Operating Systems–II: CS3523
Spring 2018
Programming Assignment 2:
Implementing Critical Section Solution using Test_And_Set, Test_And_Set with Bounded Waiting and Compare_And_Swap in C++.

Submission Date:31/01/2018, ~~9:00 PM~~ 11:00 PM

Goal: The goal of this assignment is to implement a multithreaded solution for critical section problem using the three techniques discussed in the class: Test_And_Set, Test_And_Set with Bounded Waiting and Compare_And_Swap. You have to implement these three algorithms and compare the time taken for each thread to access the critical section. Implement all the algos the algos in **C++**.

Details: As mentioned above, you have to implement the critical section problem using the soluitions discussed in Chapter 5 of the book: Test_And_Set, Test_And_Set with Bounded Waiting and Compare_And_Swap 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 n threads, which execute the testCS function. Each of these threads, will enter critical section (CS) k times. The pseudocode of the testCS function is as follows:

```
void main()
{
    // create 'n' testCS threads
}
```

Listing 1: Main Thread

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

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

        /*
        Write your code for
```

```
         * test_and_set()
         * test_and_set()  with bounded waiting
         * compare_and_swap()
         here.
         */
         enterTime = getSysTime();
         cout<<i<<"th CS Entry by Thread "<<id<<" at "<<enterTime<<endl;

        randCSTime = Random(csSeed).get();             // get random CS Time
        sleep(randCSTime);                             // simulate a thread executing in CS

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

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

        randRemTime = Random(remSeed).get();       // get random Remainder Section Time
        sleep(randRemTime);                        // simulate a thread executing in CS

    }
}
```
Listing 2: testCS Thread

Example: A sample output would be as follows:

```
1st CS Request by Thread 1 at 01:00
1st CS Entry by Thread 1 at 01:01
1st CS Exit by Thread 1 at 01:02
1st CS Request by Thread 3 at 01:03
…...
```

Input: The input to the program will be a file, named inp-params.txt, consisting of the following parameters: **n**: the number of threads, **k**: the number of times each thread tries to access the critical section, **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: 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 three output files: TAS-log.txt, TAS_bounded-log.txt, and CAS-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: TAS, TAS_bounded, and CAS

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 three algorithms.
You must run all three 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 three curves for each of the algorithms: TAS, TAS_bounded, and CAS.

Submission Format:
You have to submit the following deliverables for this assignment.

1. The source code: Assgn2-TAS<RollNo>.cpp, Assgn2-TAS_Fair<RollNo>.cpp & Assgn2-CAS<RollNo>.cpp
2. Readme: Assgn2-Readme-<RollNo>.txt
3. Report: Assgn2-Report-<RollNo>.pdf as explained above.

Name the zipped document as: Assgn2-<RollNo>.zip. If you don't follow these guidelines, then your assignment will not be evaluated. Upload all this on the drive by **31st January 2018, ~~9:00 PM~~ 11:00 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%

References:

- CPP reference link for CAS:
  http://en.cppreference.com/w/cpp/atomic/atomic_compare_exchange
- CPP reference link for Test & Set:
  http://en.cppreference.com/w/cpp/atomic/atomic_flag_test_and_set