CS4540 - Operating Systems
Assignment #3

Introduction

Write two C programs, one sequential and another parallel (using Pthreads). Each fills an array, finds the sum of its values, and records the time used by the sequential code and the parallel code.

Problem Specification

1) In both programs (sequential and parallel), each array cell is equal to its index value, that is, `array[1] = 1, array[2] = 2, …, array[i] = i, …, array[n] = n.`

2) The main task of the sequential program is to fill the array and find the sum of its values.

3) The parallel program should use, in turn, `t` = 2, 4 and 8 threads (`t` is the number of threads). Each thread fills $\lfloor 1/t \rfloor$ of the array (the floor function, also denoted as *floor(t)*, is explained here: https://en.wikipedia.org/wiki/Floor_and_ceiling_functions), adds its values together, and updates the global `sum` variable.

   Then, each thread prints the range of the values it has assigned. For example, if there are two threads and 400 array elements, the results might be:

   ```
   Thread 0 processes array elements with indices 0-199, and Thread
   1 processes array elements with indices 200-399.
   ```

4) Run your program with different values of `n` as shown in the leftmost column of the table below. For the parallel version, run the program using 2, 4, and 8 threads as shown below.

	Execution Time for Sequential Code	Execution Time for Parallel Code		
		2 Threads	4 Threads	8 Threads
1,048,576				
2,097,152				
4,194,304				
134,217,728				
268,435,456				

5) Once you have obtained all data for the above table, plot in one chart four curves showing execution times for sequential and parallel code (for the later for different numbers of threads, i.e., 2, 4, 8) as the function of *n* (with *n* on the X-axis, *execution time* on the Y-axis). You may use any application that can draw nice diagrams, for example, a spreadsheet.

6) Answer to the following questions:

   a) Does the parallel code always outperform the sequential one? Why it does or does not?

   b) Does increasing the number of threads always result in a shorter execution time? Can you think of reasons of why or why not?

Hints

1) For recording time, you may use the following code snippets:

   ```
   struct timespec start, finish;
   ```

```
double elapsed;
clock_gettime(CLOCK_MONOTONIC, &start);

/* some of your code here */

clock_gettime(CLOCK_MONOTONIC, &finish);
elapsed = (finish.tv_sec - start.tv_sec);
elapsed += (finish.tv_nsec - start.tv_nsec) / 1000000000.0;
```

2) Due to exceeding the allowed integer value in (some) array elements, you might wish to use, for example, `double` (not `int`) as the array type.

SLC Report Requirements

For each program write a full SOFTWARE LIFE CYCLE (SLC) report (analogous to the SLC report presented in class).

Please note that your reporting job is easier due to the following simplifications:

1) The PROBLEM SPECIFICATION section (Step 1) should include just a copy of the given problem(s).
2) The PROGRAM STRUCTURE DESIGN section (Step 2) will have a few modules to name (Substep *2.1. Modules and Their Basic Structure*), and few modules to provide pseudocode for (Substep *2.2.Pseudocode for the Modules*).
3) The sections for RISK ANALYSIS (Step 3), VERIFICATION (Step 4), REFINING THE PROGRAM (Step 7), PRODUCTION (Step 8) and MAINTENANCE (Step 9) can include just 1-2 sentences (analogous in the SLC report presented in class).
4) The CODING section (Step 5) should have the appropriate number of code refinement levels.
5) Due to the simple structure of the programs, the TESTING section (Step 6) should be rather easy.
6) Using mono-spaced fonts like (Courier) for source code increases code readability and quality. Please use this type of font with only source code in your report.

Coding, Running and Submission Requirements

1) Follow *C Code Style Guide* and the proper programming style it requires, including comments, blank lines, indentations, spaces, etc.
2) All programs must be compiled and executed on Ubuntu running within the Oracle VirtualBox.
3) Remember about *Assignment Submission Instructions* (guidelines), to be followed for each program of this assignment.
4) In addition to what *Assignment Submission Instructions* require, provide a *makefile*, and, if needed a *README* file explaining how to compile and run your program.

Submission Checklist

For each program you need to submit:

1) the SLC report (including 9 SLC steps, with as many pseudocode and code refinements as needed);
2) *makefile*, and, if needed a README file explaining how to compile and run the program;
3) the files created by the *script* command (including program output to the terminal, if any);
4) the output files (if any in addition to the output to the terminal);
5) the file with the filled table of results, the plots, and answers—as required in Items 4, 5 and 6 of Problem Specification.

Submit your complete assignment package (all files) via Elearning as a zip file. Use `hw<number>_<LastName>.zip` as the format for the name of the zipped file (e.g., `hw1_Smith.zip`)

-------- Good luck! -----