Name: Soliman Alnaizy

NID: so365993

Smstr: Spring 2019

Course: COP 4520

========================

Programming Assignment #2

========================

Problem 1

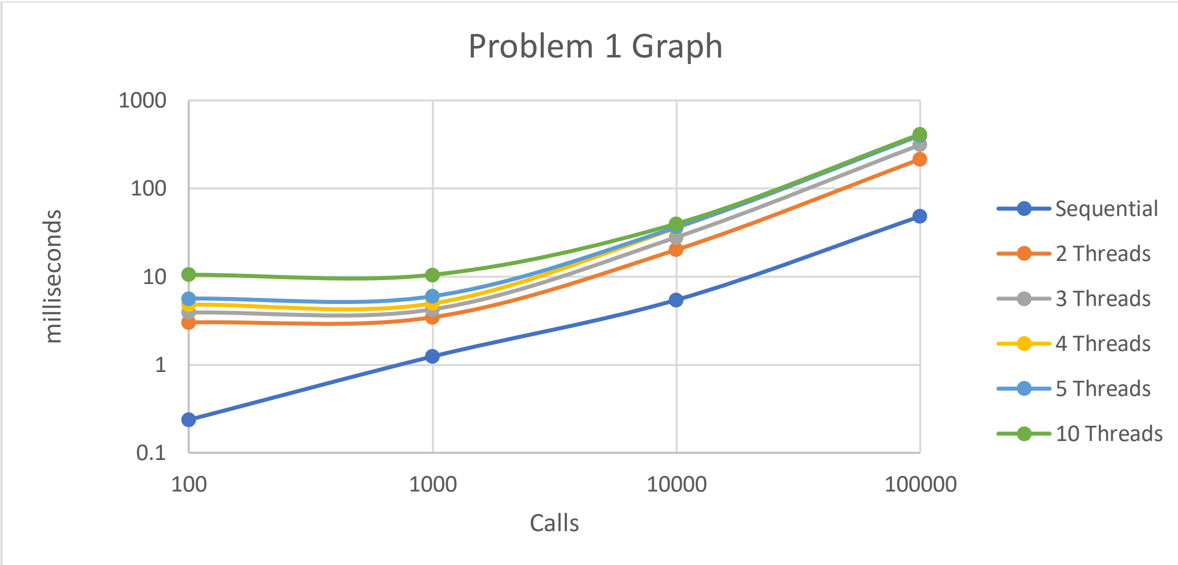
HOW TO RUN: Compile using "javac SuperAwesomeAtomicStack.java" on a Linux based terminal. I have provided a Test case in a file called "Test.java". The test case randomly_calls the push() or pop() methods 10,000 times. Compile and run the Test class_to test the SuperAweseomeAtomicStack class.

OBJECTIVE: To design a lock free stack that supports a "size" member.

EFFECIENCY: When *pop()* is called, a head-deletion is performed on a Linked List, which is an O(1) operation. When *push()* is called, a head-insertion is performed, which is also an O(1) operation. Each node in the stack takes up a constant amount of space, so a stack with n nodes would take up O(n) space. I don't think you could get more efficient that this when it comes to a stack.

CORRECTNESS: We guarantee the correctness of the stack by using a Descriptor object that encapsulates the *head* node and the *size* of the stack. Using the *compareAndSwap()* method in a while-loop, we can guarantee that the *head* is only swapped if it's the head we expect. Since we use a single *compareAndSwap()* to update both the *head* and the *size*, we also guarantee the linearizability of the *push()* and *pop()* methods.

PROGRESS: Below is a graph that shows the average time it takes for a program of a *x* threads to make *n* method calls. The method calls were 50% push() methods and 50% pop() methods. The time displayed on the graph is the average time of 1,000 trials. When plotted on a log-log graph, you can see that the time it takes is linear with the number of threads. Which is a good sign and shows that the threads are not deadlocking.



Problem 2

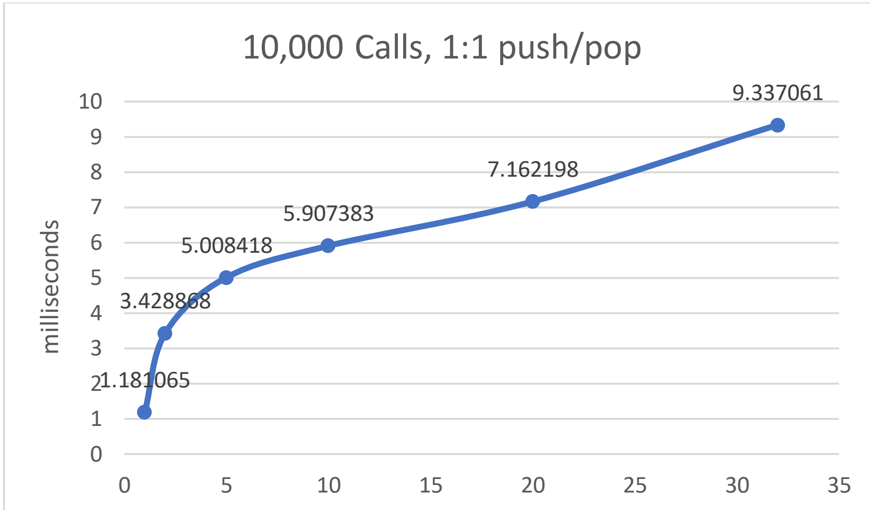
HOW TO RUN: Compile using "javac SuperAwesomeAtomicStack.java" on a Linux based terminal. I have provided a Test case in a file called "Test.java". The test case randomly calls the *push()* or *pop()* methods 10,000 times. Compile and run the Test class to test the SuperAweseomeAtomicStack class.

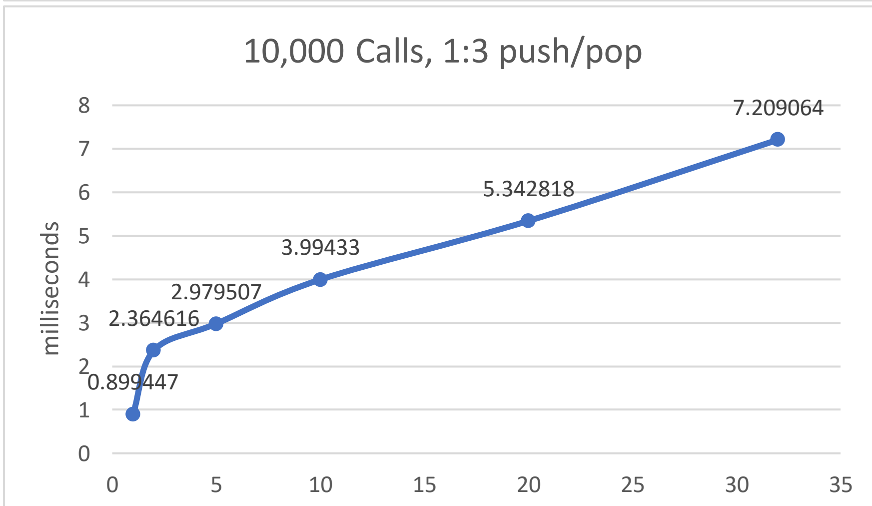
OBJECTIVE: To design a lock free stack that supports a *size* member AND a capacity limit of 10,000 nodes.

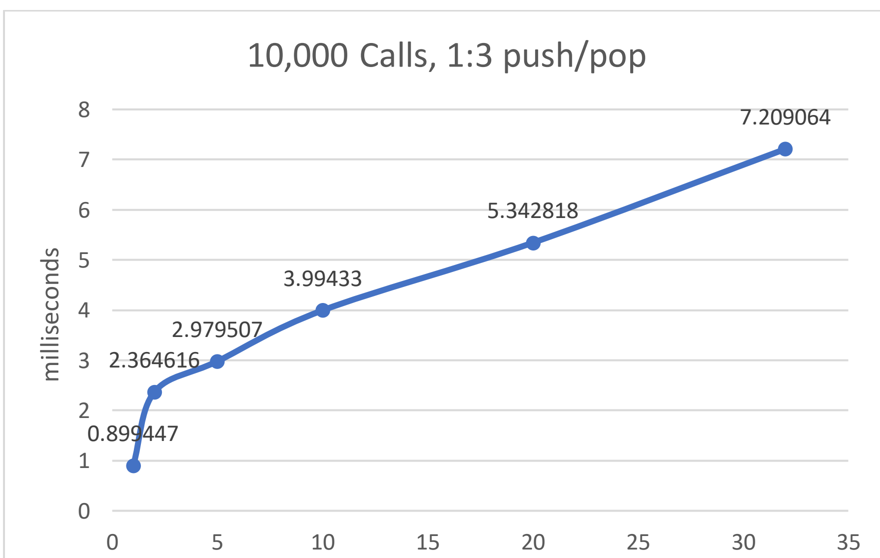
EFFECIENCY: When *pop()* is called, a head-deletion is performed on a Linked List, which is an O(1) operation. When *push()* is called, a head-insertion is performed, which is also an O(1) operation. Each node in the stack takes up a constant amount of space, so a stack with n nodes would take up O(n) space. There are also no locks used, which prevents the threads from bottlenecking at the lock.

CORRECTNESS: We guarantee the correctness of the stack by using a Descriptor object that encapsulates the *head* node and the *size* of the stack. This time, before using the *compareAndSwap()* method, we first check to see if the size of the temporarily stored head is less than the capacity. If it is, then we call CAS in a while-loop, we can guarantee that the *head* is only swapped if it's the head we expect. Since we use a single CAS to update both the *head* and the *size*, we also guarantee the linearizability of the *push()* and *pop()* methods.

PROGRESS: Below are three graphs that show the average time it takes for a program of a *x* threads to make *n* method calls. Each graph represents a certain ratio to *push()* to *pop()* methods (see title of graph to know the ratio). The time displayed on the graph is the average time of 1,000 trials. When plotted on a log-log graph, you can see that the time it takes is linear with the number of threads once the number of threads exceeds 5. Which is a good sign and shows that the threads are not deadlocking. The *push()* method also calls the *size()* method. Therefore by testing the *push()* method, we are also implicitly testing the correctness of the *size()* method.


10,000 Calls, 1:1 push/pop
milliseconds
10
9
8
7
6
5
4
3
2
1
0
0
5
10
15
20
25
30
35
1.181065
3.428868
5.008418
5.907383
7.162198
9.337061


10,000 Calls, 1:3 push/pop
milliseconds
8
7
6
5
4
3
2
1
0
0
5
10
15
20
25
30
35
0.899447
2.364616
2.979507
3.99433
5.342818
7.209064


10,000 Calls, 1:3 push/pop
milliseconds
8
7
6
5
4
3
2
1
0
0
5
10
15
20
25
30
35
0.899447
2.364616
2.979507
3.99433
5.342818
7.209064