BILKENT UNIVERSITY

CS342 OPERATING SYSTEMS

SPRING 2021

PROJECT #4 REPORT

MUSTAFA YAŞAR 21702808

CEMRE BİLTEKİN 21802369

SECTION 1

I. OVERVIEW OF THE FILE SYSTEM

Our file system sfs uses indexed allocation for managing blocks and bitmap to manage free spaces. It also holds an open file table to have quick access to opened files and its attributes like file size and position pointer (we held it as offset value). It can perform all of its operations as specified with the files. The details of our file system structures are given below. In our system for availability check purposes, 0 is free and 1 is occupied.

- Superblock

In superblock we store the information about the blocks such as total number of blocks in the virtual disk, number of files in the virtual disk, and the number of free block count in the virtual disk. It is the first block of the disk.

- FCB

File control block stores the information such as if that instance of FCB holds any file information at all (is_available attribute), the size of the file, and the index block number of the file.

- Directory Entry (Entries of the Directory Table)

Directory entry stores the information about files that are in the directory such as the name of the file, FCB index of the file, and if that directory entry holds any file information or not (is_available).

- Open File Table Entry (of Open File Table)

Open file related structures are global variables. In the open file table entry, we store the information about opened files. There can be at most 16 files opened at the same time. The information about these files include the name of the file, position offset (for reading and writing from where it is left off), mode (read only or write only), file size, and the FCB index. It also has an indicator is_available to check if that entry is holding any file information or not.

- Bitmap

For bitmap, we have decided to create arrays of integers (32-bit integers). Then the bits of the integers represent the bits of the bitmap. We wrote bitmap operation related functions to have an ease of use about bit operations when we had to use bitmap for managing free space. Also, there should be at least 2 free blocks to create a file as checked from bitmap (because a file has to have an index block).

Moreover, our read and append operations conform to project specifications (and behaves the same way as read and write operations in C do). When the position offset reaches the end of the file for reading (reads the entire file eventually), the file should be closed and opened again to start reading it from the beginning.

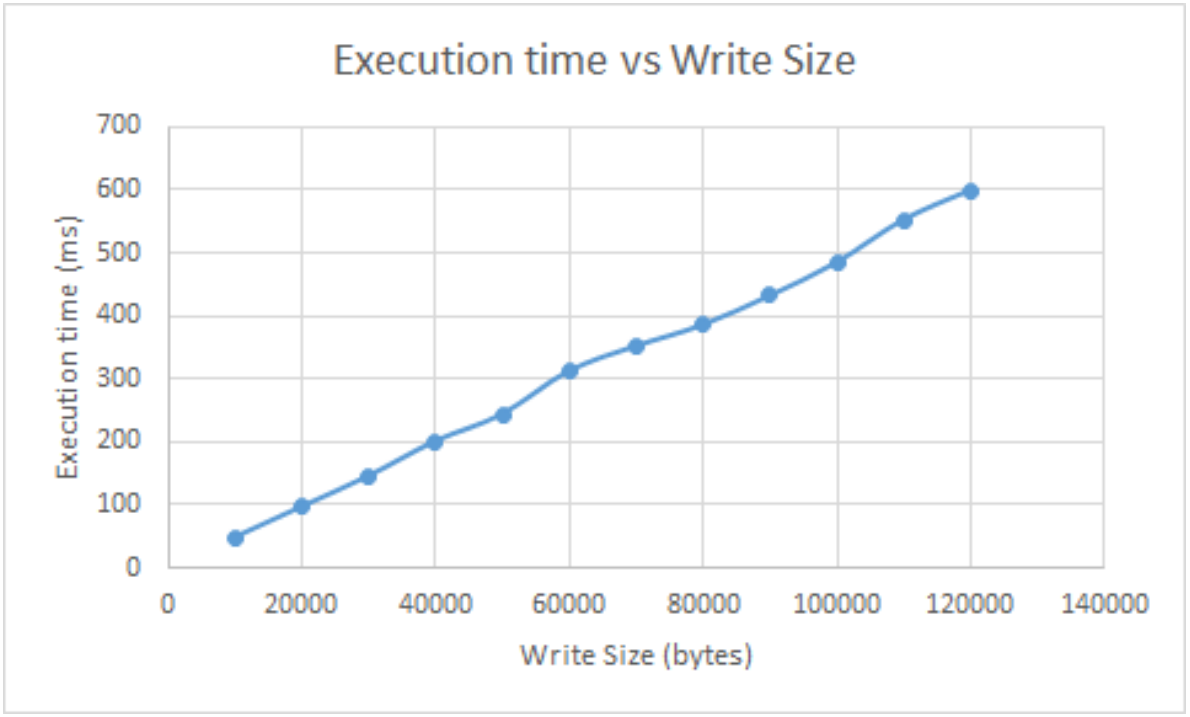
Our system does linear search for finding available block in bitmap, finding available spot for directory entry, finding available spot for FCB entry, and finding available spot in open file table (we check empty spots from beginning to end with linear iteration).

II. EXPERIMENT RESULTS AND ANALYSIS

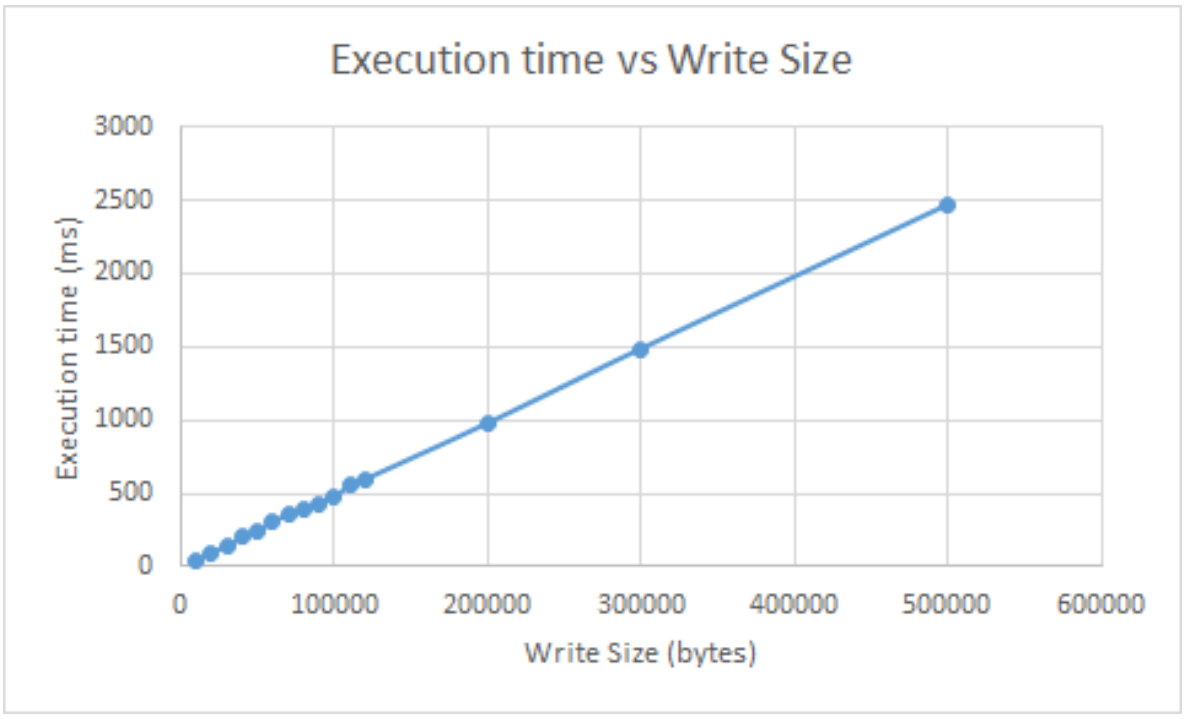
Two experiments were performed for analyzing time in the file system. First experiment measures time of writing (append) into a file, and second one measures reading time from a file. In these experiments, the file size is the independent variable under the given size constraints. First, the elapsed time was measured with small inputs, and then it was measured with larger inputs to see how the system behaves with different sized inputs.

N bytes (for writing on a file)	Execution time (ms)
10000	48.44
20000	97.23
30000	145.66
40000	201.13
50000	243.78
60000	312.48
70000	351.89
80000	385.64
90000	431.88
100000	484.57
110000	552.43
120000	599.10
200000	982.73
300000	1488.13
500000	2468.71

Table 1. Elapsed Execution Time for Writing with Variable Sized File Inputs



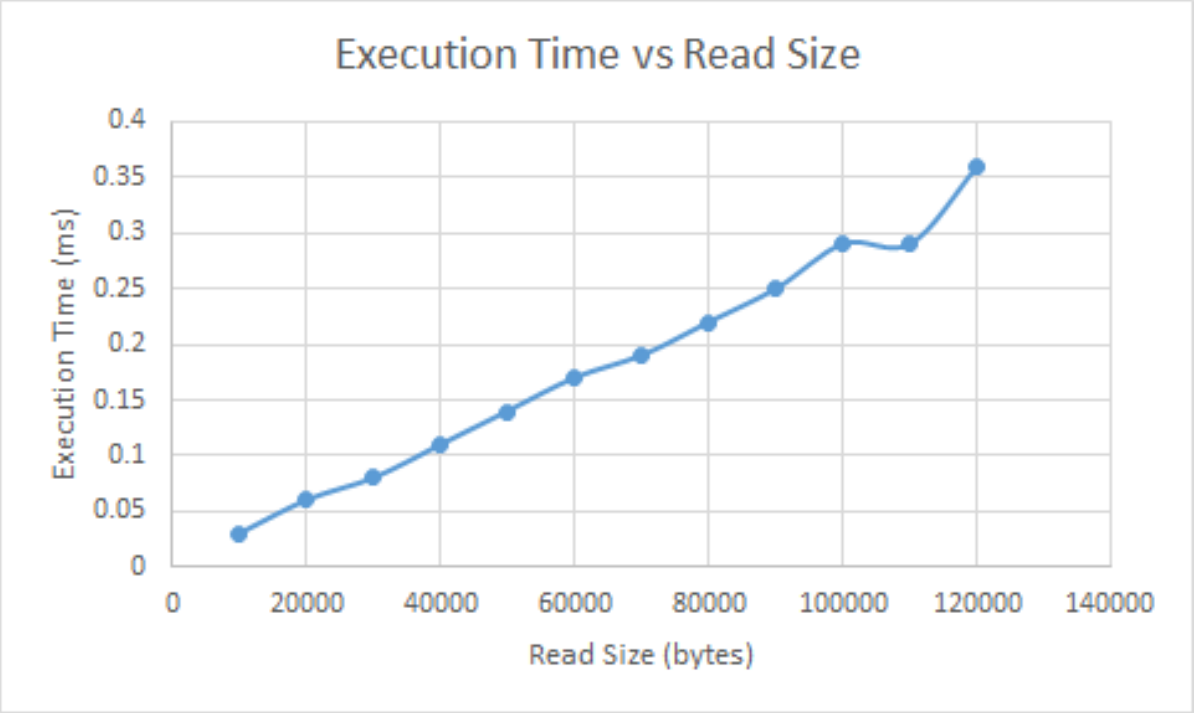
Graph 1. Elapsed Execution Time While Writing (Without Larger Inputs)



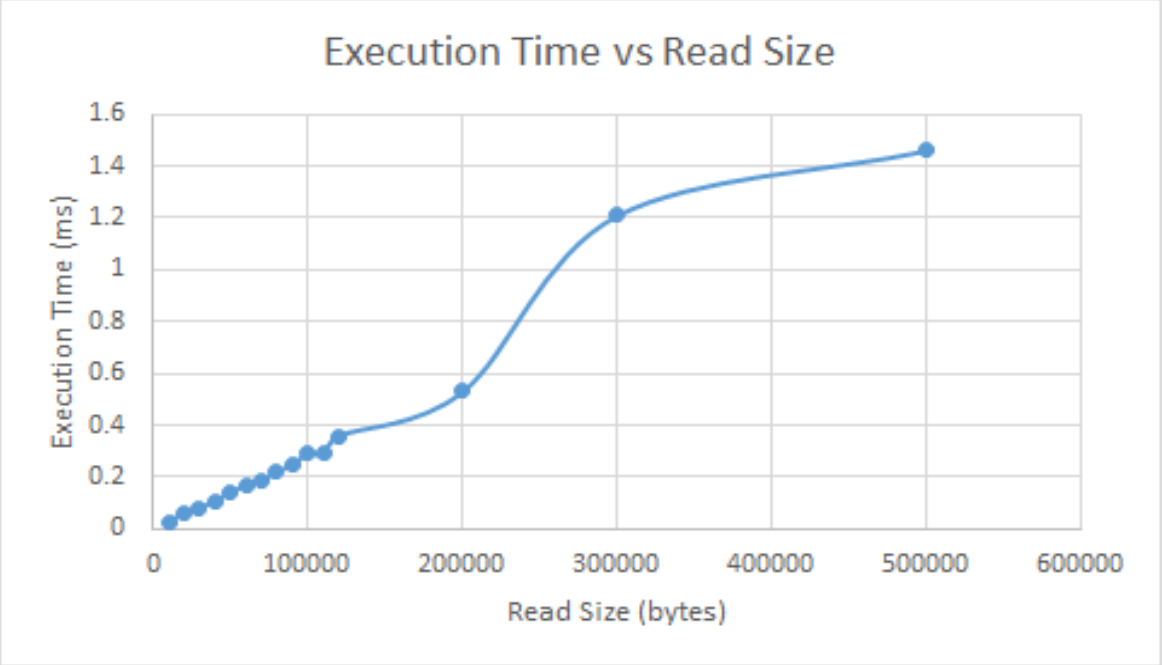
Graph 2. Elapsed Execution Time While Writing (With Larger Inputs)

N bytes (for reading a file)	Execution time (ms)
10000	0.03
20000	0.06
30000	0.08
40000	0.11
50000	0.14
60000	0.17
70000	0.19
80000	0.22
90000	0.25
100000	0.29
110000	0.29
120000	0.36
200000	0.53
300000	1.21
500000	1.46

Table 2. Elapsed Execution Time for Reading with Variable Sized File Inputs



Graph 3. Elapsed Execution Time While Reading (Without Larger Inputs)



Graph 4. Elapsed Execution Time While Reading (With Larger Inputs)

III. CONCLUSION

The first experiment results show a trend of linearity as the file size gets larger in writing. This means as the bytes we write in a file increases, the time for writing increases. This is a quick growth, and we can say that the file system's performance for writing behaves the same for both small and large file sizes.

The second experiment results show two things: a trend of linearity is shown with small file size and a logarithmic growth is shown with larger file size for reading. This means that although the elapsed time increases quickly with small file size to read, it's growth gets slower as the file size gets even larger. Thus, the file system starts to perform better as file size increases.

Comparatively, reading time is smaller than writing time. This is because, to write, we also have to read the blocks first and then write (so it includes the time of reading as well). Also, we can say that reading has an inclination to perform better with large file sizes as writing does not have that trend.