Assignment 3
Solving Producer Consumer Problem using Semaphores and Locks

Submission Date: 31st March 2025, 9:00 pm

Goal: To solve the **bounded buffer producer-consumer problem** using Semaphores and Locks as discussed in the class in **C++** and compare their performances.

Details. You have to implement the bounded buffer producer-consumer problem using semaphores and locks. In the class, we studied producer consumer problem using semaphores. This problem can be implemented using locks as well. Locks are analogous to binary semaphores. The producer-consumers problem can be solved using binary semaphores and hence locks. However, the lock based solution may be inefficient as compared to the semaphore based solution.

As mentioned above, the objective of this assignment is to compare the solutions to producer-consumer problem using locks and semaphores.

Testing the Solution. In order to test the effectiveness of your solution, the outline of a test program that creates the producer/consumer threads is given below. Once, the program starts, it creates n_p, n_c producer and consumer threads respectively. The pseudocode of the program is as follows:

Listing 1: Initialization

```
create a buffer of size capacity;
```

Listing 2: main thread

```
void main()
{
    ...
    ...
    create $n_p$ producer threads;
    create $n_c$ consumer threads;
    ...
    ...
}
```

Listing 3: producer thread

```
void producer()
{
    id = thread.getID();
    for (i=0; i < $cnt_p$; i++)
    {
        // Use semaphores or locks here
        produce the next element into item;
        place item into the $m$th location in the shared buffer;
        prodTime = getSysTime();
        cout << i << "th item: " << item << " produced by thread "
        << id << " at " << prodTime << " into buffer location " << m;
        ...
        ...
        sleep($t1$);
    }
}
```

Listing 4: consumer thread

```
void consumer()
{
    id = thread.getID();
    for (i=0; i < $cnt_c$; i++)
    {
        // Use semaphores or locks here
        read item from the $m$th location from the shared buffer;
        consTime = getSysTime();
        cout << i << "th item: " << item <<
        " read from the buffer by the thread " << id << " at "
        << consTime << " from location " << m;
        consume the element in item;
        ...
        ...
        sleep($t2$);
    }
}
```

Here $t1$ and $t2$ are delay values that are exponentially distributed with an average of μ_p, μ_c 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 all the parameters described above:$capacity, n_p, n_c, cnt_p, cnt_c, \mu_p, \mu_c$. A sample input file is: 100 10 15 15 10 8 12.

Output: Your program should output to a file in the format given in the pseudocode for each algorithm. The output of all the producers and consumers must be combined. A sample output is as follows:

```
1st item produced by thread 1 at 10:00 into buffer location 1
2nd item produced by thread 1 at 10:03 into buffer location 2
1st item produced by thread 2 at 10:04 into buffer location 3
2nd item produced by thread 2 at 10:06 into buffer location 4
1st item consumed by thread 10 at 10:07 from buffer location 1
3rd item produced by thread 2 at 10:08 into buffer location 5

  .
  .
  .
```

The output should demonstrate the following properties:

- a producer thread produces items only when there is sufficient space left in the buffer;
- a consumer thread consumes only those items that are indeed produced by a consumer thread.

Report: You have to submit a report for this assignment. This report should contain a comparison of the performance of producer and consumer threads. You must execute these threads multiple times to compare the performances and display the result in form of a graph.

You will have two plot s in the report. The y-axis for all the graphs is the average time taken for the producers and consumers. It can be seen from the description above that in each execution, the time taken by the producer thread will should be averaged over cnt_p whereas the time taken by consumer thread will be averaged over cnt_c. The details of the x-axis is described below:

1. Delay ratios: The x-axis of the graph will consist of ratio of μ_p/μ_c, i.e. the ratio of the average delays of the producer thread to the consumer thread. It will specifically consist of the following 5 values: 10, 5, 1, 0.5, 0.1.

   You can have $cnt_p = 100$, $cnt_c = 100$, $\mu_p = 10ms$ in the above experiments.

2. Thread Number ratios: The x-axis of the graph will consist of ratio of cnt_p/cnt_c, i.e. the ratio of the number of producer threads to the consumer threads. It will specifically consist of the following 5 values: 10, 5, 1, 0.5, 0.1.

   You can have $\mu_p = \mu_c = 10$, $cnt_p = 100$ in the above experiments.

Thus, the graph will have four curves: (1) a curve for the time taken by producer threads using semaphores (2) a curve for the time taken by producer threads using locks (3) a curve for the time taken by consumer threads using semaphores (4) a curve for the time taken by consumer threads using locks. Finally, you must also give an analysis of the results while explaining any anomalies observed (like in the previous assignment).

Deliverables: You have to submit the following:

- The source file containing the actual program to execute named as prod_cons-sems-<rollno>.cpp and prod_cons-locks-<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 Assn3-<rollno>.zip. Then upload it on the google classroom page of this course. Submit it by the above mentioned date.