Experiment Number: 7

Problem Statement: **a) Implementation of Classical problem Producer Consumer  using Threads and Semaphore**

**b)Implementation of Classical problem Producer Consumer  using Threads and Mutex**

NAME: Aadesh Chawla ROLLNO: 12

CLASS: TY-IT-A BATCH: B1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Using semaphore:**

#include <iostream>

#include <pthread.h>

#include <semaphore.h>

#include <vector>

using namespace std;

const int BUFFER\_SIZE = 5; // Size of the shared buffer

const int NUM\_PRODUCERS = 2;

const int NUM\_CONSUMERS = 2;

const int NUM\_ITEMS = 10; // Total number of items to produce/consume

vector<int> buffer(BUFFER\_SIZE);

sem\_t empty;  // Semaphore to track empty slots in the buffer

sem\_t full;   // Semaphore to track filled slots in the buffer

pthread\_mutex\_t mutex; // Mutex to protect the buffer

int itemCount = 0; // Counter for the number of items produced

void\* Producer(void\* id) {

    int producerId = \*((int\*)id);

    for (int i = 0; i < NUM\_ITEMS; i++) {

        // Produce item

        int item = rand() % 100;

        // Wait for an empty slot in the buffer

        sem\_wait(&empty);

        pthread\_mutex\_lock(&mutex);

        // Add item to the buffer

        buffer[itemCount] = item;

        itemCount++;

        cout << "Producer " << producerId << " produced item " << item << endl;

        pthread\_mutex\_unlock(&mutex);

        sem\_post(&full);

    }

    pthread\_exit(NULL);

}

void\* Consumer(void\* id) {

    int consumerId = \*((int\*)id);

    for (int i = 0; i < NUM\_ITEMS; i++) {

        // Wait for a filled slot in the buffer

        sem\_wait(&full);

        pthread\_mutex\_lock(&mutex);

        // Remove item from the buffer

        int item = buffer[itemCount - 1];

        itemCount--;

        cout << "Consumer " << consumerId << " consumed item " << item << endl;

        pthread\_mutex\_unlock(&mutex);

        sem\_post(&empty);

    }

    pthread\_exit(NULL);

}

int main() {

    srand(time(NULL));

    pthread\_t producers[NUM\_PRODUCERS];

    pthread\_t consumers[NUM\_CONSUMERS];

    sem\_init(&empty, 0, BUFFER\_SIZE);

    sem\_init(&full, 0, 0);

    pthread\_mutex\_init(&mutex, NULL);

    int producerIds[NUM\_PRODUCERS];

    int consumerIds[NUM\_CONSUMERS];

    for (int i = 0; i < NUM\_PRODUCERS; i++) {

        producerIds[i] = i + 1;

        pthread\_create(&producers[i], NULL, Producer, &producerIds[i]);

    }

    for (int i = 0; i < NUM\_CONSUMERS; i++) {

        consumerIds[i] = i + 1;

        pthread\_create(&consumers[i], NULL, Consumer, &consumerIds[i]);

    }

    for (int i = 0; i < NUM\_PRODUCERS; i++) {

        pthread\_join(producers[i], NULL);

    }

    for (int i = 0; i < NUM\_CONSUMERS; i++) {

        pthread\_join(consumers[i], NULL);

    }

    sem\_destroy(&empty);

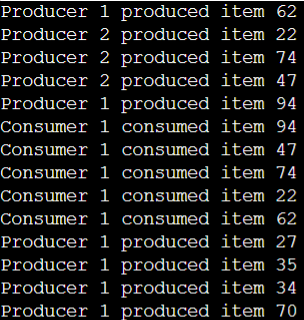
    sem\_destroy(&full);

    pthread\_mutex\_destroy(&mutex);

    return 0;

}

**Output:**



**Using Mutex:**

#include <iostream>

#include <pthread.h>

#include <queue>

using namespace std;

const int BUFFER\_SIZE = 5;

const int NUM\_PRODUCERS = 2;

const int NUM\_CONSUMERS = 2;

const int NUM\_ITEMS = 10;

queue<int> buffer;

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_cond\_t buffer\_empty = PTHREAD\_COND\_INITIALIZER;

pthread\_cond\_t buffer\_full = PTHREAD\_COND\_INITIALIZER;

void\* Producer(void\* producer\_id) {

    int producer\_num = \*((int\*)producer\_id);

    for (int i = 0; i < NUM\_ITEMS; ++i) {

        int item = i + 1;

        pthread\_mutex\_lock(&mutex);

        while (buffer.size() == BUFFER\_SIZE) {

            pthread\_cond\_wait(&buffer\_empty, &mutex);

        }

        cout << "Producer " << producer\_num << " produces item " << item << endl;

        buffer.push(item);

        pthread\_cond\_signal(&buffer\_full);

        pthread\_mutex\_unlock(&mutex);

    }

    return NULL;

}

void\* Consumer(void\* consumer\_id) {

    int consumer\_num = \*((int\*)consumer\_id);

    for (int i = 0; i < NUM\_ITEMS; ++i) {

        pthread\_mutex\_lock(&mutex);

        while (buffer.empty()) {

            pthread\_cond\_wait(&buffer\_full, &mutex);

        }

        int item = buffer.front();

        buffer.pop();

        cout << "Consumer " << consumer\_num << " consumes item " << item << endl;

        pthread\_cond\_signal(&buffer\_empty);

        pthread\_mutex\_unlock(&mutex);

    }

    return NULL;

}

int main() {

    pthread\_t producers[NUM\_PRODUCERS];

    pthread\_t consumers[NUM\_CONSUMERS];

    int producer\_ids[NUM\_PRODUCERS];

    int consumer\_ids[NUM\_CONSUMERS];

    for (int i = 0; i < NUM\_PRODUCERS; ++i) {

        producer\_ids[i] = i + 1;

        pthread\_create(&producers[i], NULL, Producer, &producer\_ids[i]);

    }

    for (int i = 0; i < NUM\_CONSUMERS; ++i) {

        consumer\_ids[i] = i + 1;

        pthread\_create(&consumers[i], NULL, Consumer, &consumer\_ids[i]);

    }

    for (int i = 0; i < NUM\_PRODUCERS; ++i) {

        pthread\_join(producers[i], NULL);

    }

    for (int i = 0; i < NUM\_CONSUMERS; ++i) {

        pthread\_join(consumers[i], NULL);

    }

    return 0;

}

**Output:**

