1. Abstract

The primitive goal of Homework 3 is to analyze the performance, reliability and safety of methods of synchronization on two different machines. In this homework, we explore three different ways to provide DRF (Data Race Free) - No synchronization, “synchronized” keyword, and atomic long array provided by java.util.concurrent.AtomicLongArray. Each thread is responsible for carrying the divided task of swapping values in the long array. Based on the sum of all values in the array, we can determine if the method of synchronization is correct or not.

Platforms’ details:

Server: lnxsrv10.seas.ucla.edu:

/proc/cpuinfo
- Model: 85
- Model name: Intel(R) Xeon(R) Silver 4116 CPU @ 2.10GHz
- CPU MHz : 2095.079
- 4 processors; each processor has 4 cores
- Cache size on each processor: 16896 KB

Java - version
- Java Version “13.0.2” 2020-01-14
- Java(™) SE Runtime Environment (build 13.0.2+8)
- Java HotSpot(™) 64 bits Server VM (Build 13.0.2+8)

/proc/meminfo
- MemTotal: 65799628 kB

Server: lnxsrv09.seas.ucla.edu

/proc/cpuinfo
- Model: 62
- Model name: Intel(R) Xeon(R) CPU E5-2640 v2 @ 2.00GHz
- Cpu Mhz: 2299.926
- 32 processors; each processor has 8 cores
- Cache size on each processor: 20480 KB

Java - Version
- Java version “13.0.2” 2020-01-14
- Java(™) SE Runtime Environment (build 13.0.2+8)
- Java Hotspot(™) 64-bit Server VM (Built 13.0.2+8)

/proc/meminfo
- MemTotal: 65755720 KB

2. Synchronization of AcmeSafeState

AcmeSafeState is a class implementation of State class that contains array long as a private data and three public methods - size(), current(), and swap() to perform swapping on the array. Rather than using a synchronized keyword to synchronize the long array shared by more than one thread, it uses atomic long array object provided by java library to synchronize swapping data, thus guarantee Data Race Free. The JDK9 versions of Java.util.concurrent.atomic classes associates Atomic objects applied to array or single element and includes various methods to atomically operate on the array or the element. Those methods are constructor to create the atomic object, get(i) to return the element of the array specified by index i, set(i,a) to set the element of the array specified by index i and value a, getAndIncrement(i) to increment the value of array specified by index i by 1, getAndDecrement(i) to decrement the value of array specified by index i by 1. These methods are performed atomically, so threads can not interleave during the operations on the array long. My AcmeSafeState uses getAndIncrement() to atomically increment the array by 1 and getAndDecrement() to atomically decrement the array by 1. To return the long array, current() allocate a local variable temp long array and copies each element of the atomic array by using get() which provides atomicity. So, the current() function guarantees a Data Race Free. As I ran the command, *time*

timeout 3600 java UnsafeMemory AcmeSafe 8 100000000 5, the value sum value is 0 which means the array is data race free.

3. Measurement Results

I ran the swaps test with the number of threads: 1, 8, 40, 80 and state array: 5, 100, 200 on the 4 state classes on lnxsrv09. In lnxsrv10, I ran the swaps test with the number of threads: 1, 8, 40, 50 and state array: 5, 100, 200 on the 4 state classes
Swap number is **10,000,000**.
Estimated time for each measurement is **ns/transaction**

Null Class					
Lnxsrv09					
size	Number of Threads				DRF
	1	8	40	80	
5	14.011	90.833	1507.1	2409.3	yes
100	13.658	189.69	1144.4	5411.6	yes
200	13.764	94.961	1736.4	3211.6	yes

Null Class					
Lnxsrv10					
size	Number of Threads				DR F
	1	8	40	50	
5	12.471	88.611	1593.6	1764.9	yes
100	12.421	108.45	1227.0	1520.8	yes
200	12.728	90.140	843.30	1836.4	yes

Synchronized					
Lnxsrv09					
Array size	Number of Threads				DR F
	1	8	40	80	
5	21.041	2484.2	10264	21115	yes
100	22.311	1767.7	5763.3	24453	yes
200	20.9315	1835.8	13065	25291	yes

Synchronized					
Lnxsrv10					
Array size	Number of Threads				DR F
	1	8	40	50	
5	17.579	524.53	2186.7	2913.1	yes
100	17.568	408.51	2369.6	3057.7	yes
200	18.512	409.57	2314.9	2960.4	yes

Unsynchronized					
Lnxsrv09					
size	Number of Threads				DR F
	1	8	40	80	
5	16.617	355.14	1458.4	4009.6	No
100	16.255	451.17	2194.9	4374.9	No
200	16.422	354.43	1376.5	4907.3	No

Unsynchronized					
Lnxsrv10					
size	Number of Threads				DRF
	1	8	40	50	
5	13.1981	286.49	2232.1	2031.0	No
100	13.1521	325.60	3154.8	2200.5	No
200	13.1173	338.77	6081.8	2185.3	No

AcmeSafeState Class					
Lnxsrv09					
size	Number of Threads				DRF
	1	8	40	80	
5	21.344	718.67	4811.2	7175.4	yes
100	29.036	799.53	2488.0	4512.7	yes
200	29.565	607.65	1906.3	4741.3	yes

AcmeSafeState Class					
Lnxsrv10					
size	Number of Threads				DRF
	1	8	40	50	
5	27.634	1261.7	12963	5555.6	yes
100	26.643	582.67	4291.7	3990.9	yes
200	26.510	597.50	2918.4	7799.5	yes

4. Measurement Analyze

Based on the performance measurements, it obviously indicates AcmeSafeState class which utilizes atomic long array perform swapping faster than Synchronized class while also retaining reliability by ensuring DRF. For both Synchronized and AcmeSafeState, as the number of thread increases, the average time spent for each swapping also increases because while a thread is locking on the array to perform operation, many others threads are waiting to obtain the lock to also perform operations. So, it increases wait time overall as more threads wait. Generally, it is also indicative that with a larger size array comes longer time per swap. Another important point to note is the server that the tests performed. Both AcmeSafeState class and Synchronized state class ran faster on Lnxsrv09 than on Lnxsrv10 because as previously pointed server 09 has more processors and more cores than server 10, thus allowing more threads to run in parallel.

5. Problems Encountered

I was able to run 80 threads on the lnxsrv09 without any problems, but i ran into a problem when i tried to run 80 threads on lnxsrv10 which says “ Failed to start thread - pthread_create failed (EAGAIN) for attributes: stacksize: 1024k, guardsize:4k, detached.” This problem occurred because lnxsrv 10 server has 4 processors, each with 4 cores which limits the number of threads able to run in parallel. However, lnxsrv 09 server has 32 processors and each processor has 8 cores.