CS 370 – Project 5A, Threads

Purpose: Become familiar with operating system interaction, threading, and race conditions.
Points: 100 40 for program and 60 for write-up
Scoring will include functionality, documentation, coding style, and write-up

Assignment

In recreational mathematics, a Evil number[1] is a positive integer that has an even number of 1's in the binary representation and an Odious number[2] is a positive integer that has an odd number of 1's in the binary representation. For example, the numbers 6, 8, 27, 1235 are Evil Numbers while 7, 21, 26, and 1234 are Odious numbers.

There are 11 types of people, Those who understand binary, those who don't understand binary, and those who are sick of seeing this binary joke.

Write a **C** language program to find the count of Evil and Odious numbers between 1 and some user provided limit. In order to improve performance, the program should use threads to perform some of the computations in parallel. The program should read the thread count option and number limit in undecimal from the command line in the following format; "./evilNums -t <threadCount> -l <limitValue> -f <1|2>". For example, for 4 threads and a limit of 3000:

```
./evilNums -t 4 -l 3000 -f 1
```

The following routines, called by the provided main, should be created:

- A boolean function, ***getArguments()***, that reads and verifies the command line arguments. This includes error and range checking based on provided parameters. The error strings are defined in the example output. Due to the potential size of the limit, unsigned long data type should be used. The function option must be 1 or 2 and is described in the applicable section below.
- A void* function, ***findEvilNumbersCnt1()*** and ***findEvilNumbersCnt1()*** that will be called as a thread function and determine the count of Evil numbers. The specific function called will be based on the user input (function option 1 or 2). The thread must perform updates to the global variables in a critical section and avoid race conditions. See below sections for additional detail. Refer to the example output for formatting details.

Implement this specific threaded problem in **C** (not C++). The program must compile and execute under Ubuntu (and will only be tested with Ubuntu). Assignments not executing under Ubuntu will not be scored. See additional guidance on following pages.

1 For more information, refer to: https://en.wikipedia.org/wiki/Evil_number
2 For more information, refer to: https://en.wikipedia.org/wiki/Odious_number

Thread Function

The general flow of the ***findEvilNumbersCnt1()*** and ***findEvilNumbersCnt2()*** thread functions is as follows:

- Obtain the next block of numbers to check (via global variable, ***currentIndex***)
  - increment the global counter by the block size
  - must be performed within a critical section (i.e., locked and unlocked)
- While the next current number in the block is ≤ ***userLimit*** (globally available);
  - Check if the number is a Evil/Odious number
    - If the number is a Evil number, increment a local Evil number count
    - If the number is a Odious number, increment a local Odious number count
- Update evil/odious count (as described below)

You should have two versions (separate copies) of this function based on the following:

> **Function Option 1** → Once an Evil/Odious number has been found, the global counters should be updated immediately. The update for the ***evilNumberCount*** and ***odiousNumberCount*** must use a lock.
>
> **Function Option 2** → Only after the count of Evil/Odious numbers for the entire block has been found, the global counters should be updated. The update for the ***evilNumberCount*** and ***odiousNumberCount*** must use a lock.

It is recommended to complete and test the program initially with only the single sequential thread before testing the parallel threads. In order to debug, you may wish to temporarily insert a direct call (non-threaded) to the ***findEvilNumbersCnt()*** function (temporarily for testing and debugging).

Global Parameters

In order to use threading functions and/or semaphores in C, you will need the below include files:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <pthread.h>
#include <time.h>
#include <stdbool.h>
#include <errno.h>                // for errno
```

The following should be declared globally and used as parameters for the program.

```
const unsigned long   MIN_THREADS = 1;
const unsigned long   MAX_THREADS = 24;
const unsigned long   MIN_LIMIT = 100;
const unsigned long   BLOCK = 5000;
unsigned long         currentIndex = 1;
unsigned long         userLimit = 0;
unsigned long         evilNumberCount = 0;
unsigned long         odiousNumberCount = 0;
```

Project 5A - Timing Script

In order to obtain the times for the write-up a timing script is provided. After you download the script file, **`p5atimer`**, set the execute permission as follows:

```
ed-vm$ chmod +x p5atimer
ed-vm$ ./p5atimer evilNums
```

The script file will expect the name of the executable as an argument (as shown). The script may take a while to execute (i.e., 10+ minutes), but no interaction is required. If needed, for older machines, they limit may be reduced. Be sure you include examples where the evil and odious values are not the same. The script will create an output file, **`p5atimes.txt`** and **`p5aresults.txt`** which will be included in the submission and the data be used create the write-up.

Compilation Options

Use the following compiler options

```
gcc -Wall -pedantic -pthread -o evilNums evilNums.c
```

Points will be removed for poor coding practices and unresolved warning messages.

Results Write-Up

When the program is working, complete additional timing and testing actions as follows;

- Use the provided script file to execute and time the working program.
  - The script will execute the program with 1, 2, 3, 4, 5, and 6 threads and create
    - the final results (in **`p5aresults.txt`**)
    - the execution times (in **`p5atimes.txt`**)
- Compute the speed-up[3] factor from the base sequential execution and the parallel execution times. Use the following formula to calculate the speed-up factor:

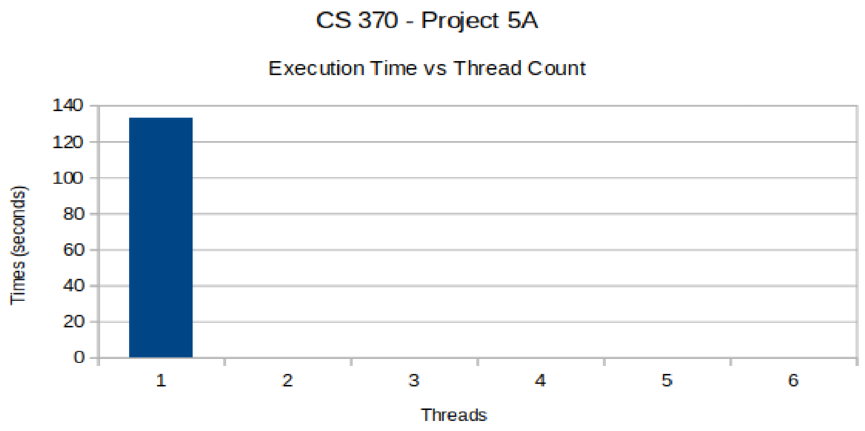
$$SpeedUp = \frac{ExecTime_{sequential}}{ExecTime_{parallel}}$$

  Note, you may use the provided spreadsheet is assist in the calculations and creation of a chart.

- Create a copy of original working program that comments out the locks (all places in both functions). Execute the modified copy of the program using 6 threads five (5) times using a limit of 2,000,000,000. Include the results (final values only) for each test in your write up along with an explanation.
- Explain the difference, if any between the two functions.

3 For more information, refer to: https://en.wikipedia.org/wiki/Speedup

- Create a final write-up as follows:
  - The writeup must include the following sections:
    - Summary
      - Summary of machine the program was tested on and the final results for the 1-6 thread executions, including final results (from **`p5aresults.txt`**) and the execution times (from **`p5atimes.txt`**). This should include a chart (see example).
    - Speed-Up
      - Explain the speed changes between 1 through 6 threads and why the speed-up factor changes as it does. Suggest some ways to improve.



    - Lock Removal
      - Explain what happened and why when the locking calls were removed.
    - Evil/Odious Counter Update Relocation
      - Explain what happened and why when the evil/odious number counters was moved as specified.

  - The explanation part of the write-up (not including the output and timing data) should be less than 500 words. Overly long explanations will be not be scored. The report should include appropriate titles and section headers. Additionally, spelling and grammar will be part of the final report score.

Example Execution
The following are some example executions:

```
ed-vm%
ed-vm% ./evilNums -t 4 -lm  -f 1
Error, invalid command line options.
ed-vm%
ed-vm% ./evilNums -tx 2 -l 100000000 -f 1
Error, invalid thread count specifier.
ed-vm%
ed-vm% ./evilNums -t 51 -l 100000000 -f 1
Error, thread count out of range.
ed-vm%
ed-vm% ./evilNums -t x -l 100000000 -f 1
Error, invalid thread count value.
ed-vm%
ed-vm% ./evilNums -t 2 -lmm 100000000 -f 1
Error, invalid limit specifier.
ed-vm%
ed-vm% ./evilNums -t 3 -l 10000x000 -f 2
Error, invalid limit value.
ed-vm%
ed-vm% ./evilNums -t 2 -l 1 -f 1
Error, limit must be > 100.
ed-vm%
ed-vm% ./evilNums
Usage: ./evilNums -t <threadCount> -l <limitValue> -f <1|2>
ed-vm%
ed-vm% ./evilNums -t 2 -l 100000000 -f 1
CS 370 - Project #5-A
Evil/Odious Numbers Program

Hardware Cores: 12
Thread Count:   2
Numbers Limit:  100000003

Please wait. Running...

Evil/Odious Numbers Results
  Evil Number Count:   50000001
  Odious Number Count: 50000002
ed-vm%
ed-vm%
```

Submission
When complete, submit:

- A copy of the final working **C** source code (not C++) file.
    - Do not submit the modified test versions (used for answering the write-up questions).
- A PDF of the write-up (as described above).

Submissions received after the due date/time will not be accepted.