CS342 Operating Systems

Project 4

Experiments Report

Ata Seren
21901575
Section-2

Osman Semih Tiryaki
21801994
Section-1

Introduction

First of all, we read everything on the document about the FAT32 and tried to understand the purposes of its priorities. After being sure of our theoretical knowledge of FAT32, we started to implement the code to access the file system. We handled the basics and achieved to implement half of the desired functions. However, we couldn’t find a way to use given path as a parameter to access the related data. The ability to use the path is required for many desired functionalities and therefore, we couldn’t fully complete the assignment. However, we tried to understand what we couldn’t achieve and explained it in this report.

Computer Specifications

We performed experiments on a laptop with Windows 10 Home, Intel i7-9750H CPU @ 2.60GHz with 4.2GHz Turbo and 16,0 GB of RAM.

We used Oracle VirtualBox VM to run the code on virtual machine with Ubuntu 20.04. I reserved 5 CPU cores and 10725MB’s of RAM for the virtual machine. In the machine, we used Linux terminal to compile and run the code and VS Code to write the code.

Experiments

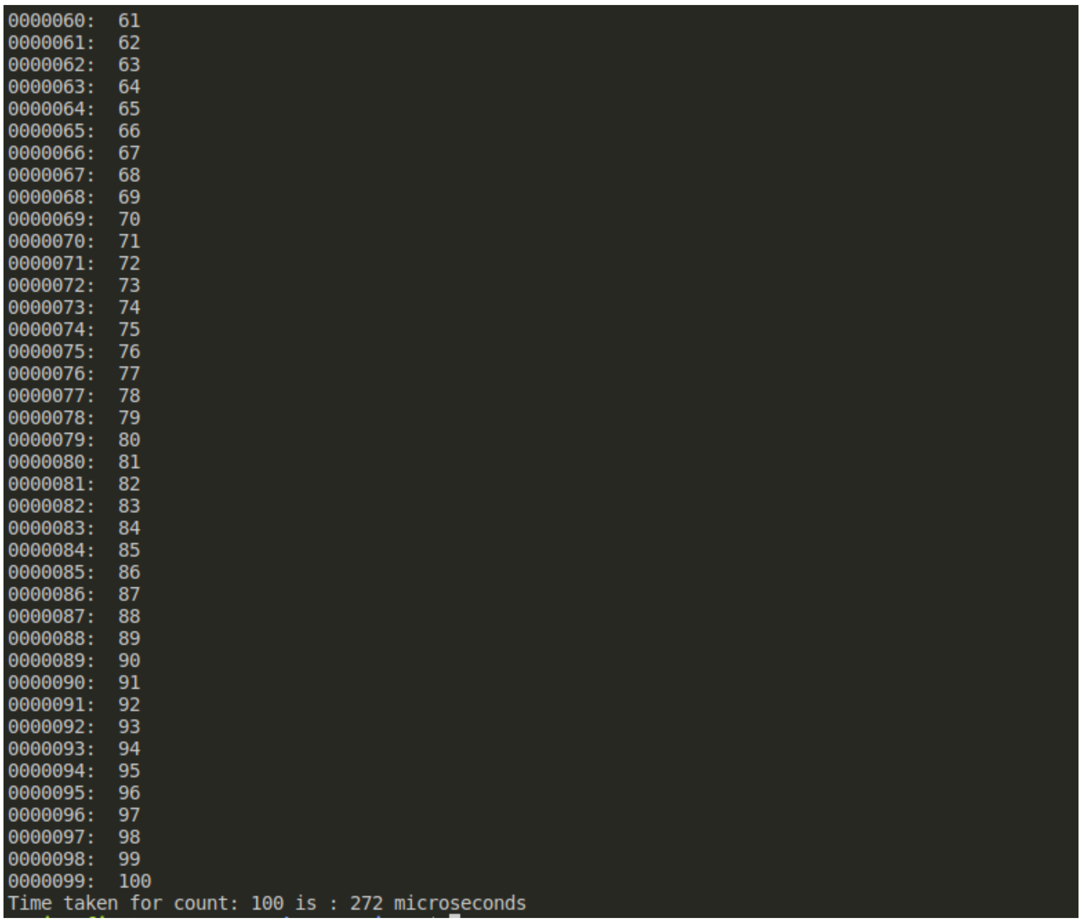
We implemented functionalities 1, 2, 3, 4, 10 and 13 and had some experiments with various parameters.

As the timing experiment, we used 10th function by changing count parameter.

Also, we had debugging and experiments on 5th function to know what the problem is.

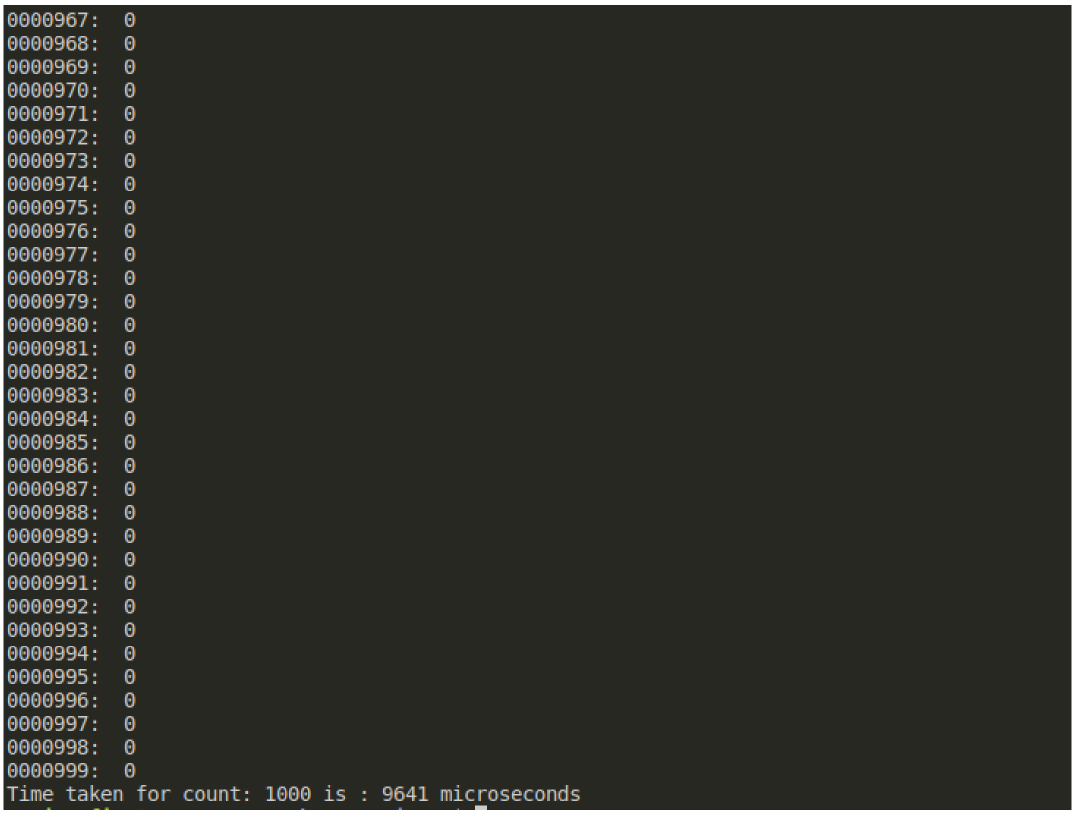
Timing experiment for 10th functionality:

Count of FAT table entries: 100



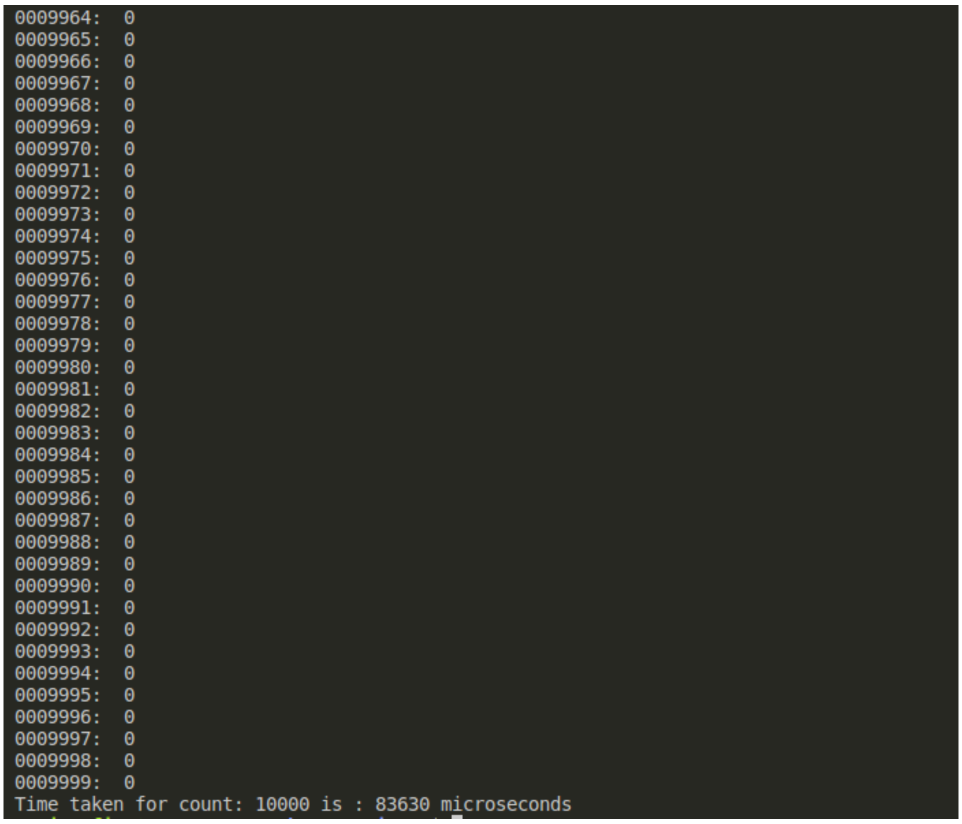
```
0000060:  61
0000061:  62
0000062:  63
0000063:  64
0000064:  65
0000065:  66
0000066:  67
0000067:  68
0000068:  69
0000069:  70
0000070:  71
0000071:  72
0000072:  73
0000073:  74
0000074:  75
0000075:  76
0000076:  77
0000077:  78
0000078:  79
0000079:  80
0000080:  81
0000081:  82
0000082:  83
0000083:  84
0000084:  85
0000085:  86
0000086:  87
0000087:  88
0000088:  89
0000089:  90
0000090:  91
0000091:  92
0000092:  93
0000093:  94
0000094:  95
0000095:  96
0000096:  97
0000097:  98
0000098:  99
0000099:  100
Time taken for count: 100 is : 272 microseconds
```

Count of FAT table entries: 1000



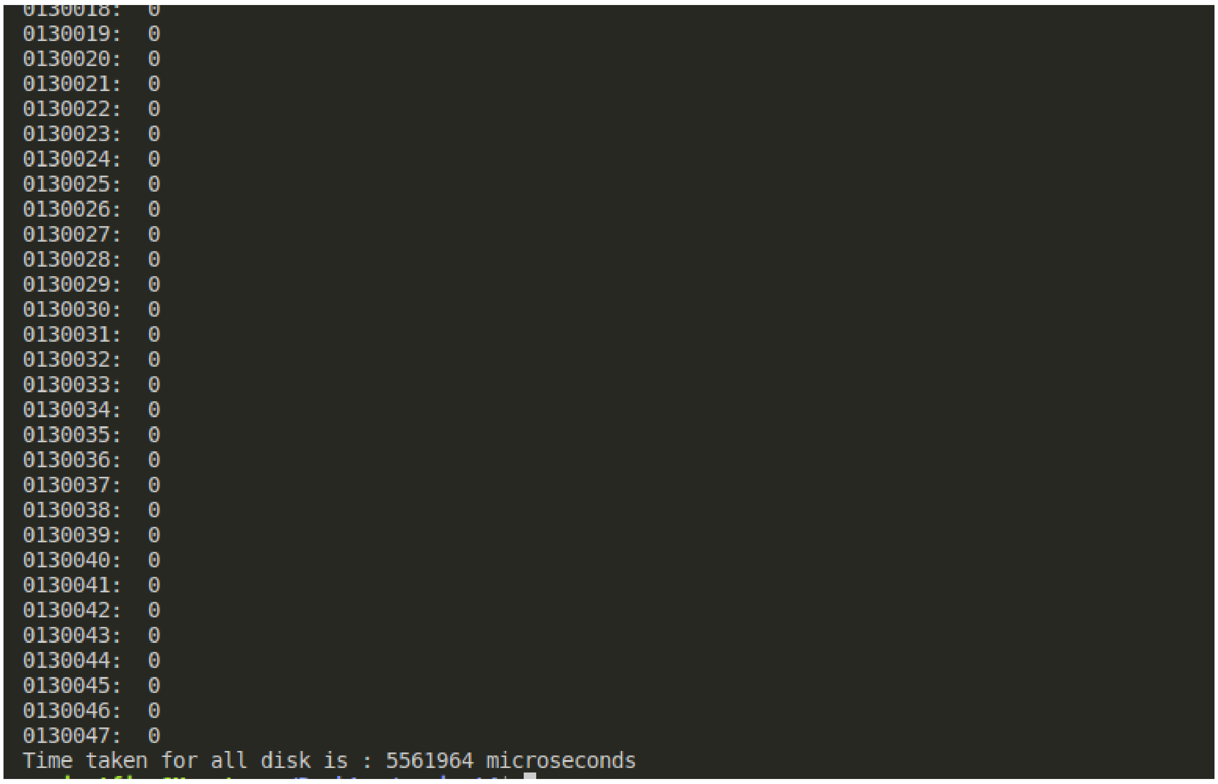
```
0000967:  0
0000968:  0
0000969:  0
0000970:  0
0000971:  0
0000972:  0
0000973:  0
0000974:  0
0000975:  0
0000976:  0
0000977:  0
0000978:  0
0000979:  0
0000980:  0
0000981:  0
0000982:  0
0000983:  0
0000984:  0
0000985:  0
0000986:  0
0000987:  0
0000988:  0
0000989:  0
0000990:  0
0000991:  0
0000992:  0
0000993:  0
0000994:  0
0000995:  0
0000996:  0
0000997:  0
0000998:  0
0000999:  0
Time taken for count: 1000 is : 9641 microseconds
```

Count of FAT table entries: 10000

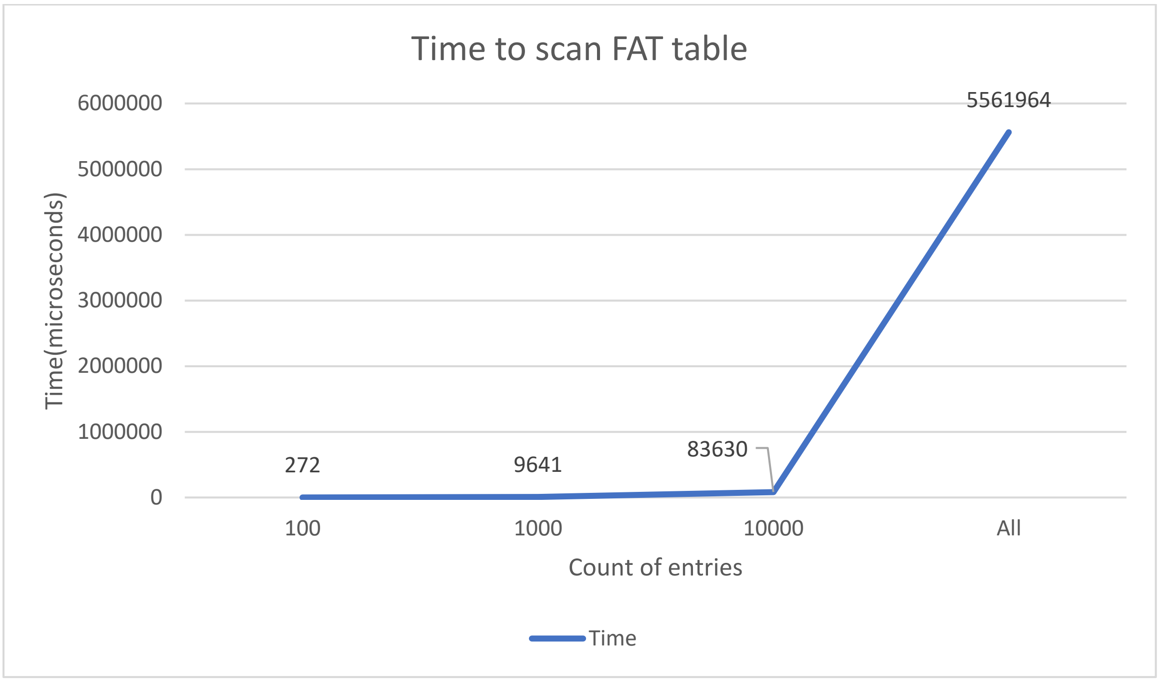


```
0009964:  0
0009965:  0
0009966:  0
0009967:  0
0009968:  0
0009969:  0
0009970:  0
0009971:  0
0009972:  0
0009973:  0
0009974:  0
0009975:  0
0009976:  0
0009977:  0
0009978:  0
0009979:  0
0009980:  0
0009981:  0
0009982:  0
0009983:  0
0009984:  0
0009985:  0
0009986:  0
0009987:  0
0009988:  0
0009989:  0
0009990:  0
0009991:  0
0009992:  0
0009993:  0
0009994:  0
0009995:  0
0009996:  0
0009997:  0
0009998:  0
0009999:  0
Time taken for count: 10000 is : 83630 microseconds
```

Measurement of all disk:



```
0130018:  0
0130019:  0
0130020:  0
0130021:  0
0130022:  0
0130023:  0
0130024:  0
0130025:  0
0130026:  0
0130027:  0
0130028:  0
0130029:  0
0130030:  0
0130031:  0
0130032:  0
0130033:  0
0130034:  0
0130035:  0
0130036:  0
0130037:  0
0130038:  0
0130039:  0
0130040:  0
0130041:  0
0130042:  0
0130043:  0
0130044:  0
0130045:  0
0130046:  0
0130047:  0
Time taken for all disk is : 5561964 microseconds
```

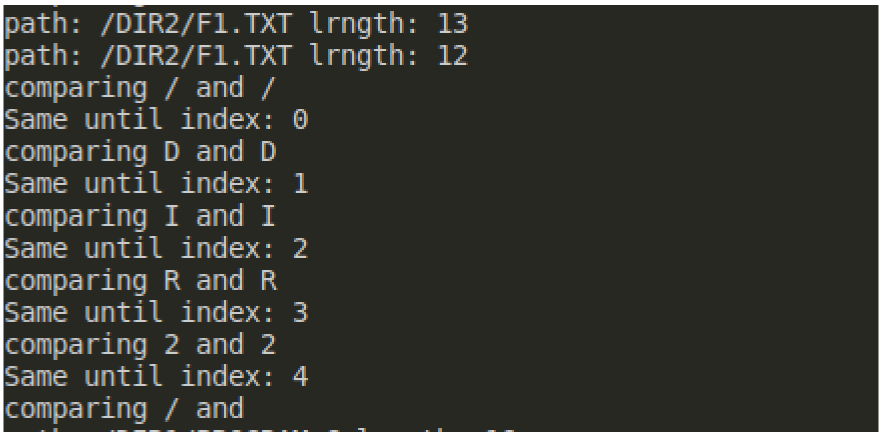


As it can be seen, time to scan the FAT table increases exponentially when count of entries increase. This exponentiality is caused because of the structure of the FAT table and our iterative algorithm.

Debugging and experiments for 5th functionality:

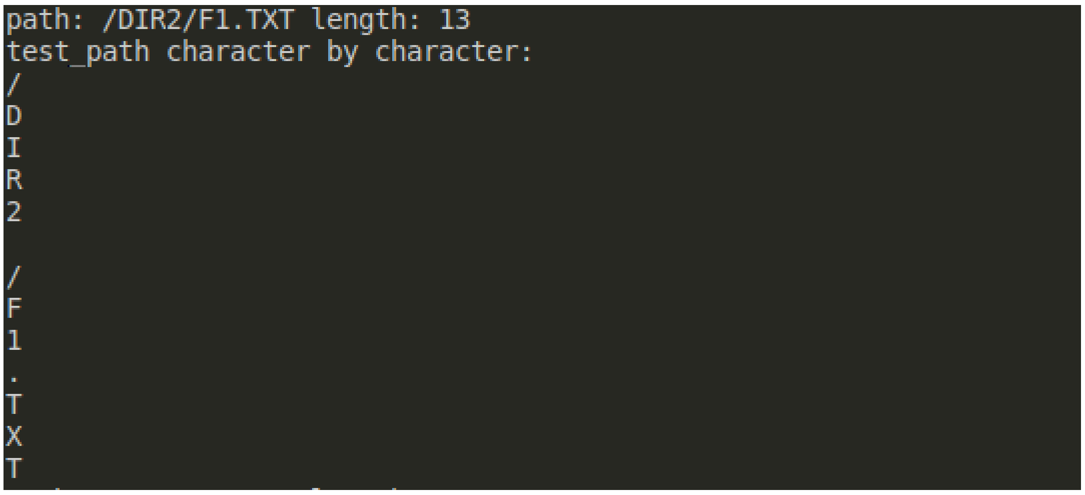
To implement this functionality, we got a lot help from 4th one since it gives us the ability to traverse the directories. In 5th functionality, we managed to reach the desired file. However, we couldn't get the number of cluster where the file belongs. We printed the desired and gathered paths and saw that they are identical. However, in comparison of any type, they do not match.

We compared them by using strcmp(), strncmp(), inspecting them char by char and even with ASCII values of chars.



```
path: /DIR2/F1.TXT lrngth: 13
path: /DIR2/F1.TXT lrngth: 12
comparing / and /
Same until index: 0
comparing D and D
Same until index: 1
comparing I and I
Same until index: 2
comparing R and R
Same until index: 3
comparing 2 and 2
Same until index: 4
comparing / and
```

Char by char comparison



```
path: /DIR2/F1.TXT length: 13
test_path character by character:
/
D
I
R
2

/
F
1
.
T
X
T
```

Printing each char to see the content

With these results, we understand that there are empty characters between chars, specifically between file or directory path names. These characters are probably appeared because of the concatenations of path names. When we inspected this empty char's ASCII value, We received 16 as decimal, which means "data link escape" character. We didn't understand how it appeared and therefore, couldn't solve it.

Results

In results of timing experiment, time to scan the FAT table increases exponentially when count of entries increase which is expected since the memory that has been inspected increases. Also, structure of the FAT32 file system and methods of accessing to it increase this time too.

In results of debugging experiment, we can see that the problem that caused us to fail to complete the implementation of the functionalities is coming from the concatenation methods we use or lack of an algorithm to make them work without errors.