CS4540 Operating Systems, Fall 2015
Assignment #3

Introduction

Write two C programs: one using sequential code and another using parallel code with threads. Both programs fill an array (but in different ways), and record time taken to do this.

Problem Specification

1) The sequential C code fills an array of size `n` with integers. Use arbitrary hard-coded integers, (e.g., `array[i] = 3` or `array[i] = 4` or `array[i] = 2`, etc.).

   Run your program with different values of `n` as shown in the leftmost column of the table below. For every value of `n` record time taken by filling the array.

n	Filling Time for Sequential Code	Filling Time for of Parallel Code
100		
1,000		
10,000		
100,000		
1,000,000		
10,000,000		

2) The parallel C code fills its array with different integers. The filling process must be done by four threads (using Pthreads). Each thread fills about ¼ of the whole array with its id. For example, `thread 0` fills the first fourth of the array (from `array[0]` to `array[floor(n/4)])` with 0s, `thread 1` fills the second fourth of the array (from `array[floor(n/4) + 1]` to `array[floor(n/2)])` with 1s, and so on (the floor function is explained here https://en.wikipedia.org/wiki/Floor_and_ceiling_functions).

   Run your program with different values of `n` as shown in the leftmost column of the table above. For every value of `n` record time taken by filling the array.

3) Once you have obtained all data for the above table, plot two curves in one chart showing times for both sequential and parallel code as the function of `n` (`n` on the X-axis, `time` on the Y-axis). You must use any program that can draw nice diagrams, a spreadsheet for example.

4) Write down your answers to the following questions:
   Does the parallel code always outperform the sequential one? Why it does or does not?

Hint: 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;
```

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 a 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) In the CODING section (Step 5) you will probably have just 2-3 code refinement levels (the first will just add function headers and trailers, as shown in class).
5) Due to the simple structures of the programs, the TESTING section (Step 6) should be pretty easy. But document it well.
6) Using mono-spaced fonts like (Courier) for source code enhances code readability and quality. Please use this type of font in your report only for the source code.

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)

IMPORTANT: In addition to the above "standard" assignment components, you must provide a file with the table, the plots, and discussion as required in Item 3 and 4 of the Program Specification.

Submit your complete assignment package (all files) via Elearning as a *zip* file. Please use **<hw#_lastname.zip>** as the naming convention for zipped file.

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