Operating System

LAB – 6

Name : **Om Santosh Gore**

Div : **AI – 'A'**

Roll No. : **56**

PRN : **12110548**

- **Problem Statement :**

  Implementation of Classical problems Producer Consumer using Threads and Semaphore.

- **Code :**

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <time.h> // Include time.h for random number seeding

#define BUFFER_SIZE 5
#define NUM_PRODUCERS 2
#define NUM_CONSUMERS 2

int buffer[BUFFER_SIZE];
int buffer_index = 0;

sem_t empty;
sem_t full;
pthread_mutex_t mutex;

void *producer(void *arg) {
   int producer_id = *((int *)arg);

   while (1) {
      int item = rand() % 1000;

      sem_wait(&empty);
      pthread_mutex_lock(&mutex);
```

```
        buffer[buffer_index] = item;
        buffer_index++;

        pthread_mutex_unlock(&mutex);
        sem_post(&full);

        printf("Producer %d produced item %d\n", producer_id, item);
        sleep(1); // Sleep to simulate variable production time
    }

    return NULL;
}

void *consumer(void *arg) {
    int consumer_id = *((int *)arg);

    while (1) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);

        int item = buffer[buffer_index - 1];
        buffer_index--;

        pthread_mutex_unlock(&mutex);
        sem_post(&empty);

        printf("Consumer %d consumed item %d\n", consumer_id, item);
        sleep(2); // Sleep to simulate variable consumption time
    }

    return NULL;
}

int main() {
    srand(time(NULL)); // Seed the random number generator
    pthread_t producer_threads[NUM_PRODUCERS];
    pthread_t consumer_threads[NUM_CONSUMERS];
    int producer_ids[NUM_PRODUCERS];
    int consumer_ids[NUM_CONSUMERS];

    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    pthread_mutex_init(&mutex, NULL);

    for (int i = 0; i < NUM_PRODUCERS; i++) {
```

```
    producer_ids[i] = i + 1;
    pthread_create(&producer_threads[i], NULL, producer, &producer_ids[i]);
  }

  for (int i = 0; i < NUM_CONSUMERS; i++) {
    consumer_ids[i] = i + 1;
    pthread_create(&consumer_threads[i], NULL, consumer, &consumer_ids[i]);
  }

  for (int i = 0; i < NUM_PRODUCERS; i++) {
    pthread_join(producer_threads[i], NULL);
  }
  for (int i = 0; i < NUM_CONSUMERS; i++) {
    pthread_join(consumer_threads[i], NULL);
  }

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

  return 0;
}
```

- **Output :**