CS 131 - Homework 3 Report

1. Testing Platform

Java Version

openjdk version "1.8.0_181"
OpenJDK Runtime Environment (build 1.8.0_181-b13)
OpenJDK 64-Bit Server VM (build 25.181-b13, mixed mode)

CPU and Memory

SEASNet lnxsrv09: 8-core Intel(R) Xeon(R) CPU E5-2640 v2 @ 2.00 GHz
64GiB RAM

2. Thread Number Test Results

Constants

10,000,000 operations on a 500 byte array, each with an initial value of 50. The maximum value for any element is 127.

Variables

Concurrency Strategy, and number of threads.

Measurements

- Average real time for one thread to perform a successful swap, including overhead.
- Average number of errors. This value is not exact, since repeated errors can cancel out. Nevertheless, it is a reasonable measure of concurrency issues.

Synchronized

	Time per transition (ns)	Average number of errors
8 Threads	1939.31	0
16 Threads	3538.97	0
32 Threads	9074.63	0

Unsynchronized

	Time per transition (ns)	Average number of errors
8 Threads	471.535	334
16 Threads	943.842	1710
32 Threads	1456.48	4796

GetNSet

	Time per transition (ns)	Average number of errors
8 Threads	890.072	2056
16 Threads	1749.96	4555
32 Threads	3216.85	4922

BetterSafe

	Time per transition (ns)	Average number of errors
8 Threads	673.661	0
16 Threads	1426.70	0
32 Threads	2554.47	0

3. Operation Number Test Results

Constants

16 threads working on a 500 byte array, each with an initial value of 50. The maximum value for any element is 127.

Variables

Concurrency Strategy, and number of operations.

CS 131 - Homework 3 Report

Synchronized

	Time per transition (ns)	Average number of errors
1,000 Swaps	70983.9	0
100,000 Swaps	9868.59	0
10,000,000 Swaps	3919.14	0

Unsynchronized

	Time per transition (ns)	Average number of errors
1,000 Swaps	68620.2	2
100,000 Swaps	3590.16	41
10,000,000 Swaps	984.896	1520

GetNSet

	Time per transition (ns)	Average number of errors
1,000 Swaps	66918.8	2
100,000 Swaps	4930.99	191
10,000,000 Swaps	2281.08	2835

BetterSafe

	Time per transition (ns)	Average number of errors
1,000 Swaps	88313.7	0
100,000 Swaps	8968.57	0
10,000,000 Swaps	1560.43	0

4. Array Length Test Results

Constants

16 threads doing 10,000,000 swaps on a variable length array, each element with an initial value of 50. The maximum value for any element is 127.

Variables

Concurrency Strategy, and length of array.

Synchronized

	Time per transition (ns)	Average number of errors
50 bytes	3255.84	0
500 bytes	4061.57	0
5000 bytes	5096.66	0

Unsynchronized

	Time per transition (ns)	Average number of errors
50 bytes	1239.66	89
500 bytes	1047.29	1610
5000 bytes	996.556	531

GetNSet

	Time per transition (ns)	Average number of errors
50 bytes	1897.58	2208
500 bytes	2193.14	2917

CS 131 - Homework 3 Report

5000 bytes	771.332	132

BetterSafe

	Time per transition (ns)	Average number of errors
50 bytes	1299.03	0
500 bytes	1626.42	0
5000 bytes	2584.05	0

5. Comparisons

Unsurprisingly, the Synchronized strategy takes the longest with all thread amounts, all array lengths, and most operation amounts (it performs better than BetterSafe when performing only 1000 operations).. It makes up for this by being one of the only Data Race Free (DRF) options. It does this through the java 'synchronized' keyword, which locks the swap() method, allowing only one thread to enter the method at a time. Any threads that try to enter the method while another thread is in it will be blocked until no threads are using the function.

Unsynchronized is the fastest strategy (excluding Null, which doesn't actually do any work), but is not Data Race Free. It does this by ignoring all concurrency threats and just implementing the swap() function naively.

GetNSet is faster than Synchronized, because it doesn't lock the function, but this also means that it isn't Data Race Free. At the same time, GetNSet is slower than Unsynchronized, and seems to have more errors. This is because GetNSet has pretty much the same implementation as Unsynchronized, but uses atomic set functions, which likely include some overhead over default assignment operations. With very large arrays, GetNSet's performance improves dramatically, placing it ahead of all other strategies. It's high number of errors, however, makes it ill-suited to the needs of GDI.

BetterSafe is faster than Synchronized when performing a large number of operations, and is also Data Race Free. It achieves this by locking a smaller critical section than the Synchronized strategy does. A smaller locked section means that there is less of a chance of threads colliding and blocking, and also that when they do, the lock is held for a shorter time. Since Synchronized uses the java 'synchronized' keyword and locks the entire function, it extends the critical section to include some of the overhead of setting up the swap() function and returning its result to the caller. BetterSafe, on the other hand, only locks the bare minimum critical section. Even though this is a minor difference, it obviously adds up over 10,000,000 operations.

Like all the other strategies, BetterSafe performs worse as the size of the array increases, but it still performs better than Synchronized regardless of array size

6. Conclusions

The only Data Race Free strategies are Synchronized and BetterSafe (and Null, but that doesn't count). When performing massive amounts of swaps (as a big company probably will), BetterSafe performs almost 3 times better than Synchronized, which makes it an obvious candidate for GDI's implementation.

Though the specification does allow a small amount of errors, both GetNSet and Unsynchronized have too many errors to be feasible, and provide only relatively small performance benefits over BetterSafe. As such, I would recommend BetterSafe for GDI's implementation.