CSE 2431/5431
Instructor: Luan Duong, Ph.D.

LAB 3: THREADS AND LOCKS

Electronic Submission (Due):
11:59 pm, Tuesday, April 1, 2025 (April's Fool, but this is NOT a joke)
Difficulty: * (3 stars)**
Point: 30 points (6% total grade)

1. Goal

1) This lab will start a series of using threads, locks, semaphores, deadlocks, …
2) Particularly, in this lab, you will **learn how to create threads and utilize locks** first. We do not base on one piece of codes, instead, you will be given several pieces of code for different experiments.

2. Creating Threads [5 points]

In the class **(Lecture_Topic_7_Concurrency_pt_1), page 8 to 15**, we have known that creating threads and running threads is indeed pretty tricky. In this lab, I have provided a faulty program as **`createThreads.c`**. **You need to fix it with the malloc approach.**

3. Using Locks: [25 points] (Lecture_Topic_7_Concurrency_pt_2: mutex lock)

First take a look at the provided **`sequential.c`**. It generates a million random integers between 0 and 100 and then calculates the sum of these integers. Compile it and run it. Note that we provide a deterministic seed to the random generator, so every time you run it, you should get the same result.

Experiment 1 [5 points]: We hope to parallelize the computation by creating 10 threads and let each thread sum 100,000 numbers. Take a look at **`parallel1.c`**. It creates 10 separate arrays, each holding 100,000 numbers. You need to **add code to create threads, pass one array to each thread, and let each thread perform its work**. **In Experiment 1, let's try it without synchronization.** Each thread should do the following work:

```
for(i=……){
    sum += array[i];
}
```

Compile and run your program. Does it generate the same result as the sequential version? Run it multiple times. Does it always generate the same result?

Experiment 2 [10 points]: Now let's add proper synchronization to your code. **You need to define a lock in your program** and **change each thread's code** to the following **(this is a pseudocode, please refer to the mutex lock slide in your lecture notes):**

```
for(i=……){
  lock()
  sum += array[i];
  unlock()
}
```

Put your code in `parallel2.c`. Compile and run your program. Does it generate the same result as the sequential version? Run it multiple times. Does it always generate the same result?

Experiment 3 [10 points]: The previous program needs *to lock and unlock many times*. Could you think of a better solution that does not need to hold the lock for multiple times or for a long time? Put your code in **parallel3.c**

4. Test

For experiments 2 and 3, the result of a correct multi-threaded version should always generate the same result as the sequential version.
Since a faulty multi-threaded program may just occasionally show a wrong result, you need to run your program multiple times.

5. Submission Instruction

a. In this project, you need to submit all your codes, your **single** makefile and your report together in a **zip** file.
b. What code files? **4 files**: **createThread.c, parallel1.c, parallel2.c, parallel3.c**

c. You are responsible **to add any necessary libraries**. Please make sure that you have all the libraries added and make sure it successfully compiled before submitting your files.
d. How about Makefile? **A Single Makefile**, but this Makefile needs to generate **four (4) output files** called **`createThreads, parallel1, parallel2, parallel3`**
e. Report? **Have a simple report** with your name and your OSU dot number, noting down the answer for the questions from Experiment 1, Experiment 2 and Experiment 3 (corresponding to the program parallel1, parallel2, and parallel 3).
f. Pack all your file in a zip file and submit it on Carmen **on or before the due date.**
g. **Any program that does not compile will receive a zero.** Graders/I will not spend any time to fix your code due to simple errors you introduce at the last minute. It is your responsibility to leave yourself enough time to ensure that your code can be compiled, run, and tested well before the due date.