HW2: Pthreads #1

Due Midnight 1/30/2020

For this assignment you will write a naive multithreaded parallel prime number generator using pthreads. Basically the goal is this: launch `NUM_THREADS` individual threads, each engaged in a potentially time-consuming brute force algorithm (described below) and finally join on all the threads collecting the generated numbers.

(A followup tweak to this assignment will involve using synchronization primitives to wait until `NUM_RESULTS` are generated by the threads, and then programatically terminating the program.)

Rubric

Program compiles with no warnings	30
Code launches a configurable number of threads and collects results	30
Results are correct	30
Code style is professional* and the submission contains the correct files	10

*Professional code style demonstrates consistent indentation and good identifiers (variable and function) names. Strive for readable code, using comments only when an algorithm is unclear. Document subroutine definitions in block comments before the subroutine definition.

Please note: file names should not contain spaces. Code that fails to compile will not be graded.

Thread task

The task given to your threads is naive and computationally expensive, by design. Each thread should perform the following steps:

1. randomly generate an `unsigned long int`
2. test for primality by looking for factors with brute force iteration (iteratively divide by numbers until a remainder is zero).
3. repeat 1-2 until a number is found or `MAX_TRIES` has been reached.

Upon completion of this algorithm, the thread should return a value to the spawning thread (via `return` or `pthread_exit`) that is either the random prime, or a 0 value indicating that `MAX_TRIES` has been reached.

In my code, I'm setting `NUM_THREADS` to 100 and `MAX_TRIES` to 10. Make these variable or preprocessor macros so that you can experiment.

Random number generation

To generate your `unsigned long int`, use the following code snippet. I have verified that this works in Cygwin and MacOS (`#include <stdlib.h>`). On linux, it requires installation of the `libbsd-devel` package, or equivalent (`#include <bsd/stdlib.h>`). If you find a better approach, let me know!

Note: this code uses modulo to limit the maximum random value, to better control the time of execution. Experiment with `MAXVAL`! Mine is set to `99999999999999999`.

```
unsigned long int to_test;
arc4random_buf(&to_test, sizeof(to_test));
to_test = to_test % MAXVAL;
```

In this code snippet, `to_test` contains the value that you will test for primality.

Sample output

Here's the output of my program with 30 threads and `MAX_TRIES` set to 10:

```
thread 0 reports no prime
thread 1 reports no prime
thread 2 reports no prime
thread 3 reports no prime
thread 4 reports no prime
thread 5 reports no prime
thread 6 reports no prime
thread 7 reports no prime
thread 8 reports no prime
thread 9 reports that 47638751574261677 is prime
thread 10 reports that 41373699410331529 is prime
thread 11 reports no prime
thread 12 reports no prime
thread 13 reports that 27187083255359519 is prime
thread 14 reports no prime
thread 15 reports that 9563539263957823 is prime
thread 16 reports no prime
thread 17 reports no prime
thread 18 reports no prime
thread 19 reports that 74646019533938029 is prime
```

```
thread 20 reports no prime
thread 21 reports that 22684060040044307 is prime
thread 22 reports no prime
thread 23 reports no prime
thread 24 reports no prime
thread 25 reports no prime
thread 26 reports no prime
thread 27 reports no prime
thread 28 reports that 77387671998363191 is prime
thread 29 reports no prime
```

You can easily validate the prime numbers generated using https://www.wolframalpha.com .

Turn in . . .

A zip file containing your single source code file and a text file showing the contents of your Bash prompt, compiling the code using `-Wall` and `-O3` flags, and running the program with 100 threads.