CS2750 System programming and Tools Spring 2019

Project #5 [60 points]

Due date is Thursday, May 9th

Task:
In this project you are writing a program that will create some shared memory and then launch a user process (using fork and exec) to do some work. The user process and the master process will very briefly communicate and then they both will terminate. The point of this project is to test launching, execing and basic shared memory usage. Doing this project's task without using the appropriate system calls will result in virtually no credit. Examples of shared memory using multiple processes (though not of fork/exec) can be found at /accounts/facstaff/hauschildm/scripts/C/shm/t

In particular, your project will involve 2 executables, worker and master, with a makefile that should compile both of them. You will need to use the `all` target to do so (to make both of those executables) as described in class. When master launches, it should take in two possible command line arguments, -h and -n. -h would simply output what command line arguments we take. -n will be followed by a number, so for example a possible call would be:
./master -n 99

If -n is not used, then it should default to 100.

Master should start by allocating a shared memory region to hold an integer. It should then set that shared memory location to 0. (Make sure and set the shared memory before doing any of the following fork/exec). Then it should do a single `fork`() (Do not do this in a loop). This fork should be followed by code that determines if it is the parent or child. The child should immediately `exec` the worker executable, giving it as a command line argument the parameter passed as -n (or the default value). Master should then continually look at the shared memory location, waiting for it to become any other value than 0. Once it changes, it should print that value to the screen, then set the shared memory value back to 0 and then `wait` for the child to end.

When the user process is launched, it should take its command line argument given to it and try and find the largest prime smaller than it. So for example, if given 99, it should find the prime 97. It should then attach to shared memory, and see if a 0 is there in shared memory. If that shared memory is not set to 0, it should output an error message and terminate. If it sees a 0, it should output a message indicating it saw a zero. Then it should set that shared memory location to the value of the prime that we found (in this case, 97). Then the user process should, in a loop, check to see when the memory location gets set back to 0. When it does, it should output a message to screen indicating that the memory location was reset back to 0. Then it should terminate.
Note that all output should indicate whether it is output from `master` or `worker`.

As the master is doing a `wait` for the child to end, it should then end after it detects that the child ended, outputting a message indicating that it is terminating because the child is terminating.
Before terminating, make sure to free up your shared memory.

Details:
This project must involve two executables. It must use shared memory to communicate. There should be a single makefile that compiles both of them. Points will be taken off if your project leaves shared memory in the system after running.

To check on shared memory, you can use the command as discussed in class, ipcs. To manually remove a shared memory assigned to you, you use the command ipcrm, giving it appropriate command line options.

In addition, I expect you to use local version control. I want your project to be committed at least 3 times, with differing stages of tasks done and appropriate commit messages. This is worth 10% of your credit and is the easiest part. Do not skip this.

Hints:
Do this process incrementally, testing as you go, and it should go fairly quickly. One way to do this would be:

1) Write a master that takes in appropriate command line arguments and a worker that takes in a number (as well as a makefile for them both). Have each just verify and output the results. Then test them manually by calling separately.
2) Have master fork off and then exec worker, then `wait`ing for it to end, passing it its number. Test that this passing worked correctly.
3) Write the code for the worker, given a number, to find the prime lower than it, should be easiest part!
4) Have master allocate shared memory and set it to some nonzero value and then see if child sees that value, have it output that value. This is to test if the shared memory is set up correctly.
5) Lastly, start having them communicate using the shared memory.

Submission:
Submit the 2 source files, the output of your git log as text input (or a separate text file) and your makefile as separate files in canvas (not zipped up please).