UNIVERSITY OF SCIENCE
Ho Chi Minh City

INFORMATION OF TECHNOLOGY
High quality

A
Project
Report on

FILE SYSTEM MANAGEMENT ON WINDOWS

Under the course of

CTT103 – Operating System

Semester III

Sumitted by:
Class:19CLC9

1.	Ho Ngoc Minh Duc	19127368
2.	Nguyen Phuong Vy	19127088
3.	Nguyen Kim Thi To Nga	19127219

Dr. Lê Viết Long

Academic year
(2019-2023)

Table of Contents

GROUP INFORMATION

Full Name	Student ID
Hồ Ngọc Minh Đức	19127368
Nguyễn Phương Vy	19127088
Nguyễn Kim Thị Tố Nga	19127219

TASKS DISTRIBUTION

Full Name	Tasks	Complete %
Hồ Ngọc Minh Đức	Code Fat32 and app interface	100
Nguyễn Phương Vy	Code NTFS	100
Nguyễn Kim Thị Tố Nga	Write report and find information	100

INTRODUCTION

FAT32 and **NTFS** are file systems i.e., a set of logical constructs that an operating system can use to track manage files on a disk volume. Storage hardware cannot be used without a file system, but not all file systems are universally supported by all operating systems.

All operating systems support FAT32 because it is a simple file system and has been around for a long time. NTFS is more robust and effective than FAT since it makes use of advanced data structures to improve reliability, disk space utilization and overall performance. Support for NTFS has grown but is not as universal as FAT32.

In this project, we will take a deep look on how FAT32 and NTFS file systems can manage file in our computer and build a program which can list file tree in a specified disk.

PERFORMANCE

1. FAT32

Structure of FAT32

❖ ***Boot Sector:***

First, we read the Boot Sector for important information.

The Partition Boot Sector contains information that the file system uses to access the volume. On x86-based computers, the Master Boot Record use the Partition Boot Sector on the system partition to load the operating system kernel files.

The boot record in the FAT file system contains critical information about the volume, the structure of the FAT file system itself, the OS to be booted, an executable code and other detail. Once the executable code is triggered, control is handed to the operating system loaded from the partition.

The boot record area is always located at the beginning of the FAT system. The below table describes the fields in the Boot Sector for a volume formatted with the FAT file system.

Offset	Description	Size
00h	Jump Code + NOP	3 Bytes
03h	OEM Name (Probably MSWIN4.1)	8 Bytes
0Bh	Bytes Per Sector	1 Word
0Dh	Sectors Per Cluster	1 Byte
0Eh	Reserved Sectors	1 Word
10h	Number of Copies of FAT	1 Byte
15h	Media Descriptor (F8h forHard Disks)	1 Byte

18h	Sectors Per Track	1 Word
1Ah	Number of Heads	1 Word
1Ch	Number of Hidden Sectors inPartition	1 Double Word
20h	Number of Sectors inPartition	1 Double Word
24h	Number of Sectors Per FAT	1 Double Word
28h	Flags (Bits 0-4 IndicateActive FAT Copy) (Bit 7 Indicates whether FAT Mirroringis Enabled or Disabled) (If FATMirroring is Disabled, the FAT Information is onlywritten to the copy indicated by bits 0-4)	1 Word
2Ah	Version of FAT32 Drive (HighByte = Major Version, Low Byte = Minor Version)	1 Word
2Ch	Cluster Number of the Startof the Root Directory	1 Double Word
30h	Sector Number of the FileSystem Information Sector (See Structure Below)(Referenced from the Start of the Partition)	1 Word
32h	Sector Number of the BackupBoot Sector (Referenced from the Start of the Partition)	1 Word
34h	Reserved	12 Bytes
40h	Logical Drive Number ofPartition	1 Byte
41h	Unused (Could be High Byteof Previous Entry)	1 Byte
42h	Extended Signature (29h)	1 Byte
43h	Serial Number of Partition	1 Double Word
47h	Volume Name of Partition	11 Bytes
52h	FAT Name (FAT32)	8 Bytes
5Ah	Executable Code	420 Bytes
1FEh	Boot Record Signature (55hAAh)	2 Bytes

❖ ***FAT Table (FAT1, FAT2):***

FAT should contain a format identifier and some entries, each entry represents a cluster of the user area. These entries should be numbered consecutively starting from 2, and the entry number should be equal to the cluster number of the corresponding cluster.

Each item in the FAT table should indicate the status of the corresponding cluster. FAT entries should use a set of clusters to identify each file.

FAT32	Description
0x?0000000	**Free Cluster**
0x?0000001	**Reserved Cluster**
0x?0000002 - 0x?FFFFFEF	**Used Cluster, value points to next Cluster**
0x?FFFFFF0 - 0x?FFFFFF6	**Reserved values**
0x?FFFFFF7	**Bad Cluster**
0x?FFFFFF8 - 0x?FFFFFFF	**Last Cluster in File**

.FAT Entry Values

❖ ***Root Folder (Root Directory):***

Second, following the file allocation tables is the *root* directory. This area contains 32-byte entries that describe files, subdirectories, and the volume label (if present). The root directory can have a variable size. The location of the first cluster of the root directory on the FAT32 volume is stored in the "root cluster" field of the partition boot sector. In addition, the root directory does not have any associated date/time stamps. Finally, the root directory does not contain dot and dot entries.

Each file's entry in the root directory contains the number of the first cluster assigned to that file, which is used as an entry point into the FAT. From the entry point on, each FAT slot contains the cluster number of the next cluster in the file, until a last-cluster mark is encountered.

The structure of short name entries in the root directory is illustrated in the following figure: **(Main entry)**

Offset	Size	Description
0	8B	Filename
8	3B	Extension
11	1B	File attribute
12	1B	Case
13	1B	Creation time (ms)
14	2B	Creation time
16	2B	Creation date
18	2B	Last access date
20	2B	Reserved
22	2B	Last modification time
24	2B	Last modification date
26	2B	Starting cluster
28	4B	File size

One or more long name entries are used to store filenames that are more than 13 characters and up to 255 characters. Each long filename has a corresponding asub name entry. An **sub name entry** has the same structure as a short name entry:

Offset	Size (Bytes)	Description
0	1	Sequence
1	3	First 5 characters of filename
11	1	Attributes
12	1	Reserved
13	1	Checksum
14	12	Next 6 characters of filename
26	2	Reserved
28	2	Last 2 characters of filename

❖ ***SDET(Sub directory entry table):***

After read the RDET, we continue with the SDET, which locates in the DATA area. SDET contains information of files and subdirectories of a speciffied folder. This area is organized into entries just like RDET and always has 2 entries "." and "..".

- Entry ".": directory information
- Entry "..": parent directory information

2. NTFS

Structure of NTFS

❖ ***Partition boot sector:***

Read Boot Sector following the Partition Boot Sector as mentioned below.

Format program for NTFS volume allocates **first 16 sectors** for the $Boot metadata file. First sector, in fact, is a boot sector, or VBR, the *volume boot record* with a "bootstrap" code and the following 15 sectors are the boot sector's IPL (initial program loader).

The following table contains the information of NTFS's boot sector.

Offset	Length	Field name	Purpose
0x00	3 bytes	JMP instruction	Cause execution to continue after the data structures in this boot sector
0x03	8 bytes	OEM ID	This is the magic number that indicated this is an NTFS file system
0x0B	2 bytes	Bytes per sector	The number of bytes in a disk sector
0x0D	1 byte	Sector per cluster	The number of sectors in a cluster. If the value is greater than 0x80, the amount of sectors is 2 to the power of the absolute value of considering this field to be negative.
0x0E	2 bytes	Reserved Sectors, unused	How much space is reserved by the OS at the start of disk. This is always 9.
0x10	3 bytes	Unused	This field is always 0
0x13	2 bytes	Unused by NTFS	This field is always 0
0x15	1 byte	Media Descriptor	The type of drive. 0xF8 is used to denote a hard drive (in contrast to the several sizes of floppy).

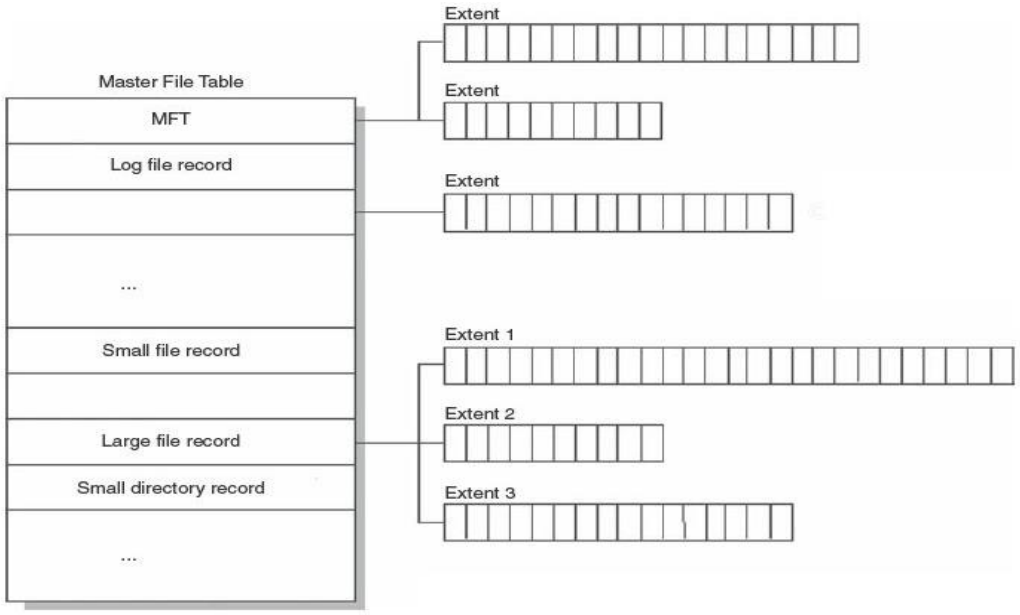
0x16	2 bytes	Unused	This field is always 0
0x18	2 bytes	Sector Per Tracks	The number of disk sectors in a drive track
0x1A	2 bytes	Number of Heads	The number of heads on the drive
0x1C	4 bytes	Hidden Sectors	The number of sectors preceding the partition
0x20	4 bytes	Unused	Not used by NTFS
0x24	4 bytes	Unused	Not used by NTFS
0x28	8 bytes	Total sectors	The partition size in sectors
0x30	8 bytes	$MFT cluster number	The cluster that contains the Master File Table
0x38	8 bytes	$MFTMirr cluster number	The cluster that contains a backup of the Master File Table
0x40	1 byte	Bytes or Clusters Per File Record Segment	A positive value denotes the number of clusters in a File Record Segment. A negative value denotes the amount of bytes in a File Record Segment, in which case the size is 2 to the power of the absolute value. (0xF6 = -10 → 2^{10} = 1024).
0x41	3 bytes	Unused	This field is not used by NTFS
0x44	1 byte	Bytes or Clusters Per Index Buffer	A positive value denotes the number of clusters in an Index Buffer. A negative value denotes the amount of bytes and it uses the same algorithm for negative numbers as the "Bytes or Clusters Per File Record Segment."
0x45	3 bytes	Unused	This field is not used by NTFS
0x48	8 bytes	Volume Serial Number	A unique random number assigned to this partition, to keep things organized.
0x50	4 bytes	Checksum, unused	Supposedly a checksum
0x54	426 bytes	Bootstrap Code	The code that loads the rest of the operating system. This is pointed to by the first 3 bytes of this sector
0x01FE	2 bytes	End-of-sector Marker	This flag indicates that this is a valid boot sector

From the Partition Boot Sector, we have the location of $MFT file.

```
MFT_start=['BytesPerSector']*['SectorsPerCluster']*['LogicalClusterNumber$MFT')
```

❖ ***Master File Table:***

MFT is a unique, database-like structured file, named the *Master File Table*, internally $MFT. The MFT contains a **record** for *every file* and *folder* on NTFS volume. First 16 entries in the MFT are reserved for NTFS metadata -- the *system files*. File attributes, size, date/time stamps, and permissions are saved in *MFT entries*. When number of files grows, the size of the MFT increases. When a file is deleted, the $MFT entry is marked free (to be reused by a new file in the future)



MFT structure

❖ ***MFT Entries:***

NTFS creates a file record for each file and a folder record for each folder created on an NTFS volume. The MFT includes a separate file record for the MFT itself. The first MFT header describes the MFT file itself.

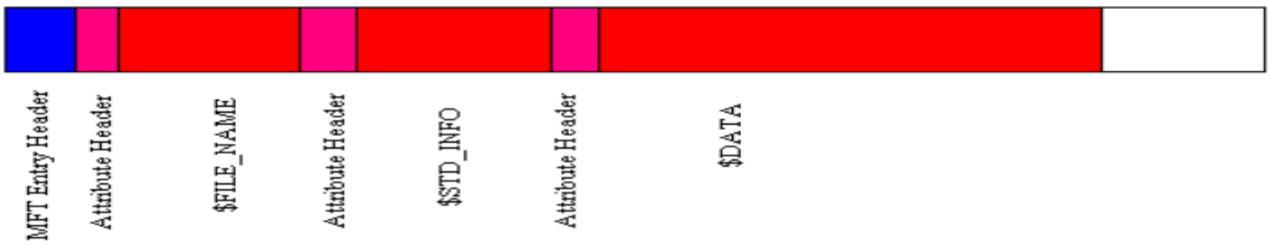
Read MFT entries sequentialy to find out if this entry is a file or folder.
The attributes of the file are kept in the MFT.

An MFT entry is very much like a database record with many optional fields.

Each $MFT entry (record) is 1,024 (0x400) bytes in length.

Each MFT entry starts with a 42-byte sized header.

Small files may be fully contained within an MFT records.



MFT Record Structure

❖ ***MFT Entry Header:***

Each MFT record, or entry, begins with ASCII string “FILE” (if there is an error in the entry, it will begin with “BAAD”) and consists of one or more attributes, each with their own identifier and structure.

If it starts with “FILE”, read MFT header then repeatedly parses the atrribute header to get type and information about these attributes based on their specified structures.

The first 42 bytes of each MFT entry comprise a header structure, and the remaining bytes depend largely on the values within the header and the various attributes contained within the entry.

The table below describes a MFT entry header.

Offset	Size	OS	Description
0x00	4		Magic Number: "FILE"
0x04	2		Offset to the update sequence.
0x06	2		Number of entries in fixup array
0x08	8		$LogFile Sequence Number (LSN)
0x10	2		Sequence number
0x12	2		Hard link count
0x14	2		Offset to first attribute
0x16	2		Flags: 0x01: record in use, 0x02 directory.
0x18	4		Used size of MFT entry
0x1C	4		Allocated size of MFT entry.
0x20	8		File reference to the base FILE record
0x28	2		Next attribute ID
0x2A	2	XP	Align to 4B boundary
0x2C	4	XP	Number of this MFT record
0x30			Attributes and fixup value

❖ ***MFT Attributes***

The data structure for the first 16 B of an attribute is the same for resident and non-resident attributes. After that, it differs. This is because non-resident attributes need to describe an arbitrary number of cluster runs, consecutive clusters that they occupy.

Offset	Size	Description	Offset	Size	Description
0x00	4	Attribute Type Identifier (see Table 4)	0x00	4	Attribute Type Identifier (see Table 4)
0x04	4	Length of Attribute (determines the location of next attribute)	0x04	4	Length of Attribute (determines the location of next attribute)
0x08	1	Non-resident flag	0x08	1	Non-resident flag
0x09	1	Lenght of name	0x09	1	Lenght of name
0x0a	2	Offset to name	0x0a	2	Offset to name
0x0c	2	Flags	0x0c	2	Flags
0x0e	2	Attribute Identifier	0x0e	2	Attribute Identifier
0x10	4	Size of content.	0x10	8	Starting Virtual Cluster Number of the runlist.
0x15	2	Offset to content	0x18	8	Ending Virtual Cluster Number of the runlist.
			0x20	2	Offset to the runlist
			0x22	2	Compression unit size
			0x24	4	unused
			0x28	8	Allocated size of the attribute content
			0x30	8	Actual size of attribute content
			0x38	8	Initialized size of the attribute content.

As mentioned previously, only the first 42 bytes are structured; after that, the rest of the MFT entry consists of one or more attribute fields. There is no formal specification or statement that says there needs to be specific attributes within an MFT entry, but for the most part, you can expect to find a $STANDARD_INFORMATION and a $FILE_NAME attribute in most MFT entries.

$STANDARD_INFORMATION			
Offset	**Size**	**OS**	**Description**
0x00	8		C Time - File Creation
0x08	8		A Time - File Altered
0x10	8		M Time - MFT Changed
0x18	8		R Time - File Read
0x20	4		DOS File Permissions
0x24	4		Maximum Number of Versions
0x28	4		Version Number
0x2C	4		Class Id
0x30	4	2K	Owner Id
0x34	4	2K	Security Id
0x38	8	2K	Quota Charged
0x40	8	2K	Update Sequence Number (USN)

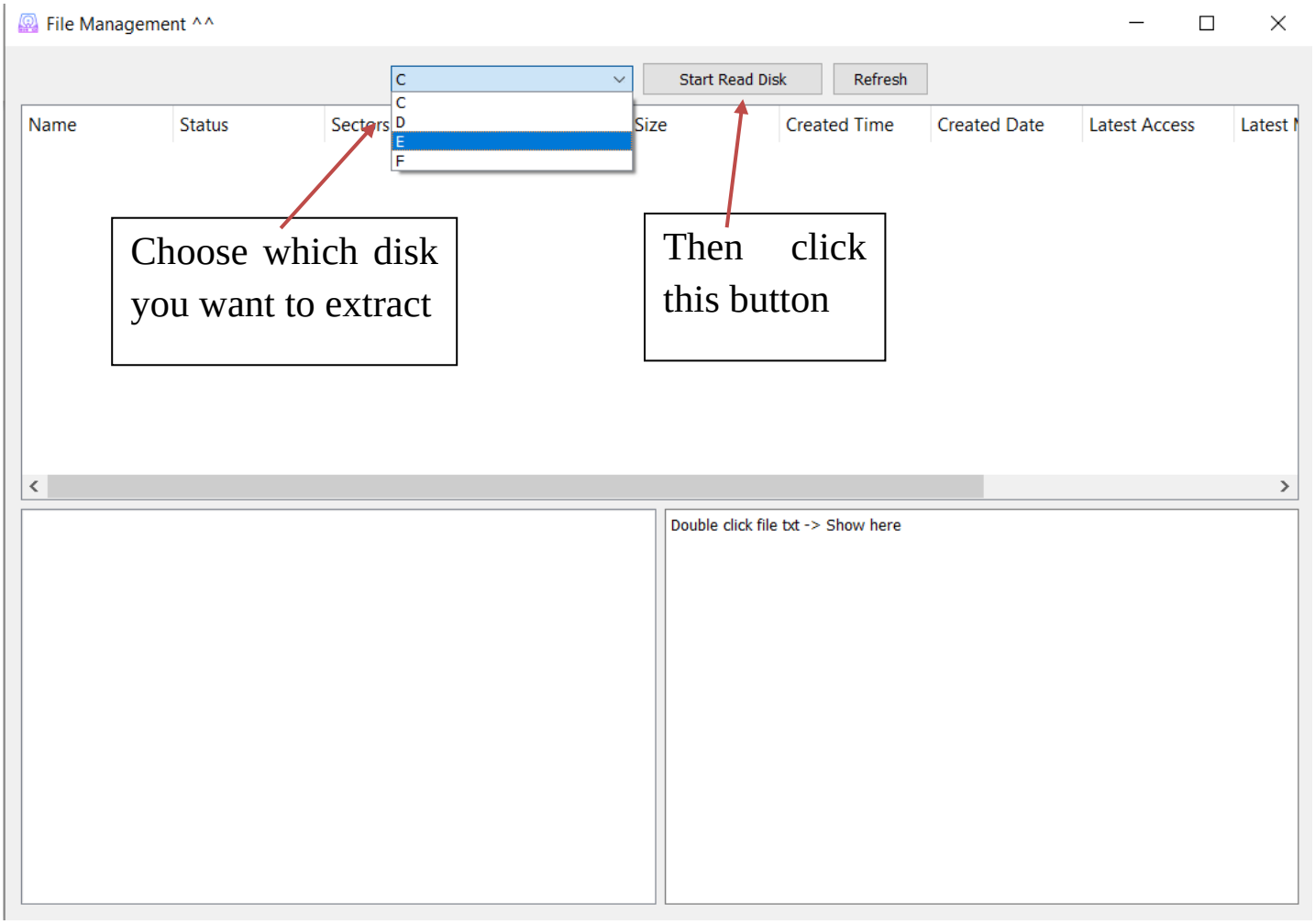
$FILE_NAME		
Offset	**Size**	**Description**
0x00	8	File Reference to parent directory
0x08	8	File creation time.
0x10	8	File modification time
0x18	8	MFT modification time
0x20	8	File access time.
0x28	8	Allocated size of file
0x30	8	Real size of file
0x38	4	Flags
0x3c	4	Used by EAs and Reparse
0x34	4	Security Id
0x40	1	Filename length in unicode characters
0x41	1	Filename namespace
0x42		File Name in Unicode

APPLICATION

We build an app which extracts file tree just like our computer.

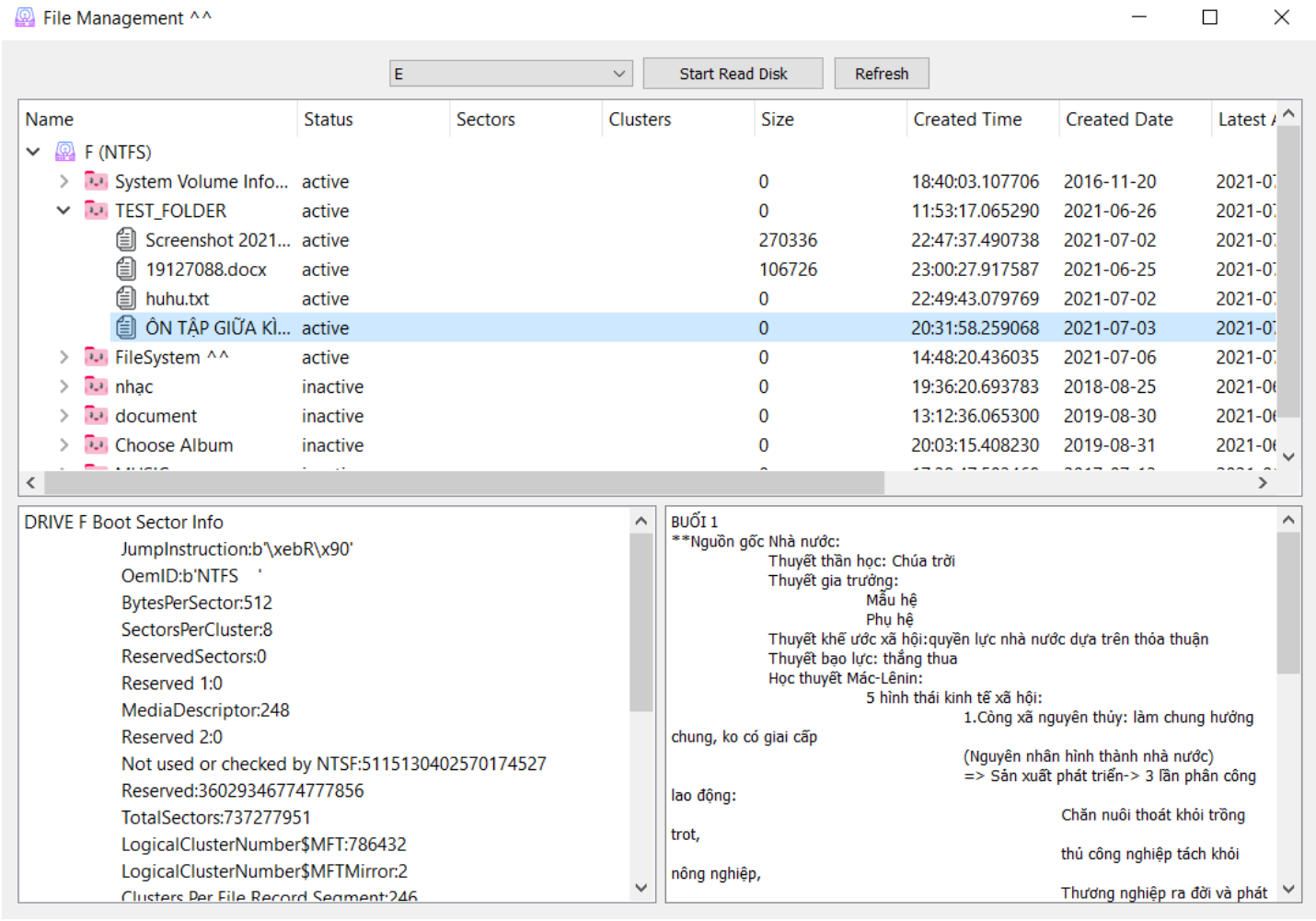
Please run file main.py to run this app.

This is the beginning interface of app.

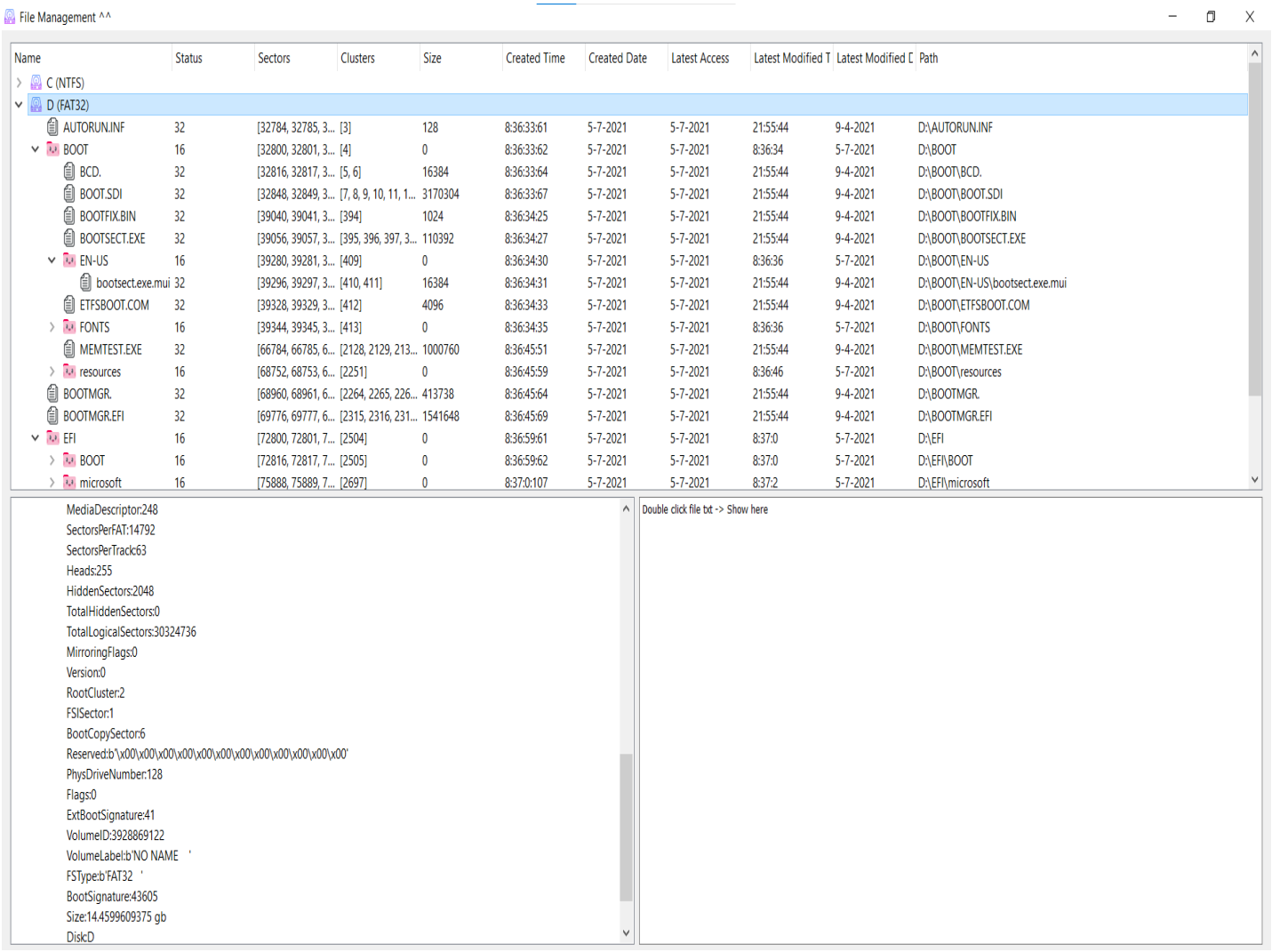


This app will show the information of Partition Boot Sector in the bottom left corner, the content of text file in the right corner and file tree in the largest area. Double click on name to open a specified file.

❖ The NTFS disk



❖ The FAT 32


File Management ^^
Name
Status
Sectors
Clusters
Size
Created Time
Created Date
Latest Access
Latest Modified T
Latest Modified D
Path
C (NTFS)
D (FAT32)
AUTORUN.INF 32 [32784, 32785, 3... [3] 128 8:36:33:61 5-7-2021 5-7-2021 21:55:44 9-4-2021 D:\AUTORUN.INF
BOOT 16 [32800, 32801, 3... [4] 0 8:36:33:62 5-7-2021 5-7-2021 8:36:34 5-7-2021 D:\BOOT
BCD. 32 [32816, 32817, 3... [5, 6] 16384 8:36:33:64 5-7-2021 5-7-2021 21:55:44 9-4-2021 D:\BOOT\BCD.
BOOT.SDI 32 [32848, 32849, 3... [7, 8, 9, 10, 11, 1... 3170304 8:36:33:67 5-7-2021 5-7-2021 21:55:44 9-4-2021 D:\BOOT\BOOT.SDI
BOOTFIX.BIN 32 [39040, 39041, 3... [394] 1024 8:36:34:25 5-7-2021 5-7-2021 21:55:44 9-4-2021 D:\BOOT\BOOTFIX.BIN
BOOTSECT.EXE 32 [39056, 39057, 3... [395, 396, 397, 3... 110392 8:36:34:27 5-7-2021 5-7-2021 21:55:44 9-4-2021 D:\BOOT\BOOTSECT.EXE
EN-US 16 [39280, 39281, 3... [409] 0 8:36:34:30 5-7-2021 5-7-2021 8:36:36 5-7-2021 D:\BOOT\EN-US
bootsect.exe.mui 32 [39296, 39297, 3... [410, 411] 16384 8:36:34:31 5-7-2021 5-7-2021 21:55:44 9-4-2021 D:\BOOT\EN-US\bootsect.exe.mui
ETFSBOOT.COM 32 [39328, 39329, 3... [412] 4096 8:36:34:33 5-7-2021 5-7-2021 21:55:44 9-4-2021 D:\BOOT\ETFSBOOT.COM
FONTS 16 [39344, 39345, 3... [413] 0 8:36:34:35 5-7-2021 5-7-2021 8:36:36 5-7-2021 D:\BOOT\FONTS
MEMTEST.EXE 32 [66784, 66785, 6... [2128, 2129, 213... 1000760 8:36:45:51 5-7-2021 5-7-2021 21:55:44 9-4-2021 D:\BOOT\MEMTEST.EXE
resources 16 [68752, 68753, 6... [2251] 0 8:36:45:59 5-7-2021 5-7-2021 8:36:46 5-7-2021 D:\BOOT\resources
BOOTMGR. 32 [68960, 68961, 6... [2264, 2265, 226... 413738 8:36:45:64 5-7-2021 5-7-2021 21:55:44 9-4-2021 D:\BOOTMGR.
BOOTMGR.EFI 32 [69776, 69777, 6... [2315, 2316, 231... 1541648 8:36:45:69 5-7-2021 5-7-2021 21:55:44 9-4-2021 D:\BOOTMGR.EFI
EFI 16 [72800, 72801, 7... [2504] 0 8:36:59:61 5-7-2021 5-7-2021 8:37:0 5-7-2021 D:\EFI
BOOT 16 [72816, 72817, 7... [2505] 0 8:36:59:62 5-7-2021 5-7-2021 8:37:0 5-7-2021 D:\EFI\BOOT
microsoft 16 [75888, 75889, 7... [2697] 0 8:37:0:107 5-7-2021 5-7-2021 8:37:2 5-7-2021 D:\EFI\microsoft
MediaDescriptor:248
SectorsPerFAT:14792
SectorsPerTrack:63
Heads:255
HiddenSectors:2048
TotalHiddenSectors:0
TotalLogicalSectors:30324736
MirroringFlags:0
Version:0
RootCluster:2
FSISector:1
BootCopySector:6
Reserved:b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
PhysDriveNumber:128
Flags:0
ExtBootSignature:41
VolumeID:3928869122
VolumeLabel:b'NO NAME '
FSType:b'FAT32 '
BootSignature:43605
Size:14.4599609375 gb
Disk:D
Double click file txt -> Show here

REFERENCES

https://www.ntfs.com/ntfs-files-types.htm

https://www.cse.scu.edu/~tschwarz/COEN252_09/Lectures/NTFS.html?fbclid=IwAR1kqquY0yY7gvaB7sLx2x1mARB5AckvqXL9vJZsj7vxgX1APenMzC1ObvQ

https://bromiley.medium.com/a-journey-into-ntfs-part-1-e2ac6a6367ec

https://www.sciencedirect.com/topics/computer-science/master-file-table

http://uw714doc.xinuos.com/en/FS_admin/_The_Root_Directory.html?fbclid=IwAR1ZwFuehQIZeaUxPtYC7hWH3NcrqCLIEbUsgIkq1zocNt3lt4fq64a81T4

https://www.cs.fsu.edu/~cop4610t/lectures/project3/Week11/Slides_week11.pdf?fbclid=IwAR0eXRufuX8LCZi9ZAq8y9rhwzzg7OOXPvNkFOyX_9AXbvuUAfhzT4kBroQ