An Introduction to DOS FAT Volume and File Structure

Mark Kampe markk@cs.ucla.edu

1. Introduction

When the first personal computers with disks became available, they were very small (a few megabytes of disk and a few dozen kilobytes of memory). A file system implementation for such machines had to impose very little overhead on disk space, and be small enough to fit in the BIOS ROM. BIOS stands for **BASIC I/O Subsystem**. Note that the first word is all upper-case. The purpose of the BIOS ROM was to provide run-time support for a BASIC interpreter (which is what Bill Gates did for a living before building DOS). DOS was never intended to provide the features and performance of real timesharing systems.

Disk and memory size have increased in the last thirty years, People now demand state-of-the-art power and functionality from their PCs. Despite the evolution that the last decades have seen, old standards die hard. Much as European train tracks maintain the same wheel spacing used by Roman chariots, most modern OSs still support DOS FAT file systems. DOS file systems are not merely around for legacy reasons. The ISO 9660 CDROM file system format is a descendent of the DOS file system.

The DOS FAT file system is worth studying because:

- It is heavily used all over the world, and is the basis for more modern file system (like 9660).
- It provides reasonable performance (large transfers and well clustered allocation) with a very simple implementation.
- It is a very successful example of "linked list" space allocation.

2. Structural Overview

All file systems include a few basic types of data structures:

- bootstrap

  code to be loaded into memory and executed when the computer is powered on. MVS volumes reserve the entire first track of the first cylinder for the boot strap.

- volume descriptors

  information describing the size, type, and layout of the file system ... and in particular how to find the other key meta-data descriptors.

- file descriptors

  information that describes a file (ownership, protection, time of last update, etc.) and points where the actual data is stored on the disk.

- free space descriptors

  lists of blocks of (currently) unused space that can be allocated to files.

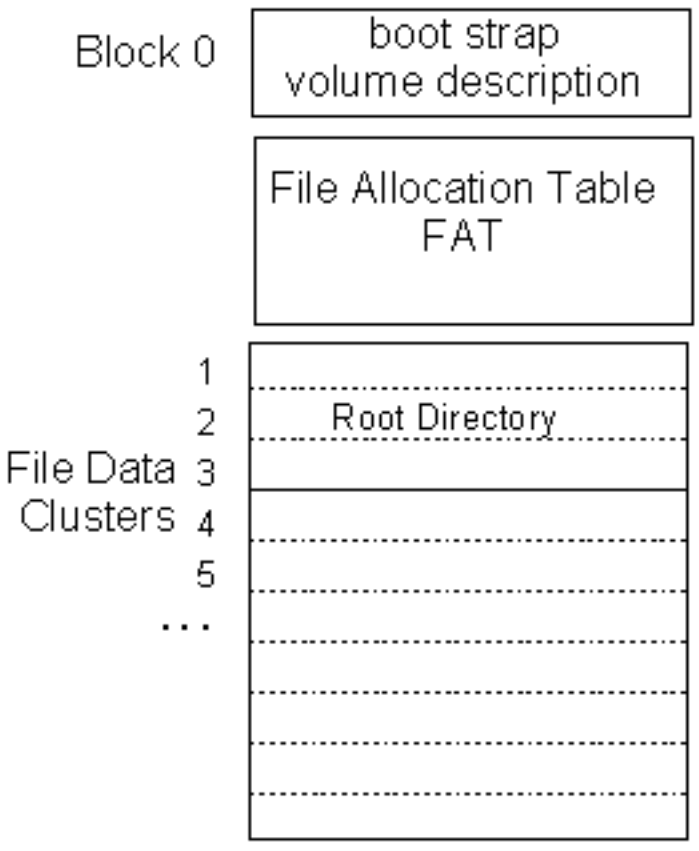
- file name descriptors

  data structures that user-chosen names with each file.

DOS FAT file systems divide the volume into fixed-sized (physical) blocks, which are grouped into larger fixed-sized (logical) block clusters.

The first block of DOS FAT volume contains the bootstrap, along with some volume description information. After this comes a much longer **File Allocation Table** (FAT from which the file system takes its name). The File Allocation Table is used, both as a free list, and to keep track of which blocks have been allocated to which files.

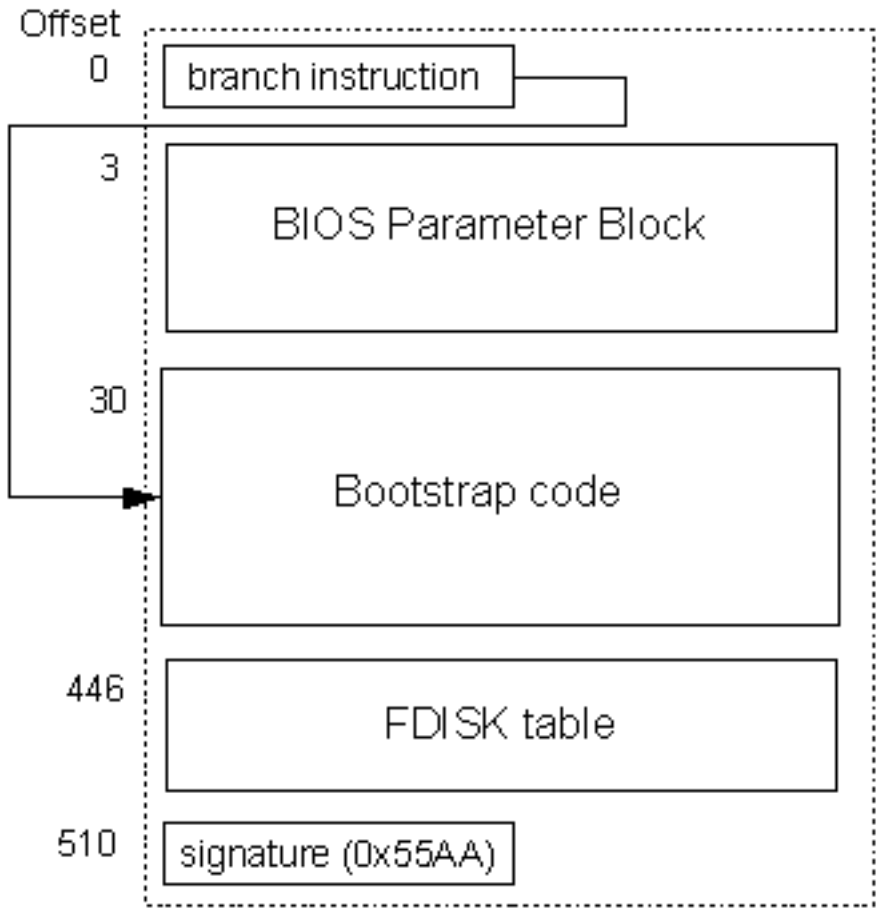
The remainder of the volume is data clusters, which can be allocated to files and directories. The first file on the volume is the root directory, the top of the tree from which all other files and directories on the volume can be reached.



3. Boot block BIOS Parameter Block and FDISK Table

Most file systems separate the first block (pure bootstrap code) from volume description information. DOS file systems often combine these into a single block. The format varies between (partitioned) hard disks and (unpartitioned) floppies, and between various releases of DOS and Windows ... but conceptually, the boot record:

- begins with a branch instruction (to the start of the real boostrap code).
- followed by a volume description (BIOS Parameter Block)
- followed by the real bootstrap code
- followed by an optional disk partitioning table
- followed by a signature (for error checking).



3.1 BIOS Parameter Block

After the first few bytes of the bootstrap comes the BIOS parameter block, which contains a brief summary of the device and file system. It describes the device geometry:

- number of bytes per (physical) sector
- number of sectors per track
- number of tracks per cylinder
- total number of sectors on the volume

It also describes the way the file system is layed out on the volume:

- number of sectors per (logical) cluster
- the number of reserved sectors (not part of file system)
- the number of Alternate File Allocation Tables
- the number of entries in the root directory

These parameters enable the OS to interpret the remainder of the file system.

3.2 FDISK Table

As disks got larger, the people at MicroSoft figured out that their customers might want to put multiple file systems on each disk. This meant they needed some way of partitioning the disk into logical sub-disks. To do this, they added a small partition table (sometimes called the FDISK table, because of the program that managed it) to the end of the boot strap block.

This FDISK table has four entries, each capable of describing one disk partition. Each entry includes

- A partition type (e.g. Primary DOS partition, UNIX partition).
- An ACTIVE indication (is this the one we boot from).
- The disk address where that partition starts and ends.
- The number of sectors contained within that partition.

Partn	Type	Active	Start (C:H:S)	End (C:H:S)	Start (logical)	Size (sectors)

1	LINUX	True	1:0:0	199:7:49	400	79,600
2	Windows NT		200:0:0	349:7:49	80,000	60,000
3	FAT 32		350:0:0	399:7:49	140,000	20,000
4	NONE					

In older versions of DOS the starting/ending addresses were specified as cylinder/sector/head. As disks got larger, this became less practical, and they moved to logical block numbers.

The addition of disk partitioning also changed the structure of the boot record. The first sector of a disk contains the Master Boot Record (MBR) which includes the FDISK table, and a bootstrap that finds the active partition, and reads in its first sector (Partition Boot Record). Most people (essentially everyone but Bill Gates :-) make their MBR bootstrap ask what system you want to boot from, and boot the active one by default after a few seconds. This gives you the opportunity to choose which OS you want to boot. Microsoft makes this decision for you ... you want to boot Windows.

The structure of the Partition Boot Record is entirely operating system and file system specific ... but for DOS FAT file system partitions, it includes a BIOS Parameter block as described above.

4. File Descriptors (directories)

In keeping with their desire for simplicity, DOS file systems combine both file description and file naming into a single file descriptor (directory entries). A DOS directory is a file (of a special type) that contains a series of fixed sized (32 byte) directory entries. Each entry describes a single file:

- an 11-byte name (8 characters of base name, plus a 3 character extension).
- a byte of attribute bits for the file, which include:
  - Is this a file, or a sub-directory.
  - Has this file changed since the last backup.
  - Is this file hidden.
  - Is this file read-only.
  - Is this a system file.
  - Does this entry describe a volume label.
- times and dates of creation and last modification, and date of last access.
- a pointer to the first logical block of the file. (This field is only 16 bits wide, and so when Microsoft introduced the FAT32 file system, they had to put the high order bits in a different part of the directory entry).
- the length (number of valid data bytes) in the file.

Name (8+3)	Attributes	Last Changed	First Cluster	Length
.	DIR	08/01/03 11:15:00	61	2,048
..	DIR	06/20/03 08:10:24	1	4,096
MARK	DIR	10/15/04 21:40:12	130	1,800
README.TXT	FILE	11/02/04 04:27:36	410	31,280

If the first character of a files name is a NULL (0x00) the directory entry is unused. The special character (0xE5) in the first character of a file name is used to indicate that a directory entry describes a deleted file. (See the section on Garbage collection below)

Note on times and dates:

DOS stores file modification times and dates as a pair of 16-bit numbers:

- 7 bits of year, 4 bits of month, 5 bits of day of month
- 5 bits of hour, 6 bits of minute, 5 bits of seconds (x2).

All file systems use dates relative to some epoch (time zero). For DOS, the epoch is midnight, New Year's Eve, January 1, 1980. A seven bit field for years means that the the DOS calendar only runs til 2107. Hopefully, nobody will still be using DOS file systems by then :-)

5. Links and Free Space (File Allocation Table)

Many file systems have very compact (e.g. bitmap) free lists, but most of them use some per-file data structure to keep track of which blocks are allocated to which file. The DOS File Allocation Table is a relatively unique design. It contains one entry for each logical block in the volume. If a block is free, this is indicated by the FAT entry. If a block is allocated to a file, the FAT entry gives the logical block number of the **next** logical block in the file.

5.1 Cluster Size and Performance

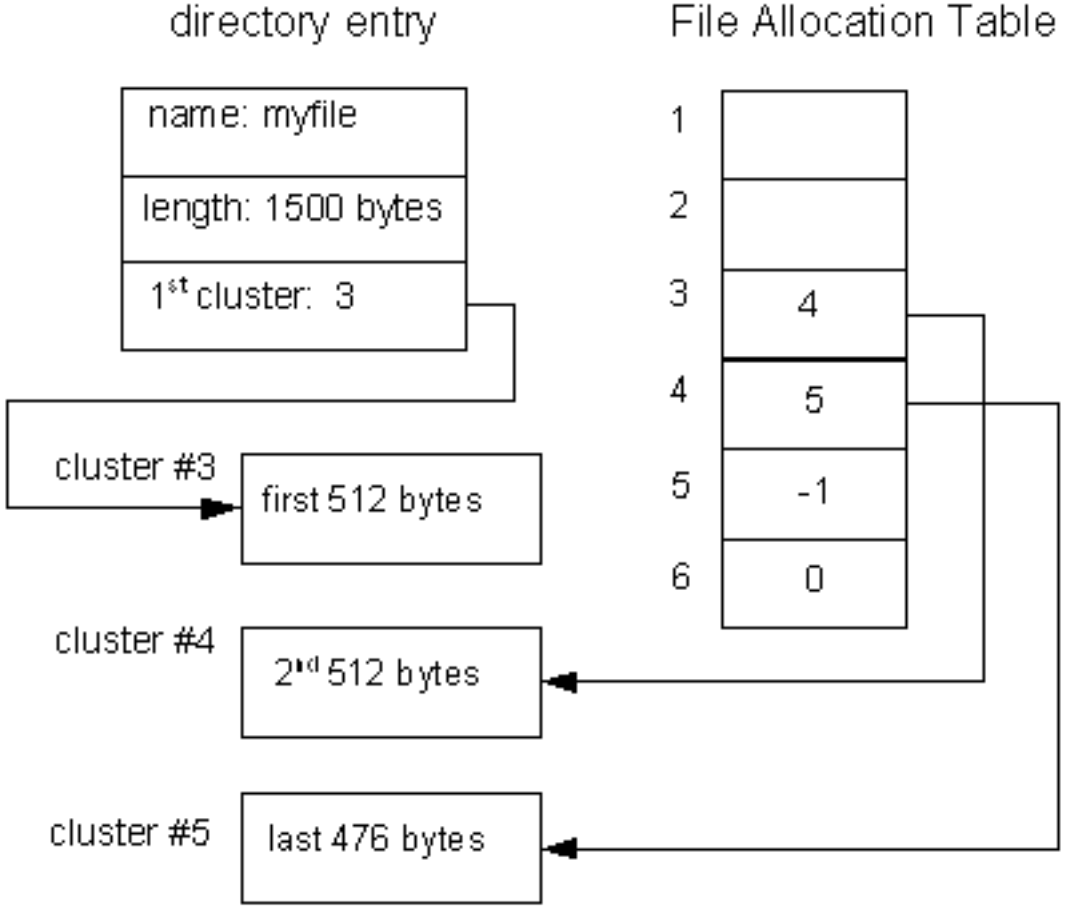
Space is allocated to files, not in (physical) blocks, but in (logical) multi-block clusters. The number of clusters per block is determined when the file system is created.

Allocating space to files in larger chunks improves I/O performance, by reducing the number of operations required to read or write a file. This comes at the cost of higher internal fragmentation (since, on average, half of the last cluster of each file is left unused). As disks have grown larger, people have become less concerned about internal fragmentation losses, and cluster sizes have increased.

The maximum number of clusters a volume can support depends on the width of the FAT entries. In the earliest FAT file systems (designed for use on floppies, and small hard drives). An 8-bit wide FAT entry would have been too small (256 * 512 = 128K bytes) to describe even the smallest floppy, but a 16-bit wide FAT entry would have been ludicrously large (8-16 Megabytes) ... so Microsoft compromised and adopted 12-bit wide FAT entries (two entries in three bytes). These were called FAT-12 file systems. As disks got larger, they created 2-byte wide (FAT-16) and 4-byte wide (FAT-32) file systems.

5.2 Next Block Pointers

A file's directory entry contains a pointer to the first cluster of that file. The File Allocation Table entry for that cluster tells us the cluster number **next** cluster in the file. When we finally get to the last cluster of the file, its FAT entry will contain a -1, indicating that there is no next block in the file.



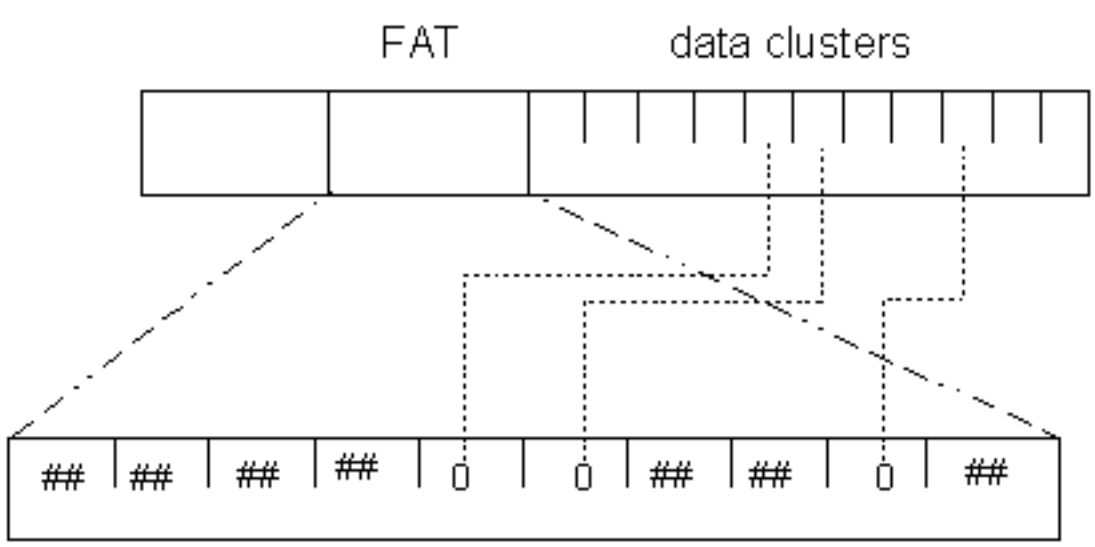
The "next block" organization of the FAT means that in order to figure out what physical cluster is the third logical block of ar file, must know the physical cluster number of the second logical block. This is not usually a problem, because almost all file access is sequential (reading the first block, and then the second, and then the third ...).

If we had to go to disk to re-read the FAT each time we needed to figure out the next block number, the file system would perform very poorly. Fortunately, the FAT is so small (e.g. 512 bytes per megabyte of file system) that the entire FAT can be kept in memory as long as a file system is in use. This means that successor block numbers can be looked up without the need to do any additional disk I/O. It is easy to imagine

5.3 Free Space

The notion of "next block" is only meaningful for clusters that are allocated to a file ... which leaves us free to use the FAT entries associated with free clusters as a free indication. Just as we reserved a value (-1) to mean **end of file** we can rerserve another value (0) to mean **this cluster is free**.

To find a free cluster, one has but to search the FAT for an entry with the value -2. If we want to find a free cluster near the clusters that are already allocated to the file, we can start our search with the FAT entry after the entry for the first cluster in the file.



Each FAT entry corresponds to one data cluster

A FAT entry of 0 means the corresponding data cluster is not allocated to any file.

5.4 Garbage Collection

Older versions of FAT file systems did not bother to free blocks when a file was deleted. Rather, they merely crossed out the first byte of the file name in the directory entry (with the reserved value 0xE5). This had the advantage of greatly reducing the amount of I/O associated with file deletion ... but it meant that DOS file systems regularly ran out of space.

When this happened, they would initiate garbage collection. Starting from the root directory, they would find every "valid" entry. They would follow the chain of next block pointers to determine which clusters were associated with each file, and recursively enumerate the contents of all sub-directories. After completing the enumeration of all allocated clusters, they inferred that any cluster not found in some file was free, and marked them as such in the File Allocation Table.

This "feature" was probably motivated by a combination of laziness and a desire for performance. It did, however, have an advantage. Since clusters were not freed when files were deleted, they could not be reallocated until after garbage colection was performed. This meant that it might be possible to recover the contents of deleted files for quite a while. The opportunity this created was large enough to enable Peter Norton to start a very successful company.

6. Descendents of the DOS file system

The DOS file system has evolved with time. Not only have wider (16- and 32-bit) FAT entries been used to support larger disks, but other features have been added. The last stand-alone DOS product was DOS 6.x. After this, all DOS support was under Windows, and along with the change to Windows came an enhanced version of the FAT file system called Virtual FAT (or simply VFAT).

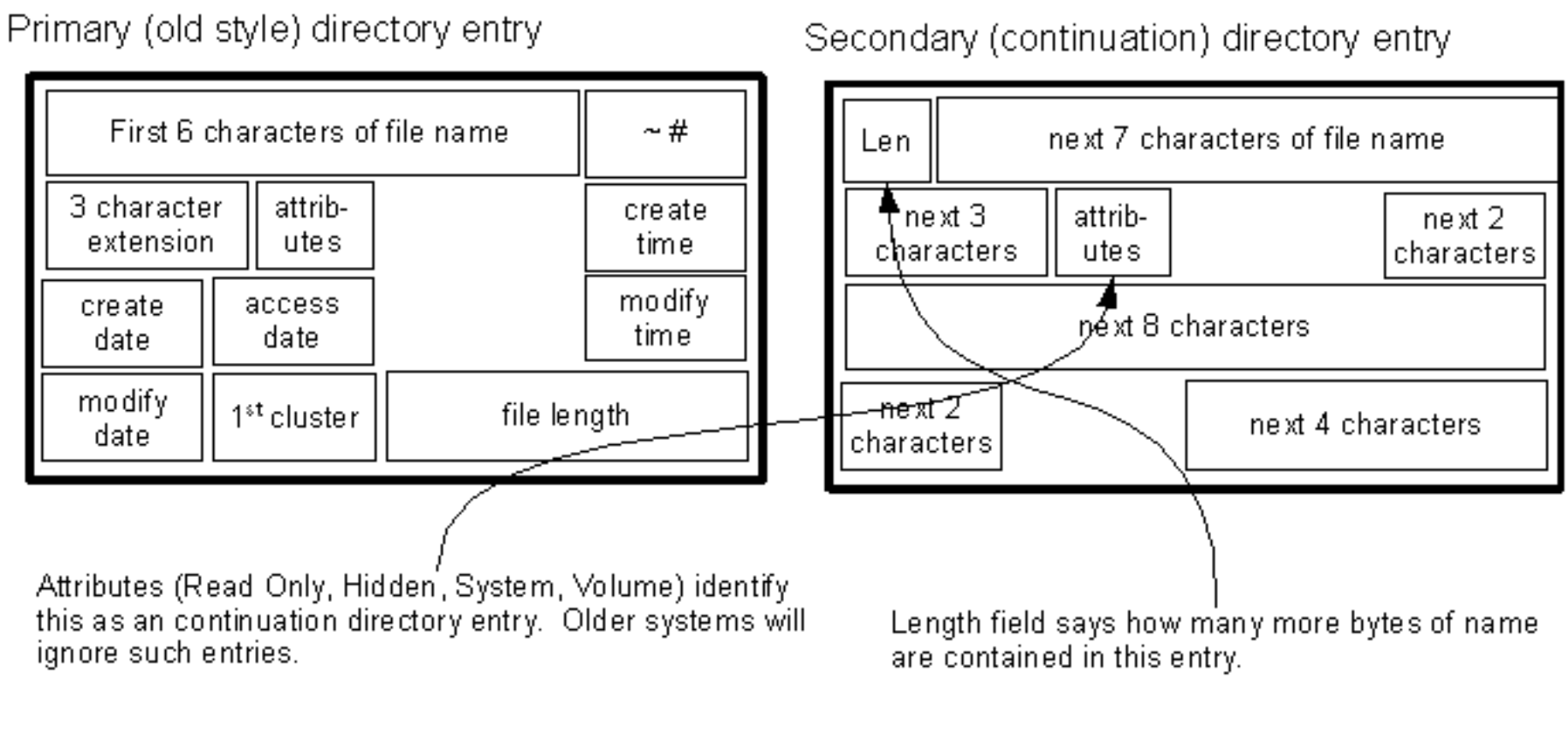
6.1 Long File Names

Most DOS and Windows systems were used for personal productivity, and their users didn't demand much in the way of file system features. Their biggest complaints were about the 8+3 file names. Windows users demanded longer and mixed-case file names.

The 32 byte directory entries didn't have enough room to hold longer names, and changing the format of DOS directories would break hundreds or even thousands of applications. It wasn't merely a matter of

importing files from old systems to new systems. DOS diskettes are commonly used to carry files between various systems ... which means that old systems still had to be able to read the new directories. They had to find a way to support longer file names without making the new files unreadable by older systems.

The solution they came up was to put the extended filenames in additional (auxiliary) directory entries. Each file would be described by an old-format directory entry, but supplementary directory entries (following the primary directory entry) could provide extenstions to the file name. To keep older systems from being confused by the new directory entries, they were tagged with a very unusual set of attributes (hidden, system, read-only, volume label). Older systems would ignore new entries, but would still be able to access new files under their 8+3 names in the old-style directory entries. New systems would recognize the new directory entries, and be able to access files by either the long or the short names.



The addition of long file names did create one problem for the old directory entries. What would you do if you had two file names that differred only after the 8th character (e.g. datafileread.c and datafilewrite.c)? If we just used the first 8 characters of the file name in the old directory entries (datafile), we would have two files with the same name, and this is illegal. For this reason, the the short file names are not merely the first eight characters of the long file names. Rather, the last two bytes of the short name were merely made unambiguous (e.g. "~1", "~2", etc).

6.2 Alternate/back-up FATs

The File Allocation Table is a very concise way of keeping track of all of the next-block pointers in the file system. If anything ever happened to the File Allocation Table, the results would be disastrous. The directory entries would tell us where the first blocks of all files were, but we would have no way of figuring out where the remainder of the data was.

Events that corrupt the File Allocation Table are extremely rare, but the consequences of such an incident are catastrophic. To protect users from such eventualities, MicroSoft added support for alternate FATs. Periodically, the primary FAT would be copied to one of the pre-reserved alternat FAT locations. Then if something bad happened to the primary FAT, it would still be possible to read most files (files created before the copy) by using the back-up FAT. This is an imperfect solution, as losing new files is bad ... but losing all files is worse.

6.3 ISO 9660

When CDs were proposed for digital storage, everyone recognized the importance of a single standard file system format. Dueling formats would raise the cost of producing new products ... and this would be a lose for everyone. To respond to this need, the International Standards Organization chartered a sub-committee to propose such a standard.

The failings of the DOS file system were widely known by this time, but, as the most widely used file system on the planet, the committee members could not ignore it. Upon examination, it became clear that the most ideomatic features of the DOS file system (the File Allocation Table) were irrelevent to a CDROM file system (which is written only once):

- We don't need to keep track of the free space on a CD ROM. We write each file contiguously, and the next file goes immidiately after the last one.
- Because files can be written contiguously, we don't need any "next block" pointers. All we need to know about a file is where its first block resides.

It was decided that ISO 9660 file systems would (like DOS file systems) have tree structured directory hierarchies, and that (like DOS) each directory entry would describe a single file (rather than having some auxiliary data structure like and I-node to do this). 9660 directory entries, like DOS directory entries, contain:

- file name (within the current directory)
- file type (e.g. file or directory)
- location of the file's first block
- number of bytes contained in the file
- time and date of creation

They did, however, learn from DOS's mistakes:

- Realizing that new information would be added to directory entries over time, they made them variable length. Each directory entry begins with a length field (giving the number of bytes in this directory entry, and thus the number of bytes until the next directory entry).
- Recognizing the need to support long file names, they also made the file name field in each entry a variable length field.
- Recognizing that, over time, people would want to associate a wide range of attributes with files, they also created a variable length extened attributes section after the file name. Much of this section has been left unused, but they defined several new attributes for files:
    - file owner
    - owning group
    - permissions
    - creation, modification, effective, and expiration times
    - record format, attributes, and length information

But, even though 9660 directory entries include much more information than DOS directory entries, it remains that 9660 volumes resemble DOS file systems much more than they resemble any other file system format. And so, the humble DOS file system is reborn in a new generation of products.

7. Summary

DOS file systems are very simple, They don't support multiple links to a file, or symbolic links, or even multi-user access control. They are also and very economical in terms of the space they take up. Free block lists, and file block pointers are combined into a single (quite compact) File Allocation Table. File descriptors are incorporated into the directory entries. And for all of these limitations, they are probably the most widely used file system format on the planet. Despite their primitiveness, DOS file systems were used as the basis for much newer CD ROM file system designs.

What can we infer from this? That most users don't need alot of fancy features, and that the DOS file system

(primitive as it may be) covers their needs pretty well.

It is also noteworthy that when Microsoft was finally forced to change the file system format to get past the 8.3 upper case file name limitations, they chose to do so with a klugy (but upwards compatible) solution using additional directory entries per file. The fact that they chose such an implementation clearly illustrates the importance of maintaining media interchangability with older systems. This too is a problem that all (successful) file systems will eventually face.

8. References

DOS file system information

- PC Guide's Overview of DOS FAT file systems. (this is a pointer to the long filename article ... but the entire library is nothing short of excellent).
- Free BSD sources, PCFS implementation, BIOS Parameter block format, and (Open Solaris) FDISK table format.
- Free BSD sources, PCFS implementation, Directory Entry format

9660 file system information

- WIkipedia Introduction to ISO 9660 file systems
- Free BSD sources, ISOFS implementation, ISO 9660 data structures.