192311035

M.RAVI SAI VINAY

EXPERIMENT-29

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

AIM:-

To write a C program to simulate the solution to the classical process synchronization problem (Producer-Consumer Problem) using semaphores.

ALGORITHM:-

1. Initialize Semaphores and Variables:
2. Use semaphores full, empty, and mutex to control synchronization.
3. Initialize full to 0, empty to the buffer size, and mutex to 1.
4. Create Threads:
5. Create producer and consumer threads.
6. Producer Logic:
7. Wait for the empty semaphore and acquire the mutex lock.
8. Add an item to the buffer.
9. Release the mutex lock and increment the full semaphore.
10. Consumer Logic:
11. Wait for the full semaphore and acquire the mutex lock.
12. Remove an item from the buffer.
13. Release the mutex lock and increment the empty semaphore.
14. Simulate the Problem:
15. Use a loop to produce and consume items.

CODE:-

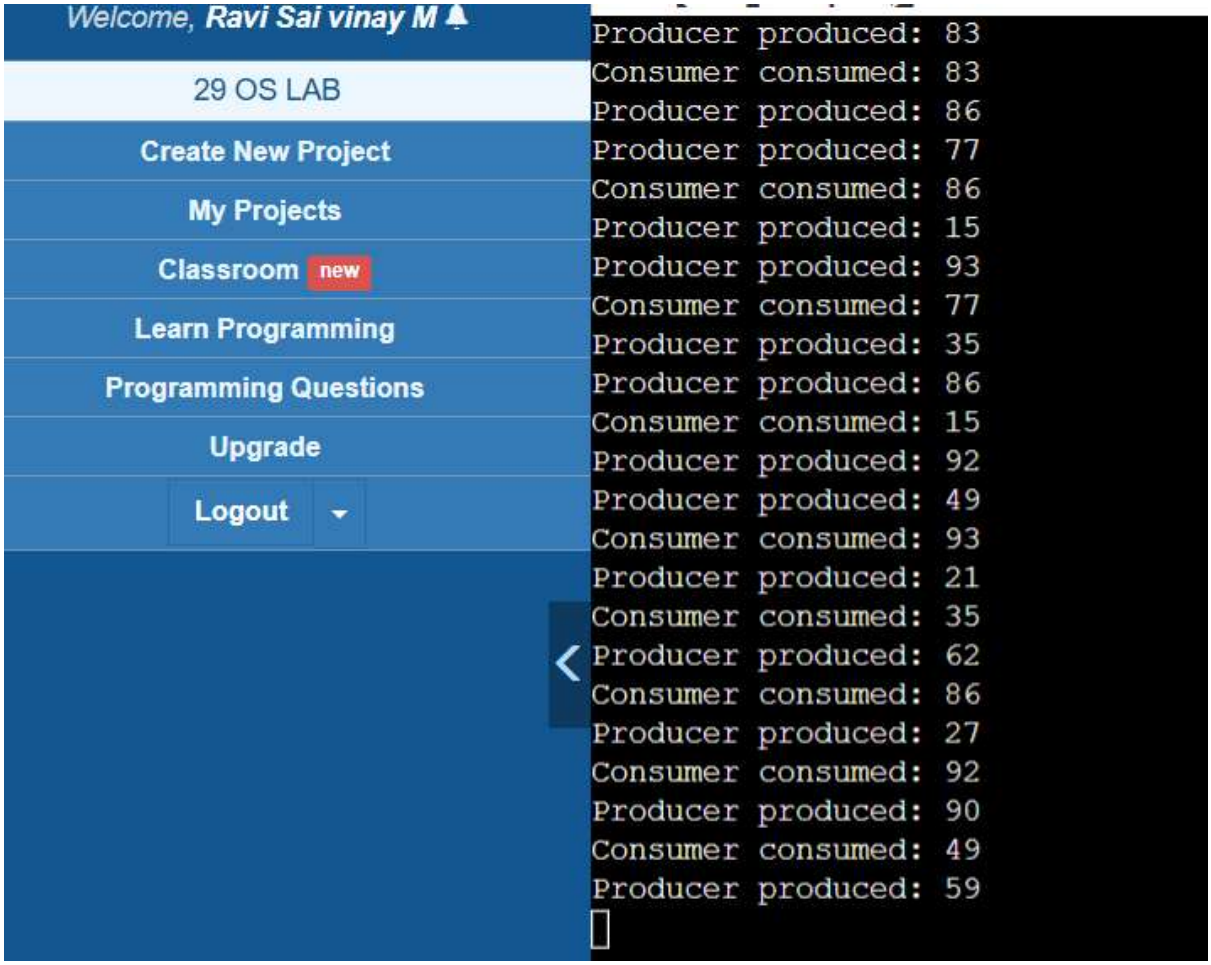
```
#include <stdio.h>

#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, mutex;

void *producer(void *arg) {
    int item;
    while (1) {
        item = rand() % 100; // Produce a random item
        sem_wait(&empty);    // Wait if buffer is full
        sem_wait(&mutex);    // Enter critical section

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

        sem_post(&mutex);    // Exit critical section
        sem_post(&full);     // Increment full count
```

```
        sleep(1);           // Simulate time
    }
}

void *consumer(void *arg) {
    int item;
    while (1) {
        sem_wait(&full);     // Wait if buffer is empty
        sem_wait(&mutex);    // Enter critical section

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

        sem_post(&mutex);    // Exit critical section
        sem_post(&empty);    // Increment empty count

        sleep(2);           // Simulate time
    }
}

int main() {
    pthread_t prod, cons;
```

```
    // Initialize semaphores
    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    sem_init(&mutex, 0, 1);

    // Create producer and consumer threads
    pthread_create(&prod, NULL, producer, NULL);
    pthread_create(&cons, NULL, consumer, NULL);

    // Join threads (this will not happen in this infinite simulation)
    pthread_join(prod, NULL);
    pthread_join(cons, NULL);

    // Destroy semaphores
    sem_destroy(&empty);
    sem_destroy(&full);
    sem_destroy(&mutex);

    return 0;
}
```

OUTPUT:-



RESULT:-

The program successfully simulated the classical process synchronization problem (Producer-Consumer) using semaphores, ensuring proper synchronization between producers and consumers accessing a shared buffer.