CS 131 Homework 3 Report

Abstract

The aim of the project was to test different types of concurrent mechanisms when swapping numbers while incrementing and decrementing. After making the mechanisms, I tested them on the SEASnet server 6.

Test Results

To test the mechanisms, I ran 10,000,000 swaps with 8, 16 and 34 threads using five initial values in the state array with a maxval of 6. The following are the results of this test with the different methods used in the project.

Model Name	Time taken for transitions (in ns)			DRF
	8 Threads	16 Threads	32 Threads	
Null	158.191	639.949	1584.19	Yes
Synchronized	2827.54	4203.07	8876.28	Yes
Unsynchronized	1354.33	-	-	No
GetNSet	1020.42	2006.68	4013.95	No
BetterSafe	789.375	1402.72	2916.42	Yes

Comparison and DRF

Synchronized appears to do the worst out of the other models, taking the longest to perform the swaps. However, it is Data Race Free (DRF) as the synchronized keyword in Java allows for blocking in the swap method, preventing other threads from preempting it. Unsynchronized would be the fastest of all the methods as it doesn't have any overhead from blocking competing threads however it is prone to race conditions and as a result would lead to infinite loops in the test conditions as Unsynchronized allows the first thread to perform the swap go first and makes it give incorrect results.

GetNSet performs better than synchronized because its implementation uses the AtomicIntegerArray, as instead of blocking the entire swap method like in Synchronized, it makes access to elements of the array atomic. However, the lack of blocking in this method still doesn't make it Data Race Free.

In the end BetterSafe is the best of all four of these models. Not only is it the fastest method due to its use of a reentrant lock to block only the critical sections of the swap function, it is also 100% reliable as BetterSafe is DRF. It is as a result faster than Synchronized and still 100% reliable. I was easily able to realize that the usage of reentrant locks would both expedite the method while keeping it reliable thanks to prior knowledge from CS111.

Problems That Had To Be Overcome

While testing, I had to swap servers twice to find a server that allowed me to use GetNSet with over 8 threads. I was finally able to get the method to work on server 6. However, during testing I was only able to produce results for unsynchronized only once where is was faster than synchronized but returned with a sum mismatch. I was never able to replicate this result and subsequent runs of the class would lead to the infinite loop due to the class not being DRF. Increasing the maxval didn't change this result and I was only able to get the time taken for transitions for the function by adding in an arbitrary println statement. However, the time results that would be returned would be much longer than what they would normally be due to the unnecessary overhead created by printing a string with every usage of the Swap function. As a result I was able to conclude that GetNSet and Unsyncronized were unreliable methods due to their unreliability on certain servers and in certain conditions due to them not being DRF as neither method employs any blocking for threads.