**Course: High Performance Computing Lab**

**Practical No. 4**

**PRN: 22510078**

**Name: Piyush Krishna Jadhav**

**Batch: B5**

**Exam Seat No:**

**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:

**Code:**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <omp.h>**

**long long fib\_task(int n) {**

**if (n < 2) return n;**

**long long x = 0, y = 0;**

***// Create tasks for the two recursive calls.***

***// Only create tasks when n is large enough to justify the overhead.***

***// Otherwise do sequential recursion.***

**const int THRESHOLD = 20; *// tune as needed***

**if (n > THRESHOLD) {**

**#pragma omp task shared(x) firstprivate(n)**

**x = fib\_task(n - 1);**

**#pragma omp task shared(y) firstprivate(n)**

**y = fib\_task(n - 2);**

**#pragma omp taskwait**

**} else {**

***// sequential recursion for small n***

**x = fib\_task(n - 1);**

**y = fib\_task(n - 2);**

**}**

**return x + y;**

**}**

**int main(int argc, char \*argv[]) {**

**int n = 40;**

**if (argc >= 2) n = atoi(argv[1]);**

**double t0 = omp\_get\_wtime();**

**long long result;**

***// Start a parallel region; one thread triggers the single region that***

***// creates the initial tasks. Other threads help execute tasks.***

**#pragma omp parallel**

**{**

**#pragma omp single**

**{**

***// The single thread initiates the task-based recursion***

**result = fib\_task(n);**

**}**

**}**

**double t1 = omp\_get\_wtime();**

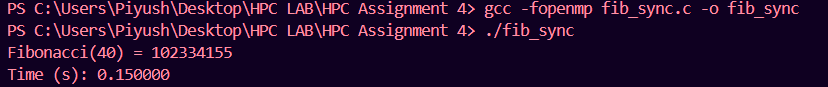
**printf("Fibonacci(%d) = %lld\n", n, result);**

**printf("Time (s): %f\n", t1 - t0);**

**return 0;**

**}**

**Screenshots:**

****

**Information:**

This program computes the Nth Fibonacci number using parallel computation with OpenMP.

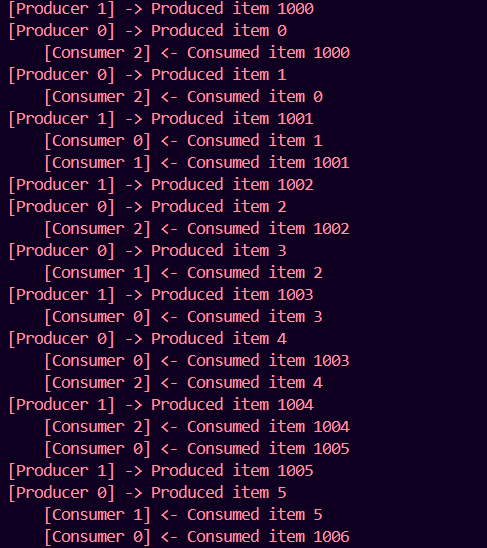
* The user inputs the value of N.
* The program uses OpenMP tasks to recursively calculate Fib(N) = Fib(N-1) + Fib(N-2).
* It demonstrates parallel recursion and measures the execution time.
* Example: For N = 40, the output is Fib(40) = 102334155.
* The program highlights the performance benefits of multithreading in solving recursive problems.

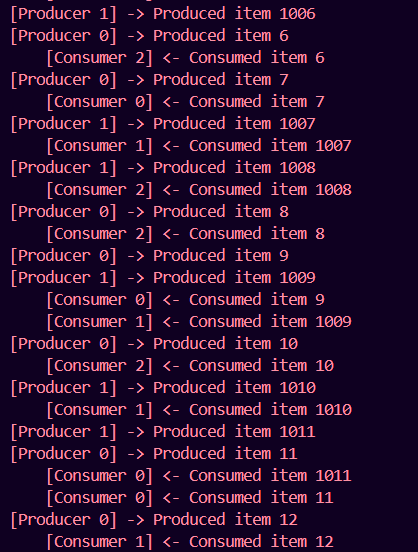
**Problem Statement 2:**

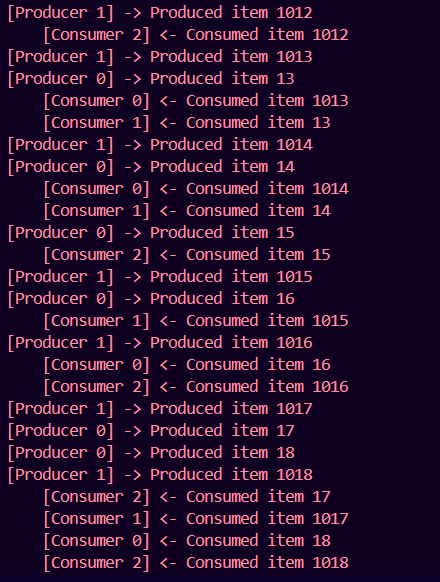
# Analyze and implement a Parallel code for below programs using OpenMP considering synchronization requirements. (Demonstrate the use of different clauses and constructs wherever applicable)

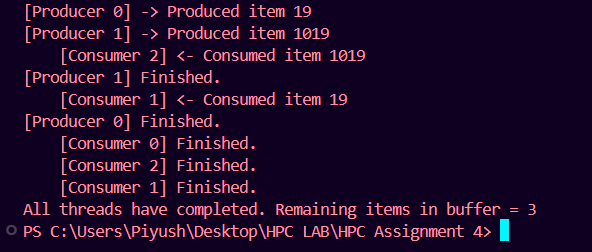
## Producer Consumer Problem

**Screenshots:**

****

****

****

****

**Information:**

This program demonstrates the Producer-Consumer Problem using OpenMP in C.

* The producer generates data items and places them into a shared buffer.
* The consumer removes items from the buffer and processes them.
* A fixed-size circular buffer is used to store items temporarily.
* Critical sections (#pragma omp critical) are applied to ensure synchronization and prevent race conditions when multiple threads access the shared buffer.
* Both producer and consumer threads execute in parallel, simulating real-world concurrent systems where synchronization is crucial.
* This example demonstrates the importance of synchronization constructs in OpenMP for safe multi-threaded execution.

**Github Link:**