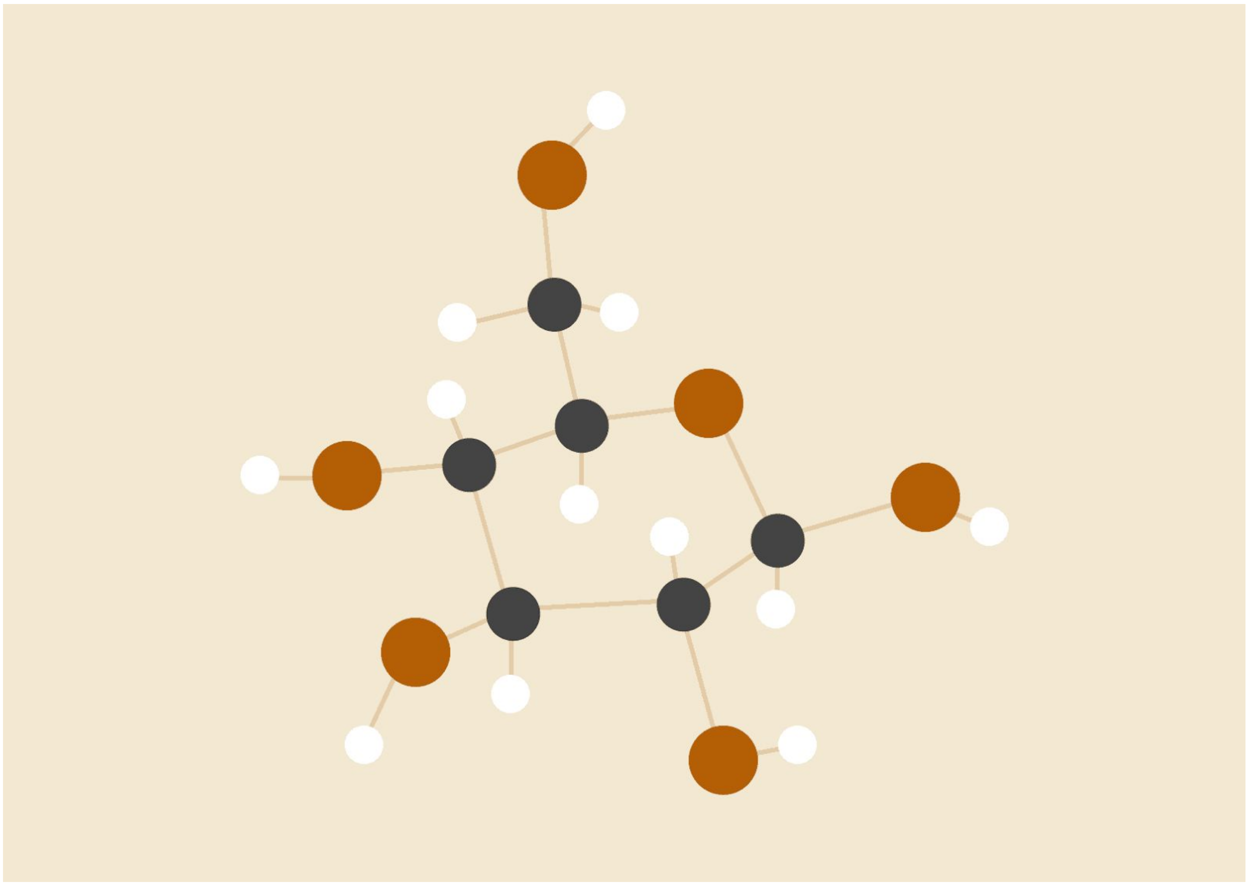
OS ASSIGNMENT 3

Solution of producer and consumer problems using semaphore and mutex



Happy Kumar

9.03.2019
CS17BTECH11018
Dr. Sathya peri

INTRODUCTION

This Assignment solves the producer and consumer problem using semaphores and mutex locks for n_p producer threads and n_c consumer threads with cnt_p and cnt_c producer thread and consumer thread frequency respectively.

HYPOTHESIS

Critical Section : Segment of code which is supposed to be accessed by only one thread at a time.

Entry Section : Segment of code which is available to all the threads and where threads contend to enter in critical section. Bounded waits guarantees that no threads waits for forever in critical section and gets a fair chance to enter in critical section.

Exit Section : Segment of code wherein thread exits critical and opens the lock for other process to enter in critical section.

Waiting Time : time that a threads spends in entry section waiting to enter in critical section.

Semaphore : an integer variable that, apart from initialization, is accessed only through two standard atomic operations: wait() and signal().

Mutex locks : mutex lock has a boolean variable available whose value indicates if the lock is available or not. If the lock is available, a call to acquire() succeeds, and the lock is then considered unavailable.

Producer and consumer problem

Code for producer :

```
while (true) {
        /* produce an item in next produced */
        while (count == BUFFER SIZE)
                ; /* do nothing */
        buffer[in] = next produced;
        in = (in + 1) % BUFFER SIZE;

        count++;

    }
```

Code for consumer :

```
    while (true) {

        while (count == 0)

            ; /* do nothing */

        next consumed = buffer[out];

        out = (out + 1) % BUFFER SIZE;

        Count--;

        /* consume the item in next consumed */

    }
```

Here if we execute the code without using any special software for hardware implementation the value of count will not consistent and will contain errors.

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 “inp-params.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 “BUFFER_SIZE” 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, they are contained in distribution1, and distribution2.
- Main threads creates n_p and n_c producer and consumer threads and each

producer thread executes the “producer” function cnt_p times, each consumer thread executes the “consumer” function cnt_c times.

- Program uses function getTime to print time whenever threads are in entry section and makes a request for critical section, threads enters in critical section and threads exits critical section.
- **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)

- In semaphore implementation two semaphores “empty” and “full” are initialized to BUFFER_SIZE and 0. Empty represents no of empty locations in buffer and full represents the no of locations filled in buffer, as the producer threads produces a item in buffer empty decrease and full increases and vice-versa .

  An extra semaphore is used to protect computation of “count” and “in” and for printing information about thread.

  Another extra semaphore is used for protecting the calculation of average waiting times for producer and consumer threads as average waiting times for producer and consumer may be disrupted by threads.

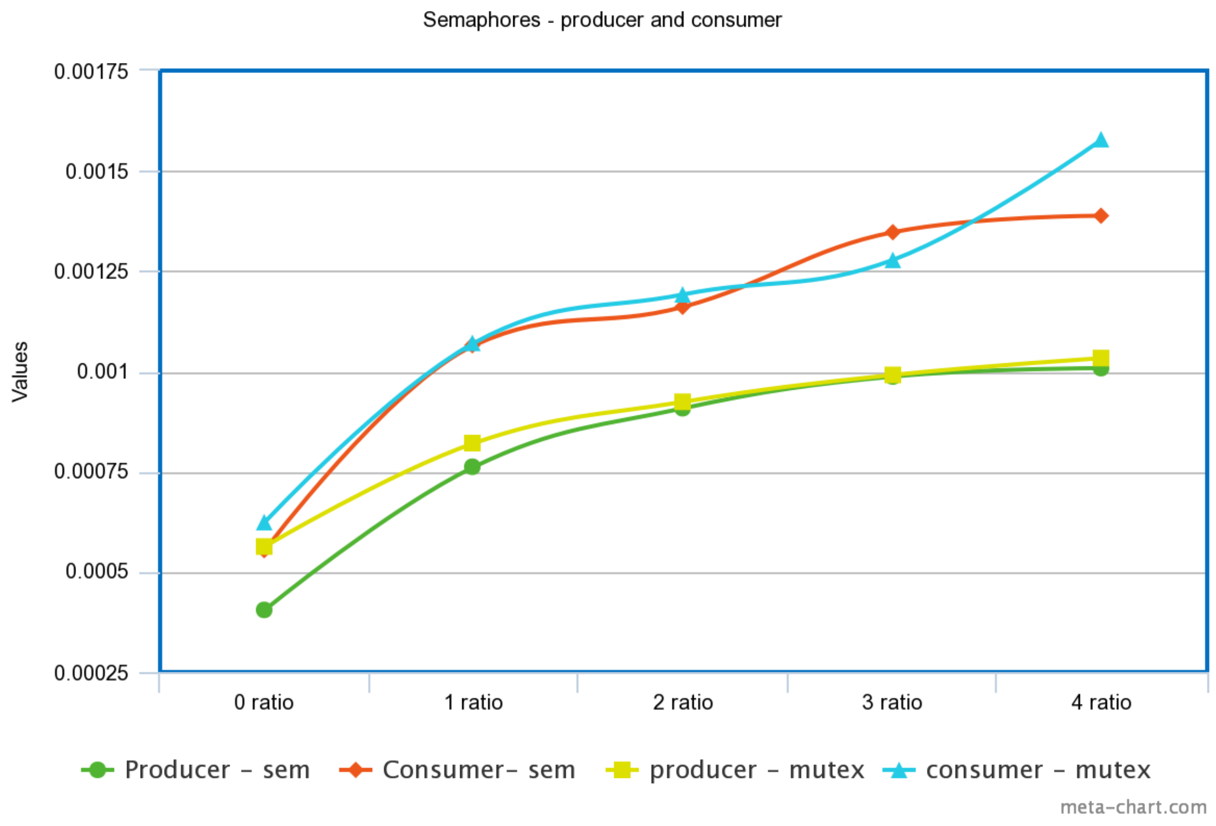
- In Mutex implementation two mutex , “mutex1” and “mutex3” are used for protecting the Critical section. While other two “mutex2” and “mutex4” is used for protecting calculation of average waiting time for producer and consumer respectively.

GRAPH

Graph is plotted between average waiting for producer and consumer threads and ratio of u_p and u_c .

The Graph contains four curve

1. Average producer waiting time for semaphore
2. Average consumer waiting time for semaphore\
3. Average producer waiting time for mutex
4. Average consumer waiting time for mutex



RESULTS

As from graph it is clear that the time that the time taken in waiting for threads in semaphore is less than the mutex.

This is evident as mutex uses spin lock implementation.

Timing Analysis

- **Waiting time of producer for semaphore < waiting time for producer for mutex**
- **Waiting time of consumer for semaphore < waiting time for consumer for mutex.**