Operating Systems

File Systems

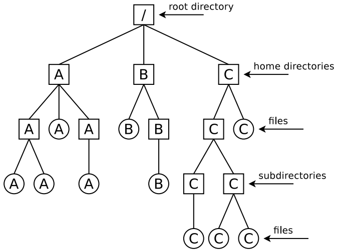
Prof. Dr. Oliver Hahm

Frankfurt University of Applied Sciences
Faculty 2: Computer Science and Engineering
oliver.hahm@fb2.fra-uas.de
https://teaching.dahahm.de

January 31, 2023

File Systems...

- organize the storage of files on data storage devices[1]
- manage file names and attributes (metadata) of files
- form a namespace, i.e., a hierarchy of directories and files
  - a namespace may comprise multiple file systems



- Absolute path names: Describe the complete path from the root to the file
- Relative path names: All paths, which do not begin with the root

- are a layer of the operating system
  - Processes and users access files via their abstract file names and not via their memory addresses

[1] Files are (semantically related) sequences of bytes

Technical Principles of File Systems

- File systems address clusters and not blocks of the storage device
  - Each file occupies an integer number of clusters
  - In literature, clusters are often called zones or blocks
- The size of the clusters is essential for the efficiency of the file system
  - The smaller the clusters are...
    - the bigger the overhead for large files
    - the bigger the usable capacity due to less internal fragmentation
  - The bigger the clusters are...
    - the smaller the overhead for large files
    - the smaller the usable capacity due to more internal fragmentation

The bigger the clusters, the more memory is lost due to internal fragmentation

- File size: 1 kB. Cluster size: 2 kB $\Longrightarrow$ 1 kB gets lost
- File size: 1 kB. Cluster size: 64 kB $\Longrightarrow$ 63 kB get lost!

- The cluster size can be specified while creating the file system

Agenda

- Block Addressing
  - Minix
  - ext2/3/4 File System
  - Analyze File Systems
- File Allocation Tables
  - FAT Evolution
  - Analyze FAT File Systems
- Extents
  - Extent-based Addressing
  - NTFS
- Journal
- Copy-on-write

Agenda

- Block Addressing
  - Minix
  - ext2/3/4 File System
  - Analyze File Systems
- File Allocation Tables
  - FAT Evolution
  - Analyze FAT File Systems
- Extents
  - Extent-based Addressing
  - NTFS
- Journal
- Copy-on-write

Basic Terminology of Unix File Systems

In Unix: Cluster size $\leq$ size of memory pages (page size)

- The page size depends on the architecture
- x86 = 4 kB, Alpha and UltraSPARC = 8 kB, Apple Silicon = 16 kB, IA-64 = 4/8/16/64 kB

- Unix file systems are based on so called inodes (*index nodes*)
- The creation of a file causes the creation of an *inode*
  - It stores a file's metadata, except the file name
    - Metadata contain, e.g., size, (group) owner, permissions, and date
  - Each inode has a unique inode number inside the file system
  - The inode contains references to the file's clusters
  - More than one file name may refer to the same inode (hardlink)
- A directory is a file, too (see slide 19)
  - Content: File name and inode number for each file in the directory
- The traditional working method of Unix file systems: Block addressing

Attention!

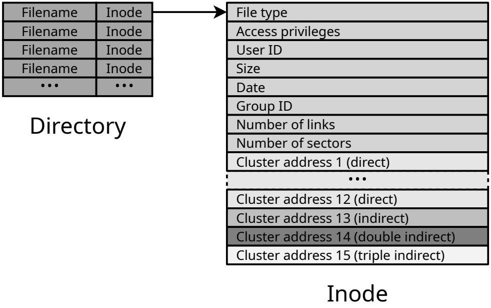
A block in Unix' terminology is a cluster on a hard disk

File Deletion

We have seen what happens on file creation.
But what about file deletion?

Block Addressing

- Each inode directly stores the numbers of up to 12 clusters



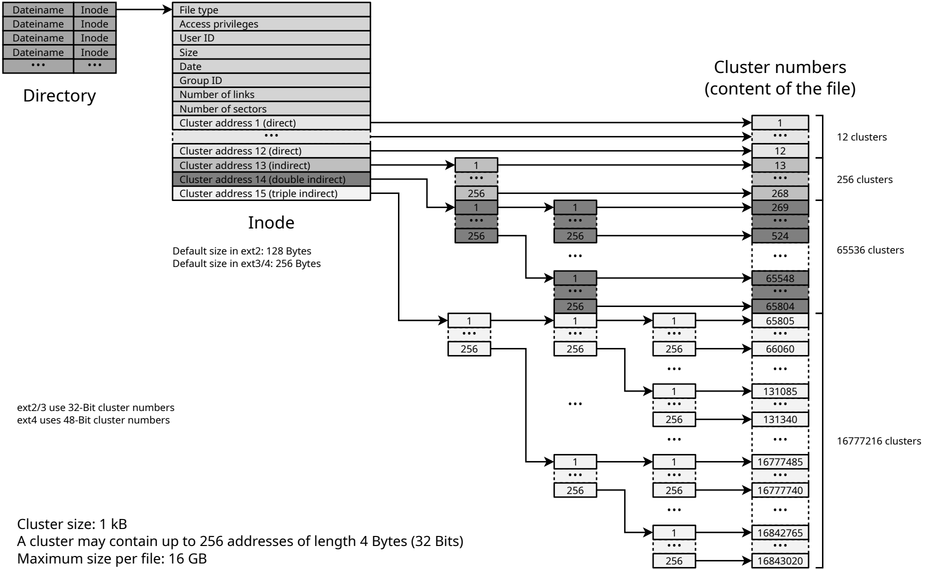
- If a file requires more clusters, these clusters are indirectly addressed
- Minix, ext2/3/4, ReiserFS and Reiser4 implement block addressing

Good explanation

`http://lwn.net/Articles/187321/`

- Scenario: No more files can be created in the file system, despite the fact that sufficient capacity is available
- Possible explanation: No more inodes are available
- The command `df -i` shows the number of existing inodes and how many are still available

Direct and indirect Addressing



ext2/3 use 32-Bit cluster numbers
ext4 uses 48-Bit cluster numbers

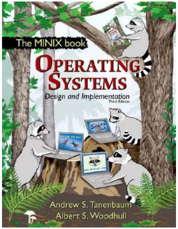
Cluster size: 1 kB
A cluster may contain up to 256 addresses of length 4 Bytes (32 Bits)
Maximum size per file: 16 GB

Agenda

Minix

The Minix operating system http://www.minix3.org

- Unix-like operating system
- Developed since 1987 by Andrew S. Tanenbaum for education purposes
  https://www.youtube.com/watch?v=bx3KuE7UjGA
- Latest revision is 3.3.0 is from 2014
- Intel chipsets post-2015 run MINIX 3 internally as the software component of the Intel Management Engine
  https://www.zdnet.com/article/minix-intels-hidden-in-chip-operating-system/
  https://linuxnews.de/2017/11/minix-in-der-intel-management-engine/
  https://itsfoss.com/fact-intel-minix-case/



- The Minix file system was the default file system for Linux until 1992
- The Minix file system causes low overhead
- Storage is represented as a linear chain of equal-sized blocks (1–8 kB)
- A Minix file system contains just 6 areas

Minix File System Structure (1/2)

Area 1	Area 2	Area 3	Area 4	Area 5	Area 6
Boot block (1 cluster)	Super block (1 cluster)	Inodes bitmap (1 cluster)	Cluster bitmap (1 cluster)	Inodes (15 clusters)	Data (remaining clusters)

- Boot block. Contains the boot loader that starts the operating system
- Super block. Contains information about the file system, e.g.,
  - reference to the root directory
  - number of inodes and clusters
- Inodes bitmap. Contains a list of all inodes with the information, whether the inode is occupied (value: 1) or free (value: 0)
- Clusters bitmap. Contains a list of all clusters with the information, whether the cluster is occupied (value: 1) or free (value: 0)

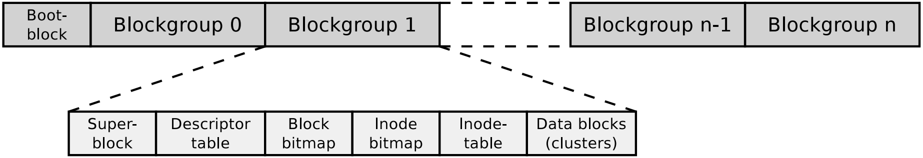
Minix File System Structure (2/2)

Area 1	Area 2	Area 3	Area 4	Area 5	Area 6
Boot block (1 cluster)	Super block (1 cluster)	Inodes bitmap (1 cluster)	Cluster bitmap (1 cluster)	Inodes (15 clusters)	Data (remaining clusters)

- Inodes. Contains the inodes with the metadata
  - Every file and every directory is represented by at least a single inode, which contains the metadata, like . . .
    - file type
    - UID/GID
    - access privileges
    - size
- Data. Contains the contents of the files and directories
  - This is the biggest part in the file system

Agenda

ext2/3/4 File System Structure



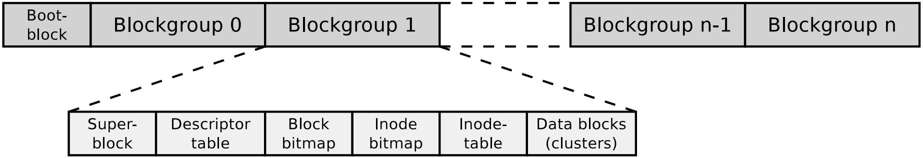
- The clusters of the file system are combined to block groups of the same size
  - The information about the metadata and free clusters of each block group are maintained in the respective block group

Maximum size of a block group: 8x cluster size in bytes

Example: If the cluster size is 4096 Bytes, each block group can contain up to 32768 clusters.
$\Longrightarrow$ The maximum block size is 32768 clusters $\times$ 4096 Bytes cluster size = 134,217,728 Bytes = 131,072 kB = 128 MB

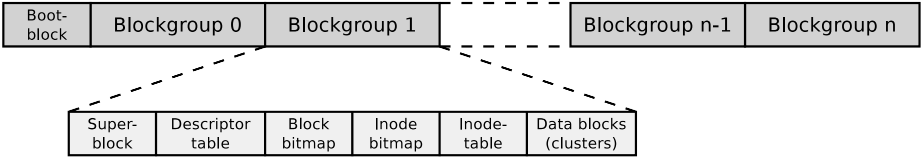
- Benefit of block groups (when using HDDs): inodes (metadata) are physically located closely to the referenced clusters

ext2/3/4 Block Group Structure (1/2)



- The first cluster of the file system contains the boot block (size: 1 kB)
  - It contains the boot manager, which starts the operating system
- Each block group contains a copy of the super block
  - This improves the data security
- The descriptor table contains among others:
  - The cluster numbers of the block bitmap and inode bitmap
  - The number of free clusters and inodes in the block group

ext2/3/4 Block Group Structure (2/2)



- Block bitmap and inode bitmap each have the size of a single cluster
  - They contain the information, which clusters and inodes in the block group are occupied
- The inode table contains the inodes of the block group
- The remaining clusters of the block group can be used for the data

Agenda

- Block Addressing
  - Minix
  - ext2/3/4 File System
  - Analyze File Systems
- File Allocation Tables
  - FAT Evolution
  - Analyze FAT File Systems
- Extents
  - Extent-based Addressing
  - NTFS
- Journal
- Copy-on-write

Analyze File Systems (1/3)

We already know. . .

Directories are files that store file names and inode numbers

Let's take a look inside. . .

With the specification of the file system and some tools, the individual fields of the directory records can be examined. For example the record of README.txt

```
# lsblk | grep sda1
sda              8:0    1  29,3G  0 disk
sda1           8:1    1  29,3G  0 part
# mkfs.ext4 /dev/sda1
# mkdir /mnt/test
# mount -t ext4 /dev/sda1 /mnt/test/
# df -h | grep test
/dev/sda1          29G     45M   28G    1% /mnt/test
# ls -l /mnt/test
insgesamt 16
drwx------ 2 root root 16384 Sep 14 09:38 lost+found
# mkdir /mnt/test/testfolder
# echo "Betriebssysteme" > /mnt/test/testfolder/README.txt
# echo "OpSys" > /mnt/test/testfolder/file2.txt
# echo "12345" > /mnt/test/testfolder/anotherfile.dat
# touch /mnt/test/testfolder/empty_file
# ls -lai /mnt/test/testfolder/
insgesamt 20
392449 drwxr-xr-x 2 root root 4096 Sep 14 09:59 .
2 drwxr-xr-x 4 root root 4096 Sep 14 09:46 ..
392452 -rw-r--r-- 1 root root    6 Sep 14 09:58 anotherfile.dat
392453 -rw-r--r-- 1 root root    0 Sep 14 09:59 empty_file
392451 -rw-r--r-- 1 root root    6 Sep 14 09:47 file2.txt
392450 -rw-r--r-- 1 root root   16 Sep 14 09:47 README.txt
```

Analyze File Systems (2/3)

We analyze the contents of `testfolder.out` with command line tools like `hexdump` or `od` (octal dump). Alternatively, it is possible to use a graphical tool such as `wxHexEditor` (see slide 36).

```
# debugfs /dev/sda1
debugfs 1.44.5 (15-Dec-2018)
debugfs:  imap <392449>
Inode 392449 is part of block group 48
located at block 1572896, offset 0x0000
debugfs:  dump testfolder testfolder.out
debugfs:  quit
# ls -l testfolder.out
-rw-r--r-- 1 root root 4096 Sep 14 10:00 testfolder.out
# hexdump -C testfolder.out
00000000  01 fd 05 00 0c 00 01 02  2e 00 00 00 02 00 00 00  |................|
00000010  0c 00 02 02 2e 2e 00 00  02 fd 05 00 14 00 0c 01  |................|
00000020  6c 69 65 73 6d 69 63 68  2e 74 78 74 03 fd 05 00  |README.txt......|
00000030  14 00 09 01 66 69 6c 65  32 2e 74 78 74 00 00 00  |....file2.txt...|
00000040  04 fd 05 00 18 00 0f 01  61 6e 6f 74 68 65 72 66  |........anotherf|
00000050  69 6c 65 2e 64 61 74 00  05 fd 05 00 9c 0f 0a 01  |ile.dat.........|
00000060  65 6d 70 74 79 5f 66 69  6c 65 00 00 00 00 00 00  |empty_file......|
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
...
```

A hex editor visualizes data in several ways

- 1st column: Number of previous bytes $\Longrightarrow$ offset or position indicator (address) inside the file in hexadecimal representation
- 2nd column: Bytes of the line in hexadecimal representation
- 3rd column: Bytes of the line in ASCII representation

Some fundamentals...

- Hexadecimal system $\Longrightarrow$ base 16
- 1 hexadecimal digit represents 4 bits
- 2 hexadecimal digits represent 1 byte

Analyze File Systems (3/3)

https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout

Offset	Size	Name	Description
0x0	4 Bytes	inode	Number of the inode that this directory record points to
0x4	2 Bytes	record length	Directory record length
0x6	1 Byte	name length	Length of the file name
0x7	1 Byte	file type	0x0 = unknown, 0x1 = regular file, 0x2 = directory, 0x3 = character-device special file, 0x4 = block-device special file, 0x5 = FIFO (named pipe), 0x6 = socket, 0x7 = symbolic link
0x8		character string	File name

Information about Big-Endian vs. Little-Endian Byte Order

x86 processors use the little-endian byte order. If a data field is several bytes long, the least significant byte (LSB) is in the first position, i.e. on the lowest memory address. The more significant bytes are on the following memory addresses

```
00000010  -- -- -- -- -- -- -- --  02 fd 05 00 14 00 0c 01  |................|
00000020  6c 69 65 73 6d 69 63 68  2e 74 78 74 -- -- -- --  |README.txt......|
```

Inode number: 02 fd 05 00 $\Longrightarrow$ 00 05 fd 02

```
$ printf "%d\n" 0x0005fd02
392450
```

Directory record length: 14 00 $\Longrightarrow$ 00 14 $\Longrightarrow$ 20 Bytes

```
$ printf "%d\n" 0x0014
20
```

File name length: 0c $\Longrightarrow$ 12 Bytes File type: 01 $\Longrightarrow$ regular file

Agenda

- Block Addressing
  - Minix
  - ext2/3/4 File System
  - Analyze File Systems
- File Allocation Tables
  - FAT Evolution
  - Analyze FAT File Systems
- Extents
  - Extent-based Addressing
  - NTFS
- Journal
- Copy-on-write

File Allocation Table (FAT)

The FAT file system was released in 1980 with QDOS, which was later renamed to MS-DOS

QDOS = Quick and Dirty Operating System

- The FAT file system is based on the data structure of the same name
- The FAT (File Allocation Table) is a table of fixed size
- The FAT contains an entry for each cluster in the file system, containing information whether the cluster is…
  - …free
  - …the storage medium is damaged at this point
  - …occupied by a file
    - In this case it stores the address of the next cluster, which belongs to the file or it is the last cluster of the file
- The clusters of a file are a linked list (cluster chain)
  $\Longrightarrow$ see slides 27 und 28

FAT File System Structure (1/2)

Area 1	Area 2	Area 3	Area 4	Area 5	Area 6
Boot block	Reserved blocks	FAT1	FAT2	Root directory	Data region

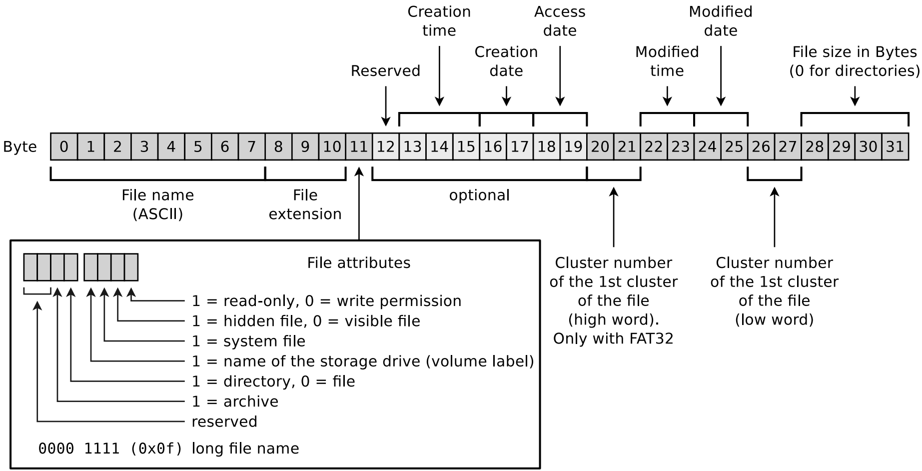
- The boot sector contains executable x86 machine code, which starts the operating system, and information about the file system:
  - Block size of the storage device (512, 1024, 2048 or 4096 Bytes)
  - Number of blocks per cluster
  - Number of blocks (sectors) on the storage device
  - Description (name) of the storage device
  - Description of the FAT version
- Between the boot block and the first FAT, optional reserved blocks may exist, e.g., for the boot manager
  - These clusters can not be used by the file system

FAT File System Structure (2/2)

Area 1	Area 2	Area 3	Area 4	Area 5	Area 6
Boot block	Reserved blocks	FAT1	FAT2	Root directory	Data region

- The File Allocation Table (FAT) stores a record for each cluster in the file system, which informs, whether the cluster is occupied or free
  - The FAT's consistency is essential for the functionality of the file system
    - Therefore, usually a copy of the FAT exists, in order to have a complete FAT as backup in case of a data loss
- In the root directory, every file and every directory is represented by an entry:
  - With FAT12 and FAT16, the root directory is located directly behind the FAT and has a fixed size
    - The maximum number of directory entries is therefore limited
  - With FAT32, the root directory can reside at any position in the data region and has a variable size
- The last region contains the actual data

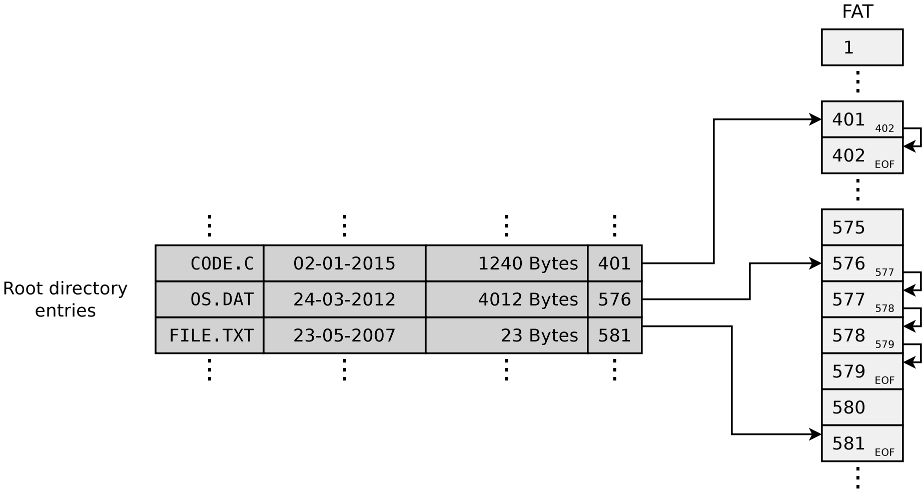
Structure of Root Directory Entries



Why is 4 GB the maximum file size on FAT32?

Only 4 Bytes are available for specifying the file size.

Root Directory and FAT

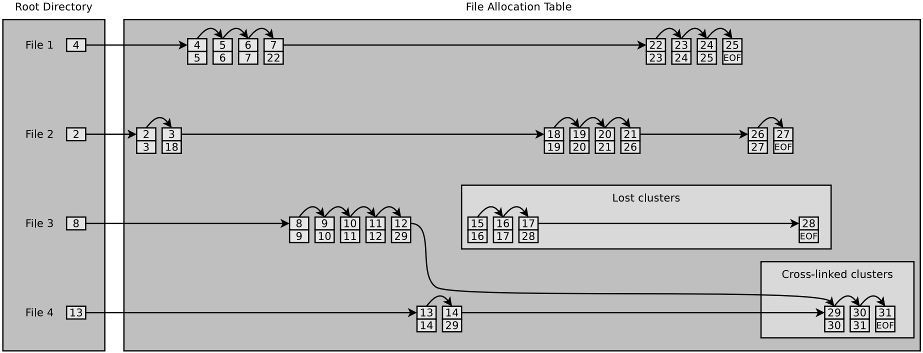


The topic FAT is well explained by...

- Betriebssysteme, *Carsten Vogt*, 1st edition, Spektrum Akademischer Verlag (2001), P. 178-179

Risk of File System Inconsistencies

- Typical problems of file systems based on a FAT:
  - lost clusters
  - cross-linked clusters



Agenda

FAT12

Released in 1980 with the first QDOS release

- Length of the cluster numbers: 12 bits
  - Up to $2^{12} = 4096$ clusters can be addressed
- Cluster size: 512 Bytes to 4 kB
- Supports storage media (partitions) up to 16 MB

$2^{12} * 4\,\text{kB cluster size} = 16384\,\text{kB} = 16\,\text{MB maximum file system size}$

- File names are supported only in 8.3 format
  - Up to 8 characters can be used to represent the file name and 3 characters for the file name extension

FAT16

- Released in 1983 because it was foreseeable that an address space of 16 MB is insufficient
- Up to $2^{16} = 65524$ clusters can be addressed
  - 12 clusters are reserved
- Cluster size: 512 Bytes to 256 kB
- File names are supported only in 8.3 format

Sources:
http://support.microsoft.com/kb/140365/de
http://secrets.mysfyts.com/index.asp?Page=Fat
http://web.allensmith.net/Storage/HDDlimit/FAT16.htm

Partition size	Cluster size
up to 31 MB	512 Bytes
32 MB - 63 MB	1 kB
64 MB - 127 MB	2 kB
128 MB - 255 MB	4 kB
256 MB - 511 MB	8 kB
512 MB - 1 GB	16 kB
1 GB - 2 GB	32 kB
2 GB - 4 GB	64 kB
4 GB - 8 GB	128 kB
8 GB - 16 GB	256 kB

The table contains default cluster sizes of Windows 2000/XP/Vista/7/8/10. The cluster size can be manually specified during the file system creation

Some operating systems (e.g., MS-DOS and Windows 95/98/Me) do not support clusters > 32 kB

Some operating systems (e.g., Windows 2000/XP/7/8/10) do not support clusters > 64 kB

FAT32

- Released in 1997 because of the rising HDD capacities and because clusters $> 32\,\text{kB}$ waste a lot of storage
- Size of the cluster numbers records in the FAT: 32 Bits
  - 4 Bits are reserved
  - Therefore, only $2^{28} = 268,435,456$ clusters can be addressed
- Cluster size: 512 Bytes to 32 kB
- Maximum file size: 4 GB
  - Reason: Only 4 Bytes are available for indicating the file size
- Main field of application today: Mobile storage media $> 2\,\text{GB}$

Partition size	Cluster size
up to 63 MB	512 Bytes
64 MB - 127 MB	1 kB
128 MB - 255 MB	2 kB
256 MB - 511 MB	4 kB
512 MB - 1 GB	4 kB
1 GB - 2 GB	4 kB
2 GB - 4 GB	4 kB
4 GB - 8 GB	4 kB
8 GB - 16 GB	8 kB
16 GB - 32 GB	16 kB
32 GB - 2 TB	32 kB

The table contains default cluster sizes of Windows 2000/XP/Vista/7/8/10. The cluster size can be manually specified during the file system creation

Sources: http://support.microsoft.com/kb/140365/de

Longer File Names by using VFAT

- VFAT (Virtual File Allocation Table) was released in 1997
  - Extension for FAT12/16/32 to support long filenames
- Because of VFAT, Windows supported for the first time...
  - file names that do not comply with the 8.3 format
  - file names up to a length of 255 characters
- Uses the Unicode character encoding

Long file names – Long File Name Support (LFN)

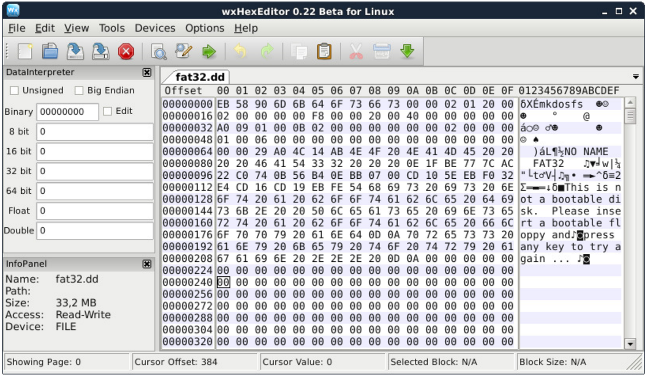
- VFAT is a good example for implementing a new feature without losing backward compatibility
- Long file names (up to 255 characters) are distributed to max. 20 pseudo-directory records
- File systems without Long File Name support ignore the pseudo-directory records and show only the shortened name
- For a VFAT records in the FAT, the first 4 bit of the file attributes field have value 1 (see slide 27)
- Special attribute: Upper/lower case is displayed, but ignored

Agenda

Analyze FAT File Systems (1/3)

```
# dd if=/dev/zero of=./fat32.dd bs=1024000 count=34
34+0 Datensätze ein
34+0 Datensätze aus
34816000 Bytes (35 MB) kopiert, 0,0213804 s, 1,6 GB/s
# mkfs.vfat -F 32 fat32.dd
mkfs.vfat 3.0.16 (01 Mar 2013)
# mkdir /mnt/fat32
# mount -o loop -t vfat fat32.dd /mnt/fat32/
# mount | grep fat32
/tmp/fat32.dd on /mnt/fat32 type vfat (rw,relatime,fmask=0022,dmask=0022,codepage=437,iocharset=utf8,
  shortname=mixed,errors=remount-ro)
# df -h | grep fat32
/dev/loop0       33M     512   33M    1% /mnt/fat32
# ls -l /mnt/fat32
insgesamt 0
# echo "Betriebssysteme" > /mnt/fat32/liesmich.txt
# cat /mnt/fat32/liesmich.txt
Betriebssysteme
# ls -l /mnt/fat32/liesmich.txt
-rwxr-xr-x 1 root root 16 Feb 28 10:45 /mnt/fat32/liesmich.txt
# umount /mnt/fat32/
# mount | grep fat32
# df -h | grep fat32
# wxHexEditor fat32.dd
```

Analyze FAT File Systems (2/3)



A hex editor visualizes data in several ways

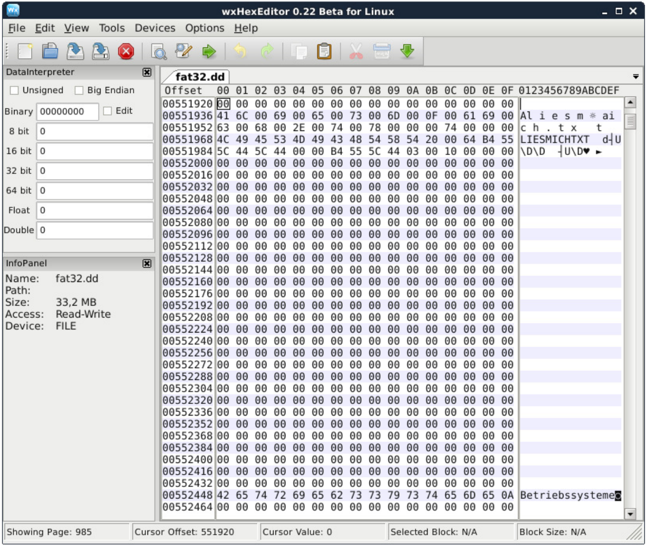
- 1st column: Number of previous bytes $\Longrightarrow$ offset
- 2nd column: Bytes of the line in hexadecimal representation
- 3rd column: Bytes of the line in ASCII representation

Some fundamentals...

- Hexadecimal system $\Longrightarrow$ base 16
- 1 hexadecimal digit represents 4 bits
- 2 hexadecimal digits represent 1 byte

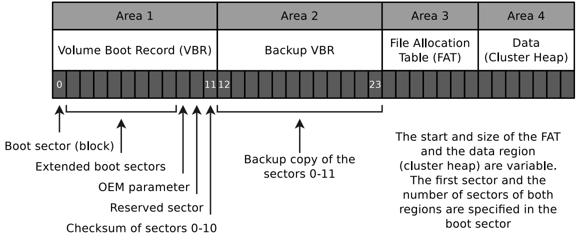
http://dorumugs.blogspot.de/2013/01/file-system-geography-fat32.html
http://www.win.tue.nl/~aeb/linux/fs/fat/fat-1.html

Analyze FAT File Systems (3/3)



exFAT

- Released in 2006 (usage is royalty-free since 2019)
- Up to $2^{32} = 4,294,967,296$ clusters can be addressed
- Cluster size: 512 Bytes to 64 MB
- Maximum file size: 16 EB (2^{64} Bytes)
- Main field of application: mobile flash memory (> 32 GB)
  - Fewer write operations than file systems with a journal (e.g., NTFS $\Longrightarrow$ slide 46)



In contrast to the other FAT file system versions, the root directory does not have a fixed position. It is located within the data area and usually does not reside there in one piece, but is fragmented.

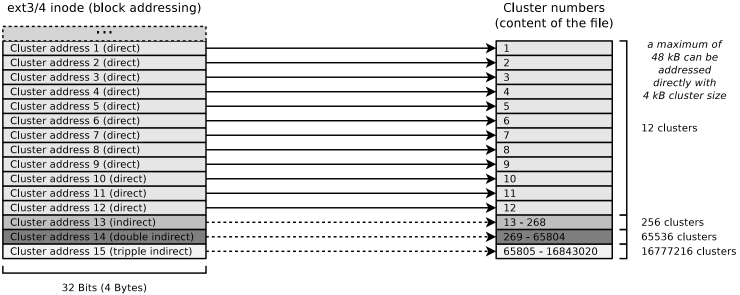
Partition size	Cluster size
up to 256 MB	4 kB
256 MB - 32 GB	32 kB
32 GB - 256 TB	128 kB

The table contains default cluster sizes of Windows 2000/XP/Vista/8/10. The cluster size can be manually specified during the file system creation
https://support.microsoft.com/de-de/kb/140365

Agenda

Problem: Metadata Overhead

- Every inode at block addressing addresses a certain number of cluster numbers directly
- If a file requires more clusters, they are indirectly addressed



- This addressing scheme causes rising overhead with rising file size
- Solution: Extents

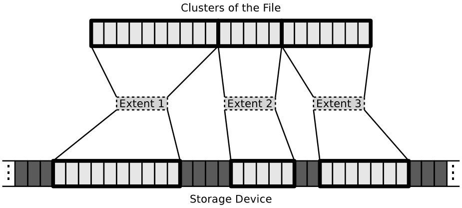
Agenda

- Block Addressing
  - Minix
  - ext2/3/4 File System
  - Analyze File Systems
- File Allocation Tables
  - FAT Evolution
  - Analyze FAT File Systems
- Extents
  - Extent-based Addressing
  - NTFS
- Journal
- Copy-on-write

Extent-based Addressing

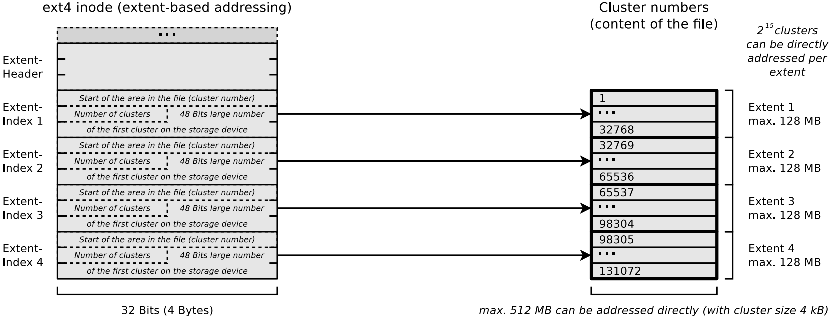
- Inodes do not address individual clusters, but instead create large areas of files to areas of contiguous blocks (extents) on the storage device
- Instead of many individual clusters numbers, only 3 values are required:
  - Start (cluster number) of the area (extent) in the file
  - Size of the area in the file (in clusters)
  - Number of the first cluster on the storage device

- Result: Lesser overhead
- Examples: JFS, XFS, btrfs, NTFS, ext4



Extents using the Example ext4

- With block addressing in ext2/3, each inode contains 15 areas with a size of 4 Bytes each ($\Longrightarrow$ 60 Bytes) for addressing clusters
- ext4 uses this 60 Bytes for an extent header (12 Bytes) and for addressing 4 extents (12 Bytes each)



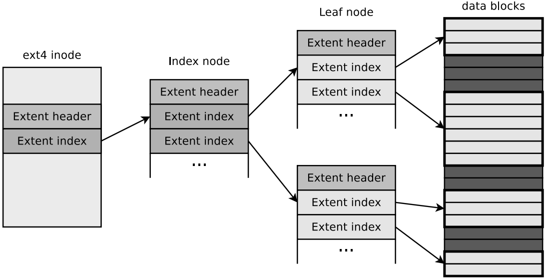
Extents cannot become larger than 128 MB (2^{15} bits) because ext4, just like its predecessors ext2 and ext3, organizes the file system clusters into so-called block groups (see slide 15) with a maximum size of 128 MB.

Maximum partition size of ext4: 2^{48} cluster numbers $\times$ 4096 Byte cluster size = 1 Exabyte

Benefit of Extents using the Example ext4

- With a maximum of 12 clusters, an ext3/4 inode can directly address 48 kB (at 4 kB cluster size)
- With 4 extents, an ext4 inode can directly address 512 MB

- If the size of a file is $>$ 512 MB, ext4 creates a tree of extents
  - The principle is analogous to indirect block addressing



Helpful descriptions of Extents in ext4...

https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout#Extent_Tree
https://www.sans.org/blog/understanding-ext4-part-3-extent-trees/
https://metebalci.com/blog/a-minimum-complete-tutorial-of-linux-ext4-file-system/
An analysis of Ext4 for digital forensics: https://www.sciencedirect.com/science/article/pii/S1742287612000357

Agenda

NTFS – New Technology File System

Several different versions of the NTFS file system exist

- NTFS 1.0: Windows NT 3.1 (released in 1993)
- NTFS 1.1: Windows NT 3.5/3.51
- NTFS 2.x: Windows NT 4.0 bis SP3
- NTFS 3.0: Windows NT 4.0 ab SP3/2000
- NTFS 3.1: Windows XP/2003/Vista/7/8/10

Recent versions of NTFS offer additional features as...

- support for quotas since version 3.x
- transparent compression
- transparent encryption (Triple-DES and AES) since version 2.x

- Cluster size: 512 Bytes to 64 kB
- NTFS offers, compared with its predecessor FAT, among others:
  - Maximum file size: 16 TB ($\Longrightarrow$ extents)
  - Maximum partition size: 256 TB ($\Longrightarrow$ extents)
  - Security features on file and directory level
- Equal to VFAT...
  - implements NTFS file names up a length of 255 Unicode characters
  - implements NTFS interoperability with the MS-DOS operating system family by storing a unique file name in the format 8.3 for each file

Structure of NTFS (1/2)

- The file system contains a Master File Table (MFT)
  - It contains the references of the files to the clusters
  - Also contains the metadata of the files (file size, file type, date of creation, date of last modification and possibly the file content)
    - The content of small files $\leq$ 900 Bytes is stored directly in the MFT

Source: How NTFS Works. Microsoft. 2003. https://technet.microsoft.com/en-us/library/cc781134(v=ws.10).aspx

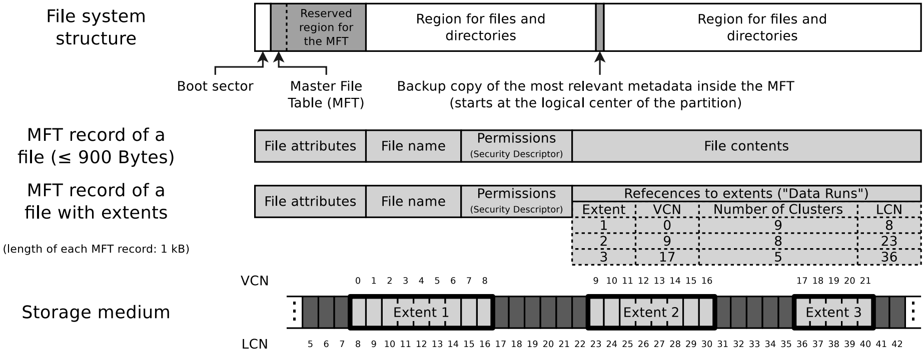
- When a partition is formatted as, a fixed space is reserved for the MFT
  - 12.5% of the partition size is reserved for the MFT by default
  - If the MFT area has no more free capacity, the file system uses additional free space in the partition for the MFT
    - This may cause fragmentation of the MFT (but has no negative effects for flash memory)

Partition size	Cluster size
< 16 TB	4 kB
16 TB - 32 TB	8 kB
32 TB - 64 TB	16 kB
64 TB - 128 TB	32 kB
128 TB - 256 TB	64 kB

The table contains default cluster sizes of Windows 2000/XP/Vista/7/8/10. The cluster size can be specified when the file system is created

Source: http://support.microsoft.com/kb/140365/de

Structure of NTFS (2/2)



If an MFT entry refers to extents (so called „Data Runs“), it stores 3 values per extent

- Start (cluster number) of the area (extent) in the file ⟹ Virtual Cluster Number (VCN)
- Size of the area in the file (in clusters) ⟹ number of clusters
- Number of the first cluster on the storage device ⟹ Logical Cluster Number (LCN)

Also a directory is a file (MFT entry) whose file contents are the numbers of the MFT entries (files) associated with the directory

Agenda

- Block Addressing
  - Minix
  - ext2/3/4 File System
  - Analyze File Systems
- File Allocation Tables
  - FAT Evolution
  - Analyze FAT File Systems
- Extents
  - Extent-based Addressing
  - NTFS
- Journal
- Copy-on-write

Problem: Write Operations

- If files or directories are created, relocated, renamed, erased, or modified, write operations in the file system are carried out
  - Write operations shall convert data from one consistent state to a new consistent state
- If a failure occurs during a write operation, the consistency of the file system must be checked
  - If the size of a file system is multiple GB, the consistency check may take several hours or days
  - Skipping the consistency check, may cause data loss
- Objective: Narrow down the data, which need to be checked by the consistency check
- Solution: Collect the write operations in a journal
  $\Longrightarrow$ Journaling file systems

Journaling File Systems

- Implement a journal, where write operations are collected before being committed to the file system
  - At fixed time intervals, the journal is closed and the write operations are carried out
- Advantage: After a crash, only the files (clusters) and metadata must be checked, for which a record exists in the journal
- Drawback: Journaling increases the number of write operations, because modifications are first written to the journal and next carried out
- 2 variants of journaling:
  - Metadata journaling
  - Full journaling

Helpful descriptions of the different journaling concepts. . .

- Analysis and Evolution of Journaling File Systems, *Vijayan Prabhakaran, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau*, 2005 USENIX Annual Technical Conference, http://www.usenix.org/legacy/events/usenix05/tech/general/full_papers/prabhakaran/prabhakaran.pdf

Metadata Journaling and Full Journaling

- Metadata journaling (*Write-Back*)
  - The journal contains only metadata (inode) modifications
    - Only the consistency of the metadata is ensured after a crash
  - Modifications to clusters are carried out by `sync()` ($\Longrightarrow$ write-back)
    - The `sync()` system call commits the page cache, that is also called = buffer cache to the HDD/SDD
  - Advantage: Consistency checks only take a few seconds
  - Drawback: Loss of data due to a system crash is still possible
  - Optional with ext3/4 and ReiserFS
  - NTFS and XFS provides only metadata journaling
- Full journaling
  - Modifications to metadata and clusters of files are written to the journal
  - Advantage: Ensures the consistency of the files
  - Drawback: All write operation must be carried out twice
  - Optional with ext3/4 and ReiserFS

The alternative is therefore high data security and high write speed

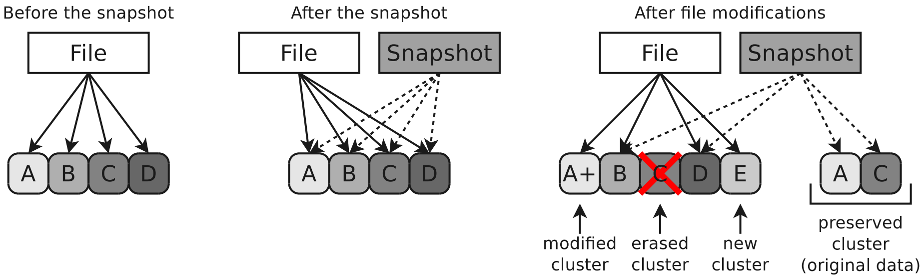
Compromise between the Variants: Ordered Journaling

- Most Linux distributions use by default a compromise between both variants
- Ordered journaling
  - The journal contains only metadata modifications
  - File modifications are carried out in the file system first and next the relevant metadata modifications are written into the journal
  - Advantage: Consistency checks only take a few seconds and high write speed equal to journaling, where only metadata is journaled
  - Drawback: Only the consistency of the metadata is ensured
    - If a crash occurs while incomplete transactions in the journal exist, new files and attachments get lost because the clusters are not yet allocated to the inodes
    - Overwritten files after a crash may have inconsistent content and maybe cannot be repaired, because no copy of the old version exists
  - Examples: Only option when using JFS, standard with ext3/4 and ReiserFS

Interesting:
`https://www.heise.de/newsticker/meldung/Kernel-Entwickler-streiten-ueber-Ext3-und-Ext4-209350.html`

Agenda

Most advanced Concept: Copy-on-write



- Write operations do not modify/erase file system contents
  - Modified data is written into free clusters
  - Afterward, the metadata is modified for the new file
- Older file versions are preserved and can be restored
  - Data security is better compared with journaling file systems
  - Snapshots can be created without delay (just metadata modification)
- Examples: ZFS, btrfs and ReFS (Resilient File System)

You should now be able to answer the following questions:

- What are the main services provided by a file systems?
- What are inodes and clusters?
- How does block addressing work?
- Why do most modern file systems use journaling?
- How does addressing via extents work?
- What is copy-on-write?