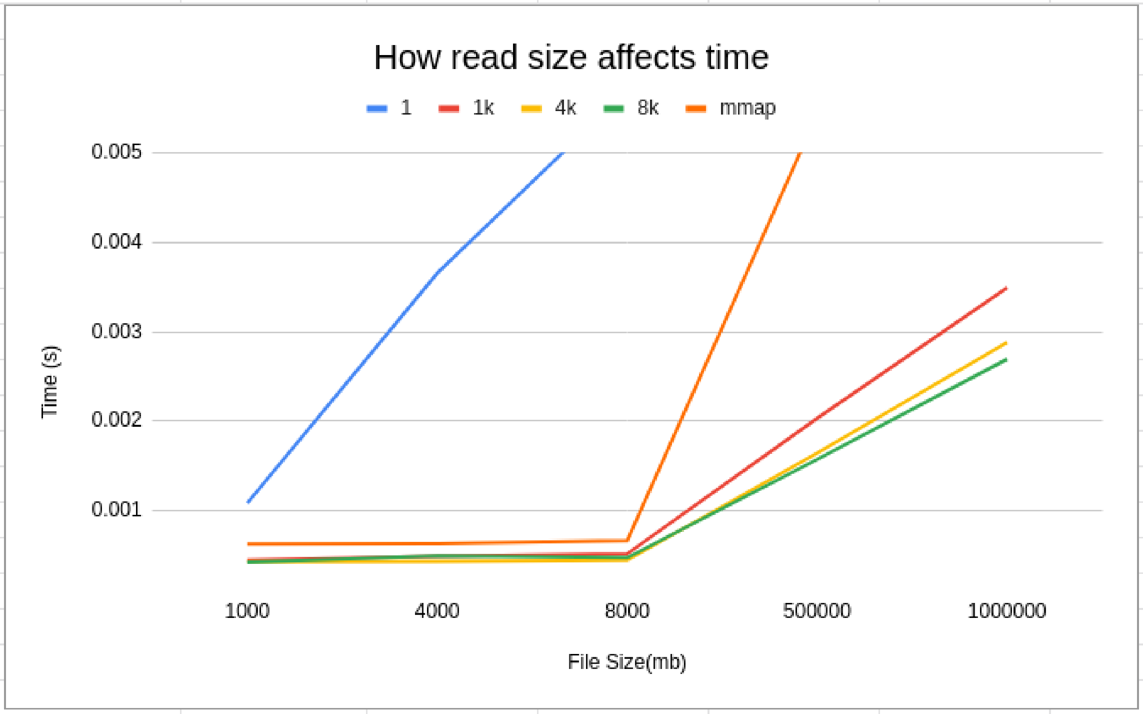
Benjamin Gilchrist

CS 3013 Operating Systems, Project 2.

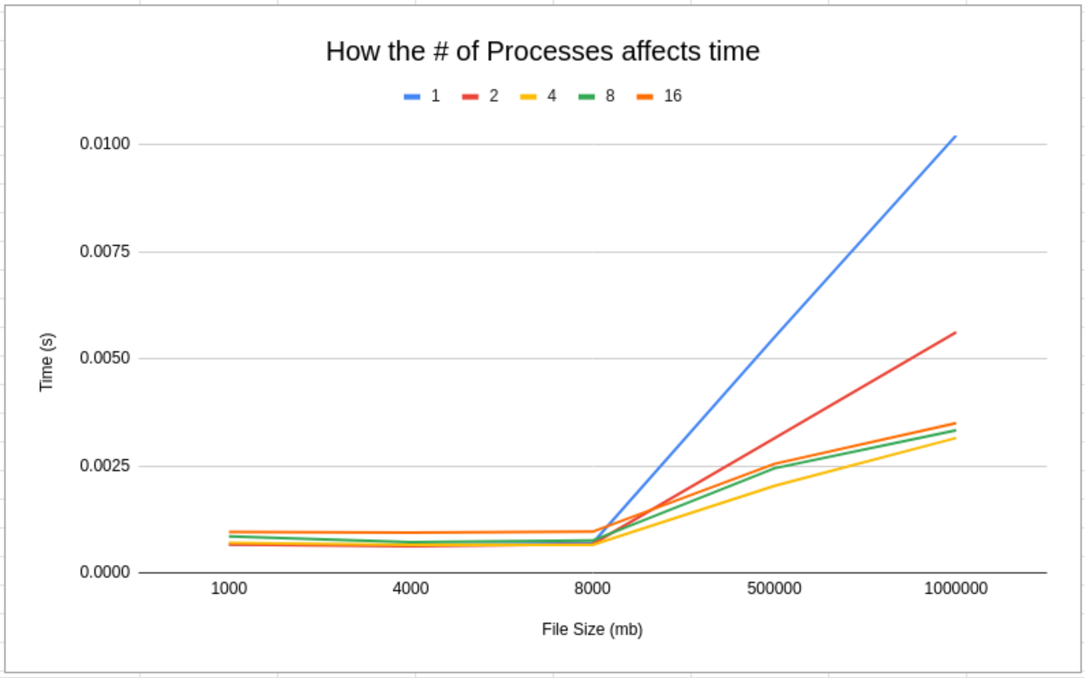
This Project was done in an oracle vm virtualbox, running ubuntu 64bit with 6 cores. Testing of standard file I/O with different read sizes shows that a read size of 1 takes the longest time regardless of file size while memory mapping takes less time but still more time than larger read sizes. At smaller file sizes read sizes of 1000 to 8000 result in similar read times. At larger file sizes (500k and 1mil) larger read sizes result in lower overall time.

File Size	Read Size:	1	1k	4k	8k	mmap
1000		0.001088	0.00045	0.00043	0.000428	0.000632
4000		0.003652	0.000492	0.000439	0.000498	0.000635
8000		0.00569	0.000521	0.00045	0.000478	0.00067
500000		0.385	0.002032	0.001645	0.001575	0.005425
1000000		0.769716	0.003491	0.002878	0.002693	0.010352



As seen by the data and graph, the most significant changes are that quickly reading 1 bit at a time goes off the graph to increase the time significantly. Then at 500k bits we can see a spread start to happen where mmap starts to take a significantly larger amount of time and 1k read size starts to take slightly longer with 4k and 8k still performing at around the same amount of time. Then at one million bits we finally see that all read sizes act according to what we would expect, with larger read sizes resulting in less time.
Next the number of processes was tested to see how it affected the runtime. With 6 cores running on the virtual machine I predicted that 6 processes would be optimal with more or less processes being less efficient as it would either not use all cores and be less efficient or have to wait for the cores to finish with their current processes until it could run. The data showed this to be the case.

File Size	Proccesses	1	2	4	8	16
1000		0.000671	0.000664	0.000701	0.000853	0.000957
4000		0.000652	0.000621	0.00066	0.000719	0.000947
8000		0.000698	0.000674	0.00066	0.000758	0.000964
500000		0.005504	0.003143	0.002033	0.002446	0.00255
1000000		0.01022	0.005622	0.003155	0.003331	0.0035



From this graph and table we can see that up to a file size of 8k bits, there is little change in the time it takes based on the number of processes, but once we get to the larger file size (500k and 1mil) we can see that our hypothesis was correct and the best times come from 4 and 8 processes (2 away from utilizing 6 cores perfectly). After that 16 processes is close behind with that many processes utilizing all 6 cores nearly 3 times perfectly. After that we can see that 2 processes are quite a bit less efficient being slower than the other 3 configurations and 1 processes being the absolute slowest option. This is likely due to the fact that one process can only run on one core and does not utilize the advantages of multicore processors.

From what I can tell my data shows a best practice of using a read size of as large as you can and utilizing as many cores as possible while not having any processes get blocked waiting for others to finish. This can be done by creating the same amount of processes as cores. Other than that it is hard to say without a whole lot more data, with many different file size and many different read sizes.