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

**Year:** 2024-25 **Semester:** 1

**Course:** High Performance Computing Lab

**Practical No. 4**

**Exam Seat No: 21510055**

**Title of practical:**

Study and Implementation of Synchronization

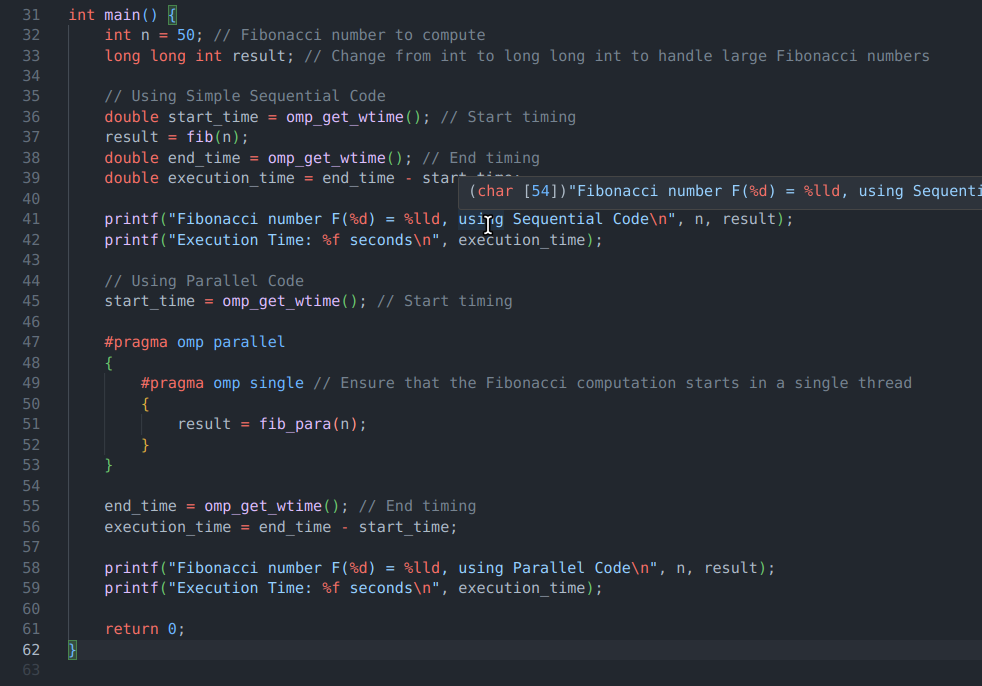
**Problem Statement 1:**

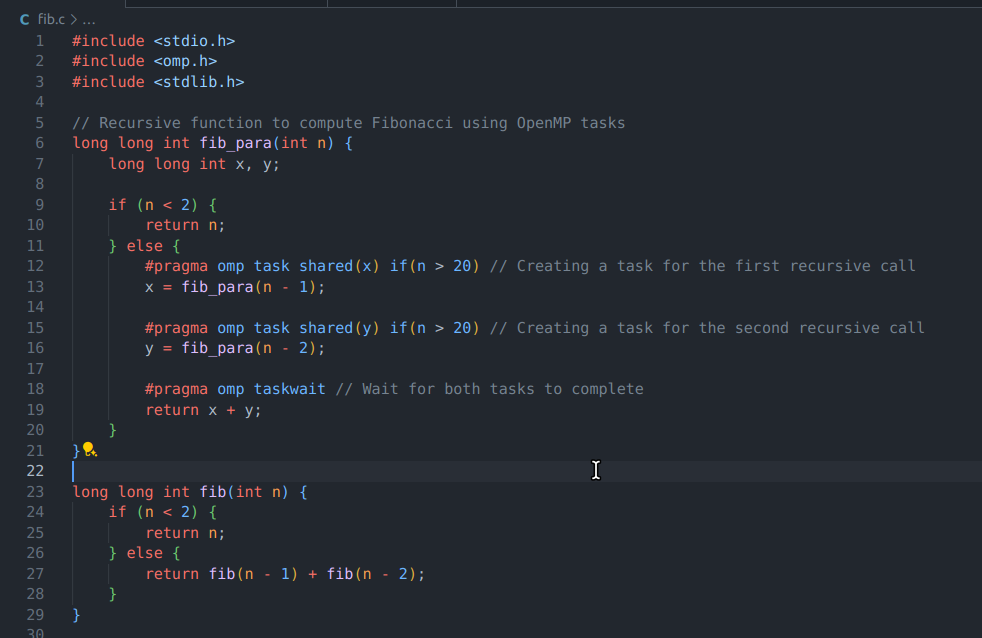
# Analyse and implement a Parallel code for below programs using OpenMP considering synchronization requirements. (Demonstrate the use of different clauses and constructs wherever applicable)

# Fibonacci Computation:

**Screenshots:**

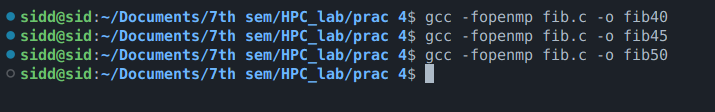
**Code :**

****

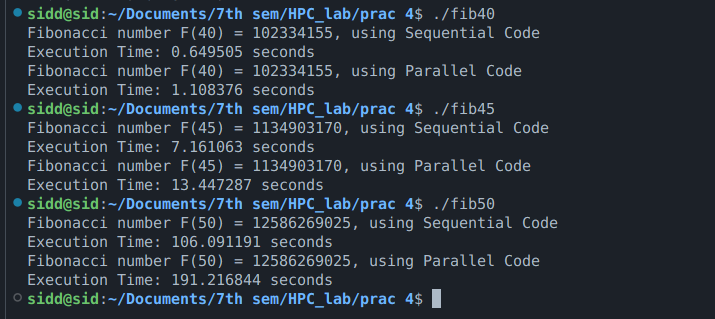
****

**Output :**

**1.**

****

**2.**



**Information:**

#pragma omp task:

This directive creates a task that can be executed in parallel with other tasks. In the fib\_para function, two tasks are created to compute fib(n-1) and fib(n-2) in parallel.

#pragma omp taskwait:

This directive ensures that the parent task (the function call) waits until the two tasks (x = fib\_para(n - 1) and y = fib\_para(n - 2)) have completed before proceeding to the next step.

if(n > 20):

The if clause with #pragma omp task ensures that tasks are only created for larger Fibonacci numbers (n > 20). For smaller numbers, the overhead of task creation may outweigh the benefits of parallelism, so the computation is done sequentially.

Still for 40th, 45th and 50th Fibonacci number OverHead of Synchronisation outweighs the efficiency of parallel task implementation as we have to wait till task on both threads are finished.

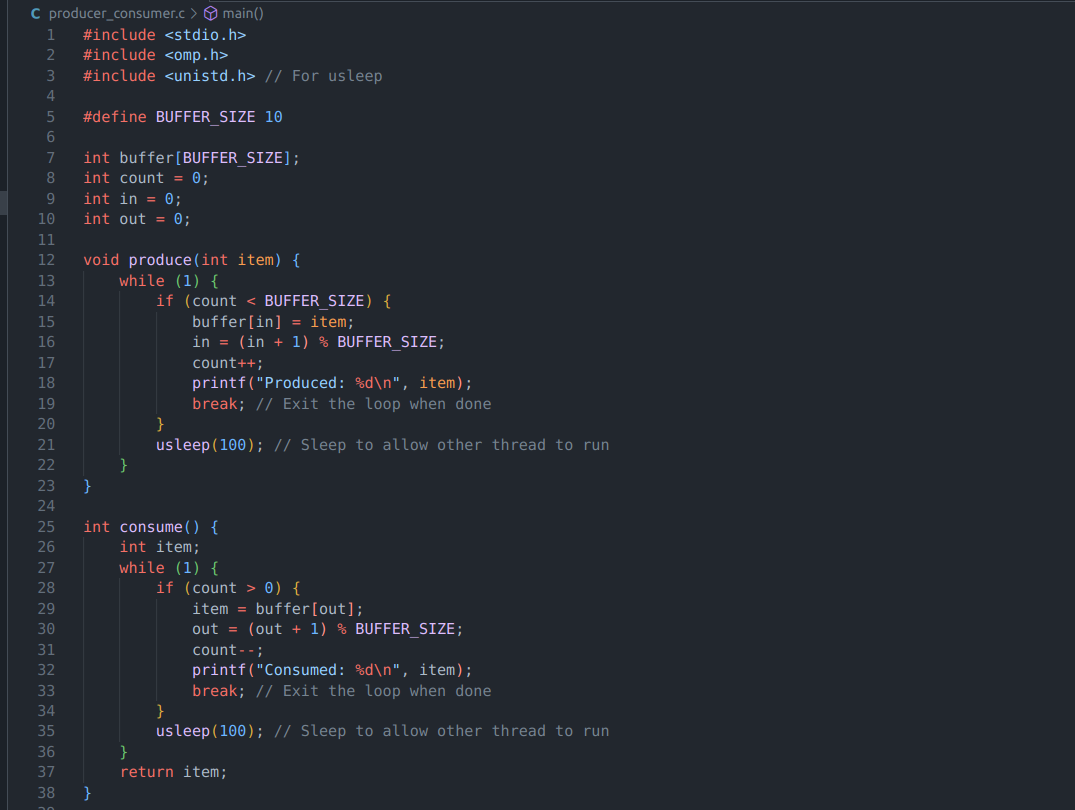
**Problem Statement 2:**

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

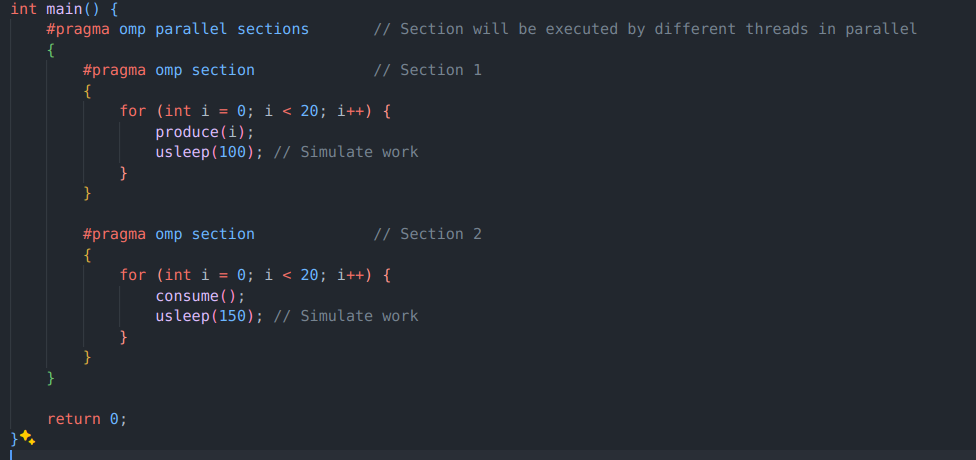
## Producer Consumer Problem

**Screenshots:**

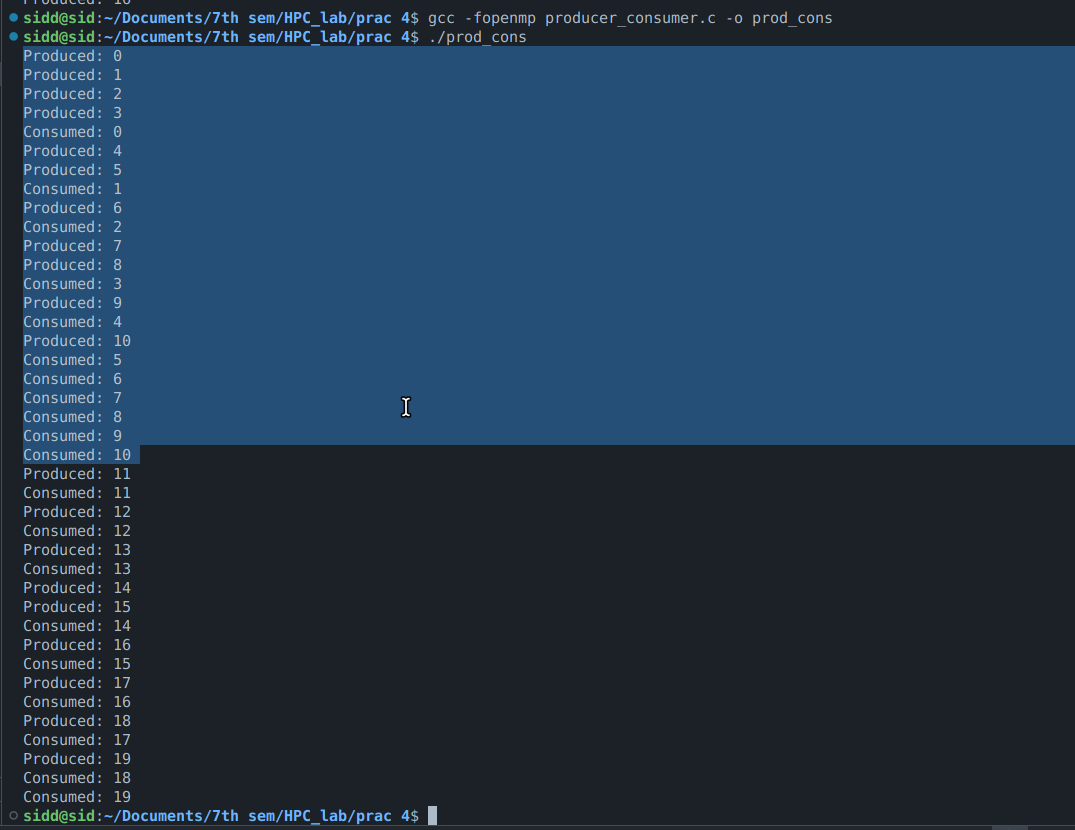
**Code :**

****

**2.**

****

**Output :**

****

**Information:**

Shared Variables:

* buffer, count, in, and out are shared among the producer and consumer threads.

produce Function:

* The producer checks if there is space in the buffer (count < BUFFER\_SIZE). If there is space, it produces an item and updates the in index and count.
* If the buffer is full, the producer sleeps for a short time (usleep(100)) to avoid busy waiting, allowing the consumer to run.

consume Function:

* The consumer checks if there is an item to consume (count > 0). If there is an item, it consumes it and updates the out index and count.
* If the buffer is empty, the consumer sleeps for a short time (usleep(100)) to avoid busy waiting, allowing the producer to run.

Parallel Sections:

* #pragma omp parallel sections is used to run the producer and consumer in parallel sections.
* The producer and consumer each perform their operations in separate sections, and usleep introduces delays to simulate concurrent execution.

**Github Link:** <https://github.com/Sid-1164/HPC_lab/tree/main/prac%204>