Multi-Core Architecture and Systems (0368-4183) Lecturer: Adam Morrison

Home Assignment 5

Due date: **June 17th, 23:59** (submit via moodle)

Submission instructions

- You may discuss with others, but submit your **own** answers and code.
- Submit one PDF (minimum 11 pt font) and one `BST.java` file with your code (**not** together in a ZIP).
- A solution submitted t seconds late will have its grade multiplied by $1 - \left(\frac{t}{60 * 60 * 24 * 7}\right)^4$.

1 Concurrent binary search tree (BST)

Design a lock-based concurrent BST data structure, using the ideas from the lazy list algorithm, and implement it in Java.

1.1 BST design

For this part, design an algorithm that meets the following requirements:

- Your BST should be a linearizable implementation of a set with the standard `insert`, `remove` and `contains` operations: An `insert`(x) inserts x into the set and returns true if x was not already in the set; otherwise, it returns false without changing the set. A `remove`(x) removes x from the set and returns true if x was in the set; otherwise, it returns false without changing the set. A `contains`(x) returns true if x is in the set; otherwise, it returns false.
- The tree can be unbalanced. There is no need to keep its height logarithmic.
- The tree must store items in all nodes, both internal (non-leaf) and leafs.
- The size of the tree (number of nodes) should be $I - R + O(1)$, where I is the number of items inserted into the tree and R is the number of items removed.
- A deletion of an item must physically remove its node from the tree, whether it is an internal node or a leaf.
- Your algorithm's scalability properties should be similar to the lazy list's. A `contains` operation must be read-only and not acquire locks. Similarly, an `insert`/`remove` should navigate to the node in the tree that it wants to modify without acquiring locks. In particular, do not use a global lock, i.e., a lock that blocks all operations regardless of which key they are searching for.
- Your algorithm should be deadlock-free.
- The `contains` method should be lock-free in the following sense: in any suffix of an execution where a `contains` takes infinitely many steps, some other operation (`contains`/`insert`/`remove`) must complete.

Hint Not having to maintain a balanced tree makes this algorithm a generalization of the lazy list (more or less) so its techniques should be handy. The main challenge is to implement `remove()`. There's no single correct solution; any approach satisfying linearizability will do.

What to submit The PDF you submit should include a write-up explaining your algorithm and why it is linearizable. The write-up can include some pseudo code to help make the explanations clear, but it's not mandatory. We will check the submitted Java implementation.

1.2 BST implementation

Implement your algorithm in Java. Your implementation should be an SC implementation, so make sure to consider the Java memory model. For locking, you can use Java's `synchronized` blocks, the `ReentrantLock` from `java.util.concurrent`, or implement a lock yourself.

You are provided with a skeleton implementation and test harness (`bst.zip`). You need to implement your algorithm in the `src/algorithms/BST.java` file of this package. The harness performs light-weight *validation* of the BST, which can help catch some bugs. See the `README.rst` file in the package for full details and instructions. In particular, **make sure** that the **getName()** method returns a string containing your id number. It is *highly recommended* to test your code with the test harness on a machine with many cores, using a workload with a lot of contention (i.e., a small tree with high frequency of updates).

The scripts in the package are customized to the TAU CS lab machines. You can work on the machine `rack-mad-01` (8 processors, each with 8 cores).

What to submit Submit **only** the `BST.java` file. Your BST implementation should be confined to this file. (You can make changes to other files for your debugging and testing, but we will test the submission with the original version of the package.)