CS5300 - Parallel & Concurrent Programming
Fall 2023
Implementing Multi Reader Multi Writer Register
Submission Date: 2nd October 2023 9:00 pm

Goal: The goal of this assignment is to implement M-VALUED MULTI-READER MULTI-WRITER (MRMW) atomic registers using MULTI-READER SINGLE-WRITER (MRSW) atomic register. Implement these algorithms in one of the following languages of your choice: **(a) C++ (b) Rust (c) Go**.

Details. The algorithm to implement the m-valued MRMW atomic register is already given in the textbook in Figure 4.14. Assume that the system by default provides m-valued MRSW registers (which is not necessarily true). You can also assume that the system accommodates a total of N threads which is known to you.

As mentioned above, you can implement these algorithms in one of the following languages of your choice: **(a) C++ (b) Rust (c) Go**. Please ensure that the language in which you are implementing your program has the support for sequential consistency enabled.

After implementation, you will have to test the performance of the algorithm with the underlying atomic MRMW implementation provided by atomics of the programming language.

Experimental Setup: You have to test the performance of your implementation with the inbuilt atomic variable implementation provided by the programming language. In case of C++, one can see the details here: https://en.cppreference.com/w/cpp/atomic. To test the performance of your implementation, develop an application, *atomic-test* as follows. Once, the program starts, it creates n threads. Each of these threads, will read/write from/to the atomic register k times. The pseudocode of the test function is as follows:

Listing 1: main thread

```
void main()
{
    ...
    create shVar = 0;    // Atomic shared variable initialized to 0
    ...
    create N testAtomic threads;
    ...
}
```

Listing 2: testAtomic thread

```
void testAtomic()
{
    int lVar;    // local variable
    int id = thread.getID();
    for (i=0; i < k; i++)
    {
```

```
                action = randomly decide to either read with probability p or
                write with probability (1-p);
                reqTime = getSysTime();
                cout << i << ‘th action requested at ’ << reqTime << ‘ by thread ’
                << id;

                if (action == read)
                {
                    // replace the following with the syntax for atomics accordingly
                    lVar = shVar.read();

                    cout << ‘Value read: ’ << lVar;
                }
                else    // Write action
                {
                    lVar = k * id;  // the value written by each thread is unique

                    // replace the following with the syntax for atomics accordingly
                    shVar.write(lVar);

                    cout << ‘Value written: ’ << lVar;
                }
                complTime = getSysTime();
                cout << i << ‘th action ‘ << action ‘completed at ’ << actEnterTime
                << ‘ by thread ’ << id;
                sleep(t1);    // Simulate performing some other operations.
            }
        }
```

Here $t1$ is a delay value exponentially distributed with an average of λ milli-seconds. The objective of having these time delays is to simulate that these threads are performing some complicated time consuming tasks.

Input: The input to the program will be a file, named inp-params.txt, consisting of the above parameters: $capacity, numOps, \lambda$. They are explained here:

- $Capacity$ is the number of threads;
- $numOps$ as the name suggests the number of operations to be performed by each thread;
- λ is the average with which a thread sleeps exponentially between different invocations on the queues.

Logfile: The program should output a log file as mentioned above. Logfile should have the entry for each write and read along with its thread_id and time. The LogFile should demonstrate the correctness of your implementation. This will be used by the TAs to check the correctness of the implementation.

Report: You have to submit a report for this assignment. The report should first explain the design of your program while explaining any complications that arose in the course of programming.

The report should also include few analysis with the plots as mentioned below:

1. **Impact of average time with increasing** $Capacity$**:** In this plot, the Y-axis will represent the average time taken for read and write operations, while the X-axis will depict the number of threads ($capacity$). Here half the threads will be performing the reads and rest half will be performing the write operation. The $capacity$ will vary from 2 to 16 in increments of 2. To maintain consistency in the experiment, all other parameters will remain constant as follows: $numOps = 5000$, $\lambda = 5$.

2. **Impact of average time with increasing $numOps$:** In this plot, the Y-axis will represent the average time taken for read and write operations, while the X-axis will depict the number of $numOps$. Here, again have the read and write to have equal probability for invoking an operation. The $numOps$ will vary from 1000 to 5000 in increments of 1000. To maintain consistency in the experiment, all other parameters will remain constant as follows: $capacity = 16$, $\lambda = 5$.

As explained above, each plot will have two curves. One curve is for the algorithm developed by you and the other curve is for inbuilt algorithm for the inbuilt atomic variable implementation provided by the programming language.

To rule out any outliers, please obtain the value of each point after averaging it over 5 times. Please give an analysis of the results in the report while explaining any anomalies observed.

Deliverables: You have to submit the following:

- The source file containing the actual program to execute. Please name it as MRMR_$\langle roll_no \rangle$.$\langle$xtn$\rangle$. Please follow this convention. Otherwise, your program will not be evaluated.
- A readme.txt that explains how to execute the program.
- The report as explained above.

Zip all three files and name them as ProgAssn3-$\langle$rollno$\rangle$.zip. Then upload it on the Google Classroom page of this course. Submit it by above mentioned deadline. Please follow the naming convention. Otherwise, your assignment will not be evaluated by the TAs.

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

1. Program Design as explained in the report - 30%
2. The Graphs obtained and the corresponding analysis shown in the report - 30%
3. Program Execution - 30%
4. Code Documentation & Indentation - 10%.

Please make sure that you your report is well written since it accounts for 60% 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.