Operating Systems-2
CS 3510 Spring 2019
Programming Assignment 3:

Solving Producer Consumer Problem using Semaphores and Locks

INSTRUCTOR: DR. SATHYA PERI

Report By:
Vijay Tadikamalla
CS17BTECH11040

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Task

To solve the bounded buffer producer-consumer problem using semaphores and Locks as discussed in the class.

Approach and Implementation

1. To achieve our above mentioned goal make two functions producer and consumer which takes thread_index as a parameter. We will pass this function along with the thread index to the each thread and calculate the average time for both semaphore and mutex.
2. Functions and data-types of the chrono library (and other libraries) like
    a. **std::chrono::system_clock, std::chrono::system_clock::now()**
    b. **std::chrono::time_point**
    c. **struct tm - Time structure**
    d. **struct tm *localtime(const time_t *timer)**

    were used to calculate the average waiting time and max waiting time.
3. **int usleep(useconds_t *usec*)** function was used to suspend the execution of the thread for microsecond intervals
4. **template <class RealType = double> class exponential_distribution**: This is a random number distribution that produces floating-point values according to an exponential distribution, which is described by the following probability density function:

    $$p(x|\lambda) = \lambda e^{-\lambda x} \quad , \; x > 0$$

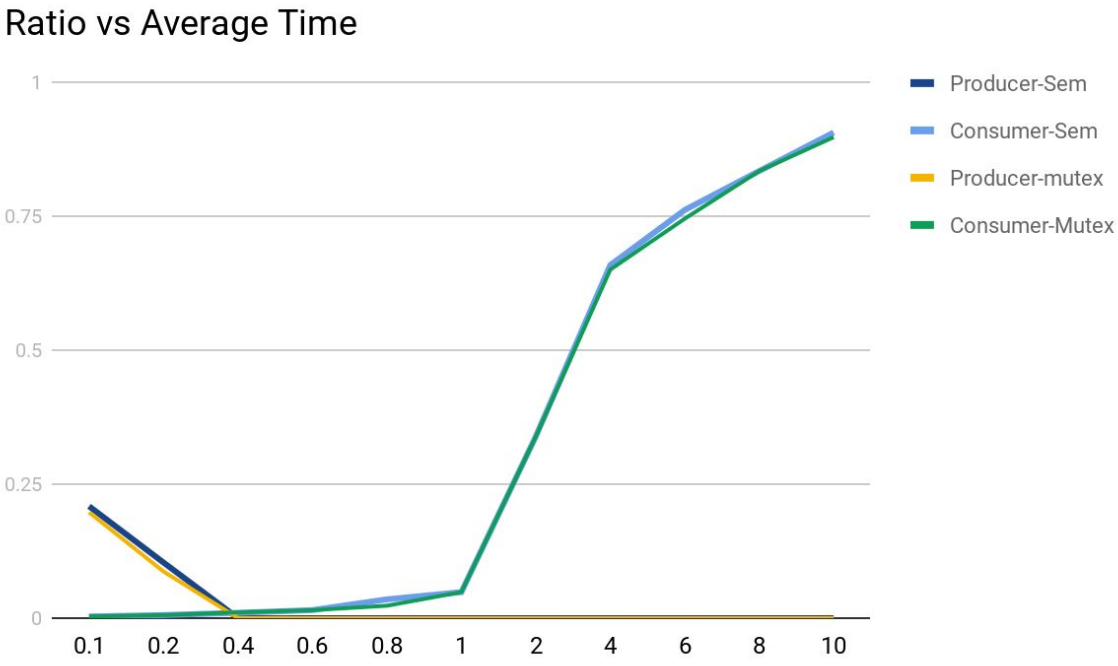
5. We make two exponential distributions and pass the value of 1/μp, 1/μc in the constructor. Later this can be used to obtain random numbers t1 and t2 with values that are exponentially distributed with an average of μp, μc seconds.

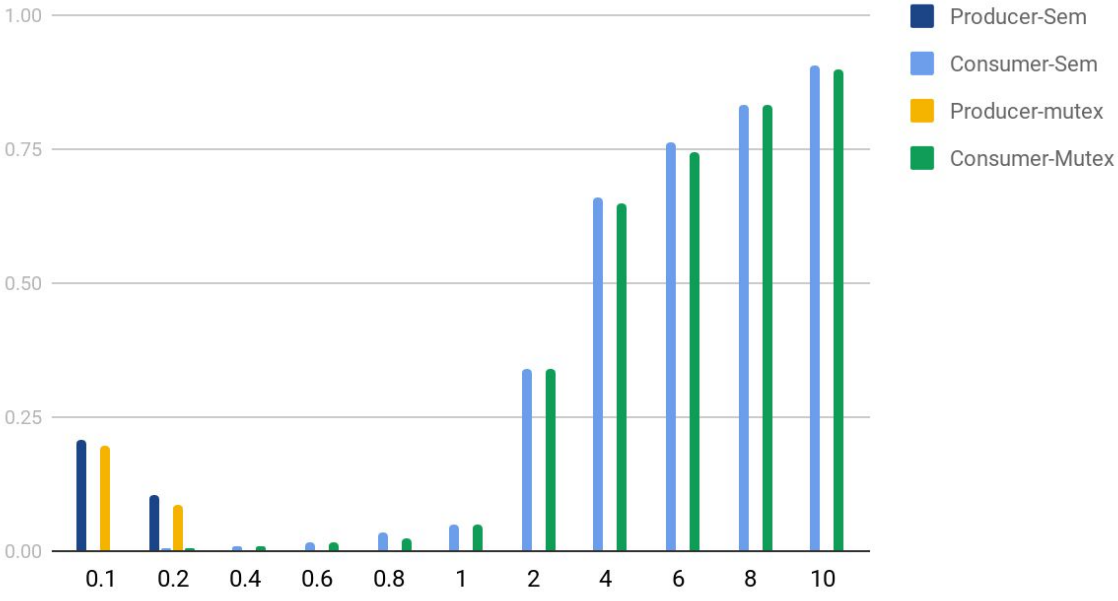
```
distribution1 = new exponential_distribution<double>(1/mu_p);
distribution2 = new exponential_distribution<double>(1/mu_c);
```

Input Parameters

Capacity =50
No of producers=10
No of consumers=10
cntp=10
cntc=10
μp + μc = 1

Graph 1 : Ratio vs Average time

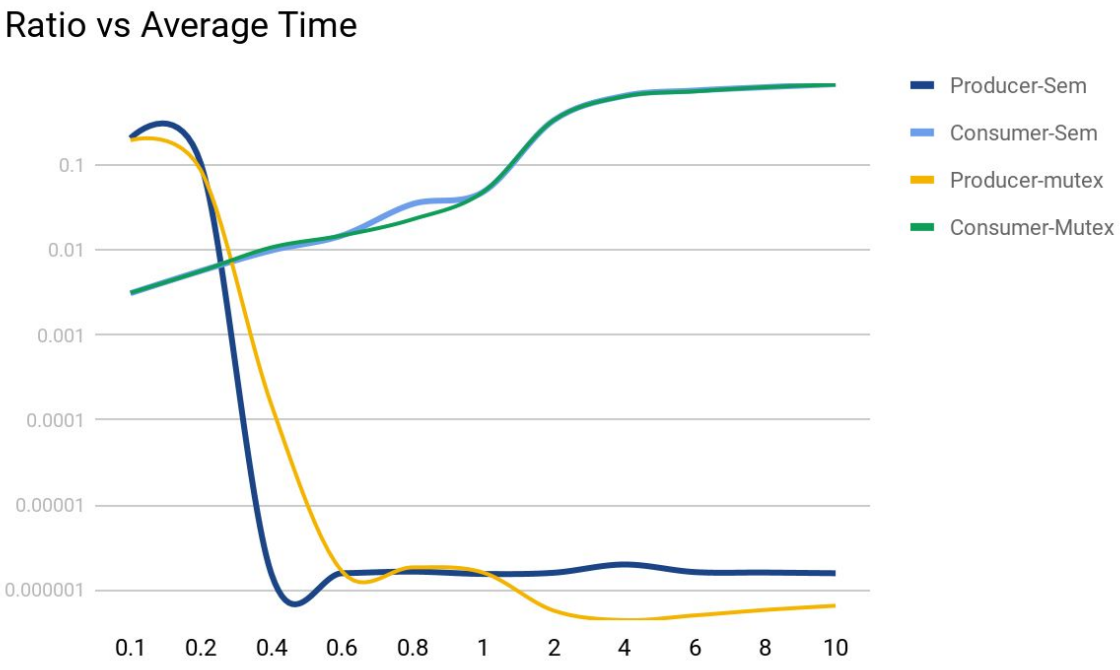




DATA

	Producer-Sem	Consumer-Sem	Producer-mutex	Consumer-Mutex
0.1	0.20873236	0.00311637	0.19734253	0.00315078
0.2	0.10371839	0.00572283	0.08698626	0.00569332
0.4	0.00000153	0.00995670	0.00015008	0.01082155
0.6	0.00000157	0.01493037	0.00000165	0.01491932
0.8	0.00000164	0.03505793	0.00000184	0.02315819
1	0.00000154	0.04857598	0.00000160	0.04854257
2	0.00000159	0.33902857	0.00000057	0.33895291
4	0.00000199	0.65866530	0.00000044	0.65026976
6	0.00000162	0.76150073	0.00000050	0.74523578
8	0.00000160	0.83416334	0.00000058	0.83414080
10	0.00000157	0.90601315	0.00000065	0.89733476

This Graph below is made using Logarithmic scale.



Output analysis of Graph

- Time taken by Producer Semaphore $\geqslant$ Time taken by Producer Mutex lock
  (Some anomalies may be observed because of some the background processes that might running on my laptop)
- Time taken by Consumer Semaphore $\approx$ Time taken by Consumer Mutex lock
- Time taken by Producer Semaphore and Producer Mutex lock is inversely proportional to μp/μc.
- Time taken by Consumer Semaphore and Consumer Mutex lock is directly proportional μp/μc.
- Ratio of time taken by producer and consumer that is,
  (Time taken by Producer)$\div$(Time taken by Consumer) is inversely proportional to μp/μc in both the cases.