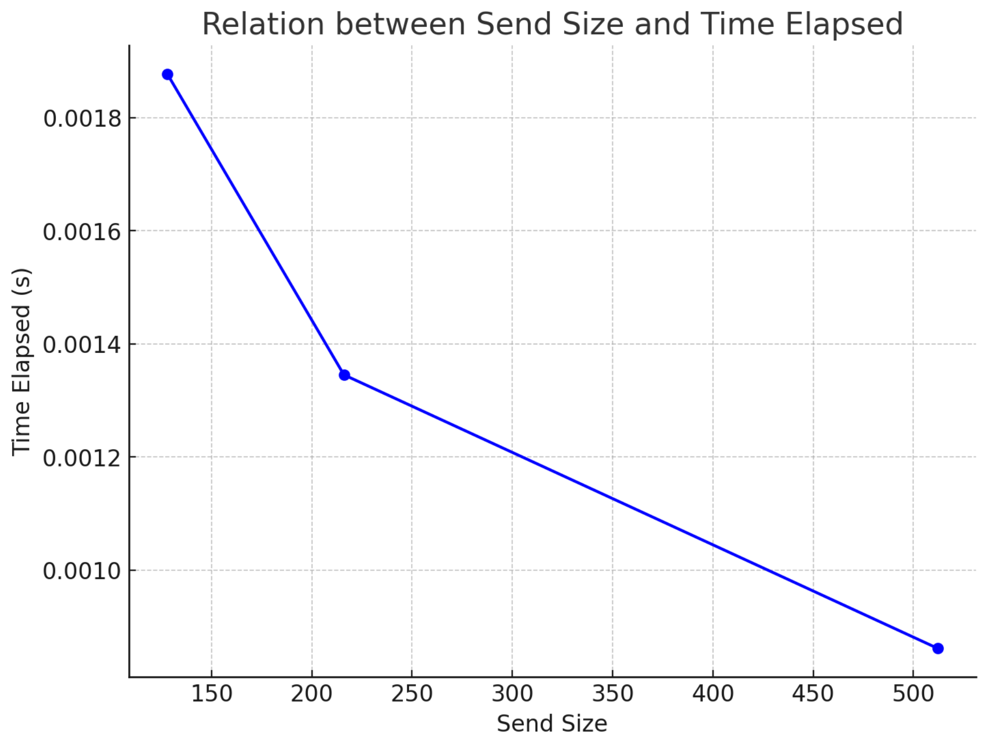
Multiprogramming by Forking

Elapsed time against sendsize, fixing other parameters

number of clients: 4
filesize: 5068 bytes

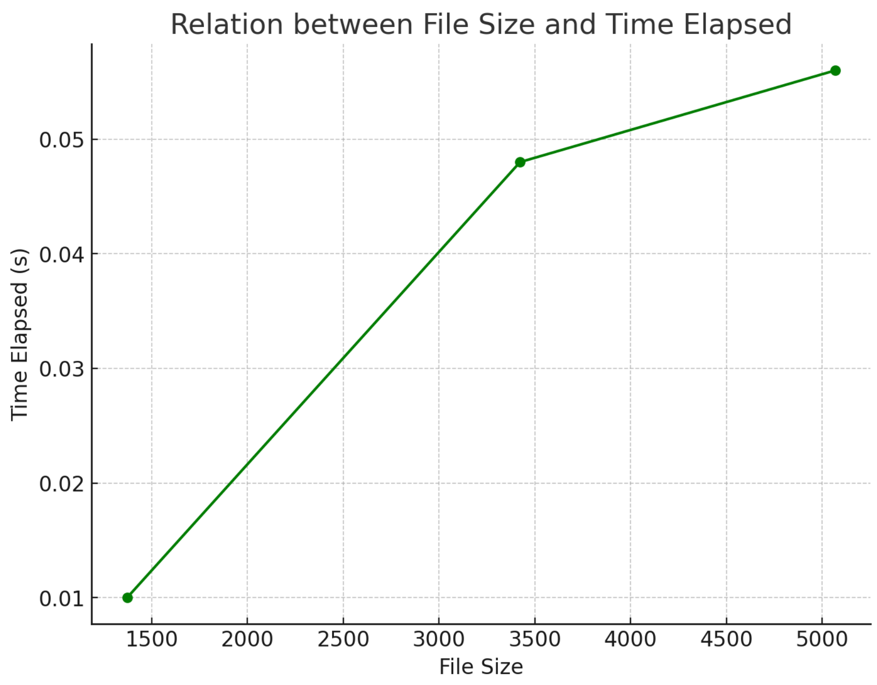
sendsize: 128, 216, 512
time elapsed: 0.001876754, 0.00134506, 0.00086146



Elapsed time against file size, fixing other parameters

number of clients: 4
sendsize: 128

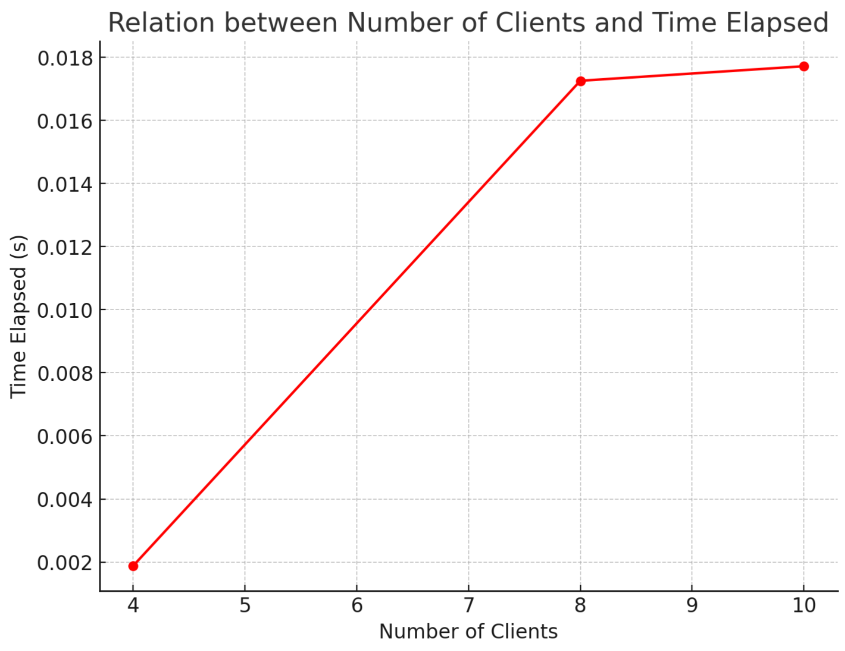
filesize: 1373 3423 5068
time elapsed: 0.010, 0.048. 0.056



Elapsed time against client number. fixing other parameters

filesize: 5068
sendsize: 128

number of clients: 4, 8, 10
elapsed time: 0.001876754, 0.0172548225, 0.017715722



Analysis:

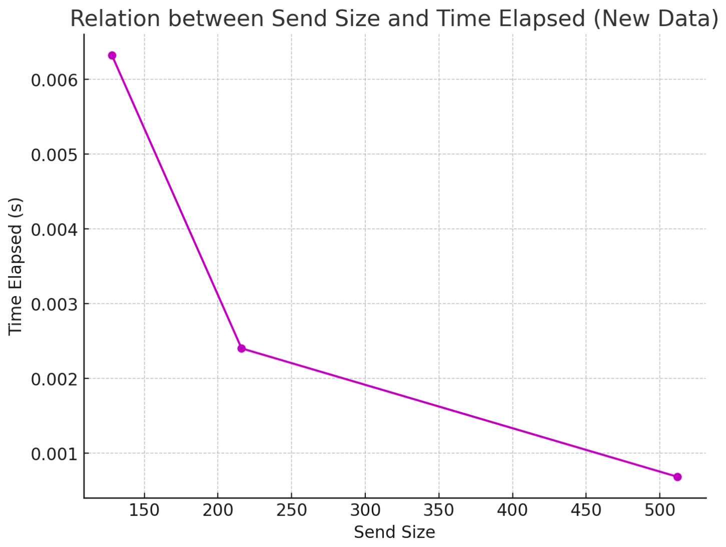
I initially hypothesized that increasing file size while send size is smaller than the file size will have a negative impact(increasing) on the elapsed time since more loop iterations would be performed. I followed a similar reasoning on increasing send size while file size was fixed, this would result in less loop operations so would have a positive impact(decreasing) on elapsed time. As you can see from the graphs, these two hypotheses proved to be right. On the send size vs time elapsed graph, we can see that there is a trend close to a linear relationship with some error. On the file size vs elapsed time graph, the trend is not so linear, but the proportional relation can be seen, I suspect this to some insufficient setting on the experiment environment, possibly hardware unpredictability. I guessed that if we were to increase client size, since there would be more context switches because of more programs demanding cpu time, causing more overhead, elapsed time would increase. The graph proves that and even suggests that the relationship is super-linear, closer to the cubic relation. When client size increases from 4 to 8, doubling time elapsed increases way more than doubling itself, closer to its triple value. When we increase client size from 8 to 10, a similar increase is there.

Multiprogramming by Multithreading

Elapsed time against sendsize, fixing other parameters

number of clients: 4
filesize: 5068 bytes

sendsize: 128, 216, 512
time elapsed: 0.006322228, 0.00240242, 0.00068439

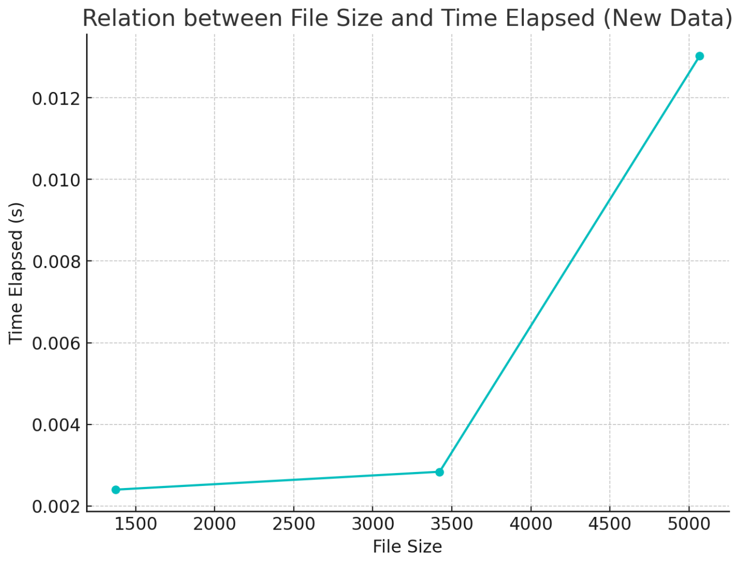


Elapsed time against file size, fixing other parameters

number of clients: 4

sendsize: 128

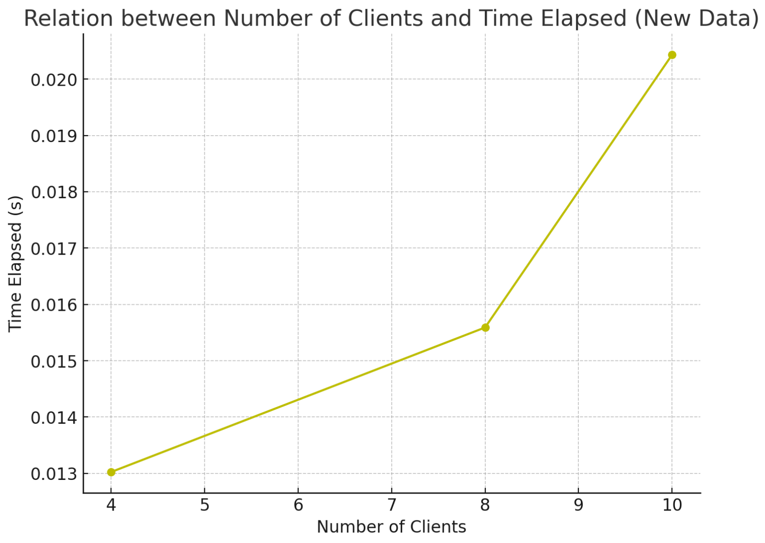
filesize: 1373 3423 5068
time elapsed: 0.00240242, 0.002840582, 0.01302273



Elapsed time against client number. fixing other parameters

filesize: 5068
sendsize: 128

number of clients: 4, 8, 10
time elapsed: 0.01302273, 0.0155896, 0.02043211



Analysis:

The behavior of the whole system is similar to the forking approach, although I've noticed some differences. First difference is that the degree of the relations changed and became more complicated. I can't say that we have linear/quadratic/cubic relations between the variables, although this might be happening because of the unpredictability of the experiment system. One important point to notice is that elapsed time does not increase as much compared to forking when we increase the client number. This might be happening due to the faster working nature of multithreading due to shared memory instead of copying the whole address space to some other process.