NAME: G. GANESH

REG NO: 192373008

EXERICSE-18

Construct a C program to simulate producer-consumer problem using semaphores.

Aim:

To simulate the Producer-Consumer problem using semaphores in C.

Algorithm:

1. Initialize Semaphores and Buffers:
    - Use two semaphores: full (count of filled slots) and empty (count of empty slots).
    - Use a mutex semaphore to ensure mutual exclusion while accessing the buffer.
    - Initialize full to 0, empty to buffer size, and mutex to 1.
2. Producer Process:
    - ➢ Wait on the empty semaphore to ensure there is space in the buffer.
    - ➢ Wait on mutex to enter the critical section.
    - ➢ Produce an item and add it to the buffer.
    - ➢ Signal mutex to exit the critical section.
    - ➢ Signal the full semaphore to indicate a filled slot.
3. Consumer Process:
    - ➢ Wait on the full semaphore to ensure there is at least one filled slot.
    - ➢ Wait on mutex to enter the critical section.
    - ➢ Consume an item from the buffer.
    - ➢ Signal mutex to exit the critical section.
    - ➢ Signal the empty semaphore to indicate an empty slot.
4. Repeat:

    Alternate between producer and consumer operations in a loop.

Procedure:

1. Initialize semaphores and the buffer.
2. Create producer and consumer threads.
3. Implement the producer and consumer logic as described in the algorithm.
4. Simulate the process by repeatedly producing and consuming items.
5. Print the state of the buffer after each operation.

Code:

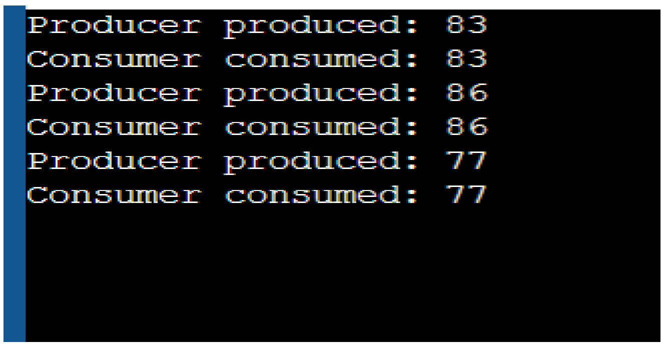
```
#include <stdio.h>
```

```
#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;
sem_t empty, full;
pthread_mutex_t mutex;
void *producer(void *param) {
    int item;
    while (1) {
        item = rand() % 100; // Produce an item
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        buffer[in] = item;
        printf("Producer produced: %d\n", item);
        in = (in + 1) % BUFFER_SIZE;
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
        sleep(1);
    }
}
void *consumer(void *param) {
    int item;
    while (1) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        item = buffer[out];
        printf("Consumer consumed: %d\n", item);
        out = (out + 1) % BUFFER_SIZE;
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
        sleep(1);
    }
}
int main() {
    pthread_t prod, cons;
    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    pthread_mutex_init(&mutex, NULL);
    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, ensuring mutual exclusion and synchronization between the producer and consumer processes.

Output:



```
Producer produced: 83
Consumer consumed: 83
Producer produced: 86
Consumer consumed: 86
Producer produced: 77
Consumer consumed: 77
```