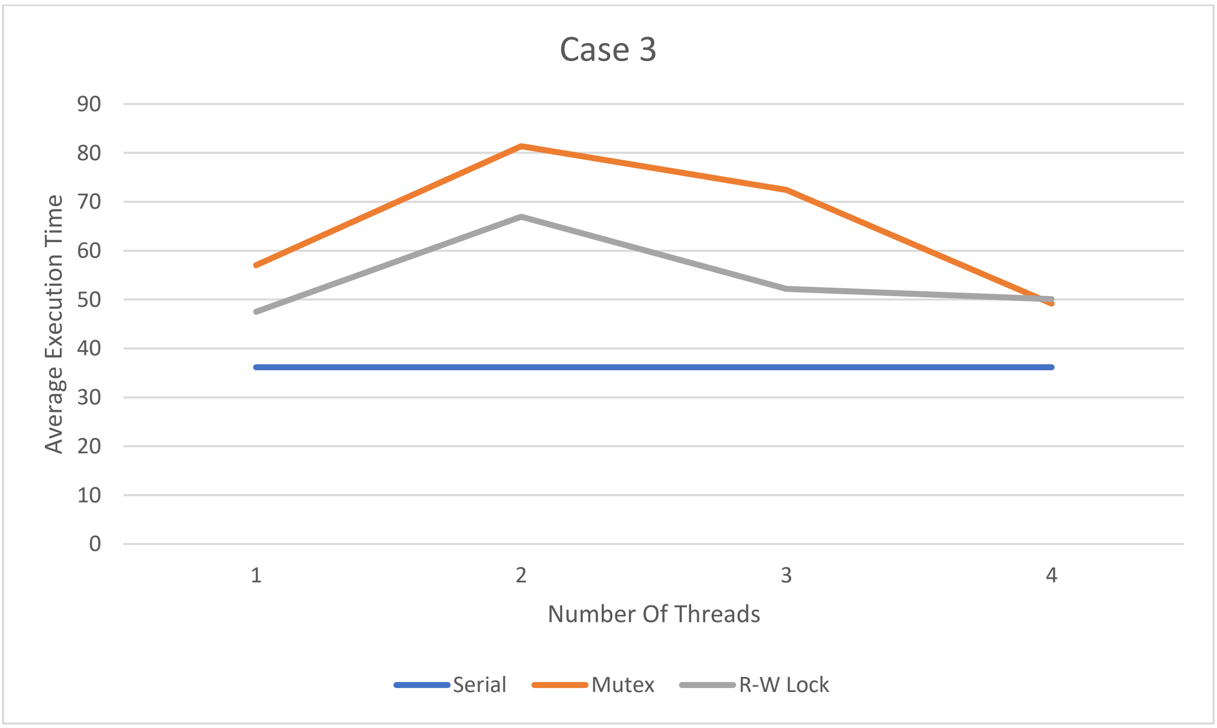
Case 2 :

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



Case 3 :

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



Step 5

Observations and Conclusions:

All times were measured in milliseconds.

One thread is all that exists in serial programming (main thread). As a result, execution time is independent of the number of threads.

There is only one mutex that manages the parallelism for the entire list. Since others are barred, only one thread can obtain the lock and access the list for any form of action. Therefore, the execution time increases regardless of the read-and-write operation percentage.

In a read-write lock, more than one thread can read the same node at once, but only one thread can write to a specific node at once. In more specific terms, writing is halted while a thread performs a read operation. Both reading and writing are prohibited while a thread is carrying out a writing operation.

As a result, the time needed to complete write operations is longer than the time needed to complete read operations. Based on observations made from the table and graph, it can be seen that the program's execution time increases when read and write blocks are used, but it decreases when no write operations are used in cases 1, 2, and 3.

As a result of the observation, it has been determined that the Read Write Lock is the optimal method for enabling parallelism for operations in a list when the number of read operations is more than the number of write operations. However, when the number of write operations is more than the number of read operations, there is no appreciable difference in execution time between a program using one mutex and a program utilizing a read and write lock.