CS 131 Homework 3. Java shared memory performance races

Ken Deng
UCLA

Abstract

This project is trying to investigate the influence of different synchronization and multi-thread techniques on performances in Java programs. A main task is to implement synchronization without "synchronized" keyword, trading reliability (race condition) for better performance.

1 Introduction

We are evaluating the performances of three kinds of state management strategies (NullState, SynchronizedState, and UnsynchronizedState) with different thread settings. The settings for our evaluation includes: number of threads, state array entries, and swap transitions.

Specifically, we are doing the following tasks:
1. Creating an array of longs consisting of zeros;
2. Randomly select two array entries, and subtract one from the first entry and minus one from the second entry.
3. Measures the CPU time for the overall operations.

The shell command we use for testing is similar to:

```
time timeout 3600 java UnsafeMemory Platform
5 Synchronized 8 100000000
```

Platform says to use platform threads; 5 says to use a state array of 5 entries; Synchronized means to test the SynchronizedState implementation; 8 means to divide the work into 8 threads of roughly equal size; 100000000 means to do 100 million swap transitions total. Also note that if the sum is not zero, then there are some reliability issues, e.g. race condition.

2 NullState

The swap function in NullState is inactive and performs no operations. This state demonstrates the quickest synchronization possible, as it lacks any synchronization mechanisms but still consistently returns a sum of zero. This occurs because the swap function is non-functional and the state array starts off initialized to zero.
2.1 Number of threads: as it increases, the total/average real time and user time both slightly increases.
2.2 Size of Array: no significant changes.
2.3 Virtual/Platform: no significant changes.

3 SynchronizedState

The swap function in SynchronizedState uses the "synchronized" keyword to ensure that the swap operation is accessed by only one thread at a time. This means that while one thread is performing a swap, no other thread can execute the swap function simultaneously.
3.1 Number of threads: as it increases along 1,8, 40, the total real time and user time both slightly increases. The sys time remains roughly the same. Yet the average real time is significantly increased.
Explanation: The decline in performance as the number of threads increases can be attributed in part to possible lock contention. This occurs when several threads try to run the same section of code simultaneously, but only one thread can advance at a time. The other threads are forced to wait until the lock is released by the thread currently in operation.
Reliability: The uses of the "synchronized" keyword provides tools like thread locks, which prevents the race condition between different threads. Thus the reliability is enhanced.
3.2 Size of Array: as it increases from 5 to 100, there is a slight decreasing tendency in the tim-

ings.
Explanation: the reason might be that as the array size is increased, there is less probability that two threads would compete for the same array element.
3.3 Virtual/Platform: virtual seems to have longer timings than platform, especially when the number of threads is larger than 1.
Explanation: the reason might be that the virtual thread is a lightweight abstraction of a task that can be bound to a platform thread and is scheduled by the Java virtual thread scheduler, which requires mounting and therefore has more workload.

4 UnsynchronizedState

The swap function in UnsynchronizedState does not employ the "synchronized" keyword to ensure that the swap operation is performed by only one thread at a time. Consequently, while one thread is performing a swap, another thread may also initiate a swap operation. This scenario illustrates the significance of synchronization and the consequences of not managing race conditions.
4.1 Number of threads: generally the timings for it are less than SynchronizedState. As we increase along 1,8, 40, the user time and total real time increase slightly, but average real time increases significantly.
Explanation: In the Unsynchronized State, the lack of synchronization overhead results in quicker overall execution times. However, the overhead associated with managing threads remains. Even when compared to itself, an increase in the number of threads will still lead to a rise in user time, real time, and average swap time.
Reliability: The lack of the "synchronized" keyword leads to race condition, which yields in the sum not being zero.
4.2 Size of Array: as it increases from 5 to 100, all the timings are increasing.
Explanation: as we are not using "synchronized" keyword, there is no more benefits of lowering the probability of two threads accessing the same array entry. Moreover, as the size of array increases, it has a more workload due to caching problem.
4.3 Virtual/Platform: all timings seem to be the same.
Explanation: as there is no synchronization, no lock would happen, and thus there are no extra delay for waiting between threads.

5 Conclusion

The UnsynchronizedState is more effective in single-threaded environments due to its higher efficiency. However, as the number of threads grows, the SynchronizedState becomes more beneficial due to its reliability, yet if there are too many threads, delays can occur from managing locks.

Platform threads are particularly effective with the SynchronizedState, while the impact on UnsynchronizedState or NullState are not significant. On the other hand, the virtual threads tend to increase system time.

Regarding the size of the state array, it represents a balance between cache efficiency and the level of lock contention. Therefore, the performance of the SynchronizedState remains stable regardless of array size, whereas the performance of the UnsynchronizedState decreases significantly as the array size grows.

Benchmark for SynchronizedState: array size = 5 or 100; number of threads = 8; Platform

Benchmark for UnsynchronizedState: array size = 5; number of threads = 1; Platform or Virtual

6 Difficulties

The testing results are not stable, especially when the number of threads is huge. This may because when running on the lnxsrv11 server, there can be many other students running other programs, resulting in workload and performance fluctuations. Also, after using "javac *.java" command to get the .class files, these files are not compiled into static machine codes, but it will be further recompiled in runtime, which causes unpredictability.

My solution involves running my program in late nights, where there are less request from other students to the server.

7 Appendix

7.1 NullState

Platform, size of state array = 5:
Threads = 1: Total real time 1.15471 s Average real swap time 11.5471 ns real 0m1.291s user 0m1.280s sys 0m0.040s
Threads = 8: Total real time 0.416577 s Average real swap time 33.3261 ns real 0m0.551s user 0m1.669s sys 0m0.044s
Threads = 40: Total real time 0.431523 s Average real swap time 172.609 ns real 0m0.569s user 0m1.710s sys 0m0.056s

Platform, size of state array = 100:
Threads = 1: Total real time 1.43610 s Average real swap time 14.3610 ns real 0m1.583s user 0m1.568s sys 0m0.042s
Threads = 8: Total real time 0.501662 s Average real swap time 40.1329 ns real 0m0.641s user 0m2.004s sys 0m0.046s
Threads = 40: Total real time 0.468432 s Average real swap time 187.373 ns real 0m0.600s user 0m1.776s sys 0m0.063s

Virtual, size of state array = 5:
Threads = 1: Total real time 1.18059 s Average real swap time 11.8059 ns real 0m1.377s user 0m1.311s sys 0m0.040s
Threads = 8: Total real time 0.420923 s Average real swap time 33.6738 ns real 0m0.558s user 0m1.721s sys 0m0.050s
Threads = 40: Total real time 0.434194 sAverage real swap time 173.678 ns real 0m0.565suser 0m1.742ssys 0m0.053s

Virtual, size of state array = 100:
Threads = 1: Total real time 1.55018 s Average real swap time 15.5018 ns real 0m1.688s user 0m1.686s sys 0m0.040s
Threads = 8: Total real time 0.597680 s Average real swap time 47.8144 ns real 0m0.742s user 0m2.099s sys 0m0.051s
Threads = 40: Total real time 0.516701 s Average real swap time 206.680 ns real 0m0.654s user 0m1.985s sys 0m0.047s

7.2 SynchronizedState

Platform, size of state array = 5:
Threads = 1: Total real time 3.36626 s Average real swap time 33.6626 ns real 0m3.505s user 0m3.474s sys 0m0.049s
Threads = 8: Total real time 6.91912 s Average real swap time 553.529 ns real 0m7.048s user 0m9.854s sys 0m0.238s
Threads = 40: Total real time 7.40415 s Average real swap time 2961.66 ns real 0m7.566s user 0m10.274s sys 0m0.112s

Platform, size of state array = 100:
Threads = 1: Total real time 3.38290 s Average real swap time 33.8290 ns real 0m3.511s user 0m3.474s sys 0m0.039s
Threads = 8: Total real time 6.07390 s Average real swap time 485.912 ns real 0m6.207s user 0m7.824s sys 0m0.178s
Threads = 40: Total real time 6.87550 s Average real swap time 2750.20 ns real 0m7.006s user 0m10.504s sys 0m0.265s

Virtual, size of state array = 5:
Threads = 1: Total real time 3.35452 s Average real swap time 33.5452 ns real 0m3.487s user 0m3.479s sys 0m0.047s
Threads = 8: Total real time 13.1645 s Average real swap time 1053.16 ns real 0m13.296s user 0m31.716s sys 0m4.602s
Threads = 40: Total real time 9.03720 s Average real swap time 3614.88 ns real 0m9.171s user 0m18.973s sys 0m3.293s

Virtual, size of state array = 100:
Threads = 1: Total real time 3.35417 s Average real swap time 33.5417 ns real 0m3.496s user 0m3.475s sys 0m0.044s
Threads = 8: Total real time 13.8203 s Average real swap time 1105.63 ns real 0m13.981s user 0m33.371s sys 0m4.461s
Threads = 40: Total real time 10.0014 s Average real swap time 4000.54 ns real 0m10.135s user 0m20.628s sys 0m3.367s

7.3 UnsynchronizedState

Platform, size of state array = 5:
Threads = 1: Total real time 1.29734 s Average real swap time 12.9734 ns real 0m1.433s user 0m1.414s sys 0m0.049s
Threads = 8: Total real time 1.99209 s Average real swap time 159.367 ns output sum mismatch (56110 != 0) real 0m2.119s user 0m7.824s sys

0m0.052s
Threads = 40: Total real time 2.53136 s Average real swap time 1012.55 ns output sum mismatch (838 != 0) real 0m2.658s user 0m9.899s sys 0m0.057s

Platform, size of state array = 100:
Threads = 1: Total real time 1.45083 s Average real swap time 14.5083 ns real 0m1.577s user 0m1.569s sys 0m0.055s
Threads = 8: Total real time 3.66226 s Average real swap time 292.981 ns output sum mismatch (-18021 != 0) real 0m3.793s user 0m14.305s sys 0m0.044s
Threads = 40: Total real time 4.10941 s Average real swap time 1643.76 ns output sum mismatch (-22037 != 0) real 0m4.245s user 0m15.225s sys 0m0.061s

Virtual, size of state array = 5:
Threads = 1: Total real time 1.29288 s Average real swap time 12.9288 ns real 0m1.446s user 0m1.445s sys 0m0.043s
Threads = 8: Total real time 2.58042 s Average real swap time 206.434 ns output sum mismatch (-16366 != 0) real 0m2.713s user 0m9.954s sys 0m0.057s
Threads = 40: Total real time 2.52382 s Average real swap time 1009.53 ns output sum mismatch (-23992 != 0) real 0m2.687s user 0m9.805s sys 0m0.058s

Virtual, size of state array = 100:
Threads = 1: Total real time 1.67815 s Average real swap time 16.7815 ns real 0m1.813s user 0m1.806s sys 0m0.044s
Threads = 8: Total real time 3.87638 s Average real swap time 310.111 ns output sum mismatch (-19867 != 0) real 0m4.007s user 0m15.282s sys 0m0.057s
Threads = 40: Total real time 3.76149 s Average real swap time 1504.60 ns output sum mismatch (-21600 != 0) real 0m3.902s user 0m14.841s sys 0m0.052s