Laurentiu Pavel

CS3013 – Project 4 Report

**Introduction to the environment**

The goal of this project is to analyze how different types of file I/O read behave on a computer. All the parts of the lab are implemented and tested on a 64-bit Ubuntu 14.04 LTS, running on 7.7 GiB of memory and an Intel Core i5-2450M CPU @ 2.50Ghz x 4. Basically all the tests are run on a multiprocessor environment. Moreover I tried to stop the processes that eat CPU resources and tried to disconnect the machine form the internet.

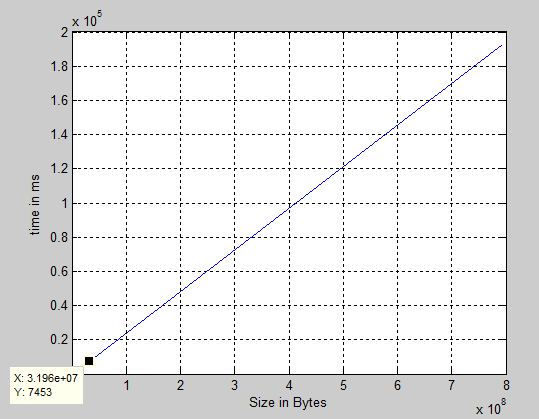
I chose 7 files with the following sizes to run the tests on: 55 bytes, 1.3kB, 19.4kB 5.4MB, 32MB, 792.7MB, 5.9G

**Results:**

A. Graphs of time performance:

1. Graph of input with a size 1 buffer

Time:

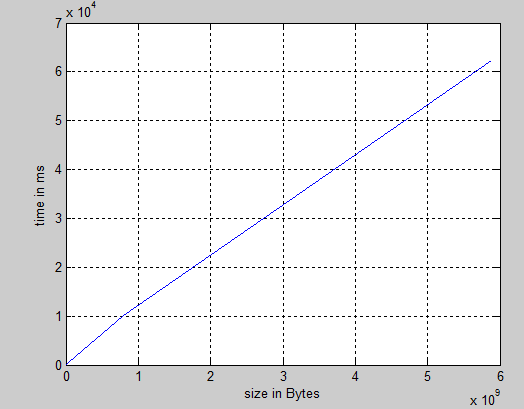


Mention: did not even run the 5.9GB file since it took a very long time for the 792.7MB

Page Faults: There were no Page Faults while running all these programs

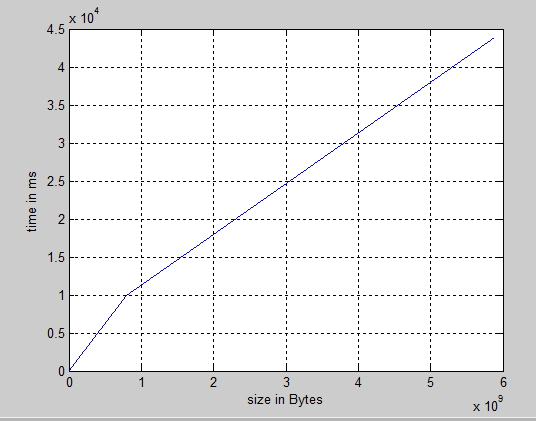
1. Graph of input with a size 1024 buffer

Time:



Page Faults: Again, no pagefaults

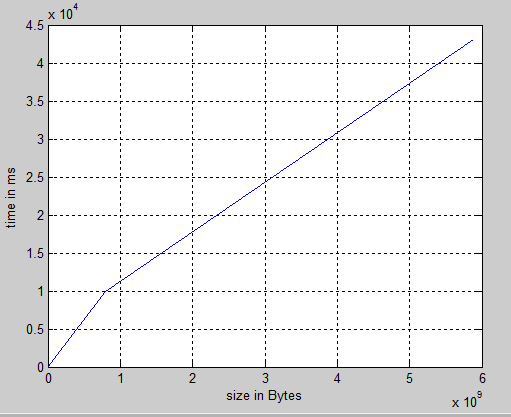
1. Graph of input with a size 2048 buffer



Page Faults: No pagefaults for this case either

1. Graph of input with a size 4096 buffer

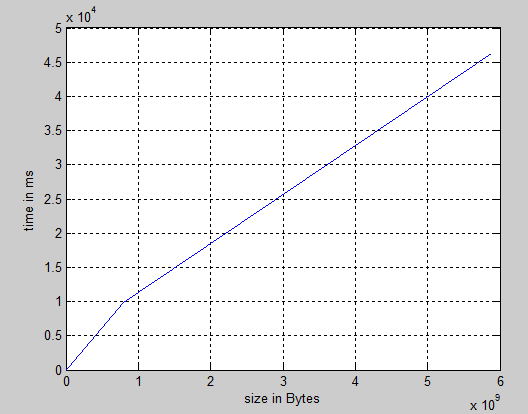
Time:



Page Faults: No Page Faults for this case

1. Graph of input with a size 8000 buffer

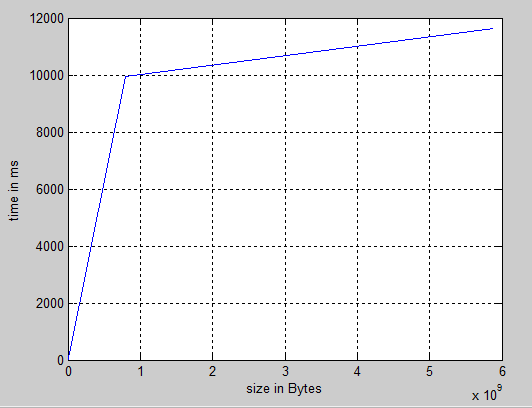
Time:



Page Faults: No Page Faults for this case either

1. Graph of Input using mmap

Time:

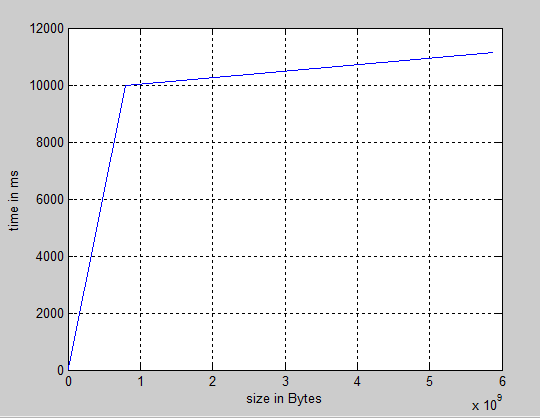


Page Faults: Again no page faults

B. Graphs of Performance using Threads

1. 1 Thread

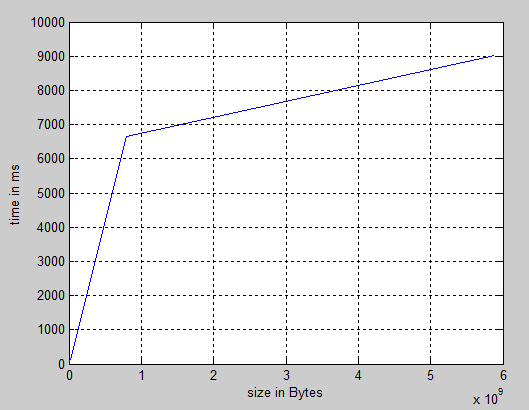
Time:



Page Faults: No page faults

2. 2 Threads

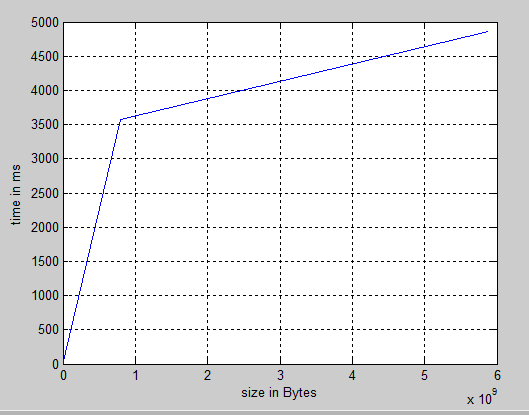
Time:



Page Faults: No page faults

3. 4 Threads

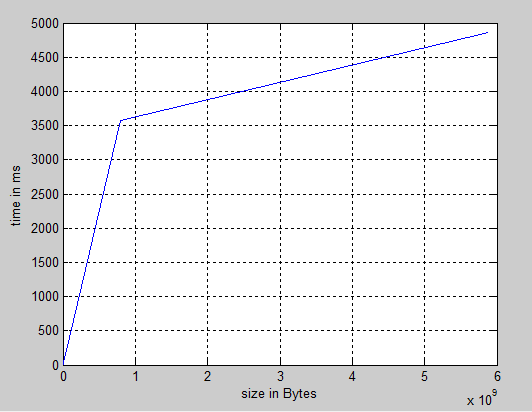
Time:



Page Faults: No page faults

4. 8 Threads

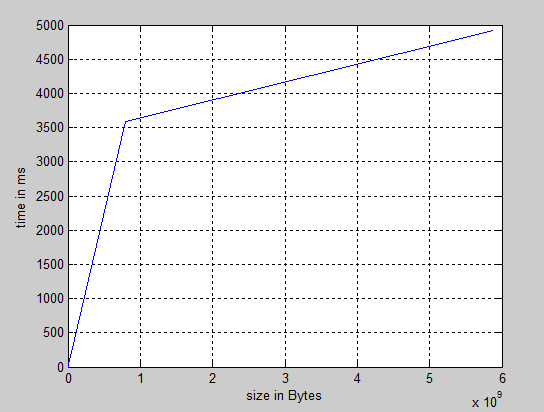
Time:



Page Faults: No page faults

5. 16 Threads

Time:



Page Faults: No page faults

**Final Observations**

First of all, there were no major page faults obtained in any of the tries. In terms of time performance, it can be seen how big of a difference increasing the size of the buffer made in part A. However, after getting to a size of 2048, there was more or less no improvement. Moreover, for the very small files, it was taking longer to read them than it was with smaller buffer sizes. Mmap appeared to behave comparable to the buffer reads for almost all the files except for the very big ones. For the very big ones (approximately 6 GB), it would perform 4 times faster. The tests show that size of buffer can make a big difference from a very small (1-8) one to a decently big one (>2048).

The multithread approach showed that for 1 thread the results are almost the same with the regular mmap read. However, 2 threads improve by a high rate (almost twice faster) that performs well with sizes between 1MB and 1GB, but for very big files (such as 6GB) the improvement is not that high anymore. However, when running on 4 threads, the improvement is considerably higher and the read is performed twice faster for almost cases. Adding more cores after 4 (8 or 16) does not really improve how fast the reading works.

Another mention: In all cases (multithreaded or not), higher buffer or more threads decrease the performance on very small files. However, since the times are very comparable and do not really make much of a difference for the user, in the case of larger files, very big improvements can be made (On a read with buffer 1, I could not even get to the end of the read of the 6GB file, but on more than 4 cores it was less than 6 seconds).

All these test cases showed that on a 4 core processor, the best performance is obtained when there are at least 4 threads doing the reading. More threads don’t harm, but less definitely do.