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

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

**Course:** High Performance Computing Lab

**Practical No. 4**

**Exam Seat No: 2020BTECS00037**

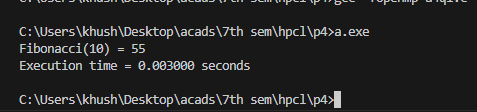
**Title of practical:**

Study and Implementation of Synchronization

**Problem Statement 1:**

# Fibonacci Computation:

**Screenshots:**

****

**Information:**

**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)

#include <stdio.h>

#include <omp.h>

long long fib(int n) {

    if (n <= 1) {

        return n;

    } else {

        long long x, y;

        #pragma omp task shared(x)

        x = fib(n - 1);

        #pragma omp task shared(y)

        y = fib(n - 2);

        #pragma omp taskwait

        return x + y;

    }

}

int main() {

    int n = 10; // Fibonacci number to compute

    long long result;

    double start\_time, end\_time;

    start\_time = omp\_get\_wtime();

    #pragma omp parallel

    #pragma omp single

    {

        result = fib(n);

    }

    end\_time = omp\_get\_wtime();

    double execution\_time = end\_time - start\_time;

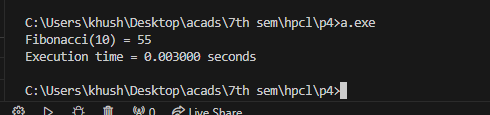
    printf("Fibonacci(%d) = %lld\n", n, result);

    printf("Execution time = %lf seconds\n", execution\_time);

    return 0;

}

**Screenshots:**

****

**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

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#define BUFFER\_SIZE 10

int buffer[BUFFER\_SIZE];

int count = 0; // Number of items in the buffer

int in = 0;    // Index for adding items to the buffer

int out = 0;   // Index for removing items from the buffer

void producer() {

    // Produce 20 items

    for (int i = 0; i < 20; i++) {

        while (count == BUFFER\_SIZE) {

#pragma omp flush(count)

        }

        buffer[in] = i;

        in = (in + 1) % BUFFER\_SIZE;

#pragma omp atomic

        count++;

        printf("Produced: %d\n", i + 1);

    }

}

void consumer() {

    // Consume 20 items

    for (int i = 0; i < 20; i++) {

        while (count == 0) {

#pragma omp flush(count)

        }

        int item = buffer[out];

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

#pragma omp atomic

        count--;

        printf("Consumed: %d\n", item + 1);

    }

}

int main() {

#pragma omp parallel sections

    {

#pragma omp section

        {

            producer();

        }

#pragma omp section

        {

            consumer();

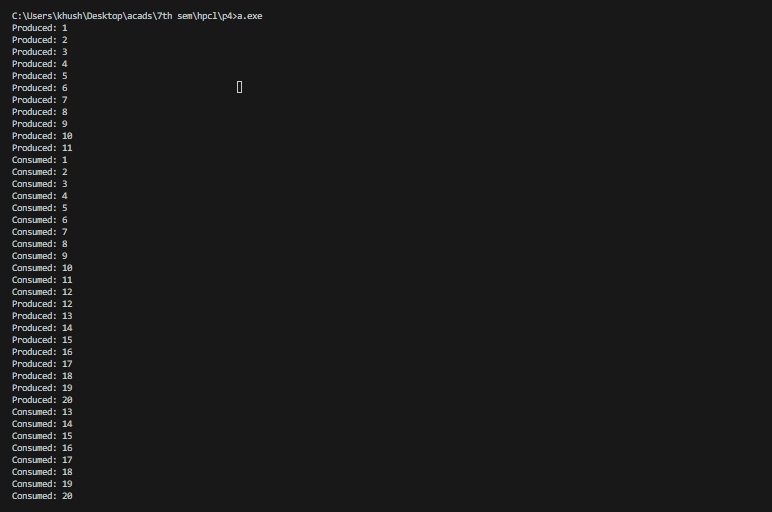
        }

    }

    return 0;

}

**Screenshots**

****