PROJECT BASED LEARNING

Producer Consumer Problem In OS

Operating System

Our Team

YASH AKOTKAR
271006

RAJDEEP CHAURASIA
271015

NETAL DAGA
271016

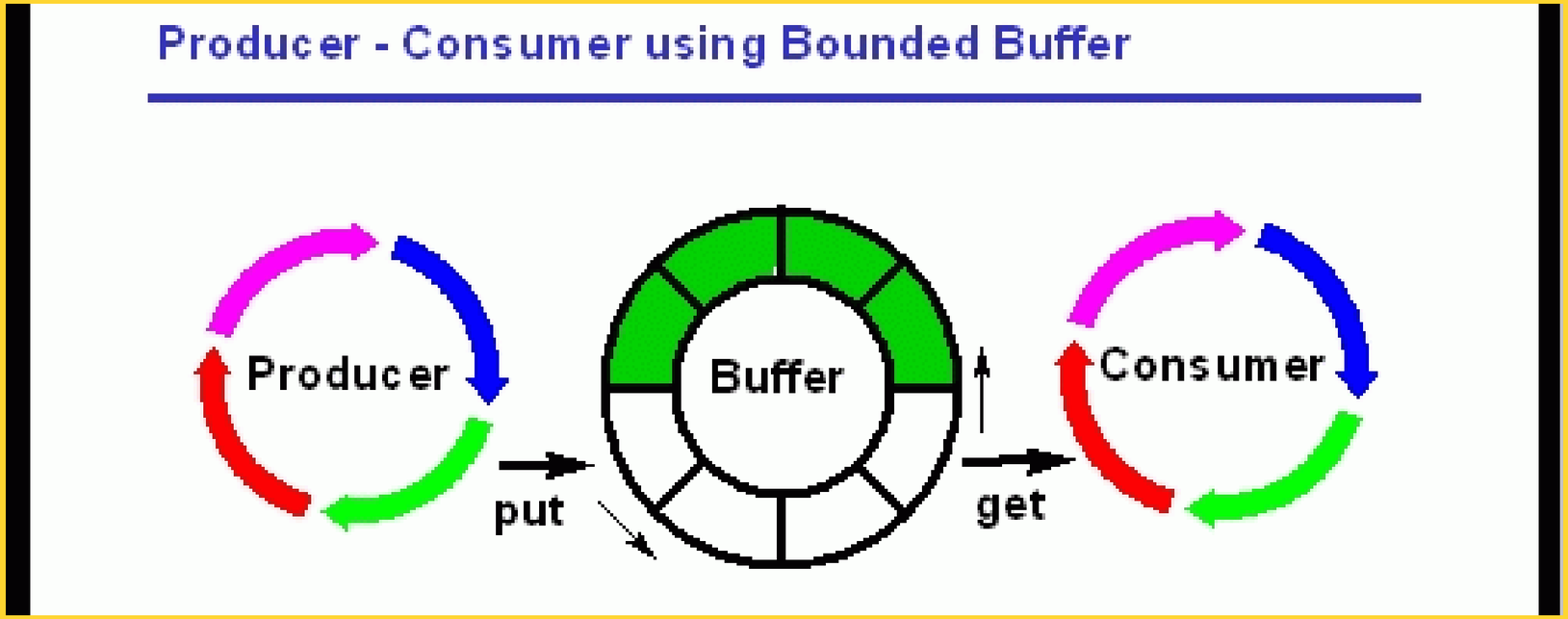
SAMARTH GHULE
271019

INTRODUCTION

- The Producer-Consumer problem is a classic problem this is used for multi-process synchronization i.e. synchronization between more than one processes. Also known as ***bounded-buffer problem.***
- Producer-Consumer problem is a classical synchronization problem in the operating system. With the presence of more than one process and limited resources in the system the synchronization problem arises.
- Producer thread: The producer's job is to generate data (a number) , put it into the buffer (array) , and repeat the same.
- Consumer thread: At the same time, the consumer is consuming the data (i.e., removing it from the buffer), one piece at a time.
- Mutual Exclusion: The producer and consumer should not access the buffer at the same time.

Before knowing what is Producer-Consumer Problem we have to know what are Producer and Consumer.

- In operating System Producer is a process which is able to produce data/item.
- Consumer is a Process that is able to consume the data/item produced by the Producer.
- Both Producer and Consumer share a common memory buffer. This buffer is a space of a certain size in the memory of the system which is used for storage. The producer produces the data into the buffer and the consumer consumes the data from the buffer.



What are the Producer-Consumer Problems?

Producer Process should not produce any data when the shared buffer is full. Consumer Process should not consume any data when the shared buffer is empty. The access to the shared buffer should be mutually exclusive i.e at a time only one process should be able to access the shared buffer and make changes to it. For consistent data synchronization between Producer and Consumer, the above problem should be resolved.

Solution For Producer Consumer Problem- To solve the Producer-Consumer problem 3 semaphores variable are used : Semaphores are variables used to indicate the number of resources available in the system at a particular time. semaphore variables are used to achieve `Process Synchronization.

- Full: The full variable is used to track the space filled in the buffer by the Producer process. It is initialized to 0 initially as initially no space is filled by the Producer process.
- Empty: The Empty variable is used to track the empty space in the buffer. The Empty variable is initially initialized to the BUFFER-SIZE as initially, the whole buffer is empty.
- Mutex: Mutex is used to achieve mutual exclusion. mutex ensures that at any particular time only the producer or the consumer is accessing the buffer. Mutex is a binary semaphore variable that has a value of 0 or 1.
- Signal(): The signal function increases the semaphore value by 1.
- Wait(): The wait operation decreases the semaphore value by 1.

Algorithm's

Producer:

- It first checks if buffer is empty.
- If buffer is full, producer will either go to sleep & keep waiting for buffer to be empty.
- Waiting forever is not the solution.
- Next time consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again.

Consumer:

- It checks if buffer is empty.
- If it finds buffer empty, consumer will go to sleep & keeps waiting for it to be full.
- The next time the producer puts the data into the buffer, it wakes up the sleeping consumer.

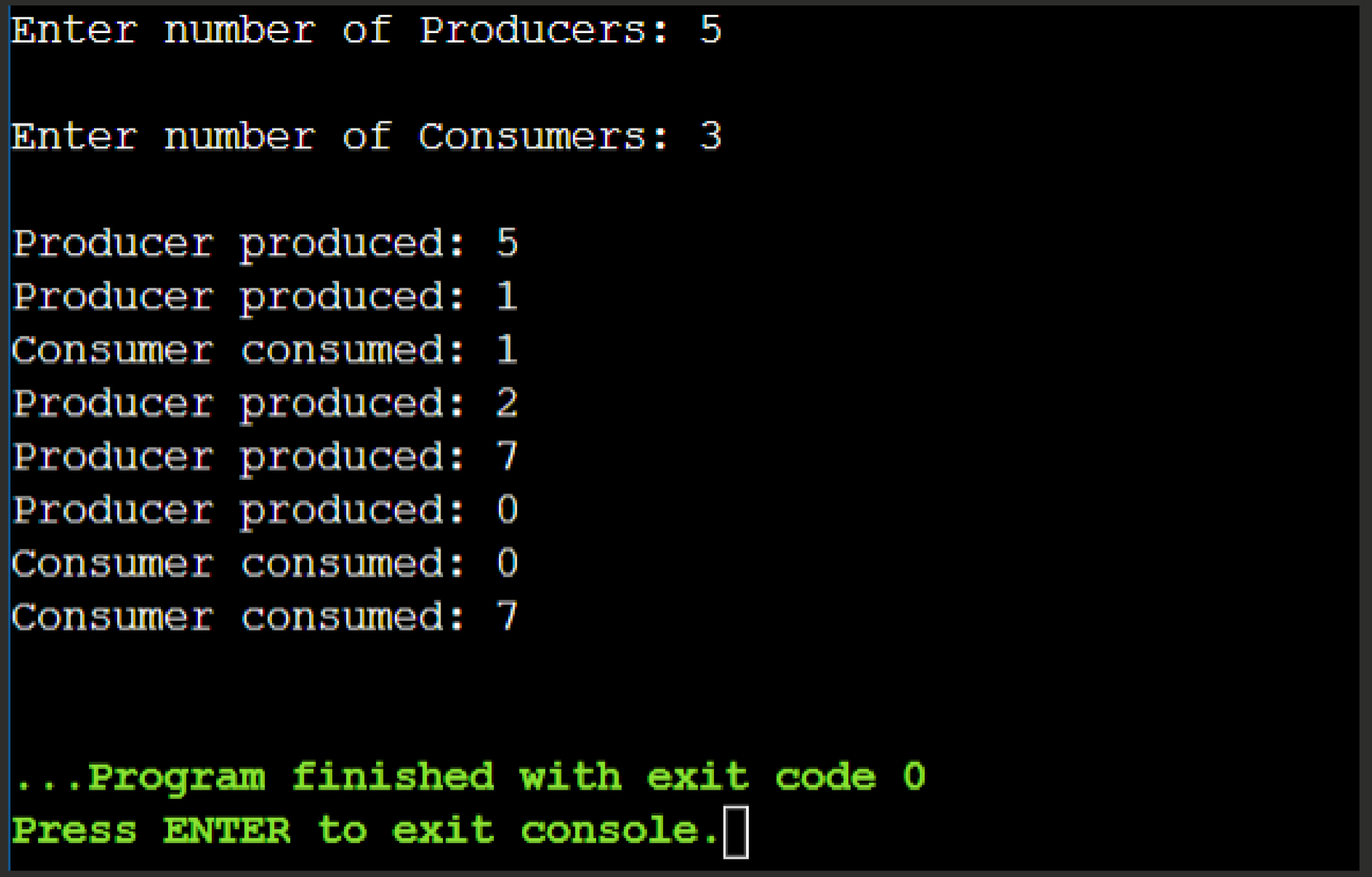
Pseudo-code

```
void Producer(){
    while(true){
        // producer produces an item/data
        wait(Empty);
        wait(mutex);
        add();
        signal(mutex);
        signal(Full);
    }
}
```

```
void Consumer() {
    while(true){
    // consumer consumes an item
        wait(Full);
        wait(mutex);
        consume();
        signal(mutex);
        signal(Empty);
    }
}
```

IMPLEMENTATION...

OUTPUT


Enter number of Producers: 5
Enter number of Consumers: 3
Producer produced: 5
Producer produced: 1
Consumer consumed: 1
Producer produced: 2
Producer produced: 7
Producer produced: 0
Consumer consumed: 0
Consumer consumed: 7
...Program finished with exit code 0
Press ENTER to exit console.

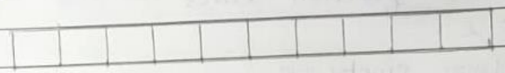
Rough Work

Buffer size = 10
Semaphore full = 0
Semaphore empty = 10

Example :-
Input :- Enter no. of items producer should produce = 5
Enter no. of items consumer should consume = 3

$n_1 = 5$ & $n_2 = 3$

Initially buffer is empty



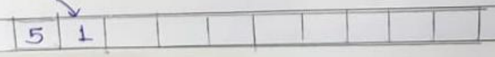
Step 1 :- Producer function runs
item = 5
Semaphore empty = 9

Inserted
item 5

Semaphore full = 1

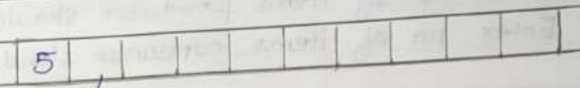
Step 2 :- Producer function runs
item = 1
Semaphore empty = 8

Inserted
item 1



Semaphore full = 2

Step 3 :- ~~Semaphore~~ Consumer function runs
item = 1
Semaphore full = 1



removed 1

Semaphore ~~full~~ empty = 9

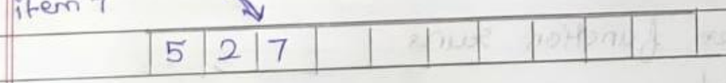
Step 4 :- Producer function runs
item = 2
semaphore empty = 8

inserted
item 2

semaphore full = 2

Step 5 :- Producer function runs
item = 7
Semaphore empty = 7

inserted
item 7



Semaphore full = 3

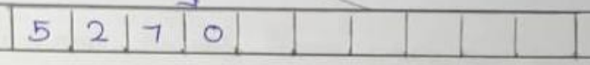
Step 6:- Producer function runs

item = 0

Semaphore empty = 6

inserted

item 0



Semaphore full = 4

Step 7:- Consumer function runs

item = 0

semaphore full = 3

removed

item 0

Semaphore empty = 7

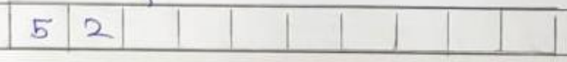
Step 8:- Consumer function runs

item = 7

Semaphore full = 2

removed

item 7



Semaphore empty = 8

Review of Research Paper

IMPLEMENTATION AND EXPERIMENTATION OF PRODUCER-CONSUMER SYNCHRONIZATION PROBLEM

Processing Models

It have clearly explained the 3 different models used in solving this problem.

Visualization of processes

Processes are easily understandable while referring to those helpful illustration through graphs.

Execution of steps

Easily explained how to build that particular simulation.

Problem explanation

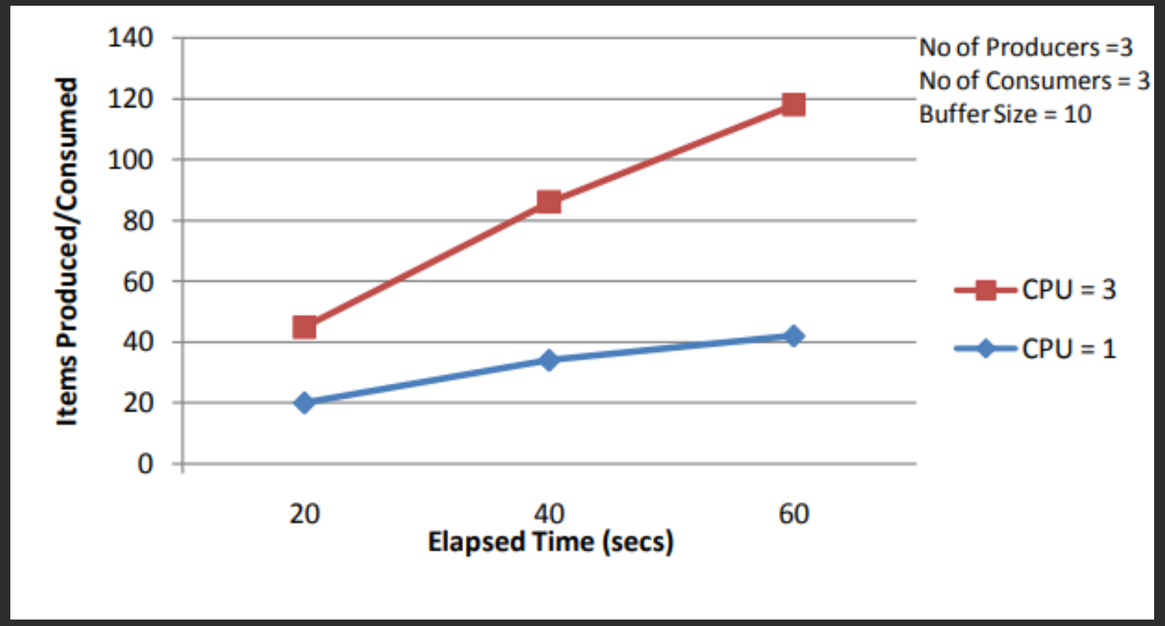
Author has clearly explained what is exact producer-consumer problem & it's different types.

Helpful & on - point

Sources were legitimate. Author was sticked to topic in overall paper & provided credible solution.

Conclusion

- Full, Empty and mutex semaphore help to solve Producer-consumer problem.
- Full semaphore checks for the number of filled space in the buffer by the producer process
- Empty semaphore checks for the number of empty spaces in the buffer.
- Mutex checks for the mutual exclusion.
- In this single process model, I have used only one CPU and the rest of the parameters such as Buffer Size, Number of Producers and Consumers must be given by the user for simulation.
- Default number of buffer is 10.



Simulation results for relationship between items produced/consumed and elapsed time (20 seconds) for single process environments.

THANK YOU!!