Project 4 Write Up

Sam Quinn

December 5, 2013

Design

pro.prime.c and thr.prime.c Programs

These programs will find the total number of primes up to a given number with the supplied number of threads or process. This program can theoretically calculate the primes up to the size of an unsigned long but I have only tested it to the size of an unsigned int. These programs will also find the happy prime numbers up to the same number.

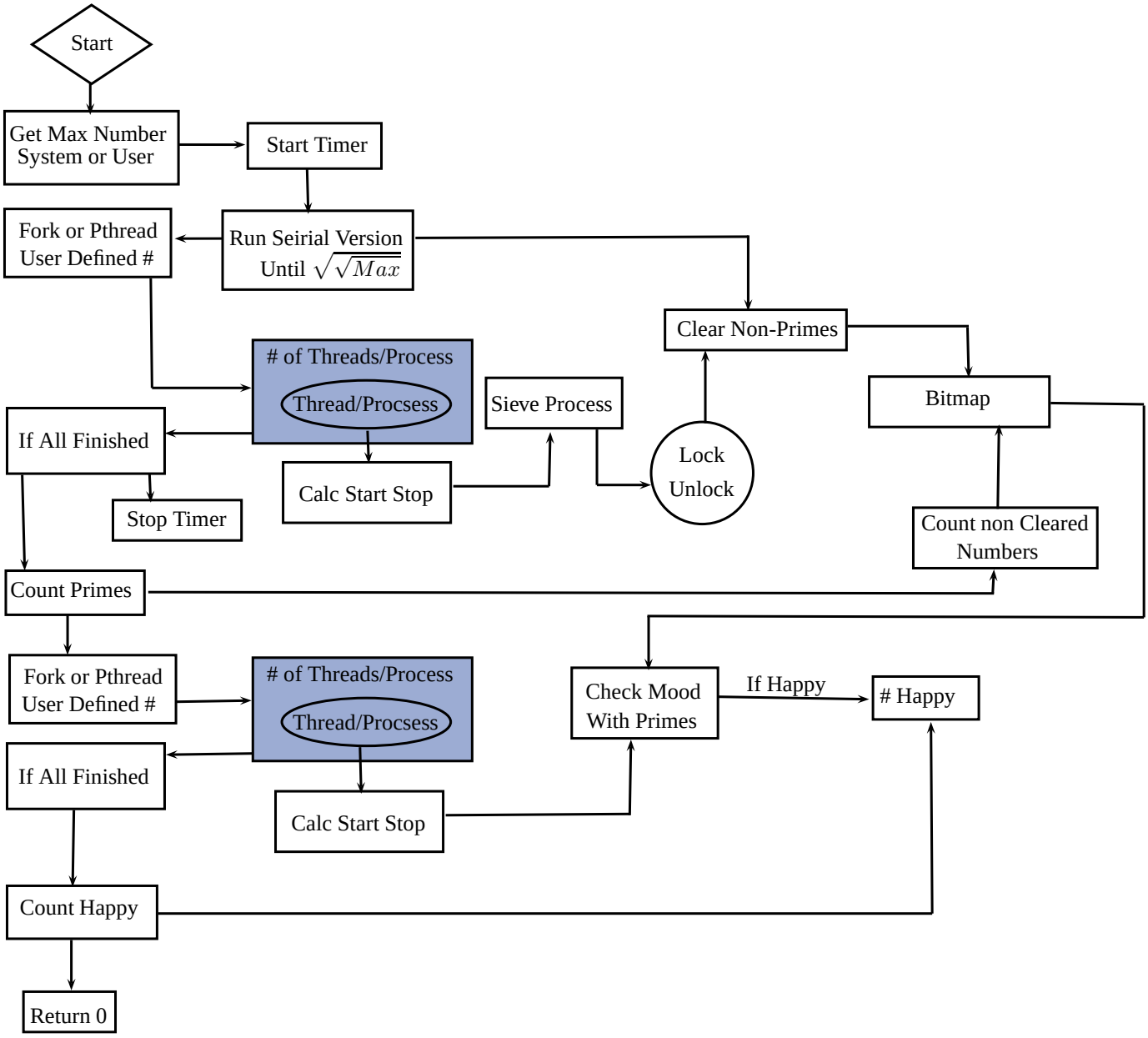
Threaded Version

- if no options/arguments, print usage
- -n is the max prime, will find all primes up to this number.
- -i will set the max prime to size of an int, will find all primes up to this number.
- -u will set the max prime to size of an unsigned int, will find all primes up to this number.
- -t will set the number of threads to use.
- -v will print the results to the terminal (NOT recommended to exceed 10,000 as for the printing process takes a lot of time.)
- -m If defined in the command line no mutexes will be used.

Process Version

- if no options/arguments, print usage
- -n is the max prime, will find all primes up to this number.
- -i will set the max prime to size of an int, will find all primes up to this number.
- -u will set the max prime to size of an unsigned int, will find all primes up to this number.
- -p will set the number of processes to use.
- -v will print the results to the terminal (NOT recommended to exceed 10,000 as for the printing process takes a lot of time.)

Functions of Program Details



Finding Primes

- Receive number from user.
- Run a serial version of Sieve only to the $\sqrt{\sqrt{Number}}$
- Spawn threads or processes giving each thread/process its number.
- Calculate what numbers that process needs to work with.
- Find the first prime with in the range calculated above.

- First square the prime and mark it as not prime.
- Then add 2∗prime to the previous number and mark as not prime too.
- Continue this process until the thread/process has iterated through its work load.

The user will either choose an arbitrary number or use one of the predefined numbers from the command line which will be set as a global variable so every function can access it with out needing to pass it around. I stored weather the number is prime or not in a bitmap that defines every odd number as prime at first. This bitmap excludes every even number because except for 2 no even number is prime. When spawning threads or processes I send the number of thread/process it is with that thread/process. The benefit of the thread/process knowing what number it is that with in each thread/process they calculate their beginning and ending range to work on. Each thread/process has a different section of numbers to work on with the last thread/process's max being the $\sqrt{Max}$. I have the work split up so theoretically no thread/process should run in to each other∗. Each thread/process will run in parallel using the Sieve of Eratosthenes formula to find all of the non primes under the number.

Note∗
However there is still a race case that comes from different non primes while getting marked off as non prime try to access the same part of the bitmap.

Finding Happy Numbers

- Spawn the amount of threads/process that the user decided
- Calculate the work space start and end number for that process
- Read from the bit map and for every prime it finds check if it is happy or not.

This function uses the work already done from the find primes function to determine if the prime is happy or sad. It will use the same process to determine the work space for each thread/process as in the find primes function. Since we have already weeded out the non primes this function will scan through the bitmap and check only the primes to determine their mood.

Work Log

```
commit 68590883a43a78b4be79f5191160cf29d1e54e3d
Author: Sam Quinn <quinnsa@os-class.engr.oregonstate.edu>
Date:   Wed Nov 27 20:59:59 2013 -0800
```

Final commit. It has been good homework. I think I let you pro_prime.c get the best of me, but you guys have done me good. Farewell, see you soon during the demo. Till then Peace my brothers!

```
commit 2dc1be6628162d334ef4f50d9c60adee3a36ae32
Author: Sam Quinn <quinnsa@os-class.engr.oregonstate.edu>
Date:   Wed Nov 27 20:04:01 2013 -0800
```

Just going through and making everything look purty for submission.

```
commit e0bee53c8fea364bfc573cb2a418ea87a82dc212
Author: Sam Quinn <quinnsa@os-class.engr.oregonstate.edu>
Date:   Wed Nov 27 10:46:53 2013 -0800
```

So I was actually wrong about being able to get rid of my mutexes it was just a coincidence that i got the right number 3 times in a row with out mutexes. I still need them. In this version of the threaded version I added a few more command line arguments like -v to print the primes and happy numbers and -m to run the program with out mutexes. mainly for timing purposes I don't want to have to wait around for 3-4 minutes ever time I run the dang thing.

```
commit 87ff777a1aecbd4b4900dc929f9b826c23108095
Author: Sam Quinn <quinnsa@os-class.engr.oregonstate.edu>
Date:   Tue Nov 26 23:42:36 2013 -0800
```

Oh yeah!!!!! I just found the reason why my code was going so slow! while working with my happy numbers function it wasnt working with my next prime function and i found the part of code that was producing race cases. I was only returning the next prime up to the Sqrt(max) not all the way up to the max. This allowed me to get rid of my mutexes which bring my runtime down from 300+ seconds to a nice number like 32! Wooohoo this is good man!

```
commit f6df108ef06abb657a37dd187c61ed9b387bdb51
Author: Sam Quinn <quinnsa@os-class.engr.oregonstate.edu>
Date:   Tue Nov 26 19:13:55 2013 -0800
```

Minor fixes on my threaded function just a few house keeping things. My process version of the program I for some reason for the life of me can't get my semaphores working it is really frustrating because after I get that working I am pretty much done.

```
commit d0156bbdc3e98fb81ed70edc8d717b9b31f3891c
Author: Sam Quinn <quinnsa@os-class.engr.oregonstate.edu>
Date:   Mon Nov 25 12:51:16 2013 -0800
```

Threaded version – pretty much works great still very slow since i have to use mutexes. I how ever did split the mutexes up into 20 separate ones so in theory there can be 20 separate access to the bitmap at a time but it still slows it down exponentially.

Process version – Have not made much progress since last commit am pretty much solely working on that now since my threaded version works well.

```
commit 68fdd6dfd8213852ac06a7a50929bd274b67194a
Author: Sam Quinn <quinnsa@os-class.engr.oregonstate.edu>
Date:   Sun Nov 24 18:41:44 2013 -0800
```

Threaded process works still no happy function though. Added the process version of the program.

```
commit 77099cc69c5371340011edee1a8ca8b6d85ee504
Author: Sam Quinn <quinnsa@os-class.engr.oregonstate.edu>
Date:   Fri Nov 22 16:46:51 2013 -0800
```

My threaded program works! have not implemented counting happy or sad numbers yet. It is also very very slow 10 minutes to calculate UINT_MAX :(

```
commit bb9d94d4815a1283b179f8c498149c2a8dd6deec
Author: Sam Quinn <quinnsa@os-class.engr.oregonstate.edu>
Date:   Thu Nov 21 00:31:29 2013 -0800
```

My threaded version works pretty well for values under 10,000 for some reason after that it becomes buggy.

```
commit 6a675cb19a7f7be52bb969b0aece7ab3a58544b5
Author: Sam Quinn <quinnsa@os-class.engr.oregonstate.edu>
Date:   Wed Nov 20 00:34:22 2013 -0800
```

Coming along I have multiple threads working and i have made beginning and end points for each thread to calculate with.

```
commit 6a9534ea07e72985119e6f6769f40614ded7a58d
Author: Sam Quinn <quinnsa@os-class.engr.oregonstate.edu>
Date:   Mon Nov 18 07:58:29 2013 -0800
```

Got my bitmap kind of working I need to add one because some numbers had a floating point decimal which makes the last bit not change.

```
commit 169c9830aa749b5303e3578b206b539b1e481018
Author: Sam Quinn <quinnsa@os-class.engr.oregonstate.edu>
Date:   Sun Nov 17 19:45:20 2013 -0800
```

Initial commit nothing really works yet!

Timings

For my Process version every time I ran the program I got a 0.00 second reading. So I have omitted the table of this program as it would be filled with 0.00's.

Table 1: Threaded Model Results

Max Number	10 Threads	32 Threads	200 Threads	500 Threads
1,000,000	0	0	0	0
100,000,000	4	3	3	4
1,000,000,000	37	31	32	33
2,147,483,647	90	85	76	73
4,294,967,295	260	229	250	273

Challenges Overcame

This program brought on a lot of frustration with both versions of the program. It was hard for me to get my program under 45 seconds. I heard a few people in class saying they got their functions to work with out using any sort of IPC, I however could never get mine to work without some sort of resource locking. In both versions of my program I have race cases on bitmap when two or more threads/processes are trying to write to the same word. This frustrated me so much is that if i get rid of the mutexes my programs runtime drops from around 230 seconds to 32 seconds. I also had a difficult time getting my semaphores working on my process version of the program. I know that you can use an unnamed semaphore if you set the pshared int to greater than 0, which I did, but for the life of me couldn't get the unnamed semaphores to work across multiple unrelated processes. The way I solved this problem was I ended up using a named semaphore which worked with out having to change much of my code. I had a very hard time getting the process version of this program working correctly. I had troubles the whole way through from creating a shared memory object to getting the right outputs. I thought this version of the program would have been easy since I have already used forking and multiple processes in past assignments, but this was not the case. I found the implicit sharing much easier to wrap my head around and far easier to implement. I am very disappointed with my self for not getting my process version working to the point i would like.