Operating Systems Project 1 Report

Group Members: Brian Monticello, Sean Olejar, Mhammed Alhayek

For part a of the project, there was only one processor necessary. The only design decision we chose to do for this part was to calculate the sum, min, and max as we pull each element from the input file, as opposed to storing all the elements in an array first and then making the calculations. We did this because it is more efficient and storing the elements in an array wasn’t necessary.

For part b, we knew if we had more than one processor, then in theory the calculations should be completed more quickly because multiple processes could be making calculations on separate parts of the array and then all the calculations could be collated. However, we knew that adding a few more processes should speed up the program, but adding too many processes would slow it down because first of all each new process requires a little overhead to spawn, and secondly, because if there are too many processes and not enough resources to allocate to all of them, then some of the processes would be doing nothing because they would be waiting for other processes to complete. After checking the timing with different numbers of processes, we chose to spawn 3 child processes, thus having a total of 4 processes operating on the array of data. After checking the timing with the different file sizes, we noticed that initially when the file sizes were small, as the file sizes got bigger, the time to compute the total actually decreased. Eventually, when the file size got much bigger, the time to compute the max, min, and sum actually increased again. We think this could be because when the file sizes are small, it takes more overhead time to spawn more processes and pass data back and forth through the pipes than it takes to just do all the calculations in one file. For implementing part b, we also chose to use recursion because we realized that the parent-child-grandchild relationship would be shaped similar to a stack where the original parent is at the bottom of the stack and the “youngest” grandchild would be at the top of the stack.

For part c…

For part d…

for the extra credit…

This project allowed us to learn many new concepts about how multiple processes work in operating systems and how they communicate with each other. Specifically, one thing we learned was that when one process spawns another using the “fork()” system call, it creates a second process that is identical to the one that spawned it, only differing in its process id. We also learned the benefits of using pipes for inter process communication as opposed to other methods such as shared memory because pipes are more secure and straight forward.