week2-run.md

School of Engineering and Applied Science, Ahmedabad University

Operating System Lab - CSE 341

Faculty: *Prof. Mansukh Savaliya*

Project Progress Report: *Week 2*

Group Number: *3*

Group Members:

Name	Enrolment Number
Yashil Depani	AU1841005
Vidish Joshi	AU1841019
Manav Patel	AU1841037

OSProject - toaruOS

An operating systems project on understanding and solving issues for the open source OS - toaruOS.

This repository and read me contains the work done, issues faced and implementation done by us on the original OS. These steps will be similar in some cases to that of the original OS and different in some others.

Index

1. Understanding FAT32 File Systems
2. Understanding File Allocation Table
3. Staring the coding part
4. Plan for next week
5. Contribution

Understanding FAT32 File Systems

FAT32 Disk Layout

FAT is a relatively simple and unsophisticated filesystem that is understood by nearly all operating systems, including Linux and MacOS, so it's usually a common choice for firmware-based projects that need to access hard drives.

Reserved Region – Includes the boot sector, the extended boot sector, the file system information sector, and a few other reserved sectors

FAT Region – A map used to traverse the data region. Contains mappings from cluster locations to cluster locations.

Data Region – Using the addresses from the FAT region, contains actual file/directory data.

The first step to reading the FAT32 filesystem is the read its first sector, called the Volume ID. The Volume ID is read using the LBA Begin address found from the partition table. LBA is Logical block addressing is a common scheme used for specifying the location of blocks of data stored on computer storage devices. We can read it with LBA = 0. LBA just numbers the sectors (512 Bytes per sector) sequentially starting at zero.

Field	Microsoft's Name	Offset	Size	Value
Bytes Per Sector	BPB_BytsPerSec	0x0B	16 Bits	Always 512 Bytes
Sectors Per Cluster	BPB_SecPerClus	0x0D	8 Bits	1,2,4,8,16,32,64,128
Number of Reserved Sectors	BPB_RsvdSecCnt	0x0E	16 Bits	Usually 0x20
Number of FATs	BPB_NumFATs	0x10	8 Bits	Always 2
Sectors Per FAT	BPB_FATSz32	0x24	32 Bits	Depends on disk size
Root Directory First Cluster	BPB_RootClus	0x2C	32 Bits	Usually 0x00000002
Signature	(none)	0x1FE	16 Bits	Always 0xAA55

Simple C code for above image:

```
fat_begin_lba = Partition_LBA_Begin + Number_of_Reserved_Sectors;

cluster_begin_lba = Partition_LBA_Begin + Number_of_Reserved_Sectors + (Number_of_FATs * Sectors_Per_FAT);

sectors_per_cluster = BPB_SecPerClus;

root_dir_first_cluster = BPB_RootClus;

lba_addr = cluster_begin_lba + (cluster_number - 2) * sectors_per_cluster;
```

Understanding File Allocation Table

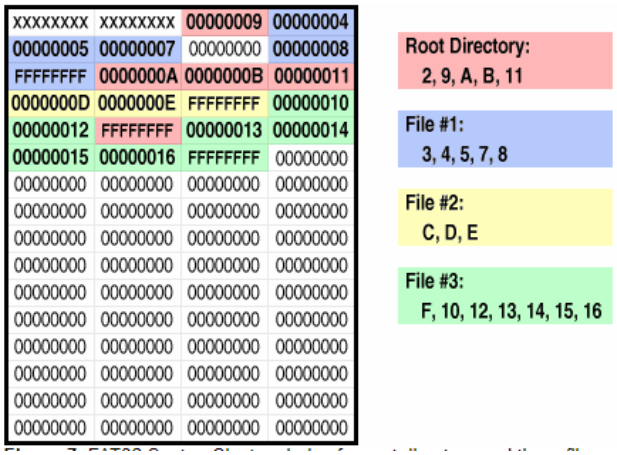
The directory entries tell us where the first cluster of each file (or subdirectory) is located on the disk, and of course you find the first cluster of the root directory from the volume ID sector. To access all the other clusters of a file beyond the first one, we need to use the File Allocation Table.

The File Allocation Table is a big array of 32 bit integers. We want to understand what the entries in this table actually mean. Each cell in this table contains *memory address pointer* to the current directory and and the files which are present inside the directories.

Here are the possible entries in a FAT cell:

- 0 - unallocated
- FF7, FFF7 or FFF FFF7 (12, 16, 32) - bad cluster
- FF8, FFF8, FFF FFF8 (12, 16, 32) - EOF
- All other values mean the cluster is allocated.

Let us understand the following File Allocation Table and the entries inside them:



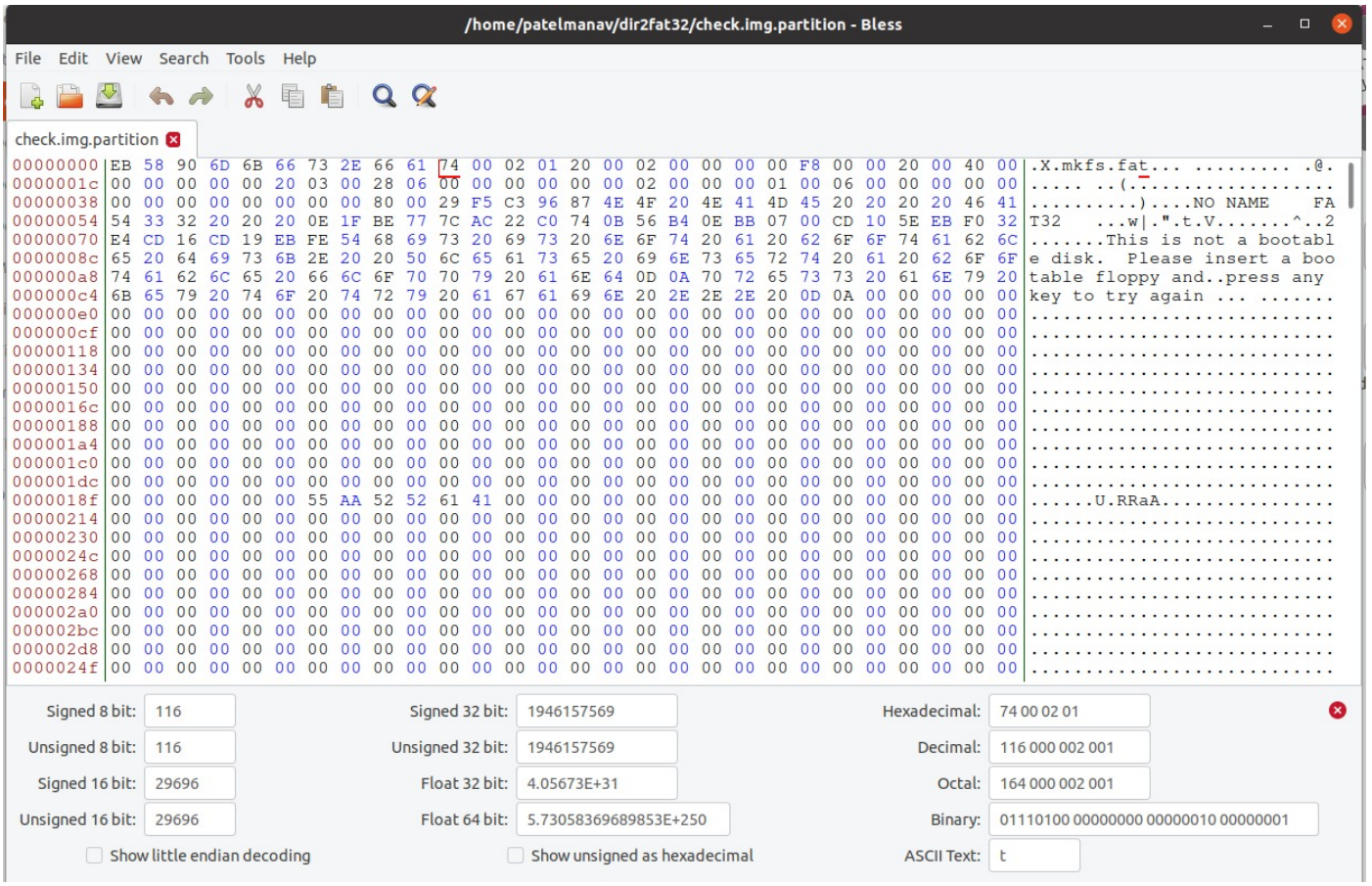
*Here, all the files and directories shown in the image are present inside the **root** directory.*

*The cells in the table are numbered starting from **0** . Here in the above table the root directory starts at memory address 2. This directory has memory link to address 9. This cell has a link to address 10 which has a link to address 11. The cell number 11 end-of-file.*

From memory address 3 starts File #1. Cell 3 has link to 4, which links to cell 5. Cell 5 has link to cell 7. Notice that cell number 6 is skipped and it contains 000000 as entry. This shows that cell 6 is unallocated. Cell 7 links to cell 8. Cell 8 is the end-of-file. Similarly we can observe for the rest of the files.

We installed **Bless** Hex editor to read binary files in a FAT32 image to read the file structure maintained in the File Allocation Table in that image.

Below is the image when a FAT image file is viewd in *Bless* editor:



Starting the Coding Part

One of our main aim for this week was to start the coding part *or* at-least understand the concepts necessary to start the coding for the project.

FAT32 file system, which was developed by Microsoft for the DOS OS, has a set of standards which are followed while using this FS. The File Allocation Table is created and maintained according to particular set of guidelines which have been found to maximize the performance of this FS.

All this standards have been written in the Microsoft Documnetation of FAT32.

Steps to read from a FAT32 image:

1. Locate, read, and extract important info from the Boot Sector
2. Locate the Root Directory, get the list of files and folders
3. Access the files and directories using information from the Root Directory and the FAT32 table

Various attributes that are required to maintain the table for each region of the memory are defined and documented in detail in this manual. Below are images from the manual the fields used in the *BIOS Parameter Block(BPB)* located in the first sector of the volume in the *Reserved Region.*

3.1 BPB structure common to FAT12, FAT16, and FAT32 implementations

The below table describes the fields in the BPB that are common to all FAT variants.

Descriptive name of field	Offset (byte)	Size (bytes)	Description
BS_jmpBoot	0	3	Jump instruction to boot code. This field has two allowed forms: **jmpBoot[0] = 0xEB, jmpBoot[1] = 0x??, jmpBoot[2] = 0x90** and **jmpBoot[0] = 0xE9, jmpBoot[1] = 0x??, jmpBoot[2] = 0x??** **0x??** indicates that any 8-bit value is allowed in that byte. What this forms is a three-byte Intel x86 unconditional branch (jump) instruction that jumps to the start of the operating system bootstrap code. This code typically occupies the rest of sector 0 of the volume following the BPB and possibly other sectors. Either of these forms is acceptable. **JmpBoot[0] = 0xEB** is the more frequently used format.

BS_OEMName	3	8	OEM Name Identifier. Can be set by a FAT implementation to any desired value. Typically this is some indication of what system formatted the volume.
BPB_BytsPerSec	11	2	Count of bytes per sector. This value may take on only the following values: 512, 1024, 2048 or 4096.
BPB_SecPerClus	13	1	Number of sectors per allocation unit. This value must be a power of 2 that is greater than 0. The legal values are 1, 2, 4, 8, 16, 32, 64, and 128.
BPB_RsvdSecCnt	14	2	Number of reserved sectors in the reserved region of the volume starting at the first sector of the volume. This field is used to align the start of the data area to integral multiples of the cluster size with respect to the start of the partition/media. This field must not be 0 and can be any non-zero value. This field should typically be used to align the start of the data area (cluster #2) to the desired alignment unit, typically cluster size.
BPB_NumFATs	16	1	The count of file allocation tables (FATs) on the volume. A value of 2 is recommended although a value of 1 is acceptable.

The code should work along with these attributes and fields since these fields will hold the information about the file structure.

Plan for next week

Read and understand the Microsoft documentation for FAT32 system.

Along with it, start coding the blocks that are understood. Refer to the documentation whenever necessary.

The following link is also useful along with the documentation as it explains concepts very clearly:

- https://www.pjrc.com/tech/8051/ide/fat32.html

This YouTube lecture series is good. Great explanation + implementation is done in this series. Refer this in the coming week.

- https://www.youtube.com/playlist?list=PLHh55M_Kq4OApWScZyPl5HhgsTJS9MZ6M

Contribution

Reading/understanding the Microsoft documentation - All 3 members

Starting to write the code - All 3 members

Understanding the working of a FAT table - All 3 members

Working around creating and manipulating .img file - All 3 members