A high level overview of the program's design goes as follows: First, command line arguments are first parsed and verified for adherence to the format specified in the project instructions. After the specific case of arguments are determined and casted to their intended data types the program is ready to begin operation. A pipe is then opened for child/parent communication, and a message queue is opened for parent/child communication. The child begins reading the first file listed by the user specified argument or default file. The child process reads the file in chunks, writing to the pipe from its readBuffer on each chunk read. Since pipes are blocking by default, it will not write to the pipe again (or even read from the current file) until the parent process has read the character buffer “off” the pipe. Once data has been written to pipe by the child, and the data has been read by the parent, the parent concatenates the readBuffer into a bufferFile via a write system call. This sequential process of child read, child write, parent read, parent write, repeats until the entire (current file) has been read by the child. Once the child has finished reading the current file it writes an end-of-transmission non-printable character to the pipe. The parent detects when this character is read from the pipe, finishes its write operation to the bufferFile (excluding the end-of-transmission character) and performs the designated counts specified by the command arguments on the cumulative reads from the pipe. This is done to avoid miscounting data that has been truncated by being passed in chunks. (specifically word count). After this, the bufferFile will be cleared and restored to its original standing for the next file sent in chunks by the child process, so as to not take the sum of all files. Once the count is performed, the parent queue sends the counts one by one in the order matching the OPTION argument, in the message queue accessible by both processes. Because the messages are sent in order, upon receival the child can appropriately understand the content being sent and can perform the correct formatting. This entire process of child read from file, child write to pipe, parent read from pipe, parent write to file, parent write N messages to queue, parent clear file, child read N messages from queue, repeat for EACH file argument given or just once if no argument is specified.
As for issues encountered, the first issue was the realization that there are no message queues on Mac OS and I’d have to test message queues in Linux (Venus or Personal VM). Upon compiling on Linux, there was an issue where I couldn’t link the mqueue.h library. After some research on Linux man pages, I found that mq_open must be linked with the ‘-lrt’ command at the end of my compile command. The next immediate issue I faced was providing incorrect arguments to mq_open. After more research I found the attributes I was trying to use for my message queue exceeded the values permissible for my account on the Venus SSH, instead I used the default values. After this I only faced logic issues with certain utility functions such as printIntString, countWords, etc. The final issue I faced was synchronization between child and parent. My child was timing out, causing a segmentation fault. The issue seemed alien, but after a long debugging process I found the child was incorrectly counting the number of messages to receive (counted - in options), and because I had changed to mq_recieve (to alleviate timeouts with blocking)

rather than mq_timedrecieve, the child was waiting indefinitely for a message it'd never received. This was easily fixed thereafter. Finally Part 2 of the project was done via system calls such as getpid() and getppid() and were rather easy to implement using my utilities. O.S information was collected by using the uname() function from the <sys/utsname.h> which writes O.S information strings into a utsname struct. Similar to objects found in higher-level languages, but lacking member functions. Very fun!

Some resources used)

https://man7.org/linux/man-pages/man3/mq_open.3.html

https://stackoverflow.com/questions/60530380/mq-open-api-return-open-invalid-argument

https://man7.org/linux/man-pages/man3/mq_timedreceive.3p.html

https://man7.org/linux/man-pages/man3/mq_receive.3p.htmls