Alif Rahi **23792468**

Write up for project 1

My software design took a very long time to perfect. It might not be perfect, however I was able to complete the requirements and get the correct output. When I first created the child process, I thought it would be easy to just send messages through the pipe and do calculations. However when I used a bigger file with paragraphs of data, my code wouldn't work. This made me realize that I needed to increase the buffer size. I multiplied the buffer size from the compile time arguements by 100. This allowed me to read and write larger files and not cut off any data.

In the child process, I declared a file descriptor and opened the input file into it. After opening the file, I used the **read()** function inside of a while loop and incremented a variable called **count** by the return value of the **read()** function until it broke out of the loop. This allowed me to count and store the length of the file into the count variable. I then passed the length and contents of the file into the write **pipe** in that respective order. This would allow me to read from the other end of the pipe in the parent process by receiving the size first. By getting the size from the **pipe** before the file itself, you have a better idea of when the file actually ends.

After the child process was finished transferring data to the pipe, I took that data out of the parent process and did some calculations using a for loop and some basic cpu bursts. Once I had my calculations, I was ready to send it off into the POSIX message queue. I obviously used **mq_open** earlier in the project however, I did not need to use it until now. We will use the return value of **mq_open** to send and receive data from and to the queue. I created one big struct called Message and included several character arrays inside of it. This allowed me to send one big msg object instead of several character arrays in the POSIX message queue. With the calculations we just computed, I used the **snprintf** function to store the designated outputs into the Message object arrays. I then put that Message object into the **mq_send** function and did a blocking send. You do a blocking send by putting NULL in the wait time parameter. This allows you to wait until the messages have been sent before doing anything else.

The child process is responsible for receiving the messages from the message queue. Upon retrieval from the message queue, we needed to display them to the output. This was very tricky at first because we weren't allowed to use **printf** or **strlen**, but I found ways around it. You will see tons of while loops in my project that are simply just to count the lengths of the strings in my project. This is important if you want to make the **write(STDFILEOUT, msg, size)** system call work correctly.

All the other parts of the project were quite easy. Part 2 consisted of making system calls such as getcwd(), gethostname(), getppid(), getpid() etc… These were really easy to compute and I didn't have much trouble with them. In the future I need to watch how I declare arrays because sometimes making a char array of size 100000 might be too big for the local stack. I would probably allocate memory in the heap instead and free it later to avoid (core segmentation fault) errors.