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

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

**Name:** Abhijeet Kamalekar

**PRN:** 2020BTECS00010

**Course:** High Performance Computing Lab

**Practical No. 4**

**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:

#include <omp.h>

#include <stdio.h>

int fibonacci(int n) {

    int x, y;

    if (n < 2) {

        return n;

    } else {

        #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 = 10;

    int result;

    #pragma omp parallel

    {

        #pragma omp single

        result = fibonacci(n);

    }

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

    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 <omp.h>

#include <stdio.h>

#include <stdlib.h>

#define BUFFER\_SIZE 10

#define NUM\_ITEMS 20

int buffer[BUFFER\_SIZE];

int count = 0;

int in = 0;

int out = 0;

void produce(int item) {

    #pragma omp critical

    {

        while (count == BUFFER\_SIZE) {

        }

        buffer[in] = item;

        in = (in + 1) % BUFFER\_SIZE;

        count++;

        printf("Produced: %d, Buffer Count: %d\n", item, count);

    }

}

int consume() {

    int item;

    #pragma omp critical

    {

        while (count == 0) {

        }

        item = buffer[out];

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

        count--;

        printf("Consumed: %d, Buffer Count: %d\n", item, count);

    }

    return item;

}

int main() {

    #pragma omp parallel sections

    {

        #pragma omp section

        {

            for (int i = 0; i < NUM\_ITEMS; i++) {

                int item = rand() % 100;

                produce(item);

            }

        }

        #pragma omp section

        {

            for (int i = 0; i < NUM\_ITEMS; i++) {

                int item = consume();

            }

        }

    }

    return 0;

}

**Screenshots:**

****

**Github Link:**  [https://github.com/AbhijeetKamalekar15/HPC-Lab.git](%20https:/github.com/AbhijeetKamalekar15/HPC-Lab.git)