Name: Daniel Ems
Date: 05/06/17
Current Module: Operating Systems
Project Name: Signaler

Project Goals:

The goal of this project is to write a program that prints prime numbers, and handles signals. Signals are sent to the program that affects the behavior of how the prime numbers are printed.

Considerations:

- How will signals be received by the program.
- What will be the best way to implement each of their actions.
- How will the program print and calculate prime numbers.
- How will the program implement command line arguments.
- What will be the best way to reverse the direction of the print.
- What will be the best way to skip prime numbers from being printed.
- What will be the best way to start incrementing from the beginning.

Initial Design:

The initial design did not differ much from the final product. The program revolves around the prime_check function. In order to make the function flexible to modification ither through signals or command line arguments, global variable were used. This allows modifications to be made from anywhere in the program. This is how the program is able to modify the direction of increment, or the starting point from either signals or command line arguments. The program is organized by placing the signal_handling, and prime_check in their own functions and error handling for command line arguments in main.

Data Flow:

Without any arguments passed, the program will start at 2 and print primes through uint_max. This is performed in a function called prime_check. Global and local variable are both used within the function in order to modify the behavior of the function as either command line arguments, or signals are sent. If signals or command line arguments are passed, the program will modify variables as necessary in order to affect the functions behavior.

If a signal is sent to the program, it is caught and handled by the program. The program has a struct that handles signals and sends them to the signal_handler where each signal is processed accordingly. Each signal affects the prime_check a different way.

If command line arguments are passed, they are handled and checked immediately before the run of prime_check. Different command line arguments are going to change different variables in prime_check. The program checks for invalid command line arguments. In such case, the program will either exit, or start with default values.

Potential Pitfalls:

One of the notable pitfalls is the inefficiency of the prime calculation. There are most certainly more efficient ways that were ultimately not utilized. Another consideration is that of global variables, or an alternative to them. This was crucial in being able to modify the behavior a function without having to pass around multiple variables. An alternative to this could have been to set flags In the function or place them in a struct.

Test Plan:

User Test:

1. Try to program a prime counter.
2. Try to have print statements for each signal.
3. Try to have each signal modify the prime counter appropriately.
4. Try to pass command line arguments.
5. Try to have the command line arguments modify the prime counter appropriately.

Test Cases:

1. Check to make sure the prime counter is accurate.
2. Check to make sure the signals sent work.
3. Check to make sure multiple signals in repetition work.
4. Test that the program exits at 2.
5. Test that the program exits at uint_max.
6. Test to make sure invalid command line arguments are handled.
7. Test to make sure invalid numbers are handled.
8. Test to make sure different combinations of commands are valid.

Conclusion:

The project was a good introductory project to operating systems, signals, and coding having recently been on break. I think the project was refreshingly straight forward. I think I could have improved by implementing some more complex data structures or methods. However, The rubric did say simple is better than complex. My only criticism is hoping that we had more literature and better understood signals than what we were given.