Rafal Stapinski

The comparisons are fair in that each process is a single threaded process, so a comparable amount of threads are executed for the two tests. The multithreaded application, however, only has to execute one write at the end of its execution, whereas the multiprocess program has to execute a write at each file it encounters.

Discrepancies occurred because of other people working at the same time as me. These were not noticeable after a while, as I ran multiple tests to find an average.

The multiprocess program was slower overall. This is due to the fact that a write occurs at each file that is being sorted, and because spawning multiple processed is slower than spawning threads. It would be possible to make the multiprocess sorter faster by having a global list and a write at the end, however this might be tricky if not impossible because of each process having its own memory in the heap.

Mergesort is the perfect option for the multithreaded program because, when keeping track of a global list in memory, merging a sorted list for a file into it is a breeze because of an already implemented merge_list function.