192311035

M.RAVI SAI VINAY

EXPERIMENT-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. Initialization:
    - Define semaphores empty, full, and mutex.
    - empty represents the number of empty slots in the buffer.
    - full represents the number of occupied slots.
    - mutex ensures mutual exclusion during buffer access.
2. Producer:
    - Wait on empty (checks if there's space) and mutex (ensures exclusive access).
    - Add an item to the buffer.
    - Signal full (indicating an item is added) and mutex.
3. Consumer:
    - Wait on full (checks if items are available) and mutex.
    - Remove an item from the buffer.
    - Signal empty (indicating a slot is freed) and mutex.
4. Repeat:
    - Continue producing and consuming until the desired condition is met.

CODE:-

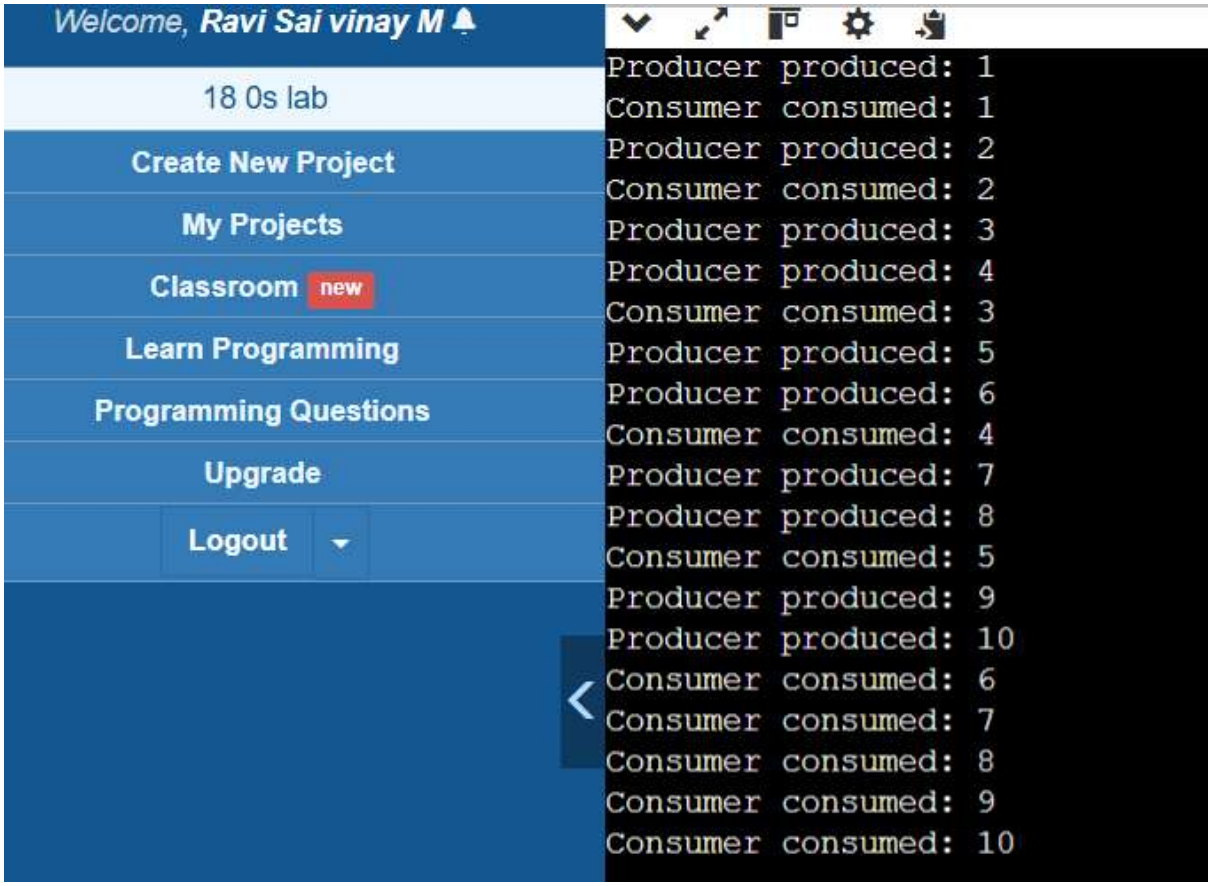
```
#include <stdio.h>
```

```
#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;
    for (int i = 0; i < 10; i++) {
        item = i + 1;
        sem_wait(&empty);
        sem_wait(&mutex);

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

        sem_post(&mutex);
        sem_post(&full);
        sleep(1);
```

```
    }
    return NULL;
}

void *consumer(void *arg) {
    int item;
    for (int i = 0; i < 10; i++) {
        sem_wait(&full);
        sem_wait(&mutex);

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

        sem_post(&mutex);
        sem_post(&empty);
        sleep(2);
    }
    return NULL;
}

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

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

    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);
    sem_destroy(&mutex);

    return 0;
}
```

OUTPUT:-



RESULT:-

The producer-consumer problem was successfully implemented using semaphores. It ensured proper synchronization, avoiding race conditions, and efficiently handled the shared buffer.