ASSIGNMENT # 03

Submitted By:

AWAIS SADDIQUI (21PWCSE1993)

SECTION A

Submitted To:

MAM DURRE NAYAB

Computer System Engineering, University of Engineering and Technology Peshawar,

Task 01:

Question 1:

Explain different scenarios where Kernel-level threads are not needed.

Answer:

When the application does not require access to kernel resources. For example, a simple application that only performs calculations does not need to access kernel resources, such as the file system or network. In this case, user-level threads can be used to improve performance.

They can be slower than user-level threads. When a thread makes a system call, it must switch from user mode to kernel mode. This can be a slow process, especially on older hardware.

Question 2:

Discuss how multithreading systems are different from multi-processing systems.

Answer:

a) *Multithreading:*

Multithreading is a technique that allows a single process to run multiple threads simultaneously. Threads are lightweight processes that share the same address space and resources.

b) *Multi-Processing:*

Multiprocessing is a technique that uses multiple CPUs to run multiple processes simultaneously. Processes are heavyweight units of execution that have their own address space and resources.

Explanation:

Threads can communicate and share data easily, which can improve performance in applications that are heavily dependent on I/O or that have a lot of parallelizable work. While processes cannot communicate or share data easily, which can limit performance in applications that require a lot of communication between threads.

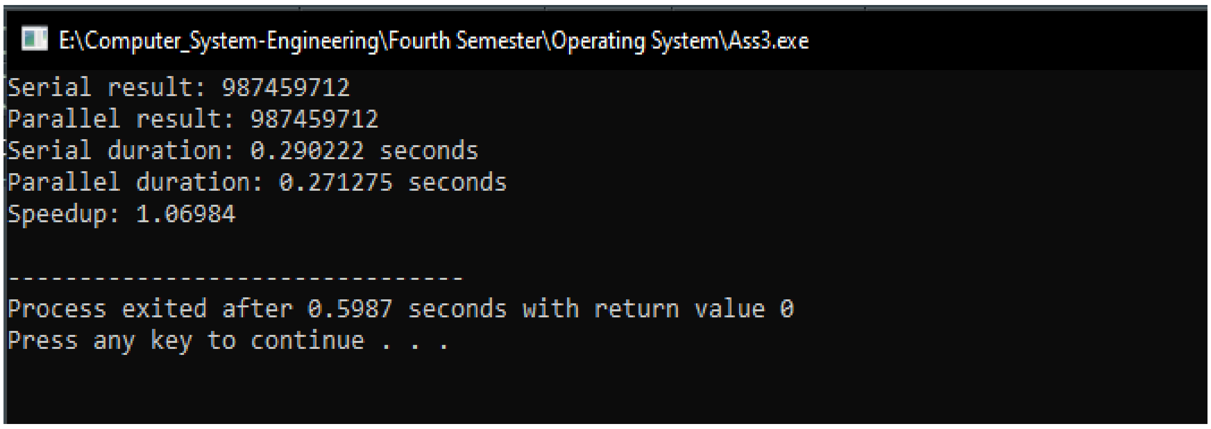
Multithreading can help to improve performance by allowing the application to continue executing other threads while it waits for I/O to complete.

Multiprocessing can help to improve performance by allowing each thread to run on its own processor, which can reduce the amount of communication between threads.

Task 02:

Question 1:

Output:



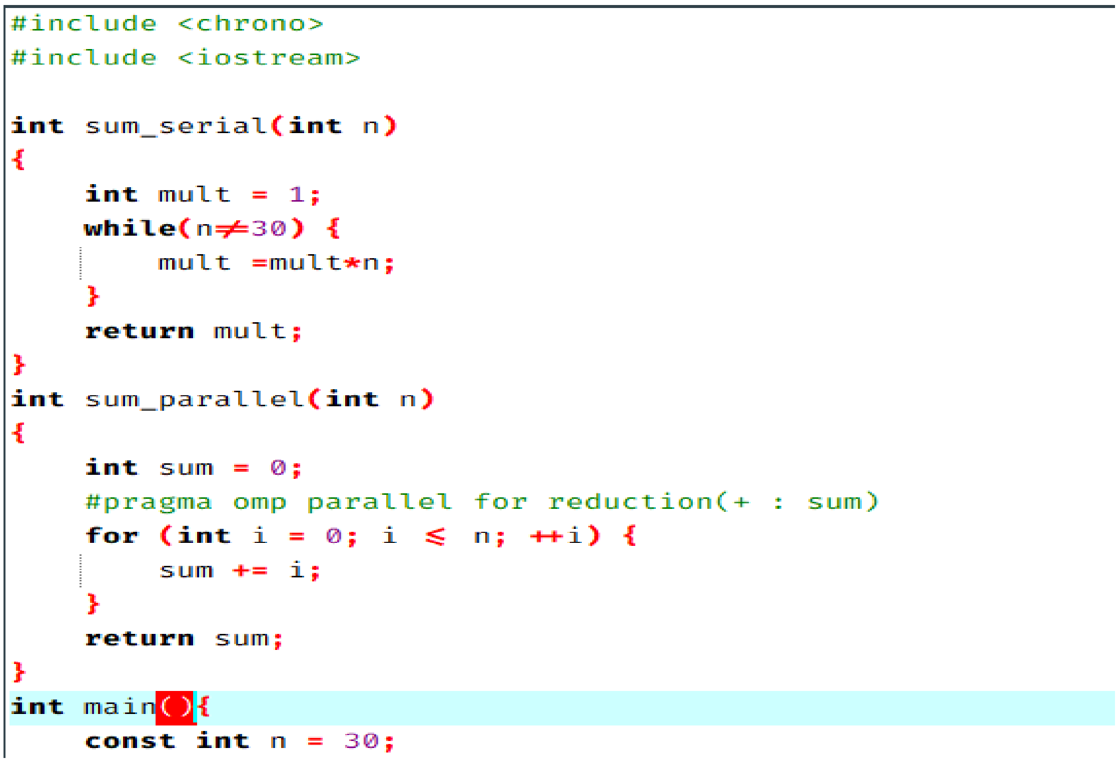
```
E:\Computer_System-Engineering\Fourth Semester\Operating System\Ass3.exe
Serial result: 987459712
Parallel result: 987459712
Serial duration: 0.290222 seconds
Parallel duration: 0.271275 seconds
Speedup: 1.06984

--------------------------------
Process exited after 0.5987 seconds with return value 0
Press any key to continue . . .
```

- This code measures the execution time and calculates the sum of numbers from 0 to n using both serial and parallel programming approaches. The sum_serial function performs the sum serially, while the sum_parallel function uses OpenMP to parallelize the computation.

- Without the use of threads (serial execution), the program will calculate the sum in a sequential manner, where each number is added to the sum one at a time.

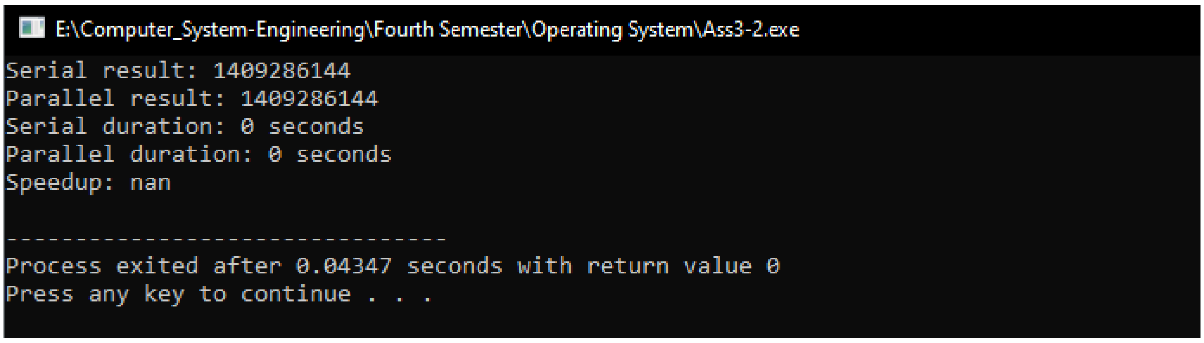
- With the use of threads (parallel execution), the program will divide the range of numbers into chunks and distribute the workload among multiple threads.

Code:



```
#include <chrono>
#include <iostream>

int sum_serial(int n)
{
    int mult = 1;
    while(n≠30) {
        mult =mult*n;
    }
    return mult;
}
int sum_parallel(int n)
{
    int sum = 0;
    #pragma omp parallel for reduction(+ : sum)
    for (int i = 0; i ≤ n; ++i) {
        sum += i;
    }
    return sum;
}
int main(){
    const int n = 30;
```

Output 3:



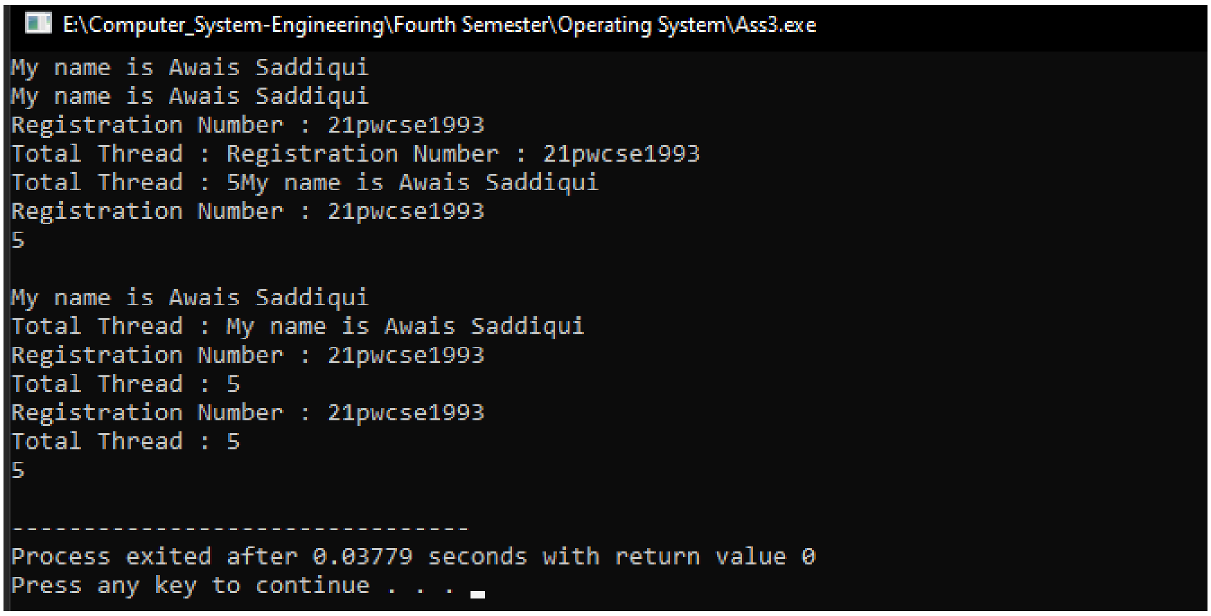
```
E:\Computer_System-Engineering\Fourth Semester\Operating System\Ass3-2.exe
Serial result: 1409286144
Parallel result: 1409286144
Serial duration: 0 seconds
Parallel duration: 0 seconds
Speedup: nan

--------------------------------
Process exited after 0.04347 seconds with return value 0
Press any key to continue . . .
```

- Without the use of threads (serial execution), the program will calculate the product in a sequential manner using a while loop. Each number from 1 to n will be multiplied to the current value of the product (mult) one at a time.
- With the use of threads (parallel execution), the program will divide the range of numbers into chunks and distribute the workload among multiple threads. Each thread will independently calculate the product for its assigned chunk.

Task 03:

Question 1:

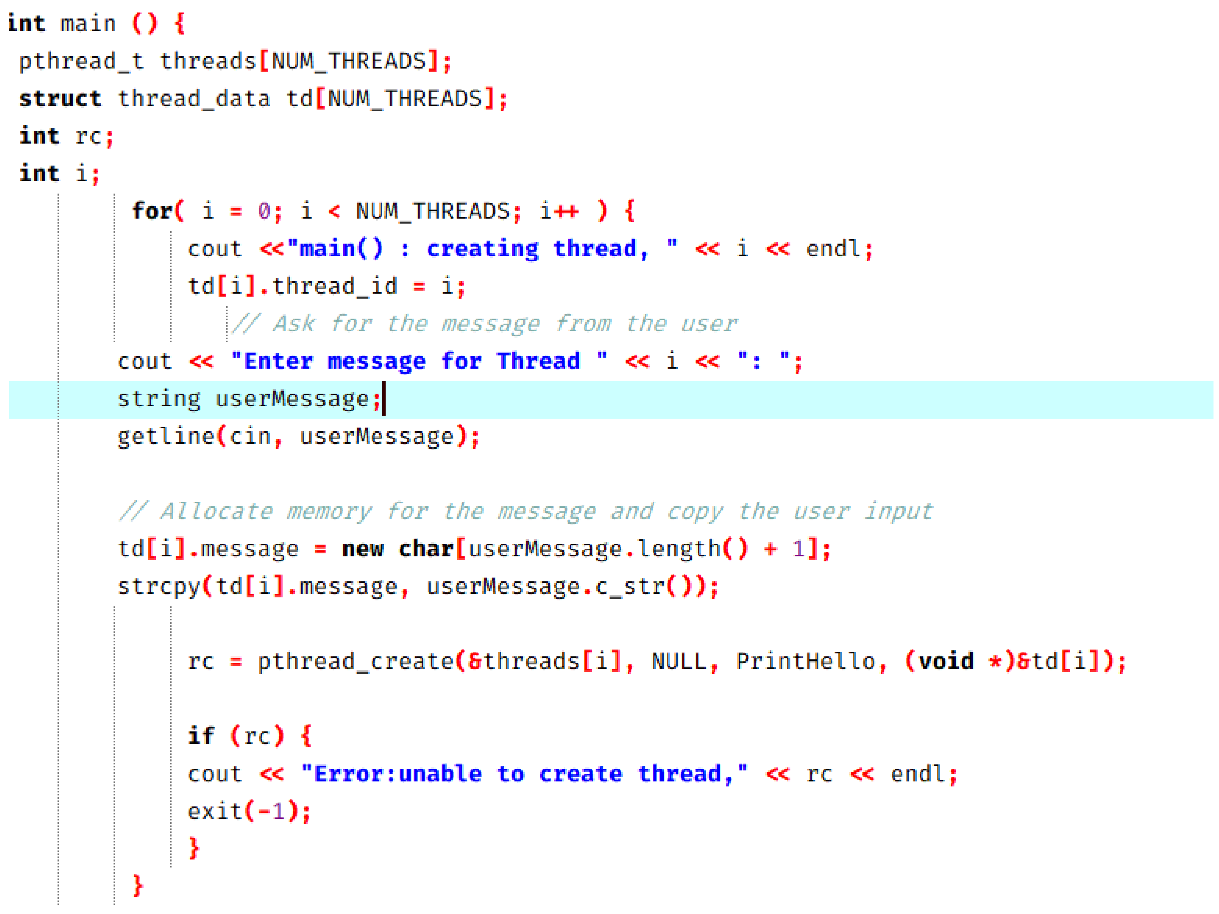
```cpp
#include <unistd.h>
using namespace std;
#define NUM_THREADS 5

void *myData(void *data){
    cout<<"My name is Awais Saddiqui\n";
    cout<<"Registration Number : 21pwcse1993\n";
    int totalThread=5;
    cout<<"Total Thread : "<<totalThread<<endl;
    pthread_exit(NULL);
}
main(){
    int rc;
    pthread_t threads[NUM_THREADS];
    for(int i=0; i<NUM_THREADS; i++){
        rc=pthread_create(&threads[i],NULL,myData,(void *)i);
        if(rc){
            cout<<"Error in threads ...."<<rc<<endl;
            exit(-1);
        }
    }
// wait for threads
    for (int i = 0; i < NUM_THREADS; ++i) {
        pthread_join(threads[i], NULL);
    }
}
```



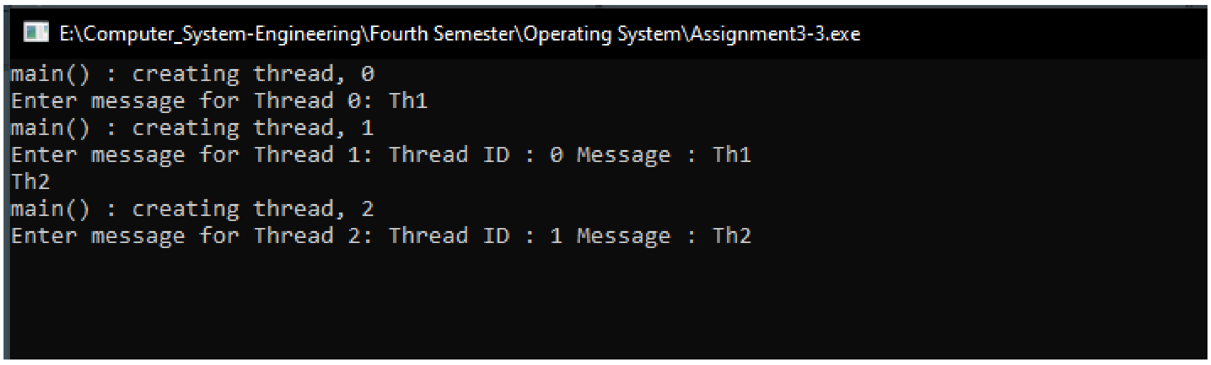
E:\Computer_System-Engineering\Fourth Semester\Operating System\Ass3.exe

```
My name is Awais Saddiqui
My name is Awais Saddiqui
Registration Number : 21pwcse1993
Total Thread : Registration Number : 21pwcse1993
Total Thread : 5My name is Awais Saddiqui
Registration Number : 21pwcse1993
5

My name is Awais Saddiqui
Total Thread : My name is Awais Saddiqui
Registration Number : 21pwcse1993
Total Thread : 5
Registration Number : 21pwcse1993
Total Thread : 5
5

--------------------------------
Process exited after 0.03779 seconds with return value 0
Press any key to continue . . .
```

Part B:



```
int main () {
 pthread_t threads[NUM_THREADS];
 struct thread_data td[NUM_THREADS];
 int rc;
 int i;
        for( i = 0; i < NUM_THREADS; i++ ) {
            cout <<"main() : creating thread, " << i << endl;
            td[i].thread_id = i;
                // Ask for the message from the user
        cout << "Enter message for Thread " << i << ": ";
        string userMessage;
        getline(cin, userMessage);

        // Allocate memory for the message and copy the user input
        td[i].message = new char[userMessage.length() + 1];
        strcpy(td[i].message, userMessage.c_str());

            rc = pthread_create(&threads[i], NULL, PrintHello, (void *)&td[i]);

            if (rc) {
            cout << "Error:unable to create thread," << rc << endl;
            exit(-1);
            }
        }
```



E:\Computer_System-Engineering\Fourth Semester\Operating System\Assignment3-3.exe

```
main() : creating thread, 0
Enter message for Thread 0: Th1
main() : creating thread, 1
Enter message for Thread 1: Thread ID : 0 Message : Th1
Th2
main() : creating thread, 2
Enter message for Thread 2: Thread ID : 1 Message : Th2
```