**CS4532 Concurrent Programming**

**Lab 1**

**120408D – MHM.Mursith**

**120615J - USI.Dissanayaka**

## **Step 1 : Design**

1. **Declaring the linked list**

A linked list will be created for each sample randomly. Number of nodes will

Decided by the user. Different random seed is used for each execution.

1. **Number of operations**

m, mMember, mInsert and mDelete values are taken from the user and number of operations will be calculated using that.

Example: m is the number of total operations.

* Number of member operations = m \* mMember
* Number of insert operations = m \* mInsert
* Number of delete operations = m \* mDelete

After calculating the number of operations, operatins will be done to the linked list. For each operation two parameters should be passed.

* Pointer to the root node.
* Value (Each time a random value will be passed)

1. **Number of threads**

User should provide the number of threads and based on that values equal

Amount of operations are divided into each thread.

Example: Assume number of thread count is 4. Then each thread has to do m/4 operations. That includes

1. (m \* mMember)/ 4 member operations.
2. (m \* mInsert)/ 4 insert operations.
3. (m \* mDelete)/ 4 delete operations.
4. **Execution time**

Execution time is calculated for each sample and finally the average and the standard deviation are calculated.

## **Step 2 : Implementation**

Implementation has been done based on the code snippets provided during the lecture.

## **Step 3 : Results**

**Specification of the machine :**

* CPU - Clock speed 2.60GHz, no of cores - 8 , cache size 6MB, CPU model - Intel Core i7-6700HQ 6th Gen processor
* Memory – 8GB DDR4 Ram at 2133MHz
* OS - Ubuntu 16.04 LTS no other user services were running.

**Applications:**

* Compiler: gcc
* Note that all the values are in seconds.

**Case 1 :** Description : n = 1000, m = 10000, member = 99%, insert = 0.5%, delete = 0.5%

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Implementation** | No of Threads | | | | | |
| 1 | | 2 | | 4 | |
| Average | Std | Average | Std | Average | Std |
| Serial Program | 0.031318 | 0.002438 |  |  |  |  |
| One Mutex for Entire List | 0.030150 | 0.001188 | 0.032652 | 0.017026 | 0.055885 | 0.002062 |
| Read-Write Lock | 0.040394 | 0.003197 | 0.016883 | 0.010805 | 0.034309 | 0.003795 |

**Case 2 :**Description : n = 1000, m = 10000, member = 90%, insert = 5%, delete = 5%

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Implementation** | No of Threads | | | | | |
| 1 | | 2 | | 4 | |
| Average | Std | Average | Std | Average | Std |
| Serial Program | 0.056773 | 0.015295 |  |  |  |  |
| One Mutex for Entire List | 0.055621 | 0.001576 | 0.072840 | 0.005531 | 0.076598 | 0.006777 |
| Read-Write Lock | 0.065335 | 0.002646 | 0.065136 | 0.002115 | 0.069082 | 0.001894 |

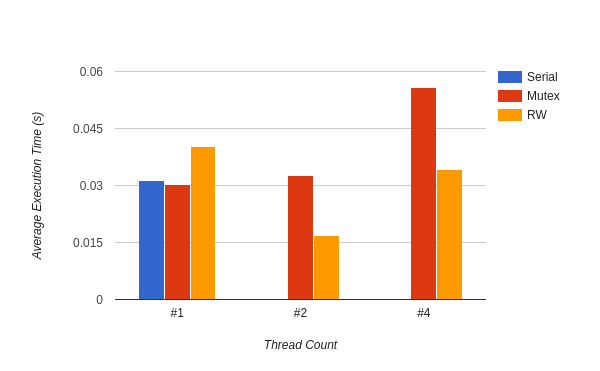
**Case 3** :Description : n = 1000, m = 10000, member = 50%, insert = 25%, delete = 25%

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Implementation** | No of Threads | | | | | |
| 1 | | 2 | | 4 | |
| Average | Std | Average | Std | Average | Std |
| Serial Program | 0.100659 | 0.018104 |  |  |  |  |
| One Mutex for Entire List | 0.100102 | 0.005263 | 0.124520 | 0.006889 | 0.131990 | 0.026560 |
| Read-Write Lock | 0.106237 | 0.003830 | 0.129624 | 0.005320 | 0.149206 | 0.013180 |

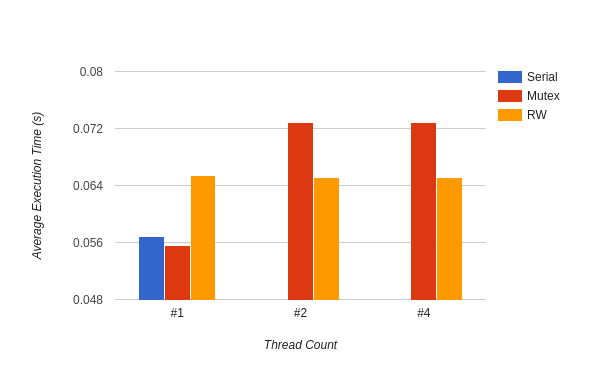
## **Step 4 : Comparison**

In graphs drawn below, *Average Execution Time*  is in **Seconds**

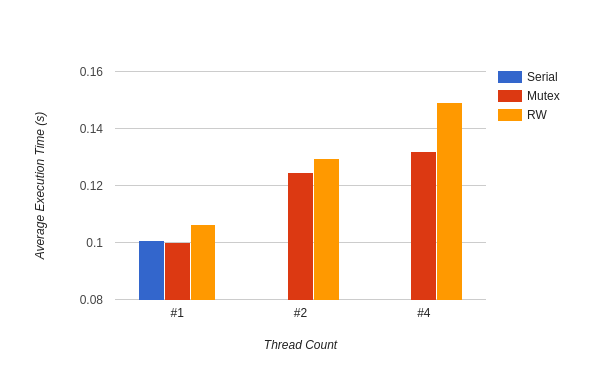
**Case 1 :** Description : n = 1000, m = 10000, member = 99%, insert = 0.5%, delete = 0.5%



**Case 2 :**Description : n = 1000, m = 10000, member = 90%, insert = 5%, delete = 5%



**Case 3** :Description : n = 1000, m = 10000, member = 50%, insert = 25%, delete = 25%



## **Step 5 : Evaluation**

* **Case 1 :**
* As the member operations dominates the execution, the time taken for the process is lower than Case 2 and Case3.
* When thread count is more than 1, RW lock program got less time as there are lesser no of write operation compared to read operations.
* Mutex get more time as it lock the entire linked list each time it locks as there are multiple threads and it switches among threads and then an overhead is gone.
* There is an interesting behaviour when there is only a single thread to the process. In single thread, Mutex gets the least time even though it has an overhead of locking and unlocking. That may be because as there is only a single thread, Mutex locks the entire list at once and do the complete computation once as the set of operations are smaller compared to the speed of CPU whereas serial program does not have an implementation of a lock thus the process could be scheduled with the other processes.
* **Case 2 :**
* Now the insert and delete operations increased. This operations takes more time than search (member) operation. Hence the execution time is obviously increased for all the cases.
* Serial program is faster, as RW lock locked the list more time than previous case. Thus the overhead time has been added to the execution time.
* Even though the RW lock time is higher than the Mutex lock, Mutex Implementation takes more time as it lock the whole linked list each time regardless of the behaviour of the operation.
* Single threaded implementation behave same as the Case 1 but RW lock program takes more time compared to other programs because of the increment of Write operations

**Case 3 :**

Percentage of operations are now same. 50% for write and 50% read. Thus RW lock operations should take more time.

When thread count is 2 or 4 the RW locks switch among threads as there are 1 to 1 Read and Write operations and RW lock takes more time to lock or unlock. Mutex take more time but only it locks the entire list each time there is an operation regardless of the behaviour of operations. Hence the time is incremented in RW lock because of the overhead time taken for locking and unlocking mechanisms.

The same behaviour is happened with Single Thread which were hppened in Case 1 and 2.