David Casente Ellis Saupe

Prof. Fietkiewicz

EECS 338

23 April 2018

Prime Number Generator and Checker

Documentation of primeGen.c:

Our main file in this final project is called primeGen.c, and will calculate first N prime numbers, which the user supplies to stdin using scanf. It will write all of the prime numbers sequentially into an output file which is called output.txt. We will accomplish this task by using a multithreaded producer and consumer, a bounded buffer, global array of primes, and the Sieve of Eratosthenes. We start with a buffered integer array of a large size where all the values are set initially to 1. The producers will go through the array eliminating composite numbers by going to indexes that are multiples of found prime numbers. They will change the value corresponding to composite numbers to zero, marking all the composite numbers, thus “producing” prime numbers. Since the Sieve of Eratosthenes is multi-threaded so we can have multiple producers going through multiples of primes; for p producers, each producer with have a unique offset and will mark only multiples of every pth prime number after the offset, i.e. 1, 1+p, 1+2p,… The consumer loops through the array checking to see if a value is a 1, meaning it is a prime number. If so, the consumer would “consume” them, putting the prime numbers into a global array where the producer can use them to eliminate more composite numbers further down in the array. If the index is not prime, the consumer will continuously reset the array by changing all values of indexes back to 1. This way when the producers or consumers reach the end of the array, they can just loop around to the beginning and the array can be reused. The producers and consumers will use counters to keep track of their actual index. The global array will hold all the prime numbers sequentially and will be written to an output file at the end. Because this uses a bounded buffer, global control variables are used to prevent either the producer or consumer from accessing invalid sections of the buffer. For example, if the consumer has not tested if 5 is prime with a buffer size of 10, the producer cannot mark 15 as composite because 5 and 15 have the same position in the bounded buffer. Synchronization of global variables is controlled with semaphores implementing the solution of the readers-writers problem that allows for multiple readers in a critical section without also starving the writers. When the consumer reaches the Nth prime, the global array will contain N primes, the threads will exit, and the prime numbers will be written to the output file.

In addition to the main idea of our code, we added some functions and initialization to streamline the process. First we added user interactions, asking the user how many cores can be used and how many primes they would like to generate. Based on the user’s response of c cores, the program will create c-1 producer threads and one consumer thread. Afterwards, the program will go about generating the primes as described above. Once prime generation is finished and the primes are written to the output file, the program will ask the user if it wants to check if a certain number is prime. The program will continue to ask and check until the user enters the number 0 at which point the program will terminate.

Sample command line execution:

./primeGen

How many cores available? (at least 2)

2

How many primes do you want to generate? (Up to 203280221)

10000

Finished finding 10000 primes, all numbers attached in output.txt

Are there are numbers you would like to check to see if prime?(Enter 0 if not)

3571

3571 is prime.

Are there are numbers you would like to check to see if prime?(Enter 0 if not)

3572

3572 is not prime.

Are there are numbers you would like to check to see if prime?(Enter 0 if not)

0

Code finished.

Group Work Breakdown:

While we both did some research on our own time, We, mainly worked on the project sitting together. First, we mapped out how we wanted to tackle the project, figuring out that we wanted to make a multi-threaded solution using a bounded buffer. In addition, we discussed possible pitfalls and difficulties in using other structures before settling on our current solution. At our next meeting, we first tackled the theoretical logic of our solution, bounced ideas off of each other, worked on how to calculate certain values, and wrote out some pseudo code to help visualize. Then we decided to divide and conquer the beta assignment, where David worked on mainly the report and output.txt and Ellis grinded out the base of our code and comments. David checked Ellis’ work from time to time and Ellis clarified David’s report questions. Since then, Ellis worked on the synchronization, the bounded buffer and major debugging. David took on the checker, file I/O, and the report. Overall, we enjoyed actively discussing our project in terms of coding as well as debugging, hypothesizing possible solutions, and creating a prime number generator that works for thousands of primes. We had a ton of fun working on this program, talking about theory and code, and hope to work on other projects outside of class in the future.