Explanation about how to implement

Member, Insert, Delete functions are added. Member function is used to check whether value is in a linked list or not. In this function, We take each node value and from head to successor nodes and check with given value.

This member function is actually a search function. Insert and delete functions are implemented. In insert function we traverse through the node where we need to put a new node before that, and change the predecessor node next point to new node and new node nest point to successor node. In deletion, we delete a node and we change the predecessor point to successor node. All checks are done inside insert, delete functions such as head deletion, if a value not exist.

Multiple threads can access the linked list in concurrent programming, In here the shared data/ resource is linked list. We might have race conditions. So this prevention can be done in two ways.

First way is using mutex and other way read and write lock. Using a mutex to lock the operations of linked list. So if a thread need to access the linked list, that will first get the mutex. With this mutex, we will not have race conditions but it is not efficient because searching a linked list will not make race conditions if write doesn't happens.

So further, we can use read write lock, With this lock when a thread search in linked list(means using member function) it will acquire the readlock. After read it will release the lock. If a thread is doing a write operation like insert, delete then it acquire write lock.
When a thread is acquired read lock other threads also can acquire the read lock not the write lock and if a thread acquires the write lock no one can acquire the read or write lock until it releases. It makes read access concurrently.

Standard deviation and average for each case

N - num of elements
mM -fraction of member
mD =fraction of delete

Op -num of operations
mI - fraction of insert

n=500 Op =10000 mM =0.99 mI=0.005 mD=0.005

Implementation	No Of Thread			
	1	2	4	8

	Avg	Std	Avg	Std	Avg	Std	Avg	Std
serial	13.41	2.83233	13.41	2.83233	13.41	2.83233	13.41	2.83233
mutex	14.78	5.58782	27.68	3.06785	26.36	1.87767	27.77	2.18745
Read-write lock	14.66	2.73111	8.35	1.43107	7.29	2.98580	7.76	1.21539

n =500, m=10000, mM =0.90, mI= 0.05, nd =0.05

Implementation	No Of Thread							
	1		2		4		8	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std
serial	21.72	3.68748	21.72	3.68748	21.72	3.68748	21.72	3.68748
mutex	19.91	3.69026	36.26	3.50359	37.64	1.95670	41.90	3.99621
Read-write lock	17.88	2.59478	17.49	1.25122	22.05	1.95670	24.15	3.9962

n =500, m=10000, mM =0.50, mI= 0.25, nd =0.25

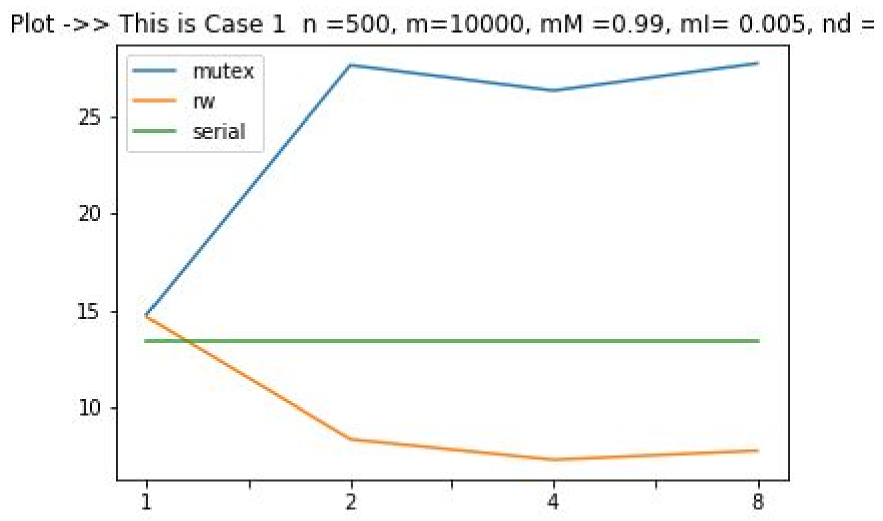
Implementation	No Of Thread							
	1		2		4		8	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std
serial	54.50	10.1846	54.50	10.1846	54.50	10.1846	54.50	
mutex	48.78	5.28497	75.64	6.15542	93.42	3.60213	95.19	3.84450
Read-write lock	48.27	6.79253	77.11	6.63917	93.42	19.1169	95.19	5.43779

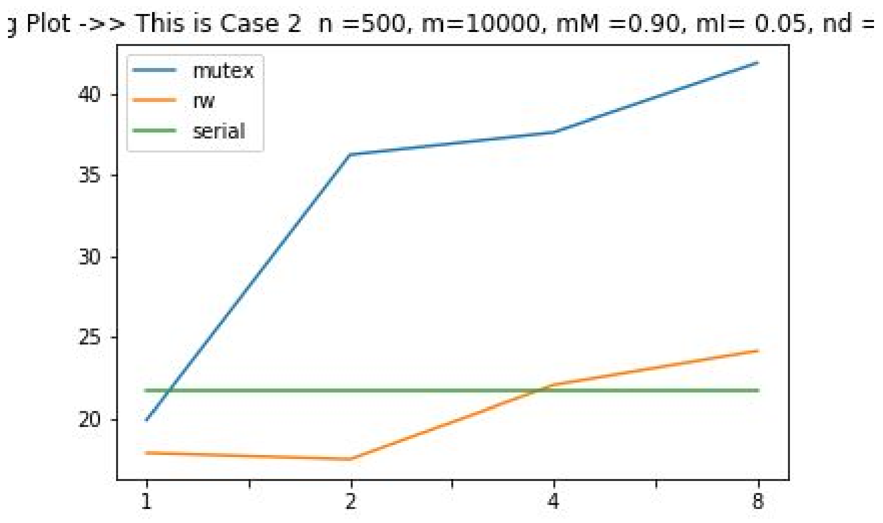
Hardware Specification

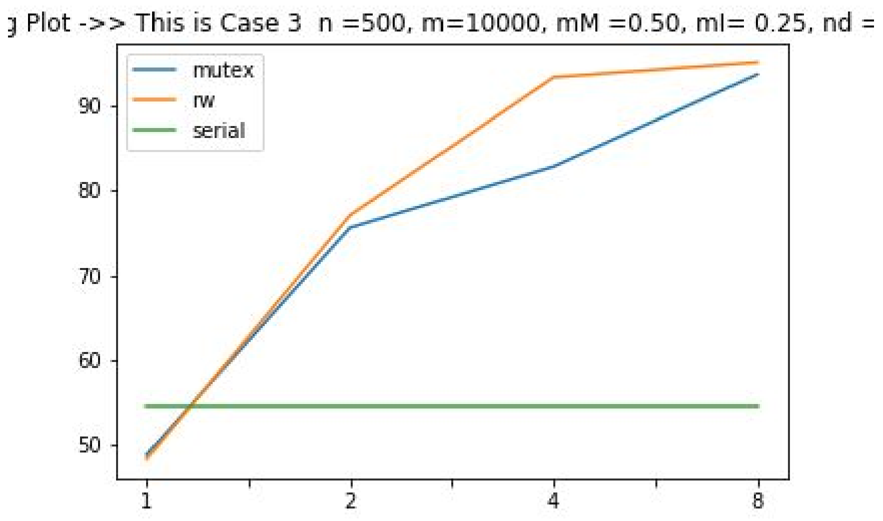
Operating System	Ubuntu 20.04
Architecture	x86_64
Model name	Intel(R) Core(TM) i5-7200U CPU @2.50GHz

CPU(s)	4
Thread(s) per core	2
Core(s) per socket	2
Vendor ID	GenuineIntel

Plotting Average time with number of threads







Observations and Conclusions

- If we look at the time taken for the execution for parallel processing is efficient in more operations with the read process not the write process
- Always, mutex has more time than normal serial programming due to switching overheads. Mutex locks for each operation so it is absolutely like serial processing for linked lists with overheads.
- If we look at the read write lock, that works well when the read processing such as searching happens. In reading, Linked list can be accessed by multiple threads.
- If you have more read requests and less write requests then the parallel programming with r/w lock is better. If you have a significant write process than parallel programming waste in that case. Above graph is shown detaily that.

160596K(Sivaram R.)
160605P(Sriyoukan S.)