Name: Muhammed Orhun Gale **ID**: 26754

CS 403-534 Programming Assignment #1 Report

Multiset

Implementation

As the concurrent multiset, I implemented the “**concurrent fine-grained sorted linked-list**” implementation from the Chapter 9 of the book The Art of Multiprocessor Programming by Herlihy and Shavit with some modifications to ensure certain aspects of the requirements. Main reason why I did these modifications is the requirement which insists that remove operations from the concurrent linked-list had to be blocking. In order to ensure this feature of remove operation, I used condition variables as locks and since I am removing only from the head of the linked list, I used the head’s condition variable to **wait** if the linked list is empty (besides sentinel nodes, head and tail). However, since the condition variable releases the lock, a race condition could happen between a sleeping node waking up and another node which passes the emptiness check. In order to avoid this race condition, I used a **guard lock (mutex)** to make the pointer change part of the remove operation atomic.

Linearizability

Since the remove operation of my concurrent multiset is blocking, a remove operation cannot be executed (may start but cannot finish) before executing an add operation. Therefore, in any execution history for an add/remove operation couple, the linearization point of the add operation would appear before the remove operation even if the executions are overlapping i.e. started concurrently or execution of remove operation started beforehand. This feature can ensure the validity of executions if operations that target the same object are executed concurrently or without the interference of an operation that targets another object even if the executions are not concurrent.

If one executes operations with a strict sequential order such as add(a)-->add(b)-->remove():a where key of the b is bigger than a, there cannot be a valid execution since the set is descending sorted and remove will always remove the b rather than a. On the other hand, if the linearization points of the same type (add or remove) operations are selected correctly for any set of concurrent executions, one can find a valid and therefore linearizable execution.

Leader Selection Algorithm

To synchronize the leader selection algorithm among threads, I placed two barriers. I placed the first barrier to the end of the maximum vote finding part of the nodeWorks function in order to synchronize the round transitions. With this implementation, nodes wait for each other after deciding a leader or going for a new round. This ensures that messages of different rounds do not interleave. The second barrier that I placed is between sending messages and reading from inbox function calls on the nodeWorks function. I placed it in order to synchronize the reading and writing phases among threads.