Başak Amasya – 26628

CS307 – HW4 – Report

The aim of this homework was to use three different approaches for reading a file and counting the number of occurrences of character 'a', analyzing performances' of these techniques and comparing their features.

The first method was using fstream in C++. I chose fstream class which is an input/output stream class to operate on files, so it supports both reading and writing operations. I used built-in eof() and get() library functions to read the file character by character sequentially. An advantage of fstream library is that it offers internalization, meaning it has equivalent classes that support wchar_t (wide characters). Also, C++ provides us the advantages of an object-oriented language and fstream is generally considered as more robust than corresponding C libraries. This method performs a lot of I/O operations to read from the disk and a lot of copying for sequential reading. A disadvantage I observed is, this code performs little slower than fopen of C, which is the second method we tried, which also does lots of I/O operations to access the disk.

The second technique was using fopen in C. I kept a file pointer and I opened the file in "r" mode as file access mode since only reading the file was necessary. I again used the EOF character to understand that I have reached the end of the file while reading and built-in getc() function to read the file character by character. One of this method's advantages is, fopen provides buffered I/O which performs relatively faster than non-buffered I/O. Also, fopen returns the pointer of type FILE, so it is easier to handle the file for other operations. However, C++ fstream is usually considered more safe and simpler to use than this technique.

Last method was using memory mapping in C. mmap function is used for this purpose which provides process address space mapping. Memory mapping method actually uses virtual memory, but the program sees this as if it's reading from main memory. That is why, it's much more efficient than first two methods. Biggest advantage of memory mapping is reading from and writing to the file is easier and more practical. Since I gave 0 as the address (first parameter of mmap), kernel maps the file to where it sees fit which can also be an advantage especially in terms of speed. A disadvantage can be seen as, it's relatively harder to implement but still not very challenging since built-in functions of C is used. Another disadvantage can be the overhead of creating the memory mapping, but this is much less than the lots of copying times of sequential access patterns of previous methods, especially for larger or frequently accessed files. Important point to note when using memory mapping is that we have to un-map the file after we are done using it. If we do not explicitly call the munmap function with corresponding address and file size in C, the file stays there which is not desired. For this homework, we didn't work with multiple processes, the file was not need to be shared. So, I used MAP_PRIVATE flag for the mmap function. But memory mapping would still be a great approach if the file was used by multiple processes, this time we would use MAP_SHARED flag.

In terms of time efficiency, slowest method was the first one where we used fstream of C++ (approximately 7 seconds). fopen of C was slightly faster but still slow (approximately 5 seconds). Third method which was memory mapping was certainly a lot faster (approximately 2 seconds). The main reason for this difference is that unlike the first two methods, we don't need to access the disk so frequently to read the file byte by byte while using memory mapped files.

In terms of impact of the file size, since the file we worked with (loremipsum.txt) was really big, its size had an observable impact on the performance, and I was able to see the difference clearly. This was my first time working with memory mapping and I remembered from the lectures and the recitation that memory mapped files are expected to work faster, but I was still impressed when I observed this myself. However, when I also tested the codes with relatively smaller files, the difference between these techniques was not so apparent. That is why, we can say that both the file size and the technique chosen is affecting the time performance.