Assignment 5
Implement Dining Philosopher's using Conditional Variables

Submission Date: 9th April 2019, 9:00 pm

Goal: To solve the **Dining Philosopher's problem** using conditional variables as discussed in the class in **C++**. You have to implement this algorithm and then compute the average and worst-case times taken for each thread to access the critical section (i.e. eat).

Details. As mentioned above, you have to develop a solution to Dining Philosopher's problem using Conditional Variables as discussed in the class 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 philosopher threads. Each of these threads, will access the Critical Section (or eat) by picking left and right chopsticks h times. The pseudocode of the application is as follows:

Listing 1: main thread

```
void main()
{
    ...
    ...
    create n philosopher threads;
    ...
    ...
}
```

Listing 2: Philospher Thread

```
void writer()
{
    int id = thread.getID();
    float randEatTime, randThinkTime;

    for(int i=0; i ≤ h ;i++)
    {

        reqTime = getSysTime();
        cout << i << "th eat request by Thread " << id <<
        " at " << reqTime << " endl";

        /*
        Write your code for the thread to pick up both the chopsticks.
        */

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

        randEatTime = exp_rand(μeat);
        sleep(randEatTime); // simulate the eating of the thread writing in CS

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

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

        randThinkTime = exp_rand(μthink);
        sleep(randThinkTime); // simulate a thread thinking
    }
}
```

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

As described above, you will measure average and worst-case time taken by threads to enters the CS (eat in this case). The worst-case time taken by a thread to enter the CS gives an indication of starvation in threads. The solution discussed in the class/book, can potentially cause some threads to starve. So in this assignment, the worst-case time taken measures the starvation.

Input: The input to the program will be a file, named inp-params.txt, consisting of the parameters discussed above which are - n: the number of philosopher threads, h: the number of times each philosopher thread tries to enter the CS (eat), μ_{CS}, μ_{Rem} as described above.

Output:

A sample output would be as follows:

```
1st eat request by Philosopher Thread 1 at 01:00
1st eat request by Philosopher Thread 2 at 01:01
1st eat request by Philosopher Thread 3 at 01:02
1st CS entry by Philosopher Thread 1 at 01:03
1st CS entry by Philosopher Thread 3 at 01:03
1st CS Exit by Philosopher Thread 1 at 01:05
1st CS Entry by Philosopher Thread 2 at 01:06
1st CS Exit by Philosopher Thread 3 at 01:05

    .
    .
    .
```

The output should neighboring threads 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: dphil-log.txt consisting of events, as described above.

2. Times.txt, consisting of the average and the worst-case times taken for a thread to gain entry to the critical section.

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: a graph with the average and worst times taken by each thread to enter the CS in the y-axis and the number of philosopher threads n varying in x-axis. Let the number of the philosopher threads n vary from 1 to 20 in the increments of 5 on the x-axis. Fix all the other parameters as: $h = 10, \mu_{CS} = 1, \mu_{Rem} = 2$. Your graph will contain two curves - one for average case and another for worst case. The report should also explain the behavior of the graphs.

Deliverables: You have to submit the following:

- The source file containing the actual program to execute named as dphil-<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.