Aayahna Herbert & Tiffani Starks
Prof. Carl Martin
CPSC/ECE 3220-002
5 March 2018

Project #1 – Write Up

While pipe() is a secure transfer of information, one problem with this function is that it is a one-way communication. Even though it was convenient for this project, if one wanted to have multiple processes writing and/or reading to one pipe, it would be problematic. With fork(), it creates a new child process and can return an error, which is helpful because it tells you if the child process was successful or not.

In our assignment, the first problem we came across was dealing with the two ends of pipe. Initially, we thought fd[0] was for reading and fd[1] was for writing; when compiling and executing, we ran into an infinite loop issue. This problem is easily resolved by switching the two variables, which then opened the door to more problems. Every time the code was ran, the producer's number of values sent was always one less than the consumer's values, which in turn affected the averages of the both. This problem came about because the way our code was implemented returned the last value read in instead of a value of zero like it would do in general cases. It was solved by simple decrementing the averages by the last value read in and the counts by one. Another problem was when the program was done writing and reading, it didn't terminate but hung instead, needing the user to terminate the program manually. This turned out to not actually be a problem and was fine. When the fork is done, there are now two processes dumping information onto one command; because the parent process finishes first and has primary control over the command line, when the child finishes last, it can't control the command line. If the user were to press the ENTER key, the command line would show up.

There are no special instructions for running our code. Just a simple gcc -Wall main.c to compile and ./a.out to execute.