**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

## 2. 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

## 3. 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