praveenkumar.R
192324012

Experiment-18:Construct a C program to simulate producer-consumer problem using semaphores.

Aim:
The aim of this program is to simulate the Producer-Consumer problem using semaphores for synchronization. In this scenario, one or more producers generate data and store it in a shared buffer, and one or more consumers consume the data from the buffer. The program will ensure mutual exclusion and synchronization using semaphores to avoid race conditions and ensure data integrity.
Procedure:

1. Initialization:
    - We use two semaphores: empty (to track empty spaces in the buffer) and full (to track filled spaces in the buffer). These semaphores help the producer and consumer to coordinate their actions.
    - mutex is a binary semaphore (used for mutual exclusion) to ensure that only one process can access the buffer at a time.
2. Producer:
    - The producer checks if the buffer has space (using the empty semaphore).
    - If space is available, the producer places an item in the buffer.
    - Then, it signals the full semaphore to notify the consumer that there is a new item to consume.
3. Consumer:
    - The consumer checks if the buffer has data (using the full semaphore).
    - If data is available, the consumer consumes an item from the buffer.
    - Then, it signals the empty semaphore to notify the producer that there is space in the buffer.
4. Synchronization:
    - The producer and consumer are synchronized using the semaphores to ensure that no producer can add to the buffer when it's full, and no consumer can consume from the buffer when it's empty.

Code

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>
#include <unistd.h>

#define MAX_BUFFER_SIZE 5

sem_t empty;
sem_t full;
sem_t mutex;

int buffer[MAX_BUFFER_SIZE];
int in = 0;
int out = 0;

void* producer(void* param) {
```

```
    int item;
    while (1) {
        item = rand() % 100;

        sem_wait(&empty);

        sem_wait(&mutex);

        buffer[in] = item;
        printf("Producer produced: %d\n", item);
        in = (in + 1) % MAX_BUFFER_SIZE;

        sem_post(&mutex);

        sem_post(&full);

        sleep(1);
    }
}

void* consumer(void* param) {
    int item;
    while (1) {
        sem_wait(&full);

        sem_wait(&mutex);

        item = buffer[out];
        printf("Consumer consumed: %d\n", item);
        out = (out + 1) % MAX_BUFFER_SIZE;

        sem_post(&mutex);

        sem_post(&empty);

        sleep(1);
    }
}

int main() {
    pthread_t prod_thread, cons_thread;

    sem_init(&empty, 0, MAX_BUFFER_SIZE);
    sem_init(&full, 0, 0);
    sem_init(&mutex, 0, 1);

    pthread_create(&prod_thread, NULL, producer, NULL);
    pthread_create(&cons_thread, NULL, consumer, NULL);

    pthread_join(prod_thread, NULL);
    pthread_join(cons_thread, NULL);
```

```
    sem_destroy(&empty);
    sem_destroy(&full);
    sem_destroy(&mutex);

    return 0;
}
```

Output:

Output

```
Producer produced: 45
Consumer consumed: 45
Producer produced: 72
Consumer consumed: 72
Producer produced: 18
Consumer consumed: 18
Producer produced: 91
Consumer consumed: 91
```