CS5300 - Parallel & Concurrent Programming
Autumn 2024
Programming Assignment 3
Comparison of Filter Lock and Bakery Locks
Submission Date: 21st September 2024, 9:00 pm

Goal: The goal of this assignment is to implement the two locking algorithms discussed in the class: Filter lock and Bakery lock. Then compare the performance of these locks by measuring two parameters: average waiting time for threads to obtain the locks and throughput. You have to implement these algorithms in C++.

Details. As explained above, you have to implement two locking algorithms: Filter lock and Bakery lock. You have to implement them in C++. Each locking algorithm implements a class consisting of two methods: lock and unlock. Your program will read the input from the file and write the output to the file as shown in the example below.

To test the performance of locking algorithms, develop an application, lock-test is as follows. Once, the program starts, it creates n threads. Each of these threads, will enter critical section (CS) k times. The pseudocode of the test function is as follows:

Listing 1: main thread

```
void main()
{
    StartTime = getSysTime();
    cout << "The start time is " << StartTime;
    ...
    ...
    // Declare a lock object which is accessed from all the threads
    Lock Test = new Lock();
    ...
    ...
    create n testCS threads;
    Wait for all n testCS threads to join;
    ...
    ...
    EndTime = getSysTime();
    cout< "The end time is " << EndTime;
}
```

Listing 2: testCS thread

```
void testCS()
{
    id = thread.getID();
    for (i=0; i < k; i++)
    {
```

```
23          reqEnterTime = getSysTime();
24          cout << i << "th CS Entry Request at " << reqEnterTime << " by thread "
25          << id;
26          Test.lock();
27          actEnterTime = getSysTime();
28          cout << i << "th CS Entery at " << actEnterTime << " by thread " << id;
29          sleep(t1);
30          reqExitTime = getSysTime();
31          cout << i << "th CS Exit Request at " << reqExitTime << " by thread "
32          << id;
33          Test.unlock();
34          actExitTime = getSysTime();
35          cout << i << "th CS Exit at " << actExitTime << " by thread " << id;
36          sleep(t2);
37      }
38  }
```

Description of the Test Program: If the lock and unlock functions work correctly then the display messages 2 and 3 of every thread will work correctly without any interleaving. Seeing these messages one can be sure of the correctness of the lock and unlock. Note that the message 1 and 4 can interleave and hence may be commented out to check the correctness.

Here $t1$ and $t2$ are delay values that are exponentially distributed with an average of $\lambda 1, \lambda 2$ milli-seconds. The objective of having these time delays is to simulate that these threads are performing some complicated time consuming tasks. The $Test$ variable declared in line 8 declared in main is an instance of Lock class and is accessible by all threads.

To determine the overall time required for executing $k * n$ tasks, you can compute it in the main function by subtracting the "StartTime" from the "EndTime." You can calculate throughput by dividing $k * n$ by the time difference, represented as $(k * n)/(EndTime - StartTime)$.

In this context, every thread calls the "testCS" function. When a thread wishes to enter the Critical Section (CS), it stores the time as reqEnterTime (or request Enter Time). And once the thread enters the CS, it records the time as actEnterTime (actual CS Entry Time). The difference between these times actEnterTime - reqEnterTime is the time taken by the thread to enter the CS which we denote as *csEnterTime*.

Similarly, when a thread wishes to exit the CS, it stores the time as reqExitTime (or request Exit Time). And once the thread exits the CS, it records the time as actExitTime (actual CS Entry Time). The difference between these times actExitTime - reqExitTime is the time taken by the thread to exit the CS which we denote as *csExitTime*.

Input: The input to the program will be a file, named inp-params.txt, consisting of all the parameters described above:$n, k, \lambda 1, \lambda 2$. A sample input file is: 100 100 5 20.

Output: Your program should output to a file in the format given in the pseudocode for each algorithm. A sample output is for Filter lock is as follows:

```
Filter lock Output:
1st CS Requested Entry at 10:00 by thread 1
1st CS Entered at 10:05 by thread 1
1st CS Requested Exit at 10:06 by thread 1
1st CS Exited at 10:06 by thread 1
1st CS Requested Entry at 10:01 by thread 2
.
.
.
```

Similar to Filter lock, you have to show the output for Bakery lock as well. The output must demonstrate the mutually exclusive execution of the threads.

Report: You have to submit a report for this assignment. This report should contain a comparison of the performance of both locking algorithms. You must run both these algorithms multiple times to compare the performances and display the result in form of a graph. You must average each point in the graph plot by 5 times.

To compare the performance of different locking algorithms in terms of throughput and average entry time, you will have to conduct the below experiments.

1. **Throughput Analysis with varying threads:** In this experiment, you will focus on scalability by increasing the threads. The y-axis will represent throughput, as previously described (tasks executed per unit time), and the x-axis will vary the parameter n (Number of threads) ranging from 2 to 64 in powers of 2. Please fix k (number of times a thread enters CS) as 15.

2. **Throughput Analysis with varying k:** In this experiment, the primary focus is on throughput analysis. The y-axis of the graph signifies throughput, measured in tasks executed per unit of time, as described earlier. Meanwhile, the x-axis is designated for varying the parameter k within the range of 5 to 25. It is essential to maintain a constant of 16 threads throughout this experiment.

3. **Average Entry Time and Worst-Case Entry Time Analysis with varying threads:** In this experiment, you will focus on the average and worst-case entry time by increasing the threads. The y-axis will measure the average time it takes for a thread to enter the critical section from the time it requested the entry. The x-axis will vary the number of threads, ranging from 2 to 64 in powers of 2. Please have k fixed at 10, which represents the number of critical section requests made by each thread.

4. **Average Entry Time and Worst-Case Analysis with varying k :** Similar to Step 3, in this experiment, you will focus on the average and worst-case entry time by varying k. The y-axis will measure the average time it takes for a thread to enter the critical section from the time it requested the entry. The x-axis will vary k, ranging from 5 to 25. Please have n fixed at 16, representing the number of threads.

Experiments 1, 2 will have two curves each corresponding Filter and Bakery Locks while Experiments 3 and 4 will have four curves corresponding to average and worst case for each of Filter and Bakery Locks.

Please maintain $\lambda 1, \lambda 2$ at values of 1 and 2, respectively, in all the experiments. Finally, you must also give an analysis of the results while explaining any anomalies observed in the report.

Deliverables: You have to submit the following:

- The source files containing the actual program to execute. Please name them as: Filter-<rollno>.cpp, Bakery-<rollno>.cpp.
- A readme.txt that explains how to execute the program
- The report as explained above

Zip the two files and name them as ProgAssn3-<rollno>.zip. Then upload it on the google classroom page of this course. Submit it by the above mentioned deadline.

Evaluation: The break-up of evaluation of your program is as follows:

1. Program design and the report - 50%
2. Program execution - 40%
3. Code documentation and indentation - 10%.

Please make sure that you your report is well written since it accounts for 50% of the marks.
Late Submission and Plagiarism Check: All assignments for this course has the late submission policy of a penalty of 10% each day after the deadline for 6 days Submission after 6 days will not be considered.

Kindly remember that all submissions are subjected to plagiarism checks.