**Laboratory 5**

Title of the Laboratory Exercise: Solution to Producer Consumer Problem using Semaphore and Mutex

1. Introduction and Purpose of Experiment

In multitasking systems, simultaneous use of critical section by multiple processes leads to data inconsistency and several other concurrency issues. By solving this problem students will be able to use Semaphore and Mutex for synchronisation purpose in concurrent programs.

1. Aim and Objectives

Aim

* To implement producer consumer problem using Semaphore and Mutex

Objectives

At the end of this lab, the student will be able to

* Use semaphore and Mutex
* Apply semaphore and Mutex in the required context
* Develop multithreaded programs with Semaphores and Mutex

1. Experimental Procedure
   * 1. Analyse the problem statement
     2. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
     3. Implement the algorithm in C language
     4. Compile the C program
     5. Test the implemented program
     6. Document the Results
     7. Analyse and discuss the outcomes of your experiment
2. Questions

Implement producer consumer problem by using the following

1. Semaphore
2. Mutex
3. Calculations/Computations/Algorithms

**Algorithm: produce( thread\_id )**

*1. start*

*2. generate random integer values between the range of 1 to 30.*

*3. While! pthread\_equal(\*(producers+i),self) and i < producer\_count increment i*

*End while loop.*

*4. Display item produced by producer.*

*5. Return item.*

*6. End function produce.*

**Algorithm:consume( item, thread\_id )**

*1. start*

*2. generate random integer values between the range of 1 to 30.*

*3.While !pthread\_equal(\*(producers+i),self) and i < producer\_count*

*increment i*

*End while loop.*

*4. Print buffer elements.*

*5. Print item consumed by consumer and buffer length.*

*6. End function consume.*

**Algorithm: Producer ()**

*1. Start*

*2. While(True)*

*i. item = produce(pthread\_self())*

*ii. decrement the number of empty slots*

*iii. lock critical section*

*iv. Increment buf\_pos*

*v. put the item at address (buf+buf\_pos)*

*vi. unlock critical section*

*vii. Increment the number of slots filled*

*viii. Delay for random number of seconds between 1 and 3.*

*3. End function producer.*

**Algorithm: Consumer ()**

*1. Start*

*2. While(True)*

*i. decrement the number of slots filled*

*ii. lock critical section*

*iii. copy value in (buf+buf\_pos) to item*

*iv. decrement buf\_pos*

*v. consume(c to pthread\_self)*

*vi. unlock critical section*

*vii. Increment the number of slots empty*

*viii. Delay for random number of seconds between 1 and 3.*

*3. End function consumer.*

**Algorithm:Main()**

**Declare threads producer and consumer.**

**Declare semaphores mutex,empty,full.**

**Declare \*buf,buf\_pos=-1, producer\_count, consumer\_count, buf\_len.**

**Declare i,error**

*1. Start.*

*2. Read number of producers and allocate memory using malloc.*

*3. Read number of consumers and allocate memory using malloc.*

*4. Read buffer capacity and allocate memory using malloc.*

*5. For i=0 to producer\_count-1*

*1. Create thread producers+I and call function producer and store return value in error.*

*i. if(error)*

*1. then print “error”*

*ii. Else*

*1. print success*

*End for loop.*

*6. For i=0 to consumer\_count-1*

*1. Create thread consumers+I and call function producer and store return value in error.*

*i. if(error)*

*1. then print “error”*

*ii. Else*

*1. print success*

*End for loop.*

*7. For i=0 to producer\_count-1*

*1. Join thread consumers+I and call function producer and store return value in error.*

*i. if(error)*

*1. then print “error”*

*ii. Else*

*1. print success*

*End for loop.*

*8. For i=0 to consumer\_count-1*

*1. Join thread consumers+I and call function producer and store return value in error.*

*i. if(error)*

*1. then print “error”*

*ii. Else*

*1. print success*

*End for loop.*

*9. Stop.*

1. **Presentation of Results**

**Code**

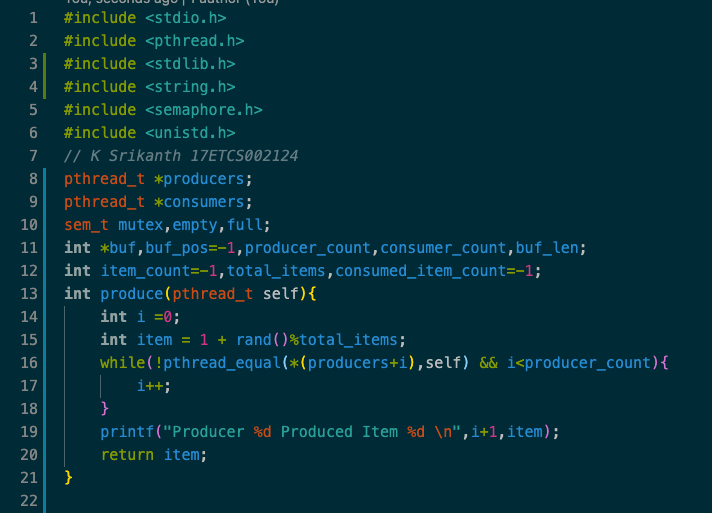
****

Figure 1 C Code for the given problem statement using semaphore and mutex

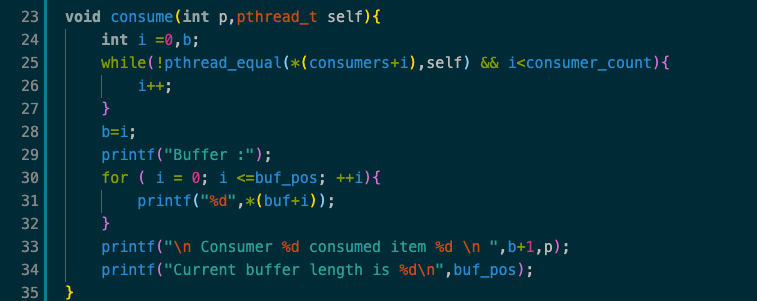
****

Figure 2 C Code for the given problem statement using semaphore and mutex continued

****

Figure 3 C Code for the given problem statement using semaphore and mutex continued

****

Figure 4 C Code for the given problem statement using semaphore and mutex continued

****

Figure 5 C Code for the given problem statement using semaphore and mutex continued

**Result**

**To Compile**

>> gcc filename.c -o filename -lpthread

**To Run the executable file**

>> ./filename

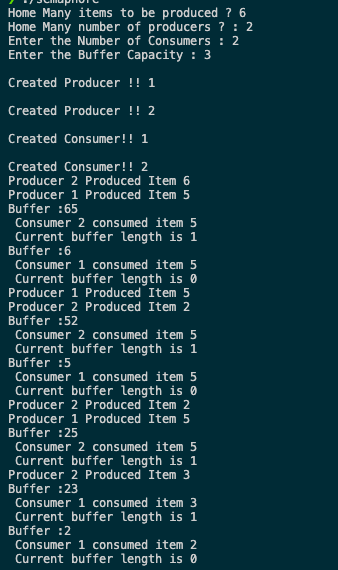


Figure 6 C output for the given problem statement using semaphore and mutex continued

1. **Analysis and Discussions**

**Semaphore**

A semaphore is a signalling mechanism and a thread that is waiting on a semaphore can be signalled

by another thread. This is different than a mutex as the mutex can be signalled only by the thread that

called the wait function. A semaphore uses two atomic operations, wait and signal for process

synchronization. The wait operation decrements the value of its argument S, if it is positive. If S is

negative or zero, then no operation is performed. The signal operation increments the value of its

argument S.

**Mutex**

Mutex is a mutual exclusion object that synchronizes access to a resource. It is created with a unique

name at the start of a program. The Mutex is a locking mechanism that makes sure only one thread can

acquire the Mutex at a time and enter the critical section. This thread only releases the Mutex when it

exits the critical section. A Mutex is different than a semaphore as it is a locking mechanism while a

semaphore is a signalling mechanism. A binary semaphore can be used as a Mutex but a

Mutex can never be used as a semaphore.

Srand(time(NULL)) makes use of the computer's internal clock to control the choice of the seed, Since

the time is continually changing, the seed is forever changing

sem\_init(sem\_t \*sem, int pshared,unsigned int value)

if pshared is 0, the value is shared among all the threads of the process, otherwise, the semaphore is

shared but should be in the shared memory

Important points to be considered in the main program:

Initialize mutex value to 1, initialize full value to 0, and initialize empty value to the length of the

buffer. Error variable lets you understand the status of thread creation. mutex is the semaphore type

variable to lock and unlock the mutex, so that one thread doesn’t interrupt the other while accessing

critical section. full value indicates number of filled slots in the buffer. empty value indicates the

empty slots in the buffer, buff\_pos to store the last filled slot position and buff to store buffer item.

1. **Conclusions**

**Advantages of using mutex:**

* Easy to implement: Mutexes are just simple locks that a thread obtains before entering its

critical section, and then releases it.

* Guarantee synchronization: Since only one thread is in its critical section at any given time,

there are no race conditions and data remain consistent.

**Advantages of using Semaphores:**

* Multiple processes and threads are allowed in their critical sections at the same time, without

interrupting one another.

* A binary semaphore can be used as a mutex. Systems that do not provide mutexes, the only

way out is to use a binary semaphore.

Based on their advantages and our problem semaphore is best suited for Producer consumer problem.

1. **Comments**

**1. Limitations of Experiments**

* Using semaphore there is no thread ownership. Any thread can take the ownership based on

the priority

* Using Mutex, if the main process dies then the racing thread which was sleeping will also die.
* Current program implements solution for producer and consumer problem using semaphores

only.

**2. Limitations in output**

The current program does not avoid racing conditions, that is why we have 8 items being

produced although the input given was 6 items.

**c. Learning happened**

* To implement producer consumer problem using Semaphore and Mutex.
* Differences, advantages and disadvantages.