Department of Computer Systems Engineering
University of Engineering and Technology
Peshawar, Pakistan

CSE-204 Operating Systems, Spring 2023
Assignments # 03 and 04

Total Marks: 20 **Submission Deadline: June 16th, 2023**

INSTRUCTIONS

1. Attempt **ALL** questions in a precise and to-the-point manner.
2. Extra details will not add any weight to the attained marks.
3. Plagiarism is strongly discouraged.
4. You can take help from available resources but do not just copy them and also do not forget to refer to the resources used.

ASSIGNMENT 03 (CHAPTER 04)

Task 01: (Topic: Threads and Concurrency) **[CLO-2]**

a. Explain different scenarios where Kernel-level threads are not needed. Justify your answer by giving reasons.

b. Discuss how multithreading systems are different from multi-processing systems. Discuss different scenarios where multi-threading systems are preferred over multi-processing systems.

Task 02: (Topic: Open Multi-processing) **[CLO-3]**

a. Execute the following code for understanding the concept of threads and its impact on loops using **Open Multi-processing** (Open MP). The code comprises two functions that calculate the sum of the first n natural numbers using a **"for loop"**: **1. sum_serial** and **2. sum_parallel.**

i. **The "sum_serial" function uses a serial implementation.**
ii. **The "sum_parallel" function uses OpenMP to parallelize the for loop.**

We then benchmark the two implementations by calling both functions with **n=100000000** and measuring the time taken to complete the task using the high_resolution_clock class from the **chrono** library. Below is the implementation of the above code:

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

// Serial programming function
```

```
 int sum_serial(int n)
{
   int sum = 0;
   for (int i = 0; i <= n; ++i) {
      sum += i;
   }
   return sum;
}

// Parallel programming function
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;
}

// Driver Function
int main()
{
     const int n = 100000000;
     auto start_time   = std::chrono::high_resolution_clock::now();
     int result_serial = sum_serial(n);
     auto end_time  = std::chrono::high_resolution_clock::now();
     std::chrono::duration<double> serial_duration = end_time - start_time;

   start_time = std::chrono::high_resolution_clock::now();

   int result_parallel = sum_parallel(n);
   end_time = std::chrono::high_resolution_clock::now();
   std::chrono::duration<double> parallel_duration
      = end_time - start_time;

   std::cout << "Serial result: " << result_serial
         << std::endl;
   std::cout << "Parallel result: " << result_parallel
         << std::endl;
   std::cout << "Serial duration: "
         << serial_duration.count() << " seconds"
```

```
        << std::endl;
  std::cout << "Parallel duration: "
        << parallel_duration.count() << " seconds"
        << std::endl;
  std::cout << "Speedup: "
        << serial_duration.count()
. / parallel_duration.count()
        << std::endl;
    return 0;
}
```

b. Comment on the output of the above code in terms of performance with and without the use of threads with logical reasoning.
c. Modify the given code by replacing the **for** loop with **while** loop, **n=30,** and the sum with **mult=mult*i**, where **i** goes from **1 to n** and the initial value of **mult** is **1.**
d. Comment on the output of the above code in terms of performance with and without the use of threads with logical reasoning.

Task 03: (Topic: Creating pThreads) **[CLO-2]**

1. Generate a simple C++ code that creates 5 threads with the **pthread_create()** routine. Each thread prints your name, registration number, and the number of threads and then terminates with a call to **pthread_exit().**
2. The following example shows how to pass multiple arguments to the threads via a structure. You can pass any data type in a thread callback because it points to void. The **pthread_join()** subroutine blocks the calling thread until the specified **'threadid'** thread terminates. When a thread is created, one of its attributes defines whether it is joinable or detached. Only threads that are created as joinable can be joined. If a thread is created as detached, it can never be joined.
   a. Execute the following code demonstrating how to wait for thread completions by using the **pthread join** routine. Comment on the output.

```
#include <iostream>
#include <cstdlib>
#include <pthread.h>
#include <unistd.h>

using namespace std;
#define NUM_THREADS 5

void *wait(void *t) {
  int i;
  long tid;
```

```
  tid = (long)t;
  sleep(1);
  cout << "Sleeping in thread " << endl;
  cout << "Thread with id : " << tid << "  ...exiting " << endl;
  pthread_exit(NULL);
}

int main () {
  int rc;
  int i;
  pthread_t threads[NUM_THREADS];
  pthread_attr_t attr;
  void *status;

  // Initialize and set thread joinable
  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

  for( i = 0; i < NUM_THREADS; i++ ) {
    cout << "main() : creating thread, " << i << endl;
    rc = pthread_create(&threads[i], &attr, wait, (void *)i );
    if (rc) {
      cout << "Error:unable to create thread," << rc << endl;
      exit(-1);
    }
  }

  // free attribute and wait for the other threads
  pthread_attr_destroy(&attr);
  for( i = 0; i < NUM_THREADS; i++ ) {
    rc = pthread_join(threads[i], &status);
    if (rc) {
      cout << "Error:unable to join," << rc << endl;
      exit(-1);
    }
    cout << "Main: completed thread id :" << i ;
    cout << "  exiting with status :" << status << endl;
  }
  cout << "Main: program exiting." << endl;
  pthread_exit(NULL);
}
```

b. Modify the following code to take the message argument from the user after each iteration.

```
#include <iostream>
#include <cstdlib>
#include <pthread.h>

using namespace std;
#define NUM_THREADS 5

struct thread_data {
  int  thread_id;
  char *message;
};
void *PrintHello(void *threadarg) {
  struct thread_data *my_data;
  my_data = (struct thread_data *) threadarg;

  cout << "Thread ID : " << my_data->thread_id ;
  cout << " Message : " << my_data->message << endl;
  pthread_exit(NULL);
}

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;
    td[i].message = "This is message";
    rc = pthread_create(&threads[i], NULL, PrintHello, (void *)&td[i]);

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

ASSIGNMENT 04 (CHAPTER 05)

Task 01: (Topic: CPU Scheduling) **[CLO-2]**

a. What are the state transitions of a process CPU burst where context switching occurs?
b. Discuss how the exponential averaging method reduces the effect of each subsequent previously estimated time value of CPU bursts by selecting the value of **alpha (α).**
c. Discuss how the **shortest job first (SJF)** scheduling is the most efficient Non-Preemptive scheduling algorithm? Give both logical and mathematical reasoning.

Task 02: (Topic: Pre-emptive/ non Pre-emptive Schedulers) **[CLO-3]**

a. Given the following six processes: P_1 -P_6 along with their **arrival time** and **burst time**.
Evaluate the following:
   i. **Waiting time per process**
   ii. **Average waiting time**
   iii. **Turnaround time per process**
   iv. **Average turnaround time**
   v. **Throughput**

   By applying the following scheduling algorithms:
   i. **Shortest Job First Scheduling**
   ii. **Shortest Remaining Job First Scheduling**
   iii. **First Come First Serve Scheduling**
   iv. Round Robin Scheduling with q= 5
   v. Priority Scheduling with priorities equal to 4,0,1,5,2,3 for P_1 -P_6 respectively

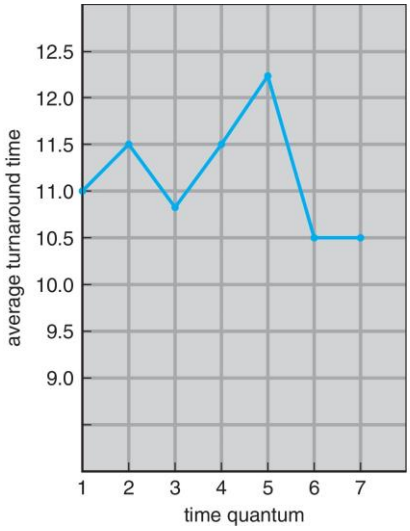
Process	Arrival Time	Burst Time
P_1	0.0	17
P_2	3.0	13
P_3	4.0	11
P_4	5.0	4
P_5	7.0	4
P_6	9.0	4

b. Which scheduling algorithm will you select and why?

Submission Deadline: June 16th, 2023

Task 03: (Topic: Round Robin Scheduling) **[CLO-3]**

a. The following graph shows the impact of changing the quantum value on the average turnaround time of the processes. Mathematically prove the values.



process	time
P_1	6
P_2	3
P_3	1
P_4	7

b. Re-generate the above graph for the updated processes burst times i.e. **7,2,5,10** for processes ***P_1 –P_4*** respectively.