Program Design

The program design of this assignment is the most important part. I attempted, and found it near impossible to actually begin to program anything without a legitimate plan behind me. Through many sketches on my whiteboard, I finally decided that the way I'm going to write this program is to have each thread or process mark off multiples of primes, and then repeat until currentprime*currentprime is greater than the max number to check until. This will work, because as soon as one thread's currentprime*currentprime is greater than the max, it means the sieve was done. I had to look at a lot of slides on how exactly the Sieve of Eratosthenes works, but now that I've read the slides, that mathematical idea makes a lot of sense. I will have to have the threads grab a new prime after they are done marking multiples, and I'm not sure how I'm going to do that yet.

I intend to use an array of unsigned characters for the bitmap: unsigned char bitmap[UINT MAX]

Deviations From My Design

Although I could declare a global array of unsigned char bitmap[UINT MAX] in my threaded version, it simply didn't work with the shared memory version. Since the shared memory object can only be so big, I had to use bitmask functions to only use one bit at a time, as opposed to a byte. My implementation using the global array wastes 7/8 of it's allocated space!

Why Did I Pick This Design?

Quite honestly, this is the only way that the parallelization of the Sieve of Eratosthenes makes sense to me. I also looked at many slides explaining block decomposition, but it didn't click in my head whatsoever. Although block decomposition may be slightly faster, I anticipate that the program will be significantly easier to design and implement using each thread to mark off multiples of increasingly large primes until currentprime*currentprime ¿ max.

Problems Faced

- Figuring out how to pass a fresh prime to each thread on each iteration of my sieving function was incredibly hard. It took me nearly two days to decipher a solution. I ended up using a queue, on which the mutex was used. I would lock the mutex before any thread was to dequeue a prime to use. The thread would then figure out what the next prime was, and then enqueue it for the next thread to pick up. After that, it would unlock and find the multiples of whichever number it dequeue'd.
- Another major issue I faced was realizing that all of my numbers(mainly loop variables) had to be initialized as longs. Originally I had them as ints, which clearly led to extreme overflow issues. I then used unsigned ints, which didn't quite work but were extremely close. Only after talking to Professor McGrath on Piazza did I realize that longs were neccesary. After I realized that, many problems I had were fixed.
- A huge issue I am facing is figuring out the size of the shared memory object. Since the bitmap must have UINT MAX usable spots, I have to designate that much room in the shared memory object, to

my understanding. However I am unable to make an object that big so far, which is leaving me with a strange segmentation fault.

- Even after it stopped seg faulting, it doesn't print the correct values into the file I specify. I cannot figure out why.
- I finally figured it out! It worked great for a while and then began printing negative numbers...
- After finishing and looking back, the hardest part of this assignment was easily the parallelization of the sieve. Although I only have about 350-400 lines total, I spent double the amount of time I have ever spent on a programming project. I put in over 30 hours of work on this assignment. I would never imagine myself saying this about any other assignment, but the 30 hours I spent on this one were legitimately fun.

Work Log

November 15 - Implemented a non parallel Sieve of Eratosthenes. Found bitmap/bitmask code and began using it in my program

November 16 - Implemented parallelize() function, which is used during pthread create(). Extremely buggy

November 17 - Removed parallelize() function and decided to make it parallel using only one function.

November 18 - Spent a few hours trying to figure out how to parallelize it, to no avail.

November 19 - GOTO November 18

November 23 - Figured out how to simply use the threads to only mark multiples, instead of running an entire sieve. Things are making more sense.

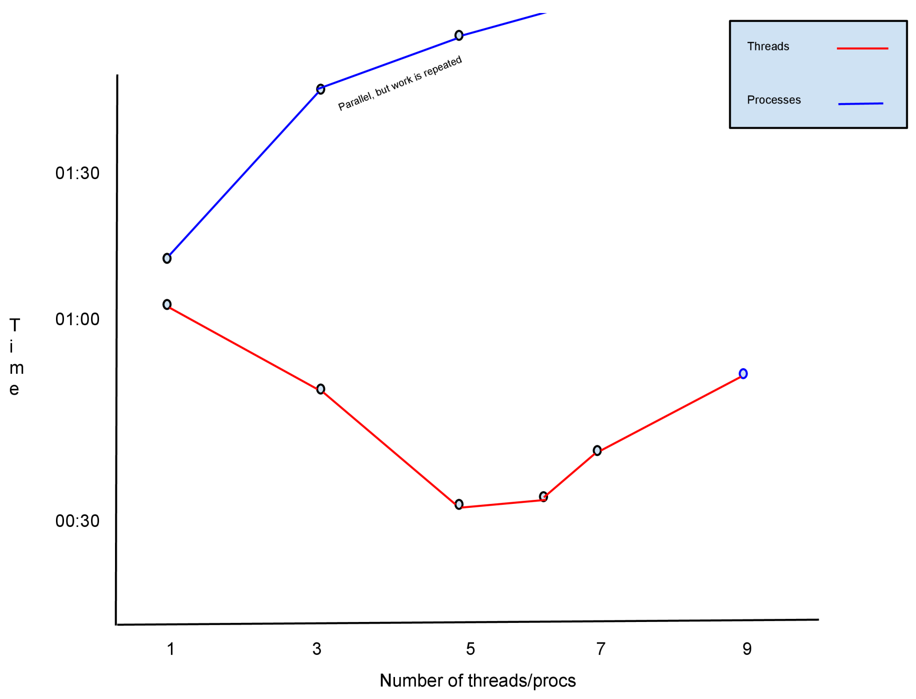
November 24 - Finished the parallelization of threads, and got runtime down to 36 seconds.

November 25 - Got mount shmem() function from the class website, and got the basics working within my process-based version of the program.

November 26 - Continued working on process/shared memory version, nothing significant accomplished.

November 27 - Spent all day in Kelley implementing the rest of the shared memory version, and got it running in approximately 2 minutes. Also implemented the happy number checking and signals.

Timings


Threads
Processes
Parallel, but work is repeated
01:30
01:00
00:30
Time
1
3
5
7
9
Number of threads/procs