CS4204 P1 Locking and Lock-free

180007044
University of St Andrews
St Andrews, UK

INTRODUCTION

In this practical, a concurrent data structure that simulates a bank account is implemented in both locking and lock-free ways in JAVA. It enables some basic logical functions including balance inquiry, depositing and withdrawing. Withdraw would fail and the method would return false if the balance is not sufficient.

To compare and analyze locking and lock-free algorithms, there are two kinds of locks applied in this practical: a mutual exclusion and a spin lock, as spin lock has some common points with both mutex and lock-free algorithms. In following parts, these three implementations would be introduced, analyzed. Based on analysis, a conclusion will be drawn in the final part after compare three algorithms.

DESIGN & IMPLEMENTATION

Mutex

To implement lock mechanism, JAVA provides a keyword: *synchronized* to block threads applying resources (named as *monitor* in JAVA) which has been held by other threads. Each object in JAVA would automatically have a monitor so that it would be accessed by only one thread at a time. [1]

Based on the mechanism discussed above, it is not complex to implement mutex. In this practical, an object variable called *mutex* is created. When a thread is calling a method for an account object, it needs to apply the monitor of mutex with the method *synchronized* before entering the formal logic.

Spin Lock

Different from mutex, threads would not be suspended or blocked when they fail to apply a spin lock. What they do is just to keep running but do nothing until the lock is available.

In this practical, a simple spin lock is implemented. An atomic Boolean variable plays the role of lock in this algorithm. At the beginning of deposit or withdraw methods, the thread would acquire the lock by TAS circularly until the thread successfully update the value of lock to true. After the methods finish their work, they would release the lock by setting its value to false and another thread could reset it and obtain the lock.

Lock-free

The implementation of the deposit method of lock-free version is quite similar to that of spin lock. A local variable of the method is initialized as the value of balance. Then the algorithm applies CAS to circularly compare its value and the balance. If they are equal, the deposit would be added to the balance, otherwise the local variable would be updated to the current value of balance and the loop would continue. In this case, it guarantees that the balance would be updated by only one thread at a time.

The logic of the withdraw method is basically as same as the deposit method, but it is a bit more complex because it is possible to fail because of the insufficient balance. Therefore, every time the value of the local variable is updated, the comparison would be made between the updated value and the amount of withdrawing. If the balance is not sufficient, the method would return false immediately. Although it may make users have to call the method again, it could prevent incorrect change of the balance.

PERFORMANCE EVALUATION

To evaluate algorithms, in this practical, 1000 threads are created and each one would execute 1 deposit and 2 withdraw operations respectively. The correctness, efficiency and CPU utility of algorithms would be evaluated.

To evaluate the efficiency of algorithms, a variable is created as an attribute of the account object as a timepoint, whose value is the time of creating the object. The difference between this time and the time that the last thread exits would be considered as run time. Besides, since the algorithms are tested in macOS system, the activity monitor would be employed to monitor the CPU utility.

Correctness

Algorithms all show high correctness to the results.

Run time

According to the test result, lock-free version shows the best efficiency. And the spin lock algorithm uses the longest time to finish the same work. Especially, sometimes spin lock would spend extremely long time and occupy almost all of CPU if there are too many threads waiting and looping.

CPU Utility

According to the feedback provided by the activity monitor, the mutex algorithm is more "CPU-consuming" than the lock-free algorithm. The spin lock generally uses as much CPU as the lock-free algorithm, however, as it is mentioned before, too many looping threads may waste a lot of computation resources.

ANALYSIS

There are both advantages and disadvantages of each algorithm.

Mutex

Although mutexes could essentially solve the problem of collision in shared memories, its mechanism may make the system spend extra computation resources to block or awake threads. This consumption would be larger if the threads are frequently blocked or awaked. Besides, the unexpected delay or interruption caused by the thread who is holding the lock would also influence other waiting threads. [2]

Spin Lock

Spin lock effectively avoids the extra consumption for blocking and awaking threads, but just like what happens in evaluation, the large number of looping threads may waste a lot of CPU and time. It also probably makes the influence caused by unexpected delay and interruption discussed above more serious and more obvious. This discussion indicates that spin locks do solve part of problems existing in mutex, but not all.

Besides, dead lock is also a vital problems of lock algorithms. It makes programmers have to take it into consideration when they design algorithms

Lock-free

The lock-free algorithm in this practical is designed more like a refined spin lock.

According to the definition of lock-free algorithms [2], it guarantees that there would be at least one thread processing the common resource, which means breakdown happening in a certain thread would not influence others. Furthermore, since it could change the value of the target variable (*balance* in this case), it would spend less time than locking algorithms. Although loops in lock-free algorithms may cause the same problems as spin locks, the situation would be better in this case because loops would not keep waiting and atomicity of the algorithm is higher.

CONCLUSION

Based on performance shown in this practical and analysis results listed above, it may be safe to draw some conclusions.

Mutex algorithms are probably more suitable for large critical sections, which would not switch the threads and make extra cost. Oppositely, spin locks could work better in programs with small critical sections so that each thread would not keep looping for long time and waste resources. So, spin locks are also not suitable for programs who may produce a large number of threads.

The problem of the free-lock algorithm is basically as same as the spin lock algorithm, but its effect is much smaller. However, other problems of locking algorithms are almost fixed. So, it could have higher priority than locking algorithms, even though it may be difficult to fit some programs because its high atomicity.

PREFERENCE

[1] Barua S. Synchronized in Java. Geeks for Geeks. Available from: https://www.geeksforgeeks.org/synchronized-in-java/

[2] Ramalete P. Lock-Free and Wait-Free, Definition and Examples. Concurrency Freaks. Available from: http://concurrencyfreaks.blogspot.com/2013/05/lock-free-and-wait-free-definition-and.html