Operating Systems–II: CS3523
Spring 2022
Programming Assignment 5:
Implement solutions to Readers-Writers problem using Semaphores

Submission Date: 11th March 2022, 9:00 pm

Goal: The goal of this assignment is to implement two solutions using semaphores for the Reader-Writers Problem. One solution with **reader preference** (Readers-Writers) and one **fair solution** (Fair-Readers-Writers). You have to implement these two algorithms and compare the average and worst-case time taken for each thread to access the critical section (shared resources).

Details. As mentioned above, you have to implement the Readers-Writers problem with reader preference (Readers-Writers) and one fair solution (Fair-Readers-Writers). Implement a multithreaded program for the above algorithms. 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 the synchronization algorithms, develop an application as shown below. Once the program starts, it creates nw writer threads and nr reader threads, which execute their respective writer and reader functions. Each of these threads will access the shared object (or Critical Section), kw or kr times, depending on whether they are a writer or a reader respectively. The pseudocode of the application is as follows:

Listing 1: main thread

```
1  void main()
2  {
3      ...
4      ...
5      create n_w writer threads;
6      create n_r reader threads;
7      ...
8      ...
9  }
```

Listing 2: Writer Thread

```
1  void writer()
2  {
3      int id = thread.getID();
4
5      for(int i=0; i ≤ kw ;i++)
6      {
7          reqTime = getSysTime();
8          cout << i << "th CS request by Writer Thread " << id <<
9          " at " << reqTime << endl;
10
11         /*
```

```
        Write your code for Readers_Writers() and
        Fair Readers_Writers() Using Semaphores here.
        */

        enterTime = getSysTime();
        cout << i << "th CS Entry by Writer Thread " << id <<
        " at "<< enterTime<< endl;

        sleep(randCSTime); // simulate a thread writing in CS

        /*
         Your code for the thread to exit the CS.
        */

        exitTime = getSysTime();
        cout << i << "th CS Exit by Writer Thread " << id <<
        " at "<< exitTime << endl;
        sleep(randRemTime); // simulate a thread executing in Remainder Section
    }
}
```

Listing 3: Reader Thread

```
void reader()
{
    int id = thread.getID();
    Random csSeed, remSeed;

    for(int i=0; i < kr ;i++)
    {
        reqTime = getSysTime();
        cout << i << " th CS request by Reader Thread "<< id <<
        " at " << reqTime << endl;

        /*
        Write your code for Readers_Writers() and
        Fair Readers_Writers() using Semaphores here.
        */

        enterTime = getSysTime();
        cout << i << " th CS Entry by Reader Thread "<< id << " at
        " << enterTime << endl;

        sleep(randCSTime); // simulate a thread reading from CS

        /*
        Your code for the thread to exit the CS.
        */

        exitTime = getSysTime();
        cout << i << " th CS Exit by Reader Thread "<< id <<" at
        "<< exitTime << endl;

        sleep(randRemTime); // simulate a thread executing in Remainder Section
    }
```

34 }

Here $randCSTime$ and $randCSTime$ are delay values that are exponentially distributed with an average of μ_{CS}, μ_{Rem} milli-seconds. The objective of having these time delays is to simulate that these threads are performing some complicated time consuming tasks.

A sample output would be as follows:
1st CS Request by Writer Thread 1 at 01:00
1st CS Entry by Writer Thread 1 at 01:01
1st CS Request by Reader Thread 2 at 01:02
1st CS Exit by Writer Thread 1 at 01:03
1st CS Entry by Reader Thread 2 at 01:04
1st CS Request by Reader Thread 3 at 01:05
1st CS Entry by Reader Thread 3 at 01:06
.
.
.

Pcode for semaphores: The pcode for normal readers-writes can be seen in the textbook while the fair readers-writers can be seen on the wikipedia page.

Input: The input to the program will be a file, named inp-params.txt, consisting of the parameters discussed above which are:

- nw: the number of writer threads,
- nr: the number of reader threads,
- kw: the number of times each writer thread tries to enter the CS,
- kr: the number of times each reader thread tries enter the CS,
- μ_{CS}, μ_{Rem} as described above.

Output: Your program must generate an output in the format to a file. A sample output would be as follows:
1st CS Request by Writer Thread 1 at 01:00
1st CS Entry by Writer Thread 1 at 01:01
1st CS Request by Reader Thread 2 at 01:02
1st CS Exit by Writer Thread 1 at 01:03
1st CS Entry by Reader Thread 2 at 01:04
1st CS Request by Reader Thread 3 at 01:05
1st CS Entry by Reader Thread 3 at 01:06
.
.

The output should demonstrate that writer and reader threads are accessing the critical section in a mutually exclusive manner. But it should allow multiple reader threads at the same time.

You program should output the following files:

1. You must display the log of all the events as shown for each of the algorithms. So your program must generate two output files: RW-log.txt and FairRW-log.txt, consisting of events, as described above.

2. Average_time.txt, consisting of the average time taken for a thread to gain entry to the Critical Section for each of the algorithms: RW and Fair-RW.

Report: You have to submit a report and readme for this assignment. The readme will explain how to compile and run your program.

The report should explain the design of your program. It should also contain a comparison graph of the performance of the two algorithms. You must run both of these algorithms multiple times to compare their performances and generate the following results graphs:

1. Average Waiting Times with Constant Writers and varying Readers: In this graph, you measure the average time taken to enter the CS by reader and writer threads with a constant number of writers. Here you vary the number of reader threads nr from 1 to 20 in the increments of 5 on the X-axis. We have all the other parameters fixed: Number of writer threads, $nw = 10$, $kr = kw = 10$. The Y-axis will have time in milli-seconds and will measure the average time taken to enter CS each for reader and writer threads. Specifically the graph will have four curves:
   (a) Average time taken by the reader threads to enter the CS for each algorithm: Readers_Writers() and Fair Readers_Writers().
   (b) Average time taken by the writer threads to enter the CS for each algorithm: Readers_Writers() and Fair Readers_Writers().
2. Average Waiting Times with Constant Readers and varying Writers: In this graph, you measure the average time taken to enter the CS by reader and writer threads with a constant number of readers. Here we vary the number of writer threads nw from 1 to 20 in the increments of 5 on the X-axis. We have all the other parameters fixed: Number of reader threads, $nr = 10$, $kr = kw = 10$. The Y-axis will have time in milli-seconds and will measure the average taken to enter CS each for reader and writer threads. The graph will have four curves:
   (a) Average time taken by the reader threads to enter the CS for each algorithm: Readers_Writers() and Fair Readers_Writers().
   (b) Average time taken by the writer threads to enter the CS for each algorithm: Readers_Writers() and Fair Readers_Writers().
3. Worst-case Waiting Times with Constant Writers and varying Readers: This graph will be similar to the graph in Step 1. In this graph, you measure the worst-case (instead of average) time taken to enter the CS by reader and writer threads with a constant number of writers. Here we vary the number of reader threads nr from 1 to 20 in the increments of 5 on the X-axis. We have all the other parameters fixed: Number of writer threads, $nw = 10$, $kr = kw = 10$. The Y-axis will have time in milli-seconds and will measure the worst-case time taken to enter CS by the reader and writer threads. The graph will have four curves:
   (a) Worst-case time taken by the reader threads to enter the CS for each algorithm: Readers_Writers() and Fair Readers_Writers().
   (b) Worst-case time time taken by the writer threads to enter the CS for each algorithm: Readers_Writers() and Fair Readers_Writers().
4. Worst-case Waiting Times with Constant Readers and varying Writers: This graph will be similar to the graph in Step 2. In this graph, you measure the worst-case time taken

to enter the CS by reader and writer threads with constant a number of readers. Here we vary the number of writer threads nw from 1 to 20 in the increments of 5 on the X-axis. We have all the other parameters fixed: Number of reader threads, $nr = 10$, $kr = kw = 10$. The Y-axis will have time in milli-seconds and will measure the worst-case time taken to enter CS each for reader and writer threads. The graph will have four curves:

(a) Worst-case time taken by the reader threads to enter the CS for each algorithm: Readers_Writers() and Fair Readers_Writers().

(b) Worst-case time taken by the writer threads to enter the CS for each algorithm: Readers_Writers() and Fair Readers_Writers().

Thus all the four graphs will contain four curves. The report should also explain the behaviour of the graphs.

Deliverables: You have to submit the following:

- The source file containing the actual program to execute named as Assn5-rw-<rollno>.cpp and Assn5-frw-<rollno>.cpp
- A readme.txt that explains how to execute the program
- The report as explained above.

Zip all the three files and name it as Assn5-<rollno>.zip. Then upload it on the google classroom page of this course. Submit it by the above mentioned date. Please follow this naming convention. Otherwise, your program will not be evaluated by the TAs.

Grading Policy:

1. Design as described in the report and analysis of the results: 50%
2. Execution of the programs based on the description in the readme: 40%
3. Code documentation and indentation: 10%

Kindly remember that all submissions are subjected to plagiarism checks.