Jaime Riley

CIS 3207- Project 4

Lab Section 4

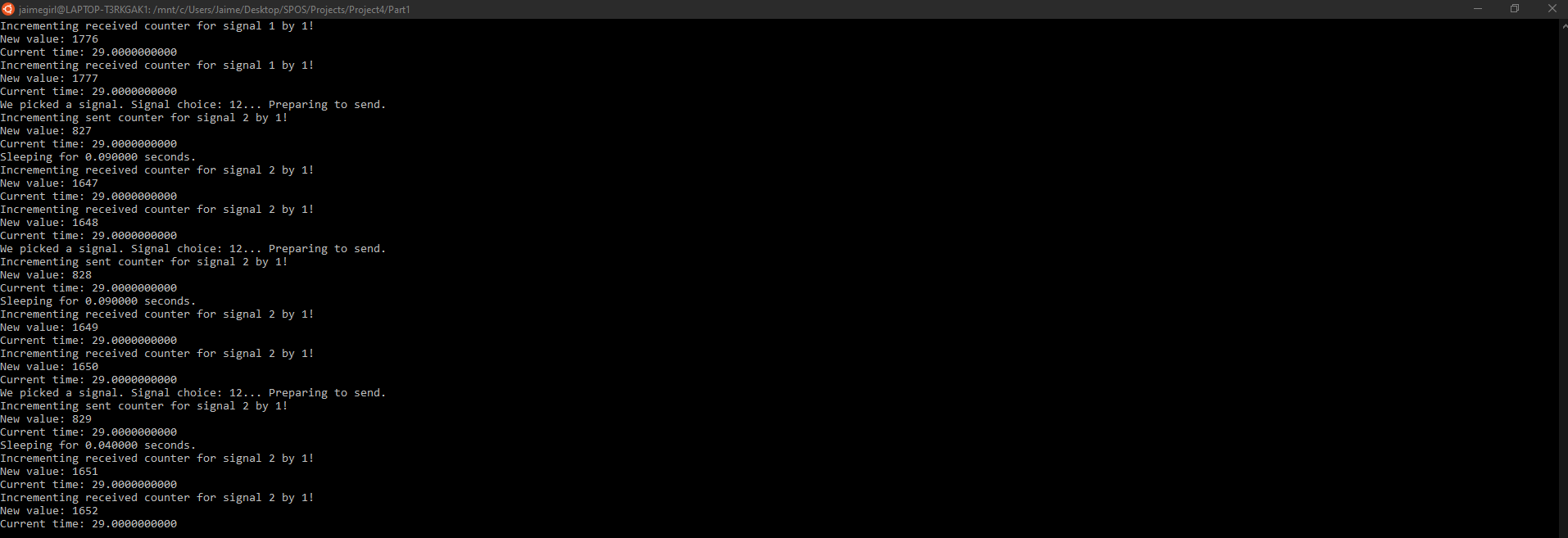
27 April 2020

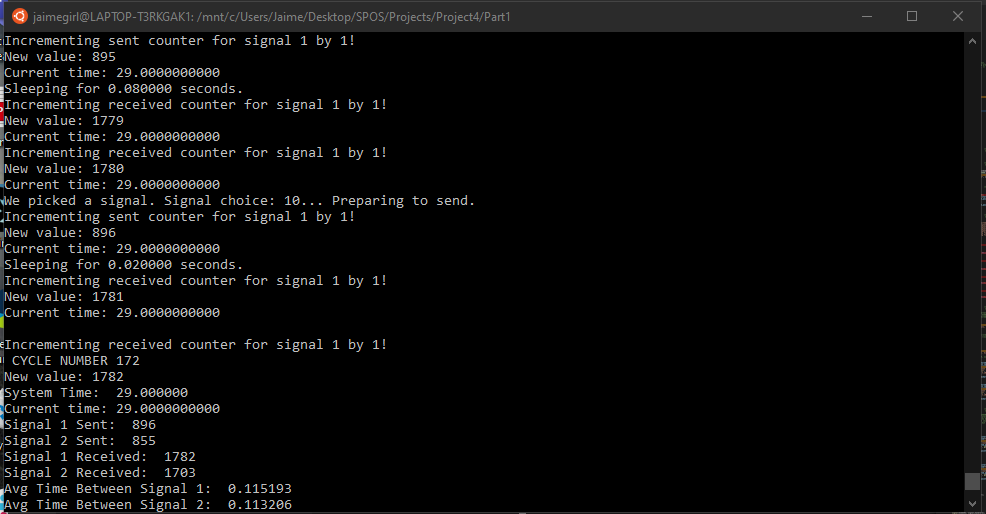
**Testing Project 4**

While developing both of my solutions for project 4, I made sure to create an #ifdef directive that would allow for the conditional compilation of my code depending on if I was testing it, or if I was doing a full execution of the program. These are similar for both my multi-process solution and multi-threaded solution, and helped me keep track of the timing, accuracy, and efficiency of my programs using print statements that give insight into the behavior of certain variables. I have included an in-depth description of both testing plans below.

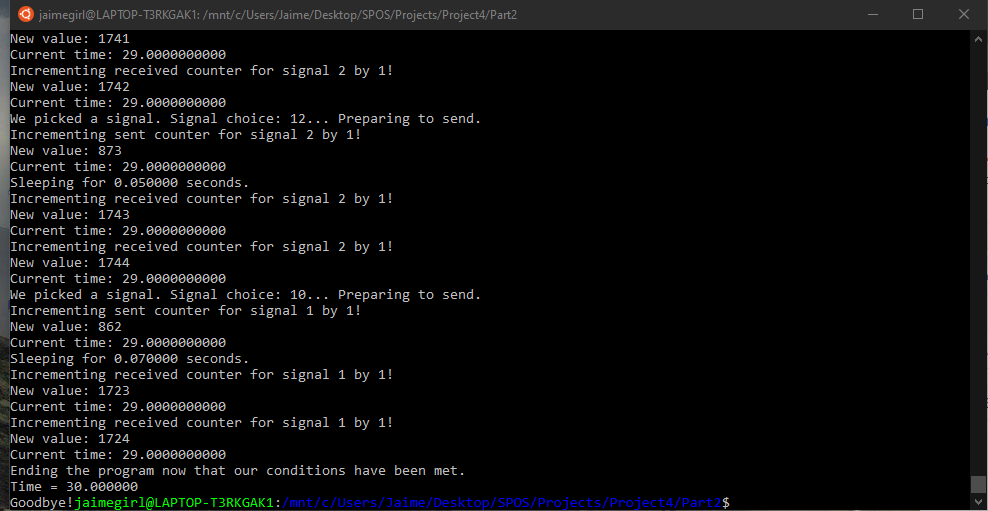
Multi-Processing Test Plan

To test my multi-processing solution, I started off small by creating a few utility functions that would make things simpler later on. This includes random\_signal(), sleep\_random\_interval(), get\_time\_elapsed(), set\_up\_report(), and all of my signal blocking and unblocking functions. I was able to make sure that these worked, with the exception of not being able to have get\_time\_elapsed() account for nanoseconds. Although this causes a bit of error in my data of both programs, the function still works properly for keeping track of time in integers and not being able to get it to work was a minor roadblock. Then, my next step was making sure that my signals were being generated properly and being sent to the correct processes. I was able to ensure that this was working appropriately by creating print statements that would print in testing mode each time a signal was sent, detailing the type of signal sent and also printing when the sent counter was incremented and the received counter was incremented. This can be seen below.



We can see also see that the number of signals received is nearly double the signals sent at any given time, since the signal is being sent to two processes that will both receive the signal and increment the count. This ensure that my program is working appropriately. The final part of my testing was making sure that the reporting process worked. Once I was able to get the process on a loop waiting to receive signals, I had to make sure that it would be reporting for every cycle of 10 signals it received. This was a bit tricky to do because of how quickly that both signals are received, and how quickly their counters are incremented, but it still functions properly no matter how many signals are passed before the data is collected and reported. I show the method of reporting below.

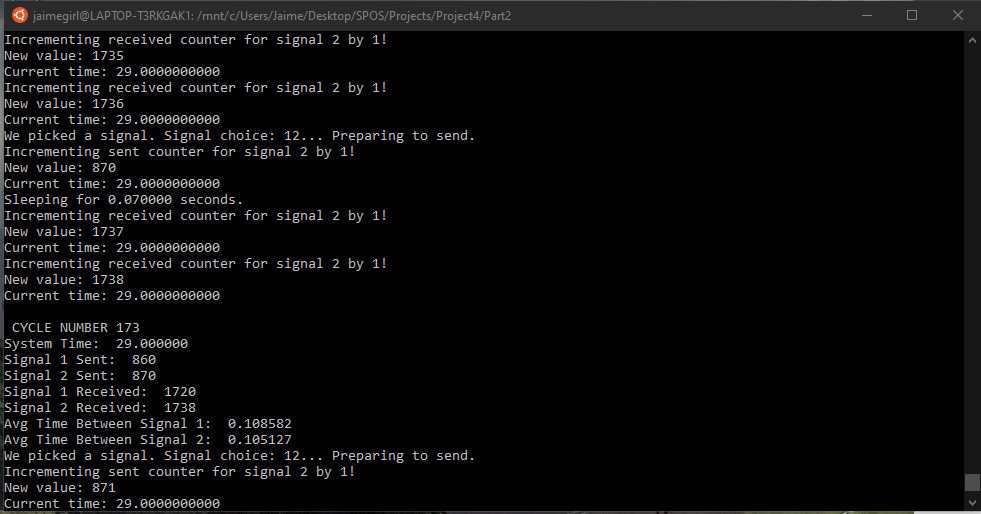
As seen by the last few cycles, the amount of time it takes between signals received each time remains small, but gets larger over time, which is expected. Also, once the program was done exiting, I had a couple print statements that would confirm the termination of child processes and the freeing of the shared memory. This is shown below.



The print statements shown in all of my above screenshots help to show that my program was working correctly, and the testing plan was successful.

**Multi-Threading Test Plan**

My multi-threading test plan was very similar to my multi-process test plan, especially because most of the functions were able to transfer over with only minor cleanup. The first step to my testing was to make sure that I was able to assign threads to run the same functions that I was using in my process solution. The functions that seamlessly transferred were random\_signal(), sleep\_random\_interval(), get\_time\_elapsed(), and set\_up\_report(). Unlike my previous solution, I no longer needed functions to block certain signals. Instead, I included the blocking within the functions themselves to enable the use of pthread\_kill(), which requires somewhat backwards thinking from when dealing with processes. I was able to test to make sure that the threads were being created properly and were running their functions with code that was somewhat similar to my previous, with print statements each time a major event was happening. The incrementing of shared variables, the time that a generating process was sleeping, and the signal chosen are all events that are accompanied by a print statement in test mode. I have showed this again below.



Given that the core logic behind the programs are the same, the way that this program executes and shows its clean up should suffice in showing that the testing of it is successful. I have included the ending of this program below.

