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

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

**Course:** High Performance Computing Lab

**Practical No. 4**

**Exam Seat No: 22520007**

**Title of practical:**

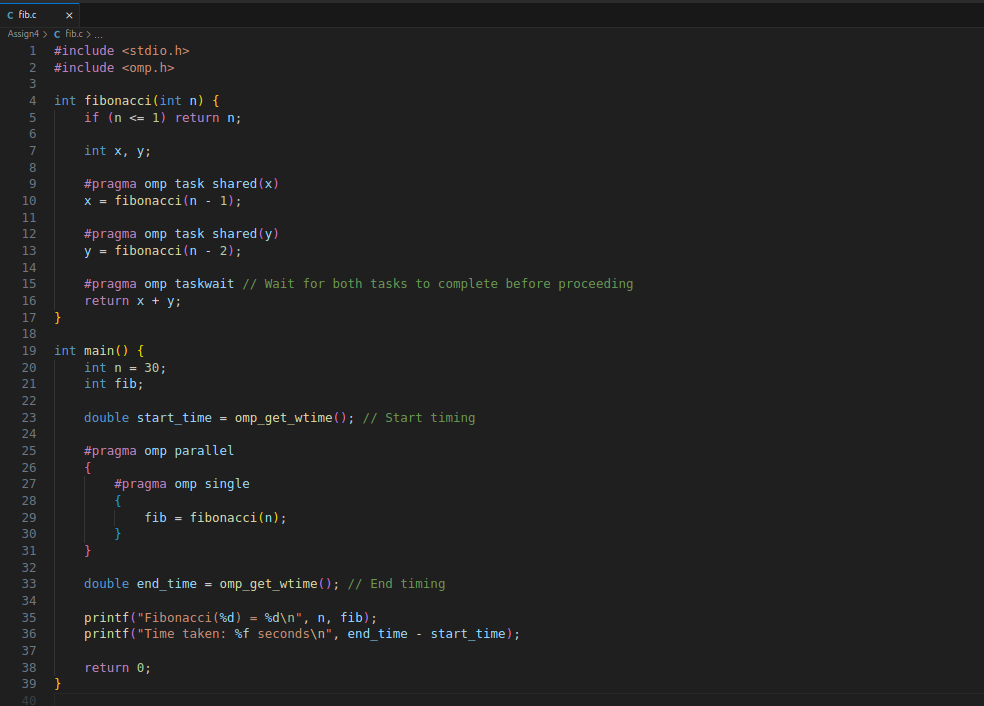
Study and Implementation of Synchronization

**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:**



### 

### 

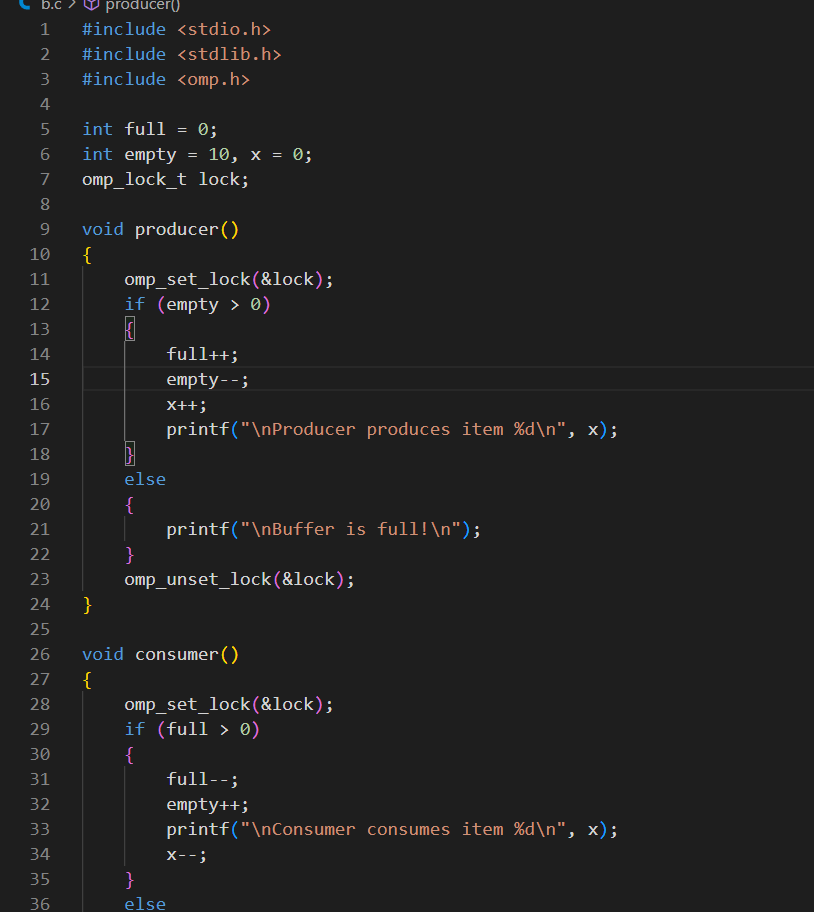
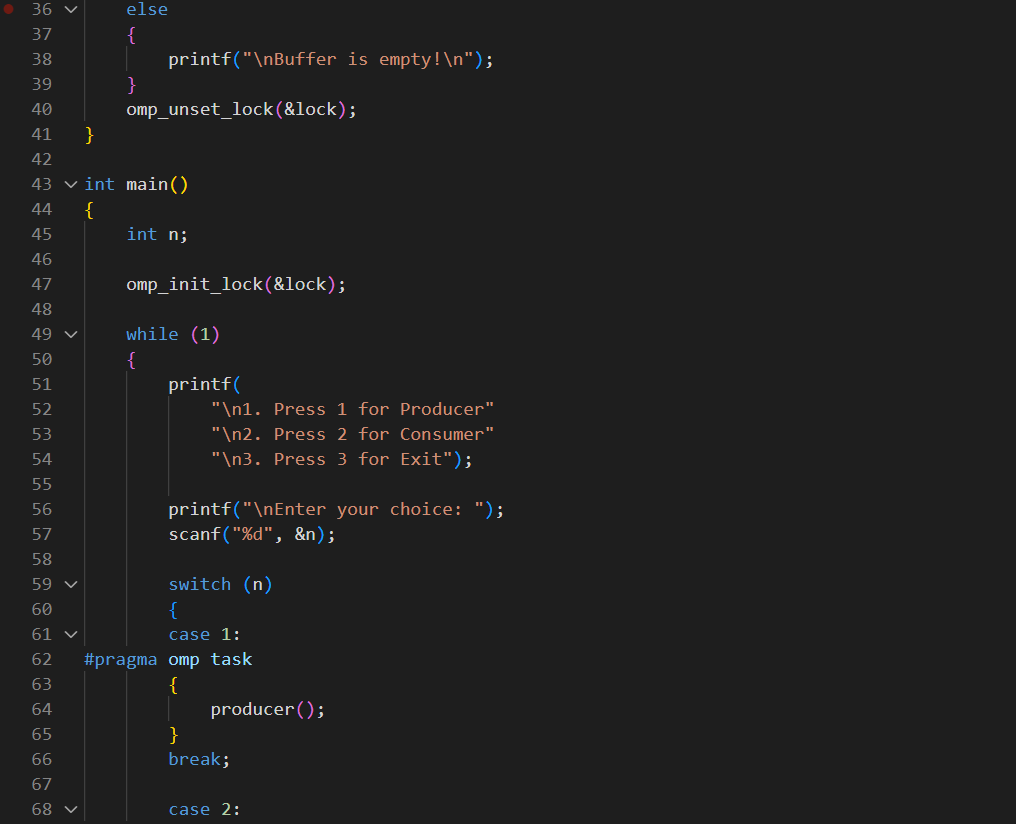
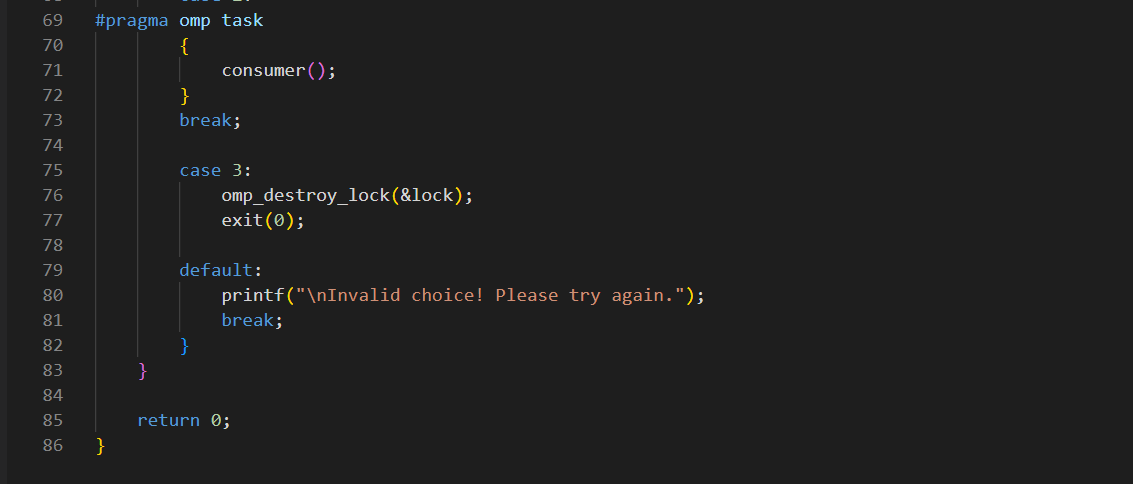
### Analysis

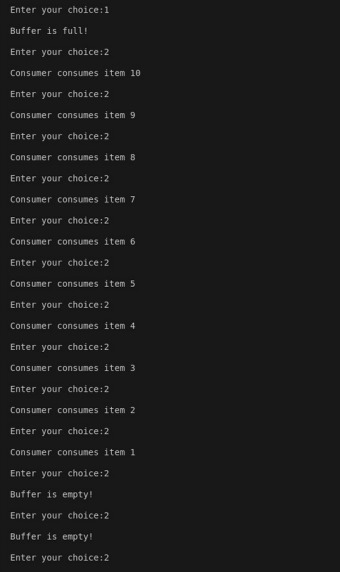
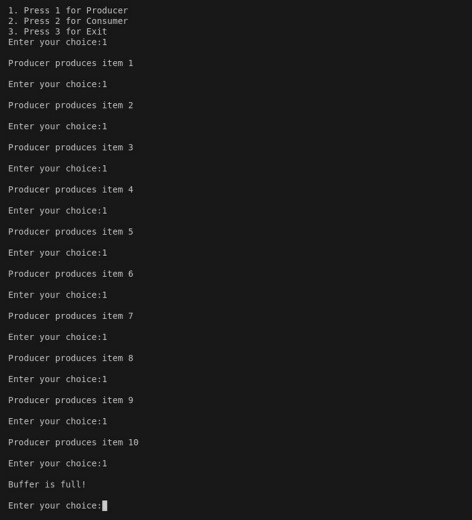
1. **Parallelism and Task Overhead**:
   * **Task Creation**: The use of #pragma omp task creates multiple tasks, which may lead to high overhead due to the creation and management of these tasks.
   * **Task Synchronization**: The #pragma omp taskwait ensures that the main thread waits for all created tasks to complete. This is necessary but can introduce additional overhead.
2. **Performance Considerations:**
   * **Scalability**: While OpenMP allows parallel execution, the recursive approach does not scale well for large n due to the exponential growth in the number of tasks.
   * **Efficiency**: Recursive approaches with OpenMP are generally inefficient for Fibonacci computation due to redundant calculations. For large n, the performance might degrade rapidly.
3. **Practical Limitations:**
   * **Segmentation Faults**: As noted in your earlier issue with larger values of n, very deep recursion and excessive task creation can lead to segmentation faults due to stack overflow.

**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:  
  
  
**



**Github Link:**