The FAT-12 Filesystem

April 5, 2016

1 Introduction

In this lab, you will gain hands-on experience reading a FAT-12 filesystem. Using information we provide, you will decode the boot sector by hand. You will then write a program to do this automatically. You may findthis website very useful as you work on this lab.

1.1 Terms

1.1.1 Sector

The smallest unit of transfer; It's also called a block. There are 512 bytes / sector on a floppy disk.

1.1.2 Boot Sector

Stores the vital information about the filesystem. The boot sector is laid out in the following way, starting at the beginning of the disk(logical sector 0, byte 0): All values are stored as unsigned little-endian numbers unless otherwise specified.

Offset	Length	Contents	Display Format
0x00	3	Binary offset of boot loader	hex
0x03	8	Volume Label (ASCII, null padded)	ASCII
0x0B	2	Bytes / sector	decimal
0x0D	1	Sectors / cluster	decimal
0x0E	2	Reserved sectors	decimal
0x10	1	Number of FATs (generally 2)	decimal
0x11	2	Number of Root Directory entries	decimal
0x13	2	Number of logical sectors	decimal
0x15	1	Medium descriptor	hex
0x16	2	Sectors per FAT	decimal
0x18	2	Sectors per track	decimal
0x1A	2	Number of heads	decimal
0x1C	2	Number of hidden sectors	decimal

1.2 Useful system calls

More information on these system calls can be found in `man 2 open`, `man 2 read`, and `man 2 lseek`.

```
int open(const char* path, int flags)
```

Opens a file at `path` and returns a file descriptor. For example, `open("/tmp/myfile", ORDONLY)` opens an existing file called `myfile` in the `tmp` directory in read-only mode. It returns a file descriptor that can be used like a handle to the open file.

```
ssize_t read(int fd, void *buf, size_t count)
```

Reads `count` bytes from the file descriptor `fd` into the memory location starting at `buf`. It starts reading from the current offset into the file. The offset is set to zero if the file has just been opened.

```
off_t lseek(int fd, off_t offset, int whence)
```

Sets the offset of the file descriptor `fd` to `offset`, depending on whence. If `whence` is `SEEK_SET`, then the offset is figured from the start of the file. If `whence` is `SEEK_CUR`, then the offset is relative to the current offset.

1.3 Inspecting binary files

You should have a few floppy disk images available in the lab folder, along with skeleton code to decode the boot sector. In Unix, devices are treated as files. For example, the floppy drive is normally found at a device file such as `/dev/fd0`. We can use a file image as a substitute for an actual floppy disk. The floppy images contain a real FAT-12, FAT-16, and FAT-32 filesystems. If you know how the byts are laid out, then you can decode the structures that describe the filesystem and work with the files. To help you get started, we're going to decode a few fields by hand.

There is a utility in Unix called `hexdump` that will show the binary contents of a file. This is useful to inspect a file containing binary data. If you use hexdump -C on one of the suppiled images, the output will look something like this(use 'hexdump -C image.fat12 | less to more easily read it)

```
00000000  eb 3c 90 6d 6b 66 73 2e  66 61 74 00 02 04 01 00  |.<.mkfs.fat.....|
00000010  02 00 02 00 08 f8 02 00  20 00 40 00 00 00 00 00  |........ .@.....|
00000020  00 00 00 00 80 00 29 1e  71 c8 d0 43 50 52 45 33  |......).q..CPRE3|
00000030  30 38 2d 31 32 20 46 41  54 31 32 20 20 20 0e 1f  |08-12 FAT12   ..|
00000040  be 5b 7c ac 22 c0 74 0b  56 b4 0e bb 07 00 cd 10  |.[|.".t.V.......|
00000050  5e eb f0 32 e4 cd 16 cd  19 eb fe 54 68 69 73 20  |^..2.......This |
...[there should be a lot more after this]
```

The first column is the hexadecimal offset into the file. Since each line contains 16 bytes, this will always end in a 0. The middle 16 columns are the hexadecimal bytes of the file. The third column contains the ASCII representation of the bytes, or a `.` if the byte is not a printable character. For example, the second byte, having an offset of 0x01, has the value 0x3c corresponding to the character `<`, while the third byte, with a value of 0x90, is not printable, so it is shown as a dot.

2 Exercises

2.1 Decoding the boot sector by hand

Using `hexdump` and the offsets in the tables above, find and decode the values for each of the following fields in the boot sector(remember that it starts at offset 0):

	Hex	Decimal
Bytes / sector		
Sectors / cluster		
Root directory entries		
Sectors / FAT		

Have the lab TA check your answers. You will use these values for debugging the next part.

3 Task for this lab

Complete the skeleton code given to decode and print the boot sector of the FAT-12 filesystem. The starting offset and size of each field can be found in the table at the beginning of the lab handout. All of the information is stored as binary values on the disk. When the values are to be printed, use the same format as the table at the beginning of the handout. The program should: - Take the name of the file to read as an argument - decode the boot sector of the file given - print out the values in the boot sector - no segmentation faults should occur - meaning all system calls should be checked if they were successful - if an error did occur, it should print out an error message detailing what caused it, and exit gracefully

4 Extra credit

Extra credit can be had for extending the boot sector parser to handle the FAT-16 and FAT-32 filesystems. Consult the OS Dev website regarding the boot sector for the other two filesystems, which can be found here.

5 License