Subject: Design Principles of Operating Systems

Subject code: CSE 3249

Assignment 5: Implementation of synchronization using semaphore:

Objective of this Assignment:

- To implement the concept of multi-threading in a process.
- To learn the use of semaphore i.e., to control access to shared resources.

1. **Producer-Consumer problem**

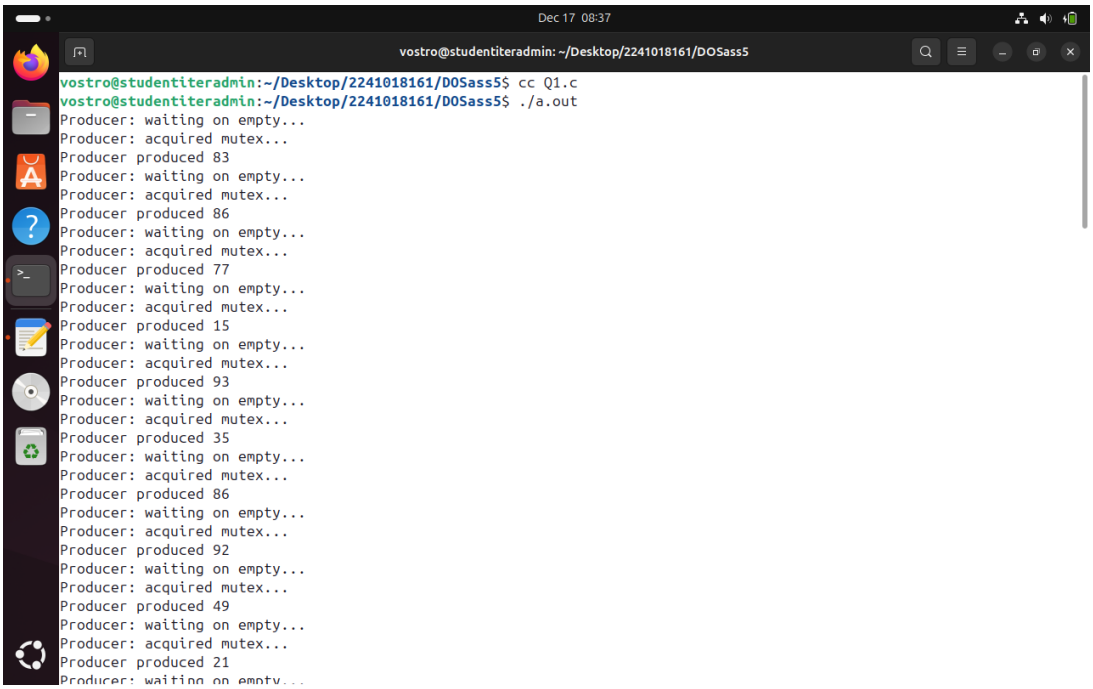
   Problem : Write a C program to implement the producer-consumer program where: • Producer generates integers from 1 to 100.
   - Consumer processes the numbers.

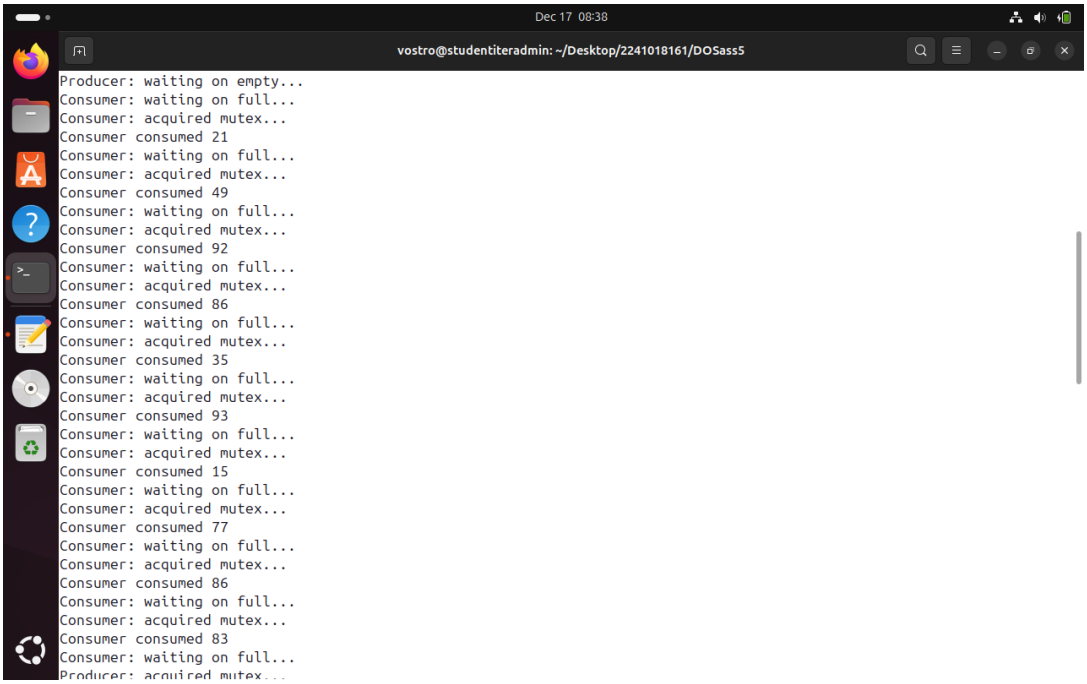
   Requirements:
   - Use a shared buffer with a maximum size of 10.
   - Use semaphores and mutex to ensure thread-safe access to the buffer.
   - Print the number that producer is producing and consumer is consuming.
   - Both producer and consumer will continue for 20 iterations

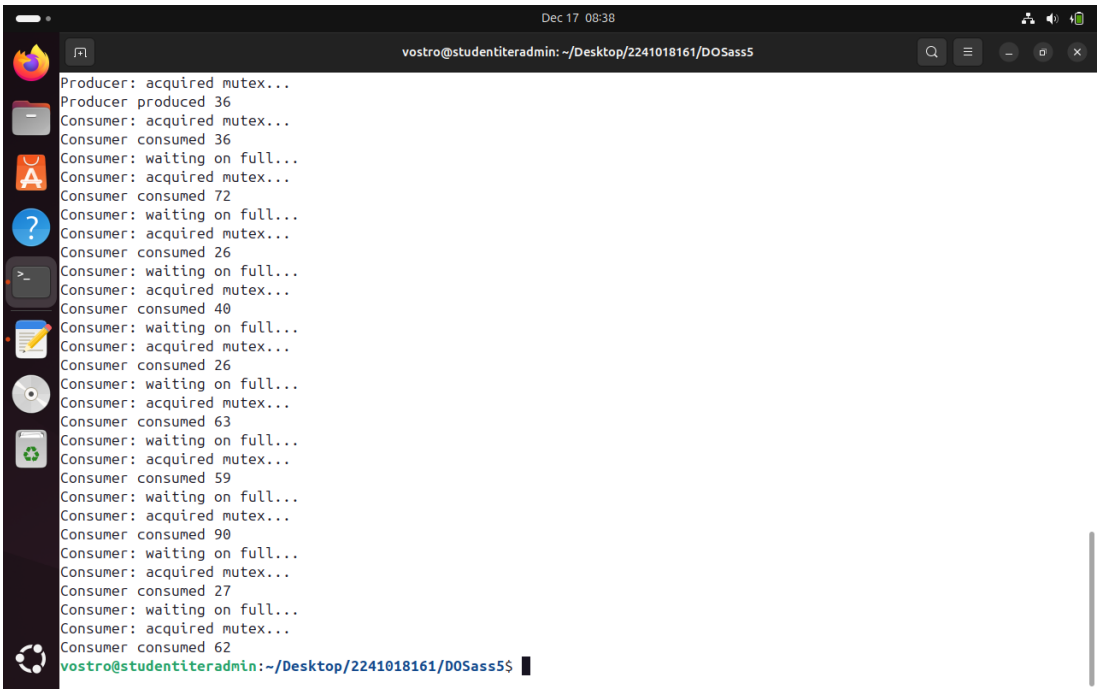
   Program:

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

#define BUFFER_SIZE 10

int buffer[BUFFER_SIZE];
int count = 0;

sem_t empty, full, mutex;

void *producer(void *param) {
    int item;
    for (int i = 0; i < 20; i++) {
        item = rand() % 100; // Produce an item
        printf("Producer: waiting on empty...\n");
        sem_wait(&empty);
        printf("Producer: acquired mutex...\n");
        sem_wait(&mutex);
        buffer[count++] = item; // Add item to the buffer
        printf("Producer produced %d\n", item);
        sem_post(&mutex);
        sem_post(&full);
    }
    pthread_exit(NULL);
}

void *consumer(void *param) {
    int item;
    for (int i = 0; i < 20; i++) {
        printf("Consumer: waiting on full...\n");
        sem_wait(&full);
        printf("Consumer: acquired mutex...\n");
        sem_wait(&mutex);
        item = buffer[--count]; // Remove item from the buffer
        printf("Consumer consumed %d\n", item);
        sem_post(&mutex);
        sem_post(&empty);
    }
    pthread_exit(NULL);
}

int main() {
    pthread_t prod, cons;

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

    // Disable buffering for immediate output
    setvbuf(stdout, NULL, _IONBF, 0);

    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;
}
```

O/P:


Dec 17 08:37
vostro@studentiteradmin: ~/Desktop/2241018161/DOSass5
vostro@studentiteradmin:~/Desktop/2241018161/DOSass5$ cc Q1.c
vostro@studentiteradmin:~/Desktop/2241018161/DOSass5$ ./a.out
Producer: waiting on empty...
Producer: acquired mutex...
Producer produced 83
Producer: waiting on empty...
Producer: acquired mutex...
Producer produced 86
Producer: waiting on empty...
Producer: acquired mutex...
Producer produced 77
Producer: waiting on empty...
Producer: acquired mutex...
Producer produced 15
Producer: waiting on empty...
Producer: acquired mutex...
Producer produced 93
Producer: waiting on empty...
Producer: acquired mutex...
Producer produced 35
Producer: waiting on empty...
Producer: acquired mutex...
Producer produced 86
Producer: waiting on empty...
Producer: acquired mutex...
Producer produced 92
Producer: waiting on empty...
Producer: acquired mutex...
Producer produced 49
Producer: waiting on empty...
Producer: acquired mutex...
Producer produced 21


Dec 17 08:38
vostro@studentiteradmin: ~/Desktop/2241018161/DOSass5
Producer: waiting on empty...
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 21
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 49
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 92
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 86
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 35
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 93
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 15
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 77
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 86
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 83
Consumer: waiting on full...


Dec 17 08:38
vostro@studentiteradmin: ~/Desktop/2241018161/DOSass5
Producer: acquired mutex...
Producer produced 36
Consumer: acquired mutex...
Consumer consumed 36
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 72
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 26
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 40
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 26
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 63
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 59
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 90
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 27
Consumer: waiting on full...
Consumer: acquired mutex...
Consumer consumed 62
vostro@studentiteradmin:~/Desktop/2241018161/DOSass5$

2. Alternating Numbers with Two Threads

Problem: Write a program to print 1, 2, 3 … upto 20. Create threads where two threads print numbers alternately.

- **Thread A** prints odd numbers: 1, 3, 5 ...
- **Thread B** prints even numbers: 2, 4, 6 ...

Requirements:

- Use semaphores to control the order of execution of the threads.
- Ensure no race conditions occur.

Program:

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

#define MAX_NUM 20

// Shared variables to track the state of the printing process
int current_num = 1; // To track which number should be printed next
pthread_mutex_t lock;
pthread_cond_t cond_odd, cond_even;

void* print_odd(void* arg) {
    while (current_num <= MAX_NUM) {
        pthread_mutex_lock(&lock);

        // Wait until it's odd's turn
        if (current_num % 2 != 1) {
            pthread_cond_wait(&cond_odd, &lock);
        }

        // Print the odd number
        if (current_num <= MAX_NUM) {
            printf("%d ", current_num);
            current_num++;
            pthread_cond_signal(&cond_even); // Signal the even thread
        }

        pthread_mutex_unlock(&lock);
    }
    return NULL;
}

void* print_even(void* arg) {
    while (current_num <= MAX_NUM) {
        pthread_mutex_lock(&lock);

        // Wait until it's even's turn
        if (current_num % 2 != 0) {
            pthread_cond_wait(&cond_even, &lock);
        }

        // Print the even number
        if (current_num <= MAX_NUM) {
            printf("%d ", current_num);
            current_num++;
            pthread_cond_signal(&cond_odd); // Signal the odd thread
        }

        pthread_mutex_unlock(&lock);
    }
    return NULL;
}
```

```
int main() {
    pthread_t thread1, thread2;

    // Initialize mutex and condition variables
    pthread_mutex_init(&lock, NULL);
    pthread_cond_init(&cond_odd, NULL);
    pthread_cond_init(&cond_even, NULL);

    // Create two threads
    pthread_create(&thread1, NULL, print_odd, NULL);
    pthread_create(&thread2, NULL, print_even, NULL);

    // Wait for both threads to finish
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    // Cleanup
    pthread_mutex_destroy(&lock);
    pthread_cond_destroy(&cond_odd);
    pthread_cond_destroy(&cond_even);

    printf("\n");
    return 0;
}
```

O/P: