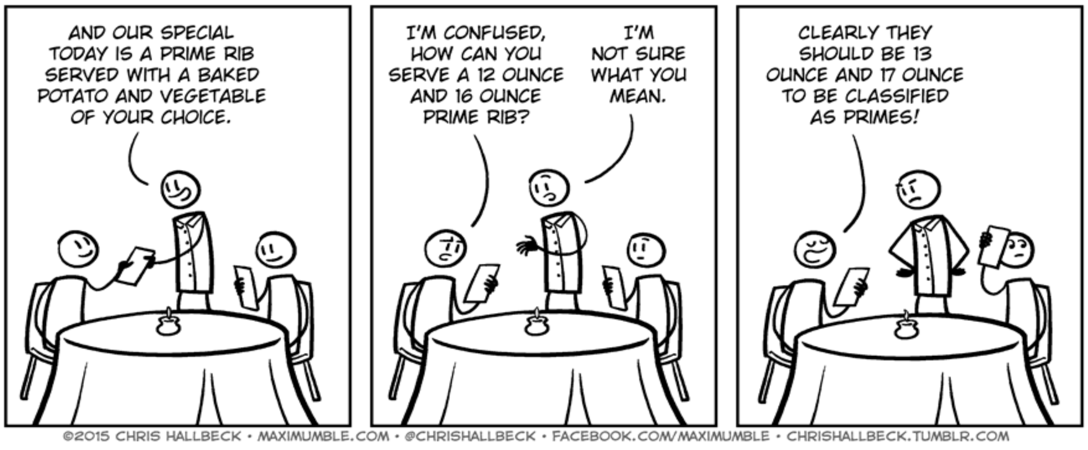
CS 370 – Threading Project

Purpose: Become more familiar with operating system interaction, threading, and race conditions.
Due: 3/2
Points: 100 60 for program and 40 for write-up
Scoring will include functionality, documentation, and coding style

Assignment:

A twin prime[1] is a prime number that is 2 more than another prime number. For example, the twin prime pair of (41, 43). In other words, a twin prime is a prime that has a prime gap of two. As such, once a prime is found, we can check to see if that number -2 or +2 is also a prime. If the -2 or +2 number is also a prime, then a twin prime pair has been found. Continue this until the limit is reached.


AND OUR SPECIAL TODAY IS A PRIME RIB SERVED WITH A BAKED POTATO AND VEGETABLE OF YOUR CHOICE.
I'M CONFUSED, HOW CAN YOU SERVE A 12 OUNCE AND 16 OUNCE PRIME RIB?
I'M NOT SURE WHAT YOU MEAN.
CLEARLY THEY SHOULD BE 13 OUNCE AND 17 OUNCE TO BE CLASSIFIED AS PRIMES!
©2015 CHRIS HALLBECK • MAXIMUMBLE.COM • @CHRISHALLBECK • FACEBOOK.COM/MAXIMUMBLE • CHRISHALLBECK.TUMBLR.COM

The goal is to find twin prime pairs. For example, there are 8 twin prime pairs between 1 and 100; (3,5), (5,7), (11,13), (17,19), (29,31), (41,43), (59,61), and (71,73).

Write a C language program to provide a count of twin primes between 1 and a user provided ***limit*** or maximum value. For example, between 1 and 10,000,000 there are exactly 58,980 twin prime numbers. In order to improve performance, the program should use threads to perform computations in parallel.

When complete, create an executable that, based on the command line option, uses either one (1) thread or four (4) threads. Also, create a simple bash script to execute the each three (3) times with a limit of 10,000,000. You will need to take the average execution time for the single threaded executions and the four (4) thread executions and compute the speed-up achieved using the formula:

$$\frac{executionSpeed_{oneThread}}{executionSpeed_{fourThreads}}$$

The reference, POSIX thread (pthread) libraries, may be useful along with the C Thread Example (last page).

1 For more information, refer to: https://en.wikipedia.org/wiki/Twin_prime

Command Line Arguments:
The program should read the sequential or parallel option and maximum number limit from the command line in the following format; "./numCount <-s|-t> <limitValue>". For example:

```
./twinPrimes -s 10000000
```

If either option is not correct, an applicable error message should be displayed. The minimum acceptable limit is 100. If no arguments are entered, a usage message should be displayed. Refer to the sample executions for examples of the appropriate error messages.

Locking
When changing global variables, a mutex must be used. For this usage, the mutex variables are declared globally.

```
// global declaration
pthread_mutex_t       myLock;    // mutex variable


// initialize myLock mutex.
pthread_mutex_init(&myLock, NULL);


// code section
pthread_mutex_lock(&myLock);
// critical section code
pthread_mutex_unlock(&myLock);
```

The program should use two mutexes; one mutex for the global counter and one for the counters.

Submission:
When complete, submit:

- A copy of the **C** source code file (not C++).

- Submit a write-up (PDF format) including the following topics:
  - Copy of the program output from the script file (above)
    - An explanation of the results (sequential and threaded) with calculated speed-up.
  - When the program is working:
    - Comment out the lock and unlock calls and execute for both sequential and threaded.
      - *Note*, use an appropriately large value for the limit.
    - Report the results.
    - Explain the difference with and without the locking calls for the parallel execution
  - *Note*, overly long explanations will be not be scored.

- A copy of the bash script use for the timing.
  - *Note*, you may halve the 10,000,000 if needed.

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

Example Execution:

The following are some example executions, including the required error checking:

```
ed-vm%
ed-vm% time ./twinPrimes -s 10000000
Twin Primes Program
Please wait. Running...

Count of Twin Primes between 1 and 10000000
is ????

real     2m11.053s
user     2m11.012s
sys      0m0.036s
ed-vm%
ed-vm%
ed-vm% time ./twinPrimes -t 10000000
Twin Primes Program
Please wait. Running...

Count of Twin Primes between 1 and 10000000
is ????

real     0m33.668s
user     2m14.147s
sys      0m0.360s
ed-vm%
ed-vm%
ed-vm% ./twinPrimes st 10000
Error, invalid command line arguments.
ed-vm%
ed-vm%
ed-vm% ./twinPrimes s 10x000
Error, invalid limit value.
ed-vm%
ed-vm%
ed-vm% ./twinPrimes -s 100.00
Error, invalid limit value.
ed-vm%
ed-vm%
ed-vm% ./twinPrimes
Usage: ./twinPrimes <-s|-p> <limitValue>
ed-vm%
ed-vm%
ed-vm% ./twinPrimes -s 99
Error, limit must be > 100.
ed-vm%
```

Note, the timing shown is for one specific machine. Actual mileage may vary.

C Thread Example

```c
// CS 370 C Thread Example. Useless example of how threads are used in C.
// use:  gcc -Wall -pedantic -g -pthread -o threadExp threadExp.c

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

//  Global variables
pthread_mutex_t           myLock;       // mutex variable
int                       j = 0;        // global shared variable

//  Thread function, just prints the global variable five times.
void * do_process() {
   int i = 0;

   pthread_mutex_lock(&myLock);
   printf("\nThread Start\n");

   j++;

   while(i < 5) {
             printf("%d", j);
             sleep(0.5);
             i++;
   }

   printf("Thread Done\n");
   pthread_mutex_unlock(&myLock);

   return    NULL;
}

int main(void)
{
   unsigned long int      thdErr1, thdErr2, mtxErr;
   pthread_t        thd1, thd2;

   printf("C Threading Example.\n");

   //  Initialize myLock mutex.
   mtxErr = pthread_mutex_init(&myLock, NULL);

   if (mtxErr != 0)
             perror("Mutex initialization failed.\n");

   //  Create two threads.
   thdErr1 = pthread_create(&thd1, NULL, &do_process, NULL);
   if (thdErr1 != 0)
             perror("Thread 1 fail to create.\n");

   thdErr2 = pthread_create(&thd2, NULL, &do_process, NULL);
   if (thdErr2 != 0)
             perror("Thread 2 fail to create.\n");

   //  Wait for threads to complete.
   pthread_join(thd1, NULL);
   pthread_join(thd2, NULL);

   //  Threads done, show final result.
   printf("\nFinal value of global variable: %d \n", j);

   return    0;
}
```