File systems

File System

- Function: main permanent data storage
  Speed bottleneck!
- Capacity not a problem nowadays: 2 TB disks even for PC.
  But backup becoming a problem.
- Logical view (view of programmer): tree structure of files together with read/write operation and creation of directories
- Physical view: sequence of blocks, which can be read and written. OS has to map logical view to physical view, must impose tree structure and assign blocks for each file

Two main possibilities to realize filesystem:

- Linked list: Each block contains pointer to next
  ⇒ Problem: random access (`seek()`) costly: have to go through whole file until desired position.
- Indexed allocation: Store pointers in one location: so-called index block (similar to page table). To cope with vastly differing file sizes, may introduce indirect index blocks.

Index blocks are called `inodes` in Unix.
Inodes store additional information about the file (eg size, permissions)

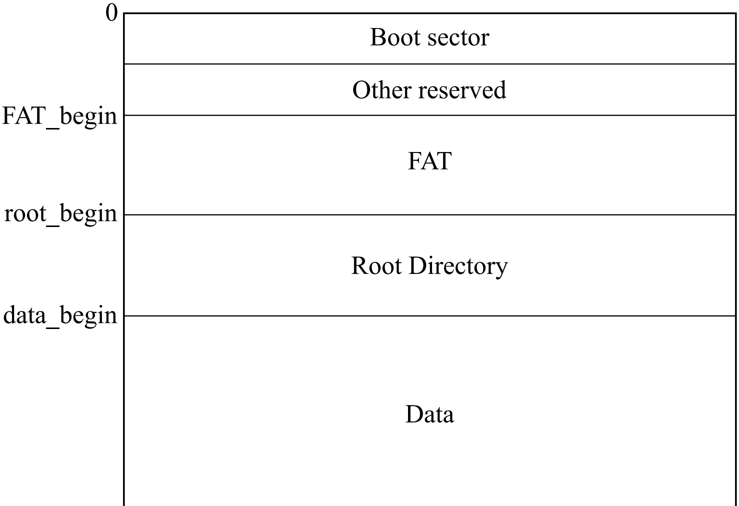
Worked Example

Worked example – based on
`http://www.tavi.co.uk/phobos/fat.html`

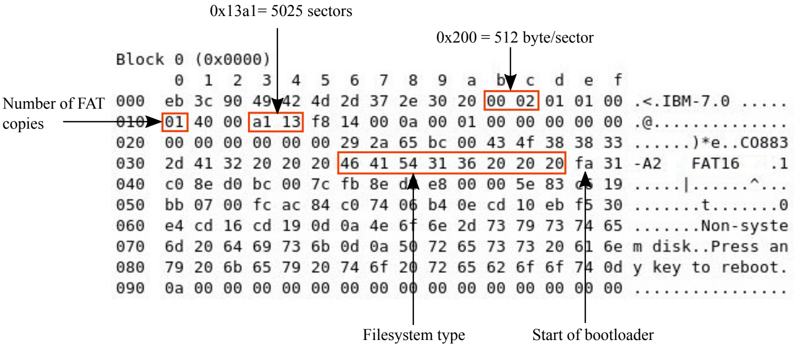
Example: FAT

- F(ile) A(llocation) T(able) – dates back to 70s.
- Useful for explaining filesystem concepts, modern filesystems are more complicated
- Variants FAT12. FAT16, FAT32 define number of bits per FAT entry – we focus on FAT16
- Sector = disk unit (e.g. 512 byte), aka block
- Cluster = multiple sectors (factor 1, 2, 4, . . . , 128) (*here*: assume cluster = 1 sector)
- Uses linked list ("cluster chain") to group clusters

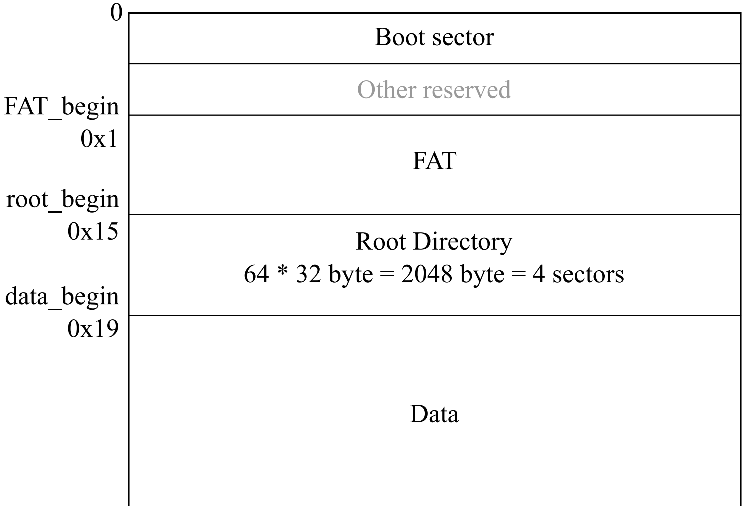
Example: FAT16 Structure



Example: FAT16 Bootsector



Example: FAT16 with Offsets



Example: File Allocation Table

FAT_begin

FFF0	FFFF	0003 $_2$	0004 $_3$
0006 $_4$	0000 $_5$	FFFF $_6$	0008 $_7$
0009 $_8$	FFFF $_9$	0000 $_A$	0000 $_B$
000F $_C$	0010 $_D$	0000 $_E$	000D $_F$
FFFF $_{10}$	0000 $_{12}$	0000 $_{13}$	0000 $_{14}$

$2 \to 3 \to 4 \to 6$ $\quad$ $7 \to 8 \to 9$ $\quad$ $C \to F \to D \to 10$

Example: File in Root Directory

```
Block 21 (0x0015)
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
000  43 4f 38 38 33 2d 41 32 20 20 20 28 00 00 00 00 C0883-A2   (....
010  00 00 00 00 00 00 91 9e 65 39 00 00 00 00 00 00 ........e9......
020  46 4f 4f 42 41 52 20 20 54 58 54 21 00 a3 91 9e FOOBAR  TXT!....
030  65 39 65 39 00 00 91 9e 65 39 c6 10 1a 00 00 00 e9e9....e9......
040  4e 45 54 57 4f 52 4b 20 56 52 53 20 00 b6 91 9e NETWORK VRS ....
050  65 39 65 39 00 00 91 9e 65 39 4e 0f 92 06 00 00 e9e9....e9N.....
060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

filename & extension

sector 0xF4E

length 0x692 = 1682 byte

Example: File in FAT

Block 16 (0x0010)

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
090	00	00	00	00	00	00	00	00	00	00	00	00	4f	0f	50	0f
0a0	51	0f	ff	ff	00	00	00	00	00	00	00	00	00	00	00	00
0b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

4 block cluster chain (2048 byte)

Example: FAT Limits

- Max. volume size: 2 GB ($2^{16} \cdot 32\,kB$)
- Max. file size: 2 GB
- Max. number of files: 65,460 (32 kB clusters)
- Max. filename length: $8 + 3$
- FAT32 / exFAT have higher limits
- Newer filesystems (NTFS, ext4) also overcome these limits, using other data structures (e.g. B-tree for dir structure, bitmap for allocation)

Further Aspects

Further aspects of filesystems

Caching

Disk blocks used for storing directories or recently used files cached in main memory

Blocks periodically written to disk

$\Rightarrow$ Big effiency gain

Inconsistency arises when system crashes

Reason why computers must be shutdown properly

Journaling File Systems

To minimise data loss at system crashes, ideas from databases are used:

- Define Transaction points: Points where cache is written to disk

  $\Rightarrow$ Have consistent state
- Keep log-file for each write-operation

  Log enough information to unravel any changes done after latest transaction point

Disk Access

Disk access contains three parts:

- Seek: head moves to appropriate track
- Latency: correct block is under head
- Transfer: data transfer

HDDs: Time necessary for seek and latency dwarfs transfer time
$\Rightarrow$ Distribution of data and scheduling algorithms have vital impact on performance for HDDs, less so for SSDs

Disk Scheduling Algorithms

Standard algorithms apply, adapted to the special situation:
1.) FCFS: easiest to implement, but: may require lots of head movements

2.) Shortest Seek Time First: Select job with minimal head movement
Problems:
- may cause starvation
- Tracks in the middle of disk preferred

Algorithm does not minimise number of head movements

3.) SCAN-scheduling: Head continuously scans the disk from en to end (lift strategy)
$\Rightarrow$ solves the fairness and starvation problem of SSTF

Improvement: LOOK-scheduling:
head only moved as far as last request (lift strategy).

Particular tasks may require different disk access algorithms
Example : Swap space management
Speed absolutely crucial $\Rightarrow$ different treatment:

- Swap space stored on separate partition
- Indirect access methods not used

- Special algorithms used for access of blocks
  Optimised for speed at the cost of space (eg increased internal fragmentation)

Linux Implementation

Interoperability with Windows and Mac requires support of different file systems (eg vfat)

$\Rightarrow$ Linux implements common interface for all filesystems

Common interface called virtual file system

virtual file system maintains

- inodes for files and directories
- caches, in particular for directories
- superblocks for file systems

All system calls (eg `open`, `read`, `write` and `close`) first go to virtual file system

If necessary, virtual file system selects appropriate operation from real file system

Disk Scheduler

Kernel makes it possible to have different schedulers for different file systems

Default scheduler (Completely Fair Queuing) based on lift strategy have in addition separate queue for disk requests for each process queues served in Round-Robin fashion

Have in addition No-op scheduler: implements FIFO

Suitable for SSD's where access time for all sectors is equal