# CS4532 Concurrent Programming Lab 2

120095L P.M.T de Silva

Design ExplanationIn the assignment the fractions of member, insert and delete functions that have to be performed is given (m\_member, m\_insert, m\_delete). Total number of operations need to perform is also given. So we can find the individual number of sub operations to be performed by simple multiplication.  
**Example:**  
Number of insert operations = m\_insert \* mNow we have the number of individual operations, we should design a method to allocate those operations to given threads.

**Solution**   
We allocate number of member, insert and delete operations per thread by dividing number of each operations by number of threads (n). So each threads have to perform m\_member / n member operations, m\_insert / n insertions and m\_delete / n delete operations.

When a thread call operation function, each time one of member, insert and delete operations will run. This will continue until number of operations reach allocate number of operations.   
For example:

Consider a thread that have 2000 member operations, 50 insert and delete operations to perform. When the operation function is called member, insert and delete functions run one time each and 2000 becomes 1999 and 50 becomes 49. This loop runs until all operations to perform becomes zero.

Thread Functionality

All operations need to perform is distributed within threads evenly. When defining a thread, we define number of operations that thread need to do. Threads keep those values locally. Each time the thread does the operations, thread decrement the value of number of each operations need to do. At calling operation function we run a while loop until all number of member, insert and delete operations need to do becomes zero. At each loop one member, insert and delete function will run. Before running a certain operation, we check the number of that operation need to perform. If this value is zero, we don’t run that operation.

**Case 1**

*n* = 1,000 and *m* = 10,000, *mMember* = 0.99, *mIndert* = 0.005, *mDelete* = 0.005

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Implemetation | No of Threads | | | | | |
| 1 | | 2 | | 3 | |
| Average | Std | Average | Std | Average | Std |
| Serial |  |  |  |  |  |  |
| One mutex for entire list |  |  |  |  |  |  |
| Read Write Lock |  |  |  |  |  |  |

**Case 2***n* = 1,000 and *m* = 10,000, *mMember* = 0.9, *mIndert* = 0.05, *mDelete* = 0.05

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Implemetation | No of Threads | | | | | |
| 1 | | 2 | | 3 | |
| Average | Std | Average | Std | Average | Std |
| Serial |  |  |  |  |  |  |
| One mutex for entire list |  |  |  |  |  |  |
| Read Write Lock |  |  |  |  |  |  |

**Case 3***n* = 1,000 and *m* = 10,000, *mMember* = 0.5, *mIndert* = 0.25, *mDelete* = 0.25

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Implemetation | No of Threads | | | | | |
| 1 | | 2 | | 3 | |
| Average | Std | Average | Std | Average | Std |
| Serial |  |  |  |  |  |  |
| One mutex for entire list |  |  |  |  |  |  |
| Read Write Lock |  |  |  |  |  |  |