Walchand College of Engineering, Sangli

Department of Computer Science and Engineering

**Class:** Final Year (Computer Science and Engineering)

**Year:** 2023-24 **Semester:** 1

**Course:** High Performance Computing Lab

**Practical No. 4**

**Exam Seat No: 2020BTECS00021**

**Title of practical:**

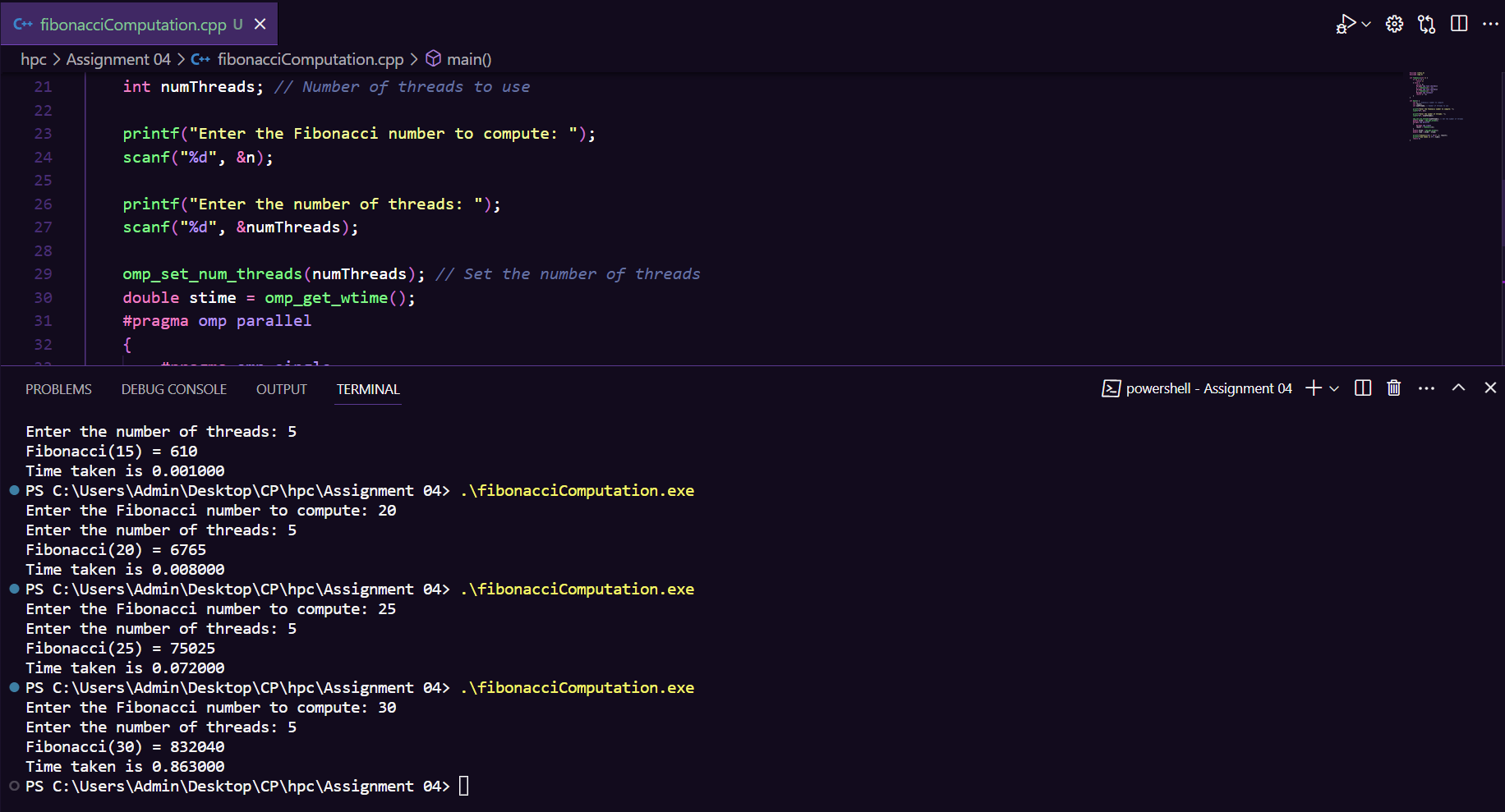
Study and Implementation of Synchronization

**Problem Statement 1:**

# Analyse and implement a Parallel code for below programs using OpenMP considering synchronization requirements. (Demonstrate the use of different clauses and constructs wherever applicable)

# Fibonacci Computation:

**Screenshots:**

****

**Information:**

#include <stdio.h>

#include <omp.h>

int fibonacci(int *n*) {

    if (*n* <= 1) {

        return *n*;

    } else {

        int x, y;

        #pragma omp task shared(x)

        x = fibonacci(*n* - 1);

        #pragma omp task shared(y)

        y = fibonacci(*n* - 2);

        #pragma omp taskwait

        return x + y;

    }

}

int main() {

    int n; *// Fibonacci number to compute*

    int result;

    int numThreads; *// Number of threads to use*

    printf("Enter the Fibonacci number to compute: ");

    scanf("%d", &n);

    printf("Enter the number of threads: ");

    scanf("%d", &numThreads);

    omp\_set\_num\_threads(numThreads); *// Set the number of threads*

    double stime = omp\_get\_wtime();

    #pragma omp parallel

    {

        #pragma omp single

        result = fibonacci(n);

    }

    double etime = omp\_get\_wtime();

    double time = etime - stime;

    printf("Fibonacci(%d) = %d\n", n, result);

    printf("Time taken is %f", time);

    return 0;

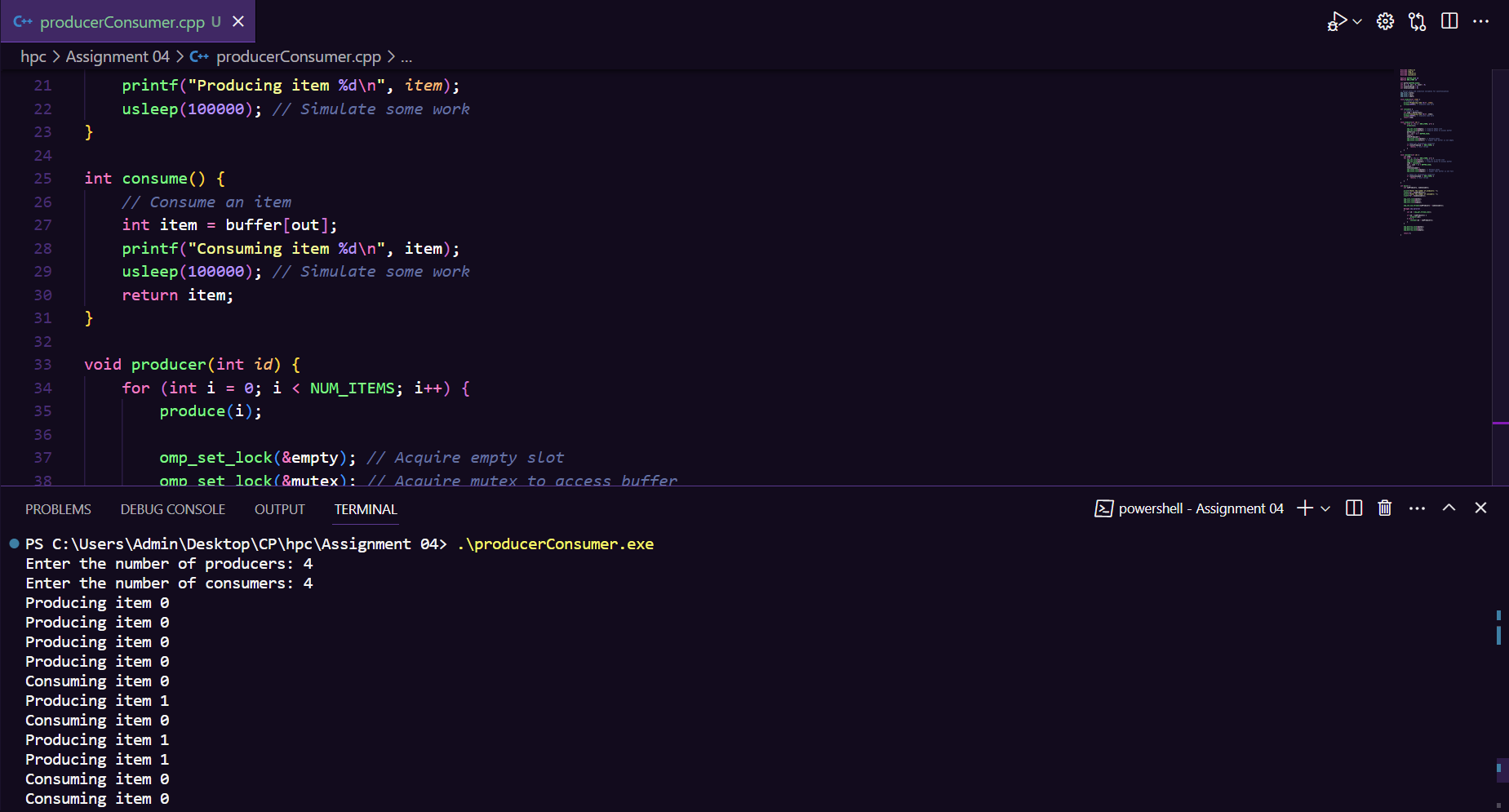
}

**Problem Statement 2:**

# Analyse and implement a Parallel code for below programs using OpenMP considering synchronization requirements. (Demonstrate the use of different clauses and constructs wherever applicable)

## Producer Consumer Problem

**Screenshots:**

****

**Information:**

#include <stdio.h>

#include <omp.h>

#include <stdlib.h>

#include <unistd.h>

#define BUFFER\_SIZE 10

#define NUM\_ITEMS 20

int buffer[BUFFER\_SIZE];

int in = 0, out = 0, count = 0;

int totalProduced = 0;

int totalConsumed = 0;

*// Define mutex and condition variables for synchronization*

omp\_lock\_t mutex;

omp\_lock\_t full;

omp\_lock\_t empty;

void produce(int *item*) {

*// Produce an item*

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

    usleep(100000); *// Simulate some work*

}

int consume() {

*// Consume an item*

    int item = buffer[out];

    printf("Consuming item %d\n", item);

    usleep(100000); *// Simulate some work*

    return item;

}

void producer(int *id*) {

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

        produce(i);

        omp\_set\_lock(&empty); *// Acquire empty slot*

        omp\_set\_lock(&mutex); *// Acquire mutex to access buffer*

        buffer[in] = i;

        in = (in + 1) % BUFFER\_SIZE;

        count++;

        totalProduced++;

        omp\_unset\_lock(&mutex); *// Release mutex*

        omp\_unset\_lock(&full); *// Signal that buffer is not empty*

*// Check for termination condition*

        if (totalProduced >= NUM\_ITEMS) {

            return; *// Exit thread*

        }

    }

}

void consumer(int *id*) {

    int item;

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

        omp\_set\_lock(&full); *// Wait for a filled slot*

        omp\_set\_lock(&mutex); *// Acquire mutex to access buffer*

        item = consume();

        out = (out + 1) % BUFFER\_SIZE;

        count--;

        totalConsumed++;

        omp\_unset\_lock(&mutex); *// Release mutex*

        omp\_unset\_lock(&empty); *// Signal that buffer is not full*

*// Check for termination condition*

        if (totalConsumed >= NUM\_ITEMS) {

            return; *// Exit thread*

        }

    }

}

int main() {

    int numProducers, numConsumers;

    printf("Enter the number of producers: ");

    scanf("%d", &numProducers);

    printf("Enter the number of consumers: ");

    scanf("%d", &numConsumers);

    omp\_init\_lock(&mutex);

    omp\_init\_lock(&full);

    omp\_init\_lock(&empty);

    omp\_set\_num\_threads(numProducers + numConsumers);

    #pragma omp parallel

    {

        int id = omp\_get\_thread\_num();

        if (id < numProducers) {

            producer(id);

        } else {

            consumer(id - numProducers);

        }

    }

    omp\_destroy\_lock(&mutex);

    omp\_destroy\_lock(&full);

    omp\_destroy\_lock(&empty);

    return 0;

}

**Github Link:**

<https://github.com/rohanChavan21/HPC-Assignments>