NAME: G. GANESH

REG NO: 192373008

EXERICSE-29

Write a C program to simulate the solution of Classical Process Synchronization Problem

Aim:

To write a C program that simulates the solution of the classical process synchronization problems, such as the **Producer-Consumer Problem** using **semaphores**.

Algorithm:

1. **Initialization of Semaphores:**
   - Use semaphores to handle synchronization between the producer and consumer processes.
   - Create a semaphore for the buffer (to manage available space) and a semaphore for counting the number of items in the buffer.
2. **Producer Process:**
   - Wait for space to be available in the buffer.
   - Produce an item.
   - Place the item in the buffer and signal the consumer that there is a new item.
3. **Consumer Process:**
   - Wait for an item to be available in the buffer.
   - Consume the item.
   - Signal the producer that there is space available for producing a new item.
4. **Shared Buffer:**
   - Use a shared buffer to simulate the storing and consuming of items between producer and consumer.

Procedure:

1. Initialize the semaphores.
2. Create the producer and consumer processes (threads).
3. The producer waits for space, produces an item, and signals the consumer.
4. The consumer waits for an item, consumes it, and signals the producer.
5. Terminate when the desired number of iterations is complete.

Code:

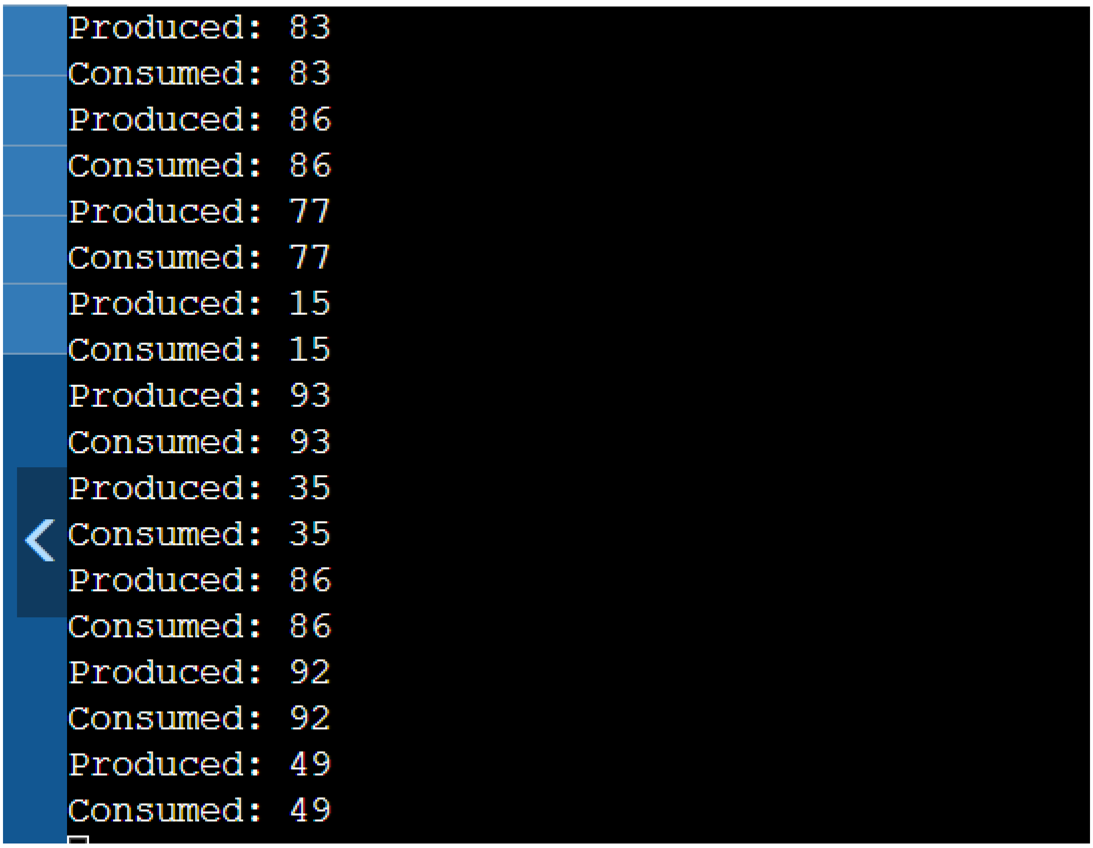
```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define BUFFER_SIZE 5
int buffer[BUFFER_SIZE];
int in = 0, out = 0;  // Indices for the producer and consumer
sem_t empty, full;
pthread_mutex_t mutex;
void* producer(void* arg) {
    int item;
    for (int i = 0; i < 10; i++) {
        item = rand() % 100;
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        buffer[in] = item;
        printf("Produced: %d\n", item);
        in = (in + 1) % BUFFER_SIZE;
        pthread_mutex_unlock(&mutex);  // Unlock buffer
        sem_post(&full);  // Signal that the buffer has a new item
        sleep(1);  // Simulate production time
    }
    return NULL;
}
void* consumer(void* arg) {
    int item;
    for (int i = 0; i < 10; i++) {
        sem_wait(&full);  // Wait for an item to be available in the buffer
        pthread_mutex_lock(&mutex);  // Lock buffer to prevent race condition
        item = buffer[out];  // Consume an item from the buffer
        printf("Consumed: %d\n", item);
        out = (out + 1) % BUFFER_SIZE;  // Update the index for the next item
        pthread_mutex_unlock(&mutex);  // Unlock buffer
        sem_post(&empty);  // Signal that there is space in the buffer
        sleep(1);  // Simulate consumption time
    }
    return NULL;
}
int main() {
    pthread_t prod, cons;

    sem_init(&empty, 0, BUFFER_SIZE);  // Initially, the buffer is empty
    sem_init(&full, 0, 0);  // Initially, no item is in the buffer
    pthread_mutex_init(&mutex, NULL);  // Initialize the mutex for critical section
    pthread_create(&prod, NULL, producer, NULL);
    pthread_create(&cons, NULL, consumer, NULL);
    pthread_join(prod, NULL);
    pthread_join(cons, NULL);
    sem_destroy(&empty);
    sem_destroy(&full);
    pthread_mutex_destroy(&mutex);
    return 0;
}
```

Result:

The program successfully simulates the Producer-Consumer problem using semaphores and mutex locks, where the producer produces items and the consumer consumes them while ensuring proper synchronization between the two processes.

Output:



```
Produced: 83
Consumed: 83
Produced: 86
Consumed: 86
Produced: 77
Consumed: 77
Produced: 15
Consumed: 15
Produced: 93
Consumed: 93
Produced: 35
Consumed: 35
Produced: 86
Consumed: 86
Produced: 92
Consumed: 92
Produced: 49
Consumed: 49
```