**Programming Assignment 2**

**Name:** Sai Naga Viswa Chaitanya Basava

**NetID:** SXB220302

**Email:** [sxb220302@utdallas.edu](mailto:sxb220302@utdallas.edu)

Experiments were executed on a 2019 MacBook pro with **12** 2.6 GHz 6-Core Intel Core i7 processors. All experiments were run for number of threads ranging from 2 to 12 and executed for an overall operation count of 1 million. The BST and Linked List experiments were warmed up to an initial size of half the key space size before running the N threads and starting the experiments.

1. **Stack**

This concurrent version of stack was implemented using a “compare and swap” instruction. For both the push and pop operations, the top pointer of the stack is tried to be updated using CAS operation. We keep on retrying until the CAS operation on the top pointer succeeds. It ensures mutual exclusion and deadlock freedom, but not starvation freedom as a thread can get stuck indefinitely trying to update top under very high contention. Also, there is no doorway, hence no FIFO guarantees amongst the operations. Experiment results are as follows

|  |  |
| --- | --- |
| A graph with a line  Description automatically generated | A graph with a line  Description automatically generated |

1. **Unbalanced Binary Search Tree**

BST was implements using a fine grained Reentrance locking at node level. At an instance, each thread would have locked out 2 nodes. For each operation, the thread would try to acquire the lock of the child in the direction of search traversal, move another level down (in the direction of search) and them unlock the parent node, so at an instance 2 nodes would have been locked out by a thread (except during remove, where 4 nodes would be locked due to the necessity to replace an internal node with in order successor). This approach allows for better concurrency compared to locking the entire tree, as operations on different traversal paths can be conducted in parallel, only the operations on same path would be seeing the effects of locking. The experiments conducted also highlight a high throughput in all the settings, which can be seen in the graphs below.

|  |  |
| --- | --- |
|  |  |

1. **Linked List**

The sorted linked list was also implemented with similar idea of locking 2 nodes at an instance. For each operation, the thread would try to acquire the locks of the predecessor and the current nodes, perform the sorted order condition check and either move forward or perform the operation, and finally release the current node followed by the predecessor. We follow the strategy of locking the next node and then letting go of the predecessor while moving forward in the list, this ensures mutual exclusion at node level, but at the overall list level we will be able to perform concurrent operations at different levels in the list. One observation from the graphs below is that this implementation isn’t scaling well as the key space size increases, but still allows for better concurrency when compared to locking out the entire linked list.

|  |  |
| --- | --- |
| A graph of different colored lines  Description automatically generated | A graph with numbers and lines  Description automatically generated |