Alif Rahi // 23792468

Project 2: WRITE UP

For this project my software design was really simple. I had 3 global arrays of size 16 along with another global array of size 16000 that stored the input file integers. In comparison to project 1, this was a lot easier because we only needed 1 process to work with. This singular process was responsible for creating 16 threads which then created 3 threads of its own (Total of 16 x 3 threads). All these threads are under the same process thus, we can share global variables. Though this may have disadvantages like needing to use a mutex lock and synchronizing code, this project didn't require that. Computations were done on different indexes of the array meaning no same variables were being changed at the same time.

My first for loop creates 16 worker threads that all go into the **routine** function. This routine function is very complex as it deals with a lot of pointers and dereferencing and freeing variables. I know I could have used strings and the stroi() function however, using pointers wasn't an issue for me. Throughout all 16 iterations of the for loop, I passed the 0 indexed value inside the loop multiplied by 1000. This would create a sequence that looks like [0, 1000, 2000, 3000 … 15000]. These are all the starting indexes of each group. I passed these starting indexes into 3 different threads within each of the 16 worker threads. The **sum**, **average** and **geometric average** computations were created in these 3 different threads. The starting index was passed into those functions as an argument. Once inside the functions, I made another variable called **end** and gave it the value of **start** plus 1000. This allowed me to do computations on 1000 numbers at a time.

After doing the computations in each of the 3 threads, I returned the values into the **routine** function and stored it into their respective array indexes. At this time, I also took the 3 computations for the group of 1000 numbers and wrote the results into my output file specified by argv[2]. This was done for all 16 threads and once every thread was finished, I calculated the max values in the main thread. The **pthread_join** creates a blocking code ensuring that all the threads finished before I moved onto the next line. This leaves my results as accurate as possible.

Lastly, after computing the max values I parsed them into string messages by using **snprintf**. This made it very simple to use the write() syscall to output the messages to the screen and to the output file.