**Name : Ruturaj Sandip Sutar Roll No : 59 Div : B Batch : 2**

1. **Producer Consumer Problem**

**Code :**

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <pthread.h>*

*#include <unistd.h>*

*#define BUFFER\_SIZE 10*

*int buffer[BUFFER\_SIZE];*

*int count = 0;*

*pthread\_mutex\_t mutex;*

*pthread\_cond\_t cond\_producer;*

*pthread\_cond\_t cond\_consumer;*

*// Number of items each producer and consumer will handle*

*#define PROCESS\_COUNT 5*

*void \*producer(void \*arg) {*

*int i;*

*for (i = 0; i < PROCESS\_COUNT; i++) {*

*pthread\_mutex\_lock(&mutex);*

*// Wait if the buffer is full*

*while (count == BUFFER\_SIZE) {*

*pthread\_cond\_wait(&cond\_producer, &mutex);*

*}*

*// Produce an item and add it to the buffer*

*buffer[count++] = i;*

*printf("Produced: %d\n", i);*

*// Signal the consumer that an item is available*

*pthread\_cond\_signal(&cond\_consumer);*

*pthread\_mutex\_unlock(&mutex);*

*sleep(1);*

*}*

*return NULL;*

*}*

*void \*consumer(void \*arg) {*

*int i;*

*for (i = 0; i < PROCESS\_COUNT; i++) {*

*pthread\_mutex\_lock(&mutex);*

*// Wait if the buffer is empty*

*while (count == 0) {*

*pthread\_cond\_wait(&cond\_consumer, &mutex);*

*}*

*// Consume an item from the buffer*

*int item = buffer[--count];*

*printf("Consumed: %d\n", item);*

*// Signal the producer that space is available*

*pthread\_cond\_signal(&cond\_producer);*

*pthread\_mutex\_unlock(&mutex);*

*sleep(1);*

*}*

*return NULL;*

*}*

*int main() {*

*pthread\_t prod, cons;*

*// Initialize mutex and condition variables*

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

*pthread\_cond\_init(&cond\_producer, NULL);*

*pthread\_cond\_init(&cond\_consumer, NULL);*

*// Create producer and consumer threads*

*pthread\_create(&prod, NULL, producer, NULL);*

*pthread\_create(&cons, NULL, consumer, NULL);*

*// Wait for both threads to finish*

*pthread\_join(prod, NULL);*

*pthread\_join(cons, NULL);*

*// Destroy mutex and condition variables*

*pthread\_mutex\_destroy(&mutex);*

*pthread\_cond\_destroy(&cond\_producer);*

*pthread\_cond\_destroy(&cond\_consumer);*

*return 0;*

*}*

**Output :**

Produced: 0

Consumed: 0

Produced: 1

Consumed: 1

Produced: 2

Consumed: 2

Produced: 3

Consumed: 3

Produced: 4

Consumed: 4

1. **Reader Writer Problem**

**Code :**

*#include* <stdio.h>

*#include* <pthread.h>

*#include* <semaphore.h>

*#include* <unistd.h>

sem\_t wrt;

pthread\_mutex\_t mutex;

int shared\_data = 0;

int reader\_count = 0;

void \*writer(void \**arg*) {

    int writer\_id = \*((int \*)*arg*);

    sem\_wait(&wrt);

    shared\_data++;

    printf("Writer %d wrote data: %d\n", writer\_id, shared\_data);

    sem\_post(&wrt);

*return* NULL;

}

void \*reader(void \**arg*) {

    int reader\_id = \*((int \*)*arg*);

    pthread\_mutex\_lock(&mutex);

    reader\_count++;

*if* (reader\_count == 1) {

        sem\_wait(&wrt);

    }

    pthread\_mutex\_unlock(&mutex);

    printf("Reader %d read data: %d\n", reader\_id, shared\_data);

    pthread\_mutex\_lock(&mutex);

*if* (reader\_count == 0) {

        sem\_post(&wrt);

    }

    pthread\_mutex\_unlock(&mutex);

*return* NULL;

}

int main() {

    pthread\_t rtid[5], wtid[5];

    int ids[5] = {1, 2, 3, 4, 5};

    sem\_init(&wrt, 0, 1);

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

*for* (int i = 0; i < 5; i++) {

        pthread\_create(&rtid[i], NULL, reader, &ids[i]);

    }

*for* (int i = 0; i < 5; i++) {

        pthread\_create(&wtid[i], NULL, writer, &ids[i]);

    }

*for* (int i = 0; i < 5; i++) {

        pthread\_join(rtid[i], NULL);

        pthread\_join(wtid[i], NULL);

    }

    sem\_destroy(&wrt);

    pthread\_mutex\_destroy(&mutex);

*return* 0;

}

**Output :**

Reader 1 read data: 0

Reader 2 read data: 0

Reader 3 read data: 0

Reader 4 read data: 0

Reader 5 read data: 0

1. **Dining philosophers problem**

**Code :**

*#include* <iostream>

*#include* <thread>

*#include* <mutex>

*#include* <condition\_variable>

*#define* N 5

*#define* THINKING 2

*#define* HUNGRY 1

*#define* EATING 0

*#define* LEFT (*phnum* + 4) % N

*#define* RIGHT (*phnum* + 1) % N

int state[N];

int phil[N] = { 0, 1, 2, 3, 4 };

std::mutex mutex;

std::condition\_variable S[N];

void test(int *phnum*)

{

*if* (state[*phnum*] == HUNGRY

        && state[LEFT] != EATING

        && state[RIGHT] != EATING) {

*// state that eating*

        state[*phnum*] = EATING;

        std::this\_thread::sleep\_for(std::chrono::milliseconds(2000));

        std::cout << "Philosopher " << *phnum* + 1 << " takes fork " << LEFT + 1 << " and " << *phnum* + 1 << std::endl;

        std::cout << "Philosopher " << *phnum* + 1 << " is Eating" << std::endl;

        S[*phnum*].notify\_all();

    }

}

void take\_fork(int *phnum*)

{

    std::unique\_lock<std::mutex> lock(mutex);

    state[*phnum*] = HUNGRY;

    std::cout << "Philosopher " << *phnum* + 1 << " is Hungry" << std::endl;

    test(*phnum*);

    S[*phnum*].wait(lock);

    std::this\_thread::sleep\_for(std::chrono::milliseconds(1000));

}

void put\_fork(int *phnum*)

{

    std::unique\_lock<std::mutex> lock(mutex);

    state[*phnum*] = THINKING;

    std::cout << "Philosopher " << *phnum* + 1 << " putting fork " << LEFT + 1 << " and " << *phnum* + 1 << " down" << std::endl;

    std::cout << "Philosopher " << *phnum* + 1 << " is thinking" << std::endl;

    test(LEFT);

    test(RIGHT);

}

void philosopher(int *num*)

{

*while* (true) {

        take\_fork(*num*);

        put\_fork(*num*);

    }

}

int main()

{

    std::thread threads[N];

*for* (int i = 0; i < N; i++) {

        threads[i] = std::thread(philosopher, i);

        std::cout << "Philosopher " << i + 1 << " is thinking" << std::endl;

    }

*for* (int i = 0; i < N; i++)

        threads[i].join();

*return* 0;

}

**Output :-**

Philosopher 1 is thinking

Philosopher 2 is thinking

Philosopher 1 is Hungry

Philosopher 3 is thinking

Philosopher 2 is Hungry

Philosopher 4 is thinking

Philosopher 3 is Hungry

Philosopher 4 is Hungry

Philosopher 5 is thinking

Philosopher 5 is Hungry

Philosopher 5 takes fork 4 and 5

Philosopher 5 is Eating