Jordan University of Science and Technology
CS375 Operating systems - Assignment 2
Summer 2019-2020

Objectives:

Student should be able to:

- ✓ Write a simple Multithreading problem that requires Task Parallelism.
- ✓ Write a simple Multithreading problem that requires Data Parallelism.
- ✓ learn how to write a multithreaded application that utilizes different software synchronization tools to prevent possible race condition scenario.

What to submit: Due: August 14, 2020 [there will be NO postponing]

- ✓ You have to work your assignment as a team with at MOST TWO members.
- ✓ At least one of the group members has to submit the report and codes on time.
- ✓ The file name of your report has to be formatted as CS375_Ass#2_X_Y.doc (or **.**docx or **.**Pdf) where **X** is the first student ID number and **Y** is the second student ID number.
- ✓ You are required to submit:
    - all assignment's parts as **fully documented source codes** (feel free to use any programming language).
    - a simple report contains **print screen for the final output of each part** and a **short discussion on the final results you got**.
    - Add whatever necessary details you need **[optional]**

After submitting your work, you should schedule an appointment. Check the discussion date(s) on the e-learning to discuss your work **on your personal computer.** Your grade will be given based on your both submission and discussion. You are expected to demonstrate any related question(s) **without refereeing to any supporting material during the discussion.**

Jordan University of Science and Technology
CS375 Operating systems - Assignment 2
Summer 2019-2020

Project Description

Refer to the textbook for Chapter 4 Programming Project: Multithreading programming problems on pages 195- 197, and 199 of the 9th edition.

- problems: 4.21, 4.24 and 4.26
- Project 2—Multithreaded Sorting Application (page 199)

You can also refer to chapter 5 notes to know how exactly you can use the synchronization tools.

Part I - Multithreaded / Task parallelism

a. Write a multithreaded program that calculates various statistical values for a list of numbers (try small/large/very large numbers). This program has to create **three** separate worker threads. One thread will determine the average of the numbers, the second will determine the maximum value, and the third will determine the minimum value. For example, suppose your list contains the Integers `90 81 78 95 79 72` and `85`, The program will report the following:

```
The average value is 82
The minimum value is 72
The maximum value is 95
```

The variables representing the average, minimum, and maximum values will be stored globally. The worker threads will set these values, and the **parent** thread will output the values once the workers have exited.

b. Write a multithreaded program that outputs prime numbers. This program should work as follows: The user will run the program and will enter a number as an input. The program will then create a **separate thread** that outputs all the prime numbers less than or equal to the number entered by the user.

c. Solve the previous problems **without using threads**. Use the **time function** with each part (a, b and c) and show, for each one, what is the exact execution time? Is there any difference? Why? You have to answer these questions **in your report.**

Part II – Multithreaded / Data parallelism

a. Write a multithreaded sorting program that works as follows: A list of integers is divided into **Four** smaller lists of equal size. **Four** separate threads (which we will term **sorting threads**) sort each sub-list using a **sorting algorithm of your choice**. (You may use any built in sorting function). The Four sub-lists are then merged by a Fifth thread — a **merging thread** — which merges the Four sub-lists into a single sorted list.
Because global data are shared cross all threads, perhaps the easiest way to set up the data is to create a global array. Each sorting thread will work on one 1/4 of this array. A second global array of the same size as the unsorted integer array will also be established. The merging thread will then merge the Four sub-lists into this second array. Graphically, this program is structured **similarly** according to Figure 4.20 if we intend to partition the array into two parts.
This programming project will require passing parameters to each of the sorting threads. In particular, it will be necessary to identify the starting index from which each thread is to begin sorting. Refer to the instructions in Project 1 for details on passing parameters to a thread. The **parent** thread will output the sorted array once all sorting threads have exited.

b. Solve the previous problem **without using threads**. Use the **time function** with each part (a and b) and show, for each one, what is the exact execution time? Is there any difference? Why? You have to answer these questions **in your report.**

Part III - Multithreaded application utilizes synchronization software tool

Write a multi-threaded application in which there are **three threads**. All threads open the same file. The content of this file is assumed to be lines that each consist of two numbers (you can start by a file that contain the line "0 0"). The first number in each line is the id of the thread that wrote the second number. Each thread performs the **following steps:**

- Open the file.
- Reads the last line in the file

- Close the file.
- Increment the second number in the last line by 1.
- Perform some computation such as finding the sum of the numbers between 1 and 1000000.
- Open the file.
- Write the thread id and the incremented number to the end of the file.
- Close the file.

Note that: at the end, all numbers (second number in each line) that are written to the file by all threads should be in sequence. Also, you should figure out what the critical region is, and to prevent the **race condition**, write your own code using:

a. Mutex lock synchronization tool

b. Semaphore synchronization tool