CS 131 Homework 3 Report

Summary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 8 | 16 | 32 |
| Null | Y | 1884.94 | 4781.57 | 7945.35 |
| Synchronized | Y | 2477.51 | 5024.01 | 11367.7 |
| Unsynchronized | N | 1496.80 | 4723.83 | 6640.89 |
| GetNSet | N | 1118.78 | 3155.60 | 7056.13 |
| BetterSafe | Y | 1317.34 | 2726.84 | 4784.66 |
| BetterSorry | N | 1678.57 | 3561.45 | 7381.75 |

The purpose of this assignment was to show the effects of concurrency when implemented in several ways. Several different versions were tried out to observe how synchronized and unsynchronized threads, as well as a few intermediate levels, behaved. The results were then recorded through testing on seasnet server 7.

Experiment

I ran 1,000,000 swaps on each implementation, running 8, 16, and 32 threads. The test data was an array of 100 byte values ranging from 0 to 127. Here is an overview of what each class is:

**Null**: Swapping does nothing, and is used to measure the total overhead of running the program.

**Synchronized**: Uses synchronized java keyword so threads run in turn, not parallel.

**Unsynchronized**: Threads all run concurrently, without safeguards.

**GetNSet**: Unsynchronized, but has added safety that retrieval and modification are an atomic unit.

**BetterSafe**: Uses a reentrant lock so that the shared resource can only be modified by one thread at a time.

**Better Sorry**: Still uses a byte array, but temporarily converts to atomic integer for swapping.

One thing to note here is the concept of Data Race Free, or DRF. Some may seem faster than others, but this is at the cost of it not always giving the correct results. This is a result of multiple threads modifying the same value, which will deliver inconsistent data every data. Next to each column, I note whether or not it is DRF.

|  |  |  |
| --- | --- | --- |
| State | DRF | Transition time per thread count (ns) |

*Figure 1: Chart comparing the different transition times at different thread counts.*

Of