Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

Class: Final Year (Computer Science and Engineering)

Year: 2023-24 **Semester:** 1

Course: High Performance Computing Lab

Practical No. 4

Exam Seat No: 2020BTECS00033

Name: Prathamesh Santosh Raje

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:

Fibonacci Computation Sequential Code:

```
#include <iostream>
#include <omp.h>
using namespace std;

long int fib(long int n)
{
  long int i, j;
  if (n<2)
    return n;
  else
    {
       i=fib(n-1);
       j=fib(n-2);
       return i+j;
```

```
    }
}

int main()
{
  long int n,ans;
  cout<<"Enter n: ";
  cin>>n;

  double itime, ftime, exec_time;
  itime = omp_get_wtime();

  ans = fib(n);

  ftime = omp_get_wtime();
  exec_time = (ftime - itime);
  printf ("fib(%ld) = %ld\n", n,ans);
  printf("\nTime taken is %f\n", exec_time);
  return 0;
}
```

Fibonacci Computation Sequential Output:

```
PS D:\Final Year B.Tech\HPC\Practical No. 4> g++ -fopenmp .\fib_seq.cpp
PS D:\Final Year B.Tech\HPC\Practical No. 4> .\a.exe
Enter n: 10
fib(10) = 55

Time taken is 0.000000
PS D:\Final Year B.Tech\HPC\Practical No. 4> .\a.exe
Enter n: 35
fib(35) = 9227465

Time taken is 0.072000
PS D:\Final Year B.Tech\HPC\Practical No. 4>
```

Fibonacci Computation Parallel Code:

```
#include <iostream>
#include <omp.h>
using namespace std;

long long int fib(long long int n)
{
```

```
  long long int i, j;
  if (n<2)
    return n;
  else
    {
       #pragma omp task shared(i) firstprivate(n)
       i=fib(n-1);

       #pragma omp task shared(j) firstprivate(n)
       j=fib(n-2);

       #pragma omp taskwait
       return i+j;
    }
}

int main()
{
  long long int n,ans;
  cout<<"Enter n: ";
  cin>>n;

  omp_set_num_threads(2);
  double itime, ftime, exec_time;
  itime = omp_get_wtime();

  #pragma omp parallel shared(n)
  {
    #pragma omp single
    ans = fib(n);
  }

  ftime = omp_get_wtime();
  exec_time = (ftime - itime);
  printf ("fib(%ld) = %ld\n", n,ans);
  printf("\nTime taken is %f\n", exec_time);
  return 0;
}
```

Fibonacci Computation Parallel Output:

```
PS D:\Final Year B.Tech\HPC\Practical No. 4> g++ -fopenmp .\fib_par.cpp
PS D:\Final Year B.Tech\HPC\Practical No. 4> .\a.exe
Enter n: 10
fib(10) = 55

Time taken is 0.002000
PS D:\Final Year B.Tech\HPC\Practical No. 4> g++ -fopenmp .\fib_par.cpp
PS D:\Final Year B.Tech\HPC\Practical No. 4> .\a.exe
Enter n: 35
fib(35) = 9227465

Time taken is 41.326000
PS D:\Final Year B.Tech\HPC\Practical No. 4>
```

Information:

Execution time for sequential code:

```
PS D:\Final Year B.Tech\HPC\Practical No. 4> g++ -fopenmp .\fib_seq.cpp
PS D:\Final Year B.Tech\HPC\Practical No. 4> .\a.exe
Enter n: 10
fib(10) = 55

Time taken is 0.000000
PS D:\Final Year B.Tech\HPC\Practical No. 4> .\a.exe
Enter n: 35
fib(35) = 9227465

Time taken is 0.072000
PS D:\Final Year B.Tech\HPC\Practical No. 4>
```

Execution time for parallel code:

```
PS D:\Final Year B.Tech\HPC\Practical No. 4> g++ -fopenmp .\fib_par.cpp
PS D:\Final Year B.Tech\HPC\Practical No. 4> .\a.exe
Enter n: 10
fib(10) = 55

Time taken is 0.002000
PS D:\Final Year B.Tech\HPC\Practical No. 4> g++ -fopenmp .\fib_par.cpp
PS D:\Final Year B.Tech\HPC\Practical No. 4> .\a.exe
Enter n: 35
fib(35) = 9227465

Time taken is 41.326000
PS D:\Final Year B.Tech\HPC\Practical No. 4>
```

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:

Producer Consumer Problem Code:

```
#include <iostream>
#include <stdlib.h>
using namespace std;
// Initialize a mutex to 1
int mutex = 1;
// Number of full slots as 0a
int full = 0;
// Number of empty slots as size
// of buffer
int empty = 10, x = 0;
// Function to produce an item and
// add it to the buffer
void producer()
{
    // Decrease mutex value by 1
    --mutex;
    // Increase the number of full
    // slots by 1
    ++full;
    // Decrease the number of empty
    // slots by 1
    --empty;
    // Item produced
    x++;
    printf("\nProducer produces item %d",x);
    // Increase mutex value by 1
    ++mutex;
```

```
}
// Function to consume an item and
// remove it from buffer
void consumer()
{
    // Decrease mutex value by 1
    --mutex;
    // Decrease the number of full
    // slots by 1
    --full;
    // Increase the number of empty
    // slots by 1
    ++empty;
    printf("\nConsumer consumes item %d",x);
    x--;
    // Increase mutex value by 1
    ++mutex;
}
// Driver Code
int main()
{
    int n, i;
    printf("\n1. Press 1 for Producer\n2. Press 2 for Consumer\n3. Press 3 
for Exit");
    // Using '#pragma omp parallel for'
    // can give wrong value due to
    // synchronization issues.
    // 'critical' specifies that code is
    // executed by only one thread at a
    // time i.e., only one thread enters
    // the critical section at a given time
    #pragma omp critical
    for (i = 1; i > 0; i++) {
    printf("\nEnter your choice:");
    scanf("%d", &n);
    // Switch Cases
    switch (n) {
        case 1:
        // If mutex is 1 and empty
        // is non-zero, then it is
        // possible to produce
```

```
        if ((mutex == 1)
        && (empty != 0)) {
        producer();
        }
        // Otherwise, print buffer
        // is full
        else {
        printf("Buffer is full!");
        }
        break;
        case 2:
        // If mutex is 1 and full
        // is non-zero, then it is
        // possible to consume
        if ((mutex == 1)
        && (full != 0)) {
        consumer();
        }
        // Otherwise, print Buffer
        // is empty
        else {
        printf("Buffer is empty!");
        }
        break;
        // Exit Condition
        case 3:
        exit(0);
        break;
    }
    }

}
```

Output:

```
PS D:\SEM 7\HPC Lab\Assignment 4> g++ -fopenmp producer_consumer.cpp
PS D:\SEM 7\HPC Lab\Assignment 4> ./a

1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit
Enter your choice:1

Producer produces item 1
Enter your choice:1

Producer produces item 2
Enter your choice:2

Consumer consumes item 2
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
Buffer is empty!
Enter your choice:3
PS D:\SEM 7\HPC Lab\Assignment 4>
```

Information:

Global Variables:

mutex: This integer variable is used as a binary semaphore to ensure mutual exclusion for accessing the shared buffer.
full: It keeps track of the number of items currently present in the buffer.
empty: This variable represents the number of empty slots available in the buffer.
x: An integer variable to keep track of the item being produced or consumed.

producer Function:

1. Decrements mutex to enter the critical section.
2. Increments full to indicate that an item has been produced.
3. Decrements empty to signify that an empty slot has been filled.
4. Increments x (the item being produced).
5. Prints a message to indicate the item produced.
6. Increments mutex to exit the critical section.

consumer Function:

1. Decrements mutex to enter the critical section.
2. Decrements full to indicate that an item has been consumed.
3. Increments empty to signify that an empty slot has been freed.
4. Decrements x (the item being consumed).
5. Prints a message to indicate the item consumed.
6. Increments mutex to exit the critical section.

main Function:

The main function is the entry point of the program.

It presents a menu to the user to choose between producing an item, consuming an item, or exiting the program.
Inside a for loop, it repeatedly asks the user for their choice and performs the corresponding action based on the choice.
The actions are protected by a critical section (#pragma omp critical) to ensure that only one thread can execute these actions at a time.

Github Link: