Chaya Chrein
CS 340- Prof. Svadlenka

Project 1 Writeup

For my project design, I incorporated dynamic shared memory which indicated to the child process how many times it should write to the output document until it was finished. This way, it wouldn't keep waiting for the producer. It only shared that updated variable, which was initialized to 0 once it completely finished reading in the file so that even if the child-out value was equal to the parent-in variable, the child wouldn't think it was finished. Aside from this shared counter variable, the parent and child process also share an in and out variable indicating if the child has caught up to the parent and should wait for the parent to continue writing, or if the parent needs to wait to overwrite the information since the child hasn't yet read it.

Throughout this project, I encountered multiple issues. One of them was printing out integers as well as figuring out how to use system calls as opposed to the functions included in libraries. Since I first wanted to understand and code the main idea of the project, I first used library functions so I had to figure out how to translate everything into system calls. Because the c write() function doesn't print out integers, each integer that I wanted to print had to be converted into its string form. To do that, I created a function called toAsc() which essentially separated each digit and added 48 to it, then turned it into its ascii character, and then put it back together to give me a string. Something I didn't realize at first was that I had to use malloc() in order to create the string to return since I wanted a variable that would still exist, meaning it could be passed, after the function finished executing.

Another issue I encountered was coordinating and communicating between parent and child. At first, I thought updating global variables would update both the parent and the child. But when I remembered that each subprocess gets its own copy I realized I had to use shared memory. Since I had some code in the wrong order, the consumer process wouldn't run until the producer process was completely finished, but after fixing the global variables and making them shared memory and rearranging the code I got it to work.

My last issue which I couldn't figure out how to fix was handling null values. For my producer items, there's a null preceding each one and after each count, and my producer and consumer don't have equal shared memory counts since the consumer process writes out the full chunk size adding nulls when there is nothing left to write. Essentially the files have the same content, but the consumer writes out an extra 8 null values, the difference between the 2 shared memory counts. I did try adding the null character at the end of the string array returned by my toAsc() function as well as appending it onto the end of the shared memory pointer, ptr, once the producer process finishes reading in the whole file but it didn't seem to help.