EXP 29: Write a C program to simulate the solution of Classical Process Synchronization Problem

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>  // Include this header for sleep()

#define BUFFER_SIZE 5
#define MAX_ITEMS 10  // Maximum number of items to be produced/consumed

// Shared buffer
int buffer[BUFFER_SIZE];
int in = 0, out = 0;
sem_t empty, full, mutex;

// Global counters
int produced_count = 0;
int consumed_count = 0;

// Producer function
void* producer(void* arg) {
    int item;
    while(produced_count < MAX_ITEMS) {
        // Produce an item
        item = rand() % 100;
        printf("Producer produced: %d\n", item);
```

```
        // Wait for an empty slot in the buffer
        sem_wait(&empty);

        // Lock the buffer for modification
        sem_wait(&mutex);

        // Add item to buffer
        buffer[in] = item;
        in = (in + 1) % BUFFER_SIZE;
        produced_count++;

        // Unlock the buffer
        sem_post(&mutex);

        // Signal that the buffer has a new item
        sem_post(&full);

        // Simulate some processing time
        sleep(1);  // Sleep for 1 second to simulate work
    }
    pthread_exit(NULL);  // Exit the producer thread
}

// Consumer function
void* consumer(void* arg) {
    int item;
    while(consumed_count < MAX_ITEMS) {
        // Wait for a full slot in the buffer
        sem_wait(&full);
```

```
        // Lock the buffer for modification
        sem_wait(&mutex);

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

        // Unlock the buffer
        sem_post(&mutex);

        // Signal that there is an empty slot in the buffer
        sem_post(&empty);

        // Simulate some processing time
        sleep(2);  // Sleep for 2 seconds to simulate work
    }
    pthread_exit(NULL);  // Exit the consumer thread
}

int main() {
    // Initialize semaphores
    sem_init(&empty, 0, BUFFER_SIZE); // Initially all slots are empty
    sem_init(&full, 0, 0);  // Initially no slots are full
    sem_init(&mutex, 0, 1); // Mutex to protect buffer

    pthread_t producer_thread, consumer_thread;
```

```
    // Create producer and consumer threads
    pthread_create(&producer_thread, NULL, producer, NULL);
    pthread_create(&consumer_thread, NULL, consumer, NULL);

    // Wait for threads to finish
    pthread_join(producer_thread, NULL);
    pthread_join(consumer_thread, NULL);

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

    return 0;
}
```

Sample Output