In this assignment, you will write a threaded program in which threads write data to a shared buffer. You must ensure that a race condition does not occur. For this assignment, you may work individually or in a group with one other student. You will write two versions of the program. The first version will use Pthreads (POSIX threads) and the second version will use C++ threads (which are built on top of Pthreads). This assignment is to be completed individually.

Assignment

Implement a multithreaded solution to find if a given number is a perfect number. (https://en.wikipedia.org/wiki/Perfect_number) N is a perfect number if the sum of all its factors, excluding itself, is N. You can find examples of perfect numbers at https://en.wikipedia.org/wiki/List_of_perfect_numbers.

You should ask the user for a number and the number of threads to use. The main program must create the required number of threads. The numbers 1 to N will be partitioned among these threads so that two threads do not work on the same number. For each number in this set, the thread will determine if the number is a factor of N. The parent thread should wait until all the threads complete. The parent thread will then determine if the input number is perfect and report the result. In addition to reporting the results, the parent thread must print the factors in the order they were added to the array.

Setup

In CodeLite on your virtual machine:

- Create two C++ projects named Project3_Pthreads and Project3_CPPthreads.
- For this project, you do not need to create a class. You may write all of your code in the main.cpp file.

Submitting Your Assignment

Submit the sourse code (.cpp files) from each of your projects in a separate zip file. Name the zip files YourLastNameYourFirstNameProject3_PThreads.zip and YourLastNameYourFirstNameProject3_CPPThreads.zip. Upload these zip file to Canvas before the due date.

Important submission notes for students working in a group:

If you worked with another student, name the zip files using the last names of both students. For example, name the Pthreads submission:

FirstPersonsLastNameSecondPersonsLastNameProject3_Pthreads.zip

Use a similar style for the C++ submission.

One person should upload the zip files to Canvas. The other person should upload a single document stating who their team member was.

Remember that late assignments are not accepted in this course. See the syllabus for details.

Implementation Notes

- I would suggest that you write a non-threaded version of the program first to make sure you understand the algorithm. You do not need to turn in the non-threaded version.
- For both threaded versions you will need to write a function that can be used as the starting routine for the thread. The Pthread version of this function must have one parameter, a void pointer. The C++ version must have three parameters: the value of the number it is checking, the starting value for potential factors, and the ending value for potential factors.
    - You really only need to check factors between 1 and number / 2. Any number greater than number / 2 will not be a factor.
    - For example, if the number I am checking is 28 and I have requested 2 threads, one thread might check factors from 1 to 7 and the second thread might check factors from 8 to 14.
- For both implementations (Pthreads and C++ threads), you will need to link in the Pthreads library. In CodeLite, you can do this by right-clicking on the project and selecting "Settings." Choose the "Linker" tab and enter *-lpthread* in "Linker Options."
- You may assume that the number we are checking can be stored in an uint64_t.
- Your threads must write the factors that they find to a shared buffer. You must implement this buffer as an array of uint64_t.
    - An array of size of 100 is sufficient for this project.

Useful information

- I have created several videos on setting up and using the PThreads and C++ threads libraries. These videos are available on Canvas.
- A useful PThreads tutorial is available at https://computing.llnl.gov/tutorials/pthreads/
    - You may find the "Passing arguments to Threads" especially helpful.
- For the C++ Threads version of your project you must use the C++ threads class (https://en.cppreference.com/w/cpp/thread/thread) and C++ mutex class (https://en.cppreference.com/w/cpp/thread/mutex).

Sample Input and Output

Here are two sample executions of the program:

Sample Execution 1

```
Enter number: 8589869056
Number of threads: 7

Number of factors is: 33
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131071
262142
524284
1048568
2097136
4194272
8388544
16777088
33554176
67108352
134216704
268433408
2147467264
1073733632
536866816
4294934528

8589869056 is a perfect number
```

Sample Execution 2

```
Enter number: 456987
Number of threads: 3
```

```
Number of factors is: 15
1
3
23
37
69
152329
111
179
537
851
2553
4117
6623
12351
19869

456987 is not a perfect number.
```

Grading Criteria (15 points possible)
Note: Your program must compile to receive credit for this assignment!

Points	Criteria
0 -1 point	**Input:** Does the main function of each version of the program prompt the user for a number to check and the number of threads and correctly read these values into variables?
0 -2 points	**Partitioning:** Does the main function of each version of the program correctly partition the potential between the threads?
0 - 3 points	**PThread Creation and Join:** Does the Pthreads version correctly create the threads? Are arguments correctly passed to the start routine? Are the threads joined correctly?
0 - 2 point	**PThread Start Routine:** Does the start routine used for the PThread version contain the type of parameter? Is the correct return type used? Are the three values (number being checked, start value, ending value) correctly extracted from the parameter?
0 -2 points	**C++ Thread Creation and Join:** Does the C++ version correctly create the threads using the C++ threads library constructor? Are arguments correctly passed to the start routine? Are the threads joined correctly?
0 - 1 point	**C++ Start Routine:** Does the start routine used for the C++ version contain the correct number and types of parameters?
0 - 2 points	**Start Routine:** Does that start routine of each version correctly write the factors into the buffer (the array)? Is a mutex used to prevent a race condition in the critical section?
0-1 point	**Output:** Does each version print the factors to standard output in the order they were added to the array? Is the correct result (perfect or not perfect) printed to standard output?
0-1 point	**Style:** Is the code easy to read? Does it follow class style guidelines (See Program Style page in Canvas.)