OPERATING SYSTEMS 2

INSTRUCTOR: DR. SATHYA PERI
Hitesh Sehrawat - ma17btech11004@iith.ac.in

Theory

Producer-Consumer Problem : The producer–consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer used as a queue. The producer's job is to generate data, put it into the buffer, and start again. At the same time, the consumer is consuming the data (i.e., removing it from the buffer), one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.

Solution of this problem using : 1) Semaphores 2) Mutex Locks

1)Semaphores : Semaphore is a simply a variable. This variable is used to solve critical section problem and to achieve process synchronization in the multi processing environment. The two most common kinds of semaphores are counting semaphores and binary semaphores.

2)Mutex locks : A mutual exclusion object (mutex) is a program object that allows multiple program threads to share the same resource, such as file access, but not simultaneously.

Important Libraries Used :

- **<pthread.h> : Used for creation and work assignment of the threads.**
- **<random> : Used for the random number generator for seeding the sleep interval.**

- **<unistd.h> : Used for calculating the current system time.**
- **<semaphore.h> : Used to use semaphores**

Implementation

Both semaphore and mutex lock use same procedure for implementation except the logical they use for mutual exclusion to ensure only one thread is executing in critical section and tries to solve producer and consumer problem.

● Programs uses input file "input.txt" to accept input values. It accepts capacity, no of producer threads, no of consumer threads, producer frequency , consumer frequency, time for producer , time for consumer threads.

● A variable "capacity" which is capacity of buffer, "buffer" is initialized dynamically and accessible by both producer and consumer threads.

● Program uses default_random_generator generator to generate random number with exponential_distribution.

● Main threads creates np and nc producer and consumer threads and each producer thread executes the "producer" function cntp times, each consumer thread executes the "consumer" function cntc times.

● Semaphore implementation

1. Semaphore are used from c++ <semaphore.h> library.

2. Semaphore are defined and initialized as :

Sem_t <name of the semaphore>

sem_init(&<name of the semaphore >, { 0, 1} , value of semaphore)

"The second argument in sem_init can be 1 or 0, 1 if semaphores are shared among process and 0 if semaphore are shared among threads.

● **Mutex lock implementation**

1. Mutex locks are used form c++ <pthread.h> library.

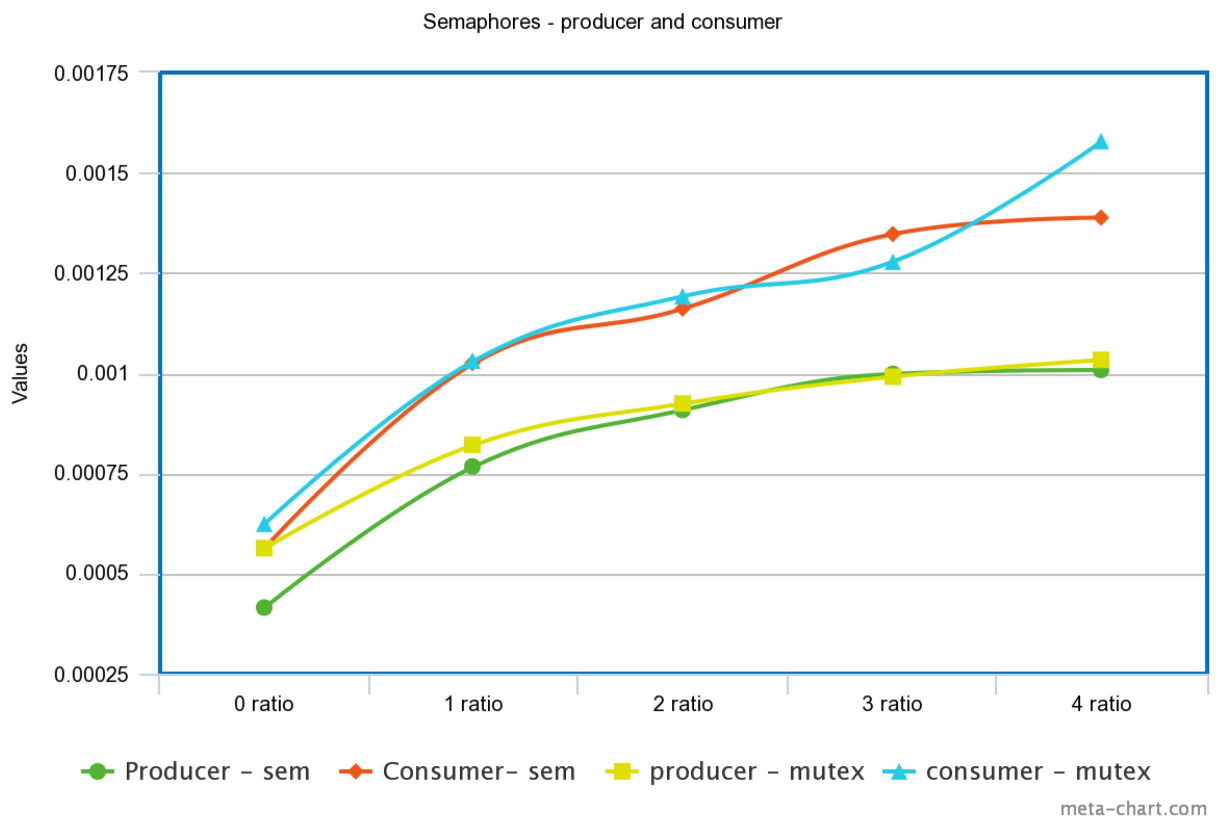
2. Mutex locks are defined and initialized as :

Pthread_mutex_t "mutex"

pthread_mutex_init(&mutex, NULL)

1) **In semaphore implementation 2 semaphores full and empty are used initialized to 0 and capacity resp. Also 1 binary semaphore "mutex" is used.**
2) **In mutex implementation 1 mutex is used along with 2 conditional parameters that take care for empty and full(buffer).**

Graphs



The Graphs contains :

(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.

- **The x-axis of the graph will consist of ratio of μp/μc, i.e. the ratio of the average delays of the producer thread to the consumer thread.**
- **The y-axis of the graph is the average time taken for the producers and consumers.**

Conclusions

The time taken in case of mutex locks is significantly similar to that of semaphores but still less than them.Since semaphore is a big struct required some more comutations than mutex.Thus mutex shows less average time in both producer and consumer cases.