CS 452 – Lab2
Distributed Systems and Process Synchronization

Note: This is not a group assignment. This lab must be attempted on an individual basis.

This lab project, to be written in C or C++, is an exercise in process synchronization and IPC. You are to implement the **Eratosthenes Prime Number Sieve** with that in mind. This will be explained after the definition of the problem given below:

The most efficient way to find all of the small primes (< 10,000,000) is by using the **Eratosthenes Sieve.**

To find all prime numbers up to ***n***, list all numbers **k, 1 < k ≤ n**, and eliminate the multiples of all primes less than or equal to the square root of **n**. The numbers that are left are the primes.

Example:

To find all primes ≤ 30, list the numbers from 2 to 30.

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

The first number 2 is prime (in green) and eliminate its multiples (in red), i.e., red numbers are not prime.

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

The first number left (still black) is 3, so it is the first odd prime. Keep it and cross out all of its multiples. Since all multiples less than 9 (i.e., 6) will already have been crossed out, we can start crossing out at 3^2=9.

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

Now the first number left (still black) is 5, the second odd prime. So keep it also and cross out all of its multiples (all multiples less than 5^2=25 have already been crossed out, and in fact 25 is the only multiple not yet crossed out).

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

The next number left, 7, is larger than the square root of 30, so there are no multiples of 7 to cross off that haven't already been crossed off (14 and 28 by 2, and 21 by 3), and therefore the sieve is complete.

Hence, all of the numbers left are primes: {**2, 3, 5, 7, 11, 13, 17, 19, 23, 29**}. Notice we just found these primes without dividing. Note that there are 25 primes less than 100. The percentage decreases for increasing values of n.

Solutions to the Eratosthenes problem can be readily googled. However, the object of this lab is to ensure that you learn how to use process communication and synchronization using the Sieve as an underlying problem. The challenge is in the design of processes/threads that will be run on separate machines and the type of information they send each other to accomplish the overall task. This experience will be useful for the Computer Networks class next semester.

Algorithm:

The first process, **Sieve[1] (on thing0)** sends all odd numbers to **Sieve[2] (on thing1).** The first number **p** that process **Sieve[i]** $(0 \leq i \leq 1)$ receives is the $\mathbf{i^{th}}$ prime. Each **Sieve[i]** subsequently passes on to the other all other numbers it receives that are not multiples of prime **p**.

Process **Sieve[i]** terminates when it runs out of odd numbers to send to the other process. When a process terminates, it remembers all the primes it has calculated so far.

How would you modify the program to terminate normally and enable it to print all primes in ascending order. Is it efficient for every process to keep a list of all the current primes?

Design your program to allow processes to be created on both ***thing0*** and ***thing1***. Think about how each process communicate and coordinate in the implementation of **Sieve[i].** Design a program such that **Sieve[i]** and **Sieve[i+1]** are running on different machines. The description above used the word processes. Perhaps threads might be more appropriate (and efficient) since we only have two machines and potentially many sieves.

You will need to decide on the following in your design:

a) The type of protocol you want to use: TCP or UDP sockets (take your pick) for processes to communicate.
b) The data structure to represent the numbers in transit to the other process.
c) The structure of your message, i.e., what fields are needed so that the server can understand what you want to do.
d) Port numbers to be used for the communication, pick a port **p** s.t.

$$9000 < \mathbf{p} < 10000$$

Output:

Your program must be compliant to the output below. Your output should have sufficient information that shows the dialog between three machines. The following is an example of two machines communicating for n = 20:

```
thing0                          thing1

Sent: 3,5,7,…,15,17,19
To: thing1.cs.uwec.edu          Recvd: 3,5,7,…,15,17,19
                                Sent: 7,11,13,17,19
                                To: thing0.cs.uwec.edu

Recd: 7,11,13,17,19                    :
Sent: 11, 13, 17, 19…                  :
To: thing1.cs.uwec.edu
  :
```

Grade Rubric:

Correctness	50%
Stress Tests (large data sets)	20%
Efficiency (fast execution time)	20%
Specifications Compliant	10%

Submission:

1. Email me a zipped directory that contains a README, source code, and sample run.
2. Drop file in the W drive under Lab2 folder.
3. Turn in a lab cover page (see below).
4. Schedule a demo.

Operating Systems Lab Assignment Cover Page

Name(s) _____ Lab Assignment No.: 2

1. Include a copy of your source code, a design document (if required), and a sample run
2. Fill in the table of contents including the names of routines and the corresponding pages.
3. Indicate the status of your program by checking one of the boxes.
4. Submit the assignment at the beginning of the class on the due date.

Program Status: (check one box)
□ Program runs with user-defined test cases. □ Program runs with some errors.
□ Program compiles and runs with no output. □ Program does not compile.

Workload:	Written By			
main.c				
Paper	N/A			
Test cases				
Program modules (classes) List only major modules that are typically more than 50 lines of code	Module name	Written by	Module name	Written by

Grading: (No late assignments)	Points	Score
Stress Test	20	
Correctness	50	
Efficiency	20	
Specifications Compliant	10	
Total	100	