CS 511: Assignment #1
Practice with UNIX Process Mechanisms

Due before class on Monday, September 28

1 Purpose

The purposes of this assignment are:

- To learn about how to create a concurrent program using `fork(2)` and UNIX inter-process communication mechanisms.
- To verify that everyone in the course has basic programming ability.

2 Assignment

The purpose of this assignment is to acquire some experience with UNIX mechanisms that are helpful in writing multi-process programs.

Write a C program named “primes” that accepts a sequence of increasing positive integers on its command line. The integers define ranges, and a child process must be created to search for primes in each range. Here are some example runs:

```
$ ./primes
usage: ./primes <increasing positive integers>
$ ./primes 44
child 32077: bottom=2, top=44
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
```

```
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
child 32077 exited correctly
$ ./primes 44 100
child 32085: bottom=2, top=44
child 32086: bottom=45, top=100
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
child 32085 exited correctly
47 is prime
53 is prime
59 is prime
61 is prime
67 is prime
71 is prime
73 is prime
79 is prime
83 is prime
89 is prime
97 is prime
child 32086 exited correctly
```

Notice that in the third run one child (process 32085) was created to search for primes in the range 2-44 while a second child (process 32086) was created to search for primes in the range 45-100. If more numbers are given as arguments then more child processes must be created; for example, the run "`./primes 44 100 205 3000`" would create four child processes searching the ranges 2-44, 45-100, 101-205, and 206-3000 respectively. You can assume that command line arguments are all positive integers in increasing order; you need not error-check this.

When a child starts, it should print its pid and its assigned range; e.g. "child 32085: bottom=2, top=44"

Once a child has discovered a prime, it should use an UNIX inter-process communication mechanism to communicate that number to the parent process. The parent process then reads the number and prints a message such as "2 is prime" for each prime, as shown above. Odd-numbered child processes (i.e., the first, third, etc.) should use a pipe to communicate with the parent; even-numbered child processes (i.e., the second, fourth, etc.) should use a FIFO to communicate with the parent. Each child process must have its own dedicated channel (i.e., pipe or FIFO) to the parent. The parent must use `select(2)` to wait for results from all children at once.

Once a child has finished searching all numbers in its assigned range it should exit, providing an exit code that is the number of primes that the child discovered and passed back to the parent. You may assume that each child process will be given a small range, so the number of primes it discovers will fit in the limited number of bits provided for an exit code; you need not error-check this.

The parent must wait on the child's exit and verify that the received exit code is indeed the number of primes that the child produced. When this is verified, the parent should print a message such as "child 32086 exited correctly." The parent should use standard macros such as `WEXITSTATUS` to read the child's exit code.

All FIFOs created by a run of this program should be removed when the program's run terminates normally.

You may copy prime-discovery code from the Internet, but all other code must be your own original code. Note that it may be a non-trivial effort to adapt Internet code to your program.

There is not much concurrency in this assignment. But to create a bit of concurrency, the parent process must create all child processes before it starts reading primes from any of the children.

3 Submission Instructions

You must work alone on this assignment. Submit your source code via Canvas by the indicated date and time. Late work is not accepted and Canvas will shut you out at the deadline. Since your clock and Canvas's clock may differ by a few minutes, be sure to submit at least several minutes before the deadline.

Your code should compile on `linux-lab.cs.stevens.edu` with no errors or warnings using a compilation line such as:

```
gcc -Wall -pedantic-errors primes.c -o primes
```

If you choose to use a different development environment, note that it might take non-trivial time to port your code from your environment to `linux-lab`, even if your code is fully working in your environment; be sure to budget for that time.

To request an account on the `linux-lab` machines, login to `www.stevens.edu/servicedesk` then choose Services, then Computer Science Department Support, then Computer Science Account Requests.

4 Advice

Be sure to start early and contact the instructors if you have questions or encounter problems. Instructor office hours are posted in Canvas. Do not assume that instructors will be able to reply to emails over the final weekend, especially Sunday night.

Do NOT write the entire program then start debugging. Instead, write a part, debug it, then write the next part, etc. Here is one possible development approach:

1. Write code that understands the command line, creates all child processes, and each child process does nothing but print out the range it must analyze.

2. Extend the program so that child processes discover the primes in their range and print them out.

3. Extend the program so that child processes communicate the primes back to the parent and the parent prints them out.

4. Extend the program to exit/wait properly.

Remove extra print statements from a section of code only once you're confident that the section is working.