Operating Systems–II: CS3523
Spring 2018
Programming Assignment 3:
Implementing Readers-Writers problem and Fair Readers-Writers problem using Semaphores in C++.

Submission Date:~~16/02/2018~~ 17/02/2018, 09:00 PM

Goal: The goal of this assignment is to implement a multithreaded solution for the above mentioned problems using Semaphores. You have to implement these two algorithms and compare the time taken for each thread to access the critical section (shared resources). Implement the algorithms in **C++**.

Details: As mentioned above, you have to implement the Readers-Writers problem and Fair Readers-Writers problem using Semaphores as discussed in Chapter 5 of the book in C++.

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 & 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:

```
 void main()
 {
         // create 'nw' writer threads
         // create 'nr' reader threads
 }
```

Listing 1: Main Thread

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

        for(int i=0; i<kw ;i++)
        {

                reqTime = getSysTime();
                cout<<i<<"th CS request by Writer Thread "<<id<<" at "<<reqTime<<endl;
                /*
                Write your code for

                * Readers_Writers() &
                * Fair Readers_Writers()
                Using Semaphores here.
                */
                enterTime = getSysTime();
                cout<<i<<"th CS Entry by Writer Thread "<<id<<" at "<<enterTime<<endl;
                randCSTime = Random(csSeed).get(); // get random CS Time
                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;
                randRemTime = Random(remSeed).get(); // get random Remainder Section Time
                sleep(randRemTime); // simulate a thread executing in Remainder Section
        }
}
```

Listing 2: Writer 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()
        * Fair Readers_Writers()
        Using Semaphores here.
        */
        enterTime = getSysTime();
        cout<<i<<"th CS Entry by Reader Thread "<<id<<" at "<<enterTime<<endl;
        randCSTime = Random(csSeed).get(); // get random CS Time
        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;
        randRemTime = Random(remSeed).get(); // get random Remainder Section Time
        sleep(randRemTime); // simulate a thread executing in Remainder Section
        }
}
```

Listing 3: Reader Thread

Example: 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
……
```

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 theCS, csSeed: the random CS time seed generator which simulates the time spent by each thread executing in CS, remSeed: the random remainder section time seed generator which simulates the time spent by each thread executing in remainder section.

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
……
```

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 report should first explain the design of your program while explaining any complications that arose in the course of programming. The report should contain a comparison graph of the performance of all two algorithms. You must run all two of these algorithms multiple times to compare their performances and display the result in form of a graph for the following parameters:

- Y-axis: Average time taken to enter the CS
- X-axis: Varying K from 1000 to 5000 in the increments of 1000. Have 10 threads execute Concurrently.

The graph must contain two curves for each of the algorithms: RW and Fair-RW.

Submission Format:

You have to submit the following deliverables for this assignment:

1. The source code: Assgn3-RW<RollNo>.cpp & Assgn3-RW_Fair<RollNo>.cpp
2. Readme: Assgn3-Readme-<RollNo>.txt
3. Report: Assgn3-Report-<RollNo>.pdf as explained above.

Name the zipped document as: Assgn3-<RollNo>.zip. If you don't follow these guidelines, then your assignment will not be evaluated. Upload all this on the classroom by **~~16/02/2018~~ 17/02/2018, 09:00 PM**.

Grading Policy:

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