**Class:** Final Year B.Tech(Computer Science and Engineering)

**Year:** 2025-26 **Semester:** 1

**Course:** High Performance Computing Lab

PRN: 22510070

Name:Suyash Yadav

**Practical No. 4**

**Exam Seat No:22510070**

**Title of practical:**

**Study and Implementation of Synchronization**

**Problem Statement 1:**

**Analyze 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:**

#include <stdio.h>

#include <omp.h>

// Recursive Fibonacci with OpenMP tasks

long long fibonacci(int n) {

    if (n < 2)

        return n;

    long long 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;

    printf("Enter N: ");

    scanf("%d", &n);

    long long fib[n];

    double start = omp\_get\_wtime();

    // Start parallel region

    #pragma omp parallel

    {

        #pragma omp single

        {

            for (int i = 0; i < n; i++) {

                fib[i] = fibonacci(i);

            }

        }

    }

    double end = omp\_get\_wtime();

    // Print series

    printf("Fibonacci series up to %d terms:\n", n);

    for (int i = 0; i < n; i++) {

        printf("%lld ", fib[i]);

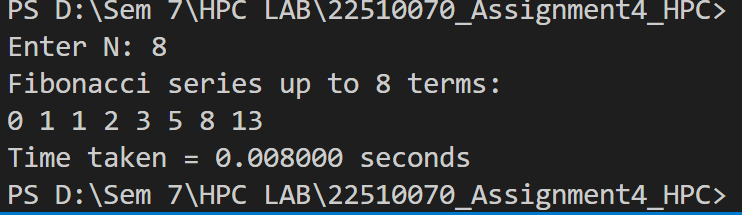
    }

    printf("\n");

    printf("Time taken = %f seconds\n", end - start);

    return 0;

}

****

**Information:**

**Fibonacci Problem: Demonstrates loop-based parallel computation with synchronization (critical) to compute Fibonacci series safely.**

**Problem Statement 2:**

**Analyze 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:**

#include <stdio.h>

#include <omp.h>

#define N 5   // number of items to produce/consume

int main() {

    int buffer[N];       // shared buffer

    int count = 0;       // number of items in buffer

    int produced = 0;    // index for producer

    int consumed = 0;    // index for consumer

    #pragma omp parallel sections shared(buffer, count, produced, consumed)

    {

        // Producer section

        #pragma omp section

        {

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

                #pragma omp critical

                {

                    buffer[produced] = i + 1;  // produce item

                    printf("Producer produced: %d\n", buffer[produced]);

                    produced++;

                    count++;

                }

                #pragma omp flush(count)  // make count visible to other threads

            }

        }

        // Consumer section

        #pragma omp section

        {

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

                int item;

                // wait until something is produced

                while (count <= 0) {

                    #pragma omp flush(count)

                }

                #pragma omp critical

                {

                    item = buffer[consumed];  // consume item

                    printf("Consumer consumed: %d\n", item);

                    consumed++;

                    count--;

                }

                #pragma omp flush(count)

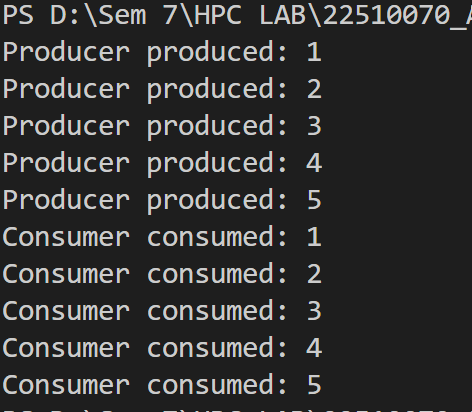
            }

        }

    }

    return 0;

}

****

**Information:**

* **Uses parallel sections → Producer in one section, Consumer in another.**
* **Uses critical to ensure only one thread modifies shared data at a time.**
* **Uses flush for memory synchronization (instead of barrier).**
* **Producer–Consumer Problem: Demonstrates task parallelism (parallel sections) with synchronization (critical, barrier) to safely manage a shared buffer.**

**Github Link: https://github.com/Suyashyadav07/HPC**