

6th Slide Set

Operating Systems

Prof. Dr. Christian Baun

Frankfurt University of Applied Sciences
(1971–2014: Fachhochschule Frankfurt am Main)
Faculty of Computer Science and Engineering
christianbaun@fb2.fra-uas.de

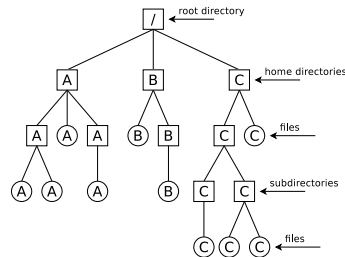
Learning Objectives of this Slide Set

- At the end of this slide set You know/understand...
 - the **functions and basic terminology of file systems**
 - what **inodes** and **clusters** are
 - how **block addressing** works
 - the **structure** of selected file systems
 - an overview about **Windows file systems** and their characteristics
 - what **journaling** is and why it is used by many file systems today
 - how addressing via **extents** works and why it is implemented by several modern file systems
 - what **copy-on-write** is
 - how **defragmentation** works and when it makes sense to defragment

Exercise sheet 6 repeats the contents of this slide set which are relevant for these learning objectives

File Systems...

- organize the storage of files on data storage devices
 - Files are sequences of Bytes of any length which belongs together with regard to content
- manage file names and attributes (metadata) of files
- form a namespace
 - Hierarchy of directories and files



- 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
- should cause only little overhead for metadata

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, the clusters are often called **zones** or **blocks**
 - This results in confusion with the sectors of the devices, which are in literature sometimes called blocks too
- The size of the clusters is essential for the efficiency of the file system
 - The smaller the clusters are. . .
 - Rising overhead for large files
 - Decreasing capacity loss due to internal fragmentation
 - The bigger the clusters are. . .
 - Decreasing overhead for large files
 - Rising capacity loss due to internal fragmentation

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

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

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

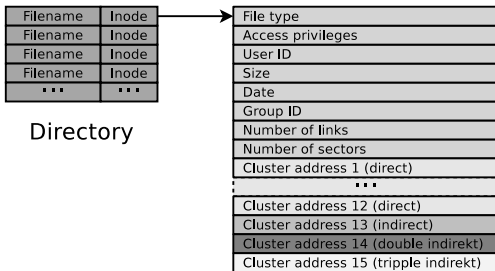
Basic Terminology of Linux File Systems

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

- The page size depends on the architecture
 - x86 = 4 kB, Alpha and Sparc = 8 kB, IA-64 = 4/8/16/64 kB
-
- The creation of a **file** causes the creation of an **Inode** (*index node*)
 - It stores a file's metadata, except the file name
 - Metadata are among others the size, UID/GID, permissions and date
 - Each inode has a unique inode number inside the file system
 - The inode contains references to the file's clusters
 - All Linux file systems base on the functional principle of inodes
 - A **directory** is a file too
 - Content: File name and inode number for each file in the directory
 - The traditional working method of Linux file systems: **Block addressing**
 - Actually, the term is misleading because file systems always address clusters and not blocks (of the volume)
 - However, the term is established in literature since decades

Block Addressing using the Example ext2/3/4

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



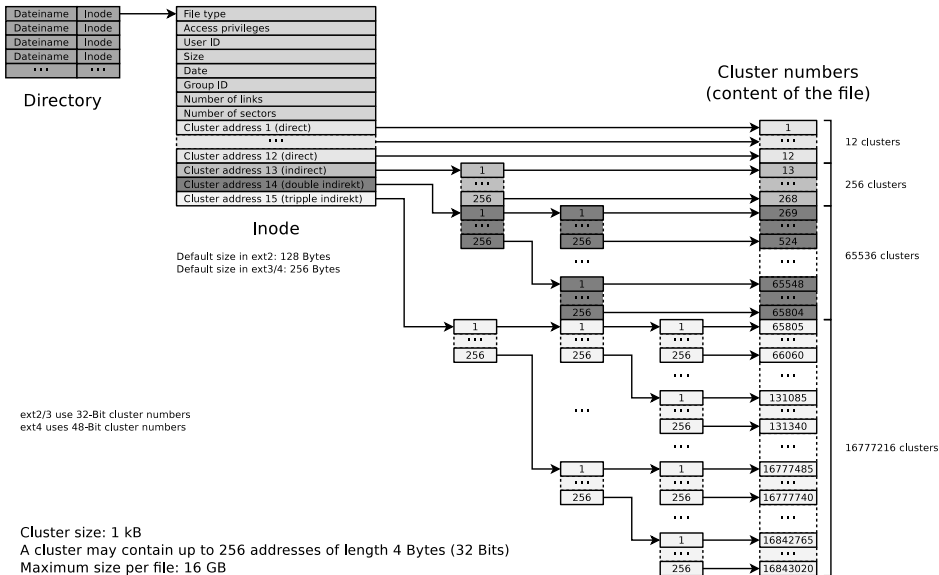
- If a file requires more clusters, these clusters are indirectly addressed
- Minix, ext2/3/4, ReiserFS and Reiser4 implement block addressing

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 using the Example ext2/3/4



Minix

The Minix operating system

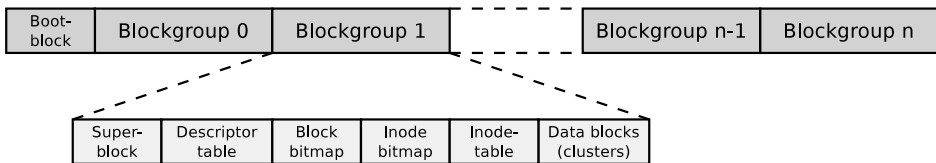
- Unix-like operating system
 - Developed since 1987 by Andrew S. Tanenbaum for education purposes
 - Latest revision is 3.3.0 is from 2014
-
- Standard Linux file system until 1992
 - Not surprising, because Minix was the basis of the development of Linux
 - The Minix file system causes low overhead
 - Useful applications „today“: Boot floppy disks and RAM disks
 - Storage is represented as a linear chain of equal-sized blocks (1-8 kB)
 - A Minix file system contains just 6 areas
 - The simple structure makes it ideal for education purposes

Minix File System Structure

| 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. 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)
- **Inodes.** Contains the inodes with the metadata
 - Every file and every directory is represented by at least a single inode, which contains the metadata
 - Metadata is among others the 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

ext2/3



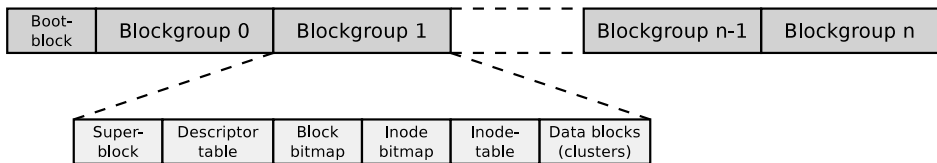
- The clusters of the file system are combined to **block groups** of the same size
 - The information about the metadata and free clusters of each block group are maintained in the respective block group

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

Example: If the cluster size is 1,024 bytes, each block group can contain up to 8,192 clusters

- Benefit of block groups (when using HDDs!): Inodes (metadata) are physically located close to the clusters, they address
 - This reduces seek times and the degree of fragmentation
 - With flash memories, the position of the data in the individual memory cells is irrelevant for the performance

ext2/3 Block Group Structure



- 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
- **Block bitmap** and **inode bitmap** are each a single cluster big
 - 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**

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 File Allocation Table (FAT) file system is based on the data structure of the same name
- The FAT (**File Allocation Table**) is a table of fixed size
- For each cluster in the file system, an entry exists in the FAT with the following information about the cluster:
 - Cluster is free or the storage medium is damaged at this point
 - Cluster is 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**)
⇒ see slides 15 und 17

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 |

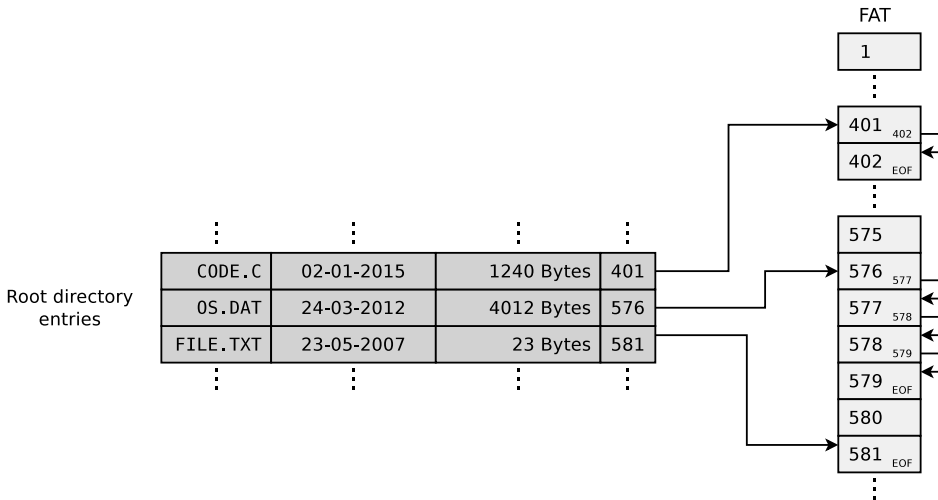
- The **boot sector** contains executable x86 machine code, which starts the operating system, and information about the file system:
 - Block size of the storage device (512, 1,024, 2,048 or 4,096 Bytes)
 - Number of blocks per cluster
 - Number of blocks (sectors) on the storage device
 - Description (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**

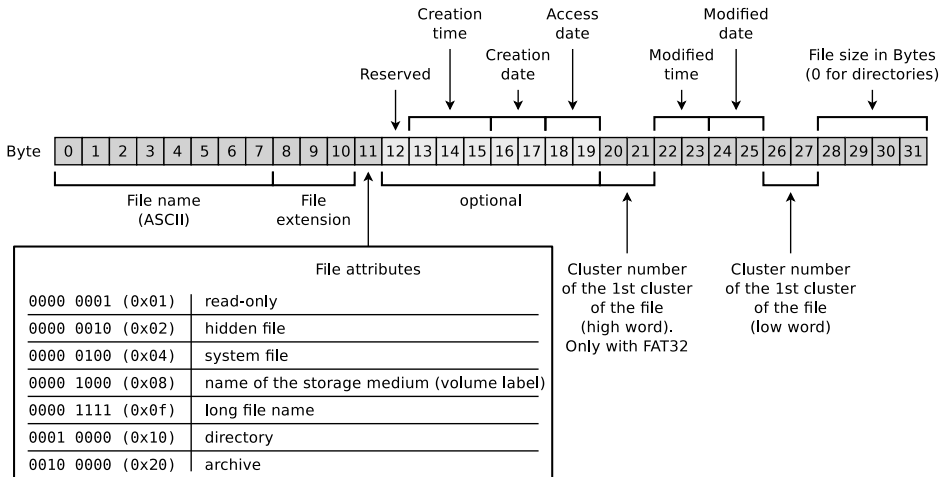
Root Directory and FAT



The topic FAT is clearly explained by...

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

Structure of Root Directory Entries

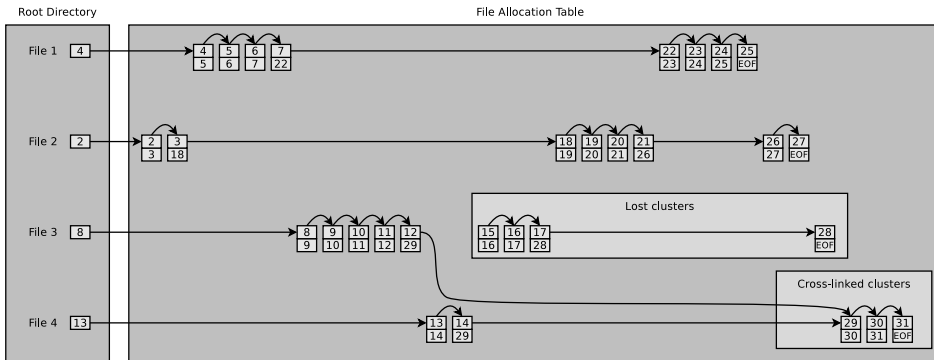


Why is 4 GB the maximum file size on FAT32?

Only 4 Bytes are available for specifying the file size.

Risk of File System Inconsistencies

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



Source: http://www.sal.ksu.edu/faculty/tim/ossf/File_sys/file_system_errors.html

FAT12

Released in 1980 with the first QDOS release

- Length of the cluster numbers: 12 bits
 - Up to $2^{12} = 4,096$ clusters can be addressed
- Cluster size: 512 Bytes to 4 kB
- Supports storage media (partitions) up to 16 MB

$2^{12} * 4 \text{ kB cluster size} = 16.384 \text{ kB} = 16 \text{ MB maximum file system size}$

- 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

Used „today“ only for DOS and Windows floppy disks

FAT16

- Released in 1983 because it was foreseeable that an address space of 16 MB is insufficient
- Up to $2^{16} = 65,536$ clusters can be addressed
 - 12 clusters are reserved
- Cluster size: 512 Bytes to 256 kB
- File names are supported only in 8.3 format
- Main field of application today: Mobile storage media ≤ 2 GB

| 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. 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 64 kB cluster size

Some operating systems (e.g. MS-DOS and Windows 2000/XP/7) do not support 128 kB and 256 kB cluster size

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

FAT32

- Released in 1997 because of the rising HDD capacities and because clusters > 32 kB waste a lot of storage
- Size of the FAT entries for the cluster numbers:
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
 - Cause: Only 4 Bytes are available for indicating the file size
- Main field of application today: Mobile storage media > 2 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. The cluster size can be manually specified during the file system creation

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 an interesting example for implementing a new functionality without losing the backward compatibility
- Long file names (up to 255 characters) are distributed to max. 20 pseudo-directory entries (see slide 22)
- File systems without Long File Name support ignore the pseudo-directory entries and show only the shortened name
- For a VFAT entry in the FAT, the first 4 bit of the **file attributes** field have value 1 (see slide 15)
- Special attribute: Upper/lower case is displayed, but ignored

Compatibility with MS-DOS

- VFAT and NTFS (see slide 34) store for every file a unique filename in 8.3 format
 - Operating systems without the VFAT extension ignore the pseudo-directory entries and only show the shortened file name
 - This way, Microsoft operating systems without NTFS and VFAT support can access files on NTFS partitions
- Challenge: The short file names must be unique
- Solution:
 - All special characters and dots inside the name are erased
 - All lowercase letters are converted to uppercase letters
 - Only the first 6 characters are kept
 - Next, a ~1 follows before the dot
 - The first 3 characters after the dot are kept and the rest is erased
 - If a file with the same name already exists, ~1 is replaced with ~2, etc.
- Example: The file A very long filename.test.pdf is displayed in MS-DOS as: AVERYL~1.pdf

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

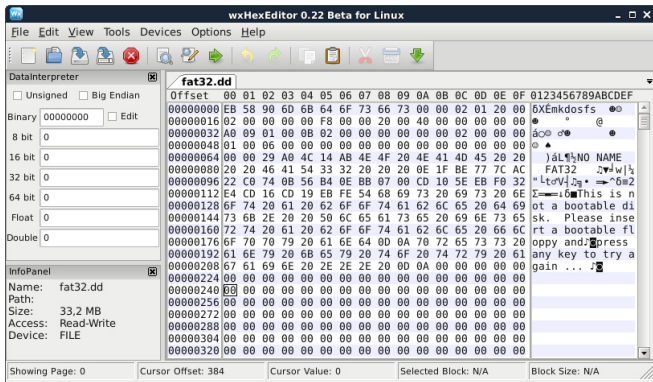
# ls -l /mnt/fat32
insgesamt 0

# 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)



Helpful information:

<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)

wxHexEditor 0.22 Beta for Linux

File Edit View Tools Devices Options Help

DataInterpreter

☐ Unsigned ☐ Big Endian

Binary 00000000 ☐ Edit

8 bit 0

16 bit 0

32 bit 0

64 bit 0

Float 0

Double 0

InfoPanel

Name: fat32.dd

Path:

Size: 33,2 MB

Access: Read-Write

Device: FILE

| Offset | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | 0123456789ABCDEF |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------------------|
| 00551920 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00551936 | 41 | 6C | 00 | 69 | 00 | 65 | 00 | 73 | 00 | 6D | 00 | 0F | 00 | 61 | 69 | 00 | A l i e s m * a i |
| 00551952 | 63 | 00 | 68 | 00 | 2E | 00 | 74 | 00 | 78 | 00 | 00 | 00 | 74 | 00 | 00 | 00 | c h . t x t |
| 00551968 | 4C | 49 | 45 | 53 | 4D | 49 | 43 | 48 | 54 | 58 | 54 | 20 | 00 | 64 | B4 | 55 | LIESMICHTEXT d\U |
| 00551984 | 5C | 44 | 5C | 44 | 00 | 00 | B4 | 55 | 5C | 44 | 03 | 00 | 10 | 00 | 00 | 00 | \D\D {U\D |
| 00552000 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552016 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552032 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552048 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552064 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552080 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552096 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552112 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552128 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552144 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552160 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552176 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552192 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552208 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552224 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552240 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552256 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552272 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552288 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552304 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552320 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552336 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552352 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552368 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552384 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552400 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552416 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552432 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00552448 | 42 | 65 | 74 | 72 | 69 | 65 | 62 | 73 | 73 | 79 | 73 | 74 | 65 | 6D | 65 | 0A | Betriebssysteme |
| 00552464 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |

Showing Page: 985 Cursor Offset: 551920 Cursor Value: 0 Selected Block: N/A Block Size: N/A

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: Implement a journal, which keeps track about all write operations \implies 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
- <http://www.ibm.com/developerworks/library/l-journaling-filesystems/index.html>

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()` (\implies write-back)
 - The `sync()` system call commits the page cache, that is also called = buffer cache (see slide 37) 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

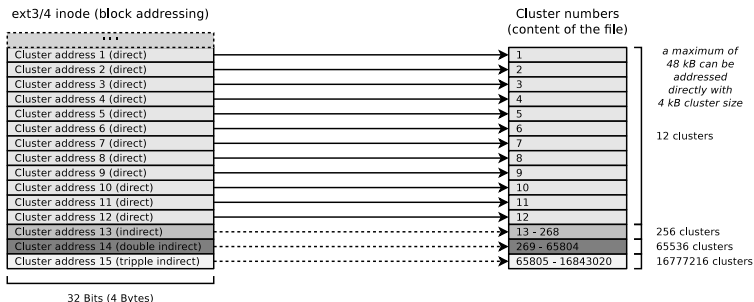
The alternative is therefore high data security and high write speed

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

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

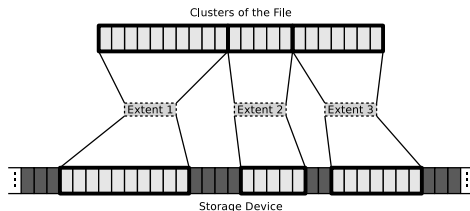


- This addressing scheme causes rising overhead with rising file size
- Solution: Extents

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

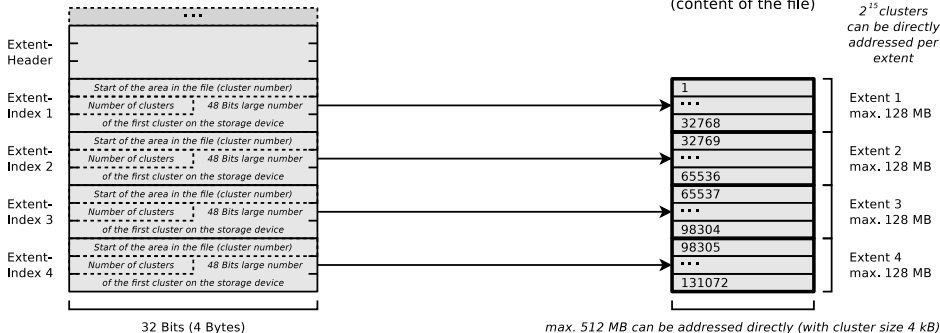
- Result: Lesser overhead
- Examples: JFS, XFS, btrfs, NTFS, ext4



Extents using the Example ext4

- With block addressing in ext3/4, each inode contains 15 areas with a size of 4 Bytes each (\Rightarrow 60 Bytes) for addressing clusters
- ext4 uses this 60 Bytes for an extent header (12 Bytes) and for addressing 4 extents (12 Bytes each)

ext4 inode (extent-based addressing)

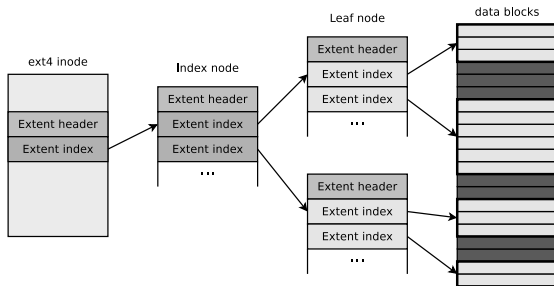


Maximum partition size of ext4: 2^{48} cluster numbers \times 4096 Byte cluster size = 1 Exabyte

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



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 (\implies extents)
 - Maximum partition size: 256 TB (\implies 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

- 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 ≤ 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](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

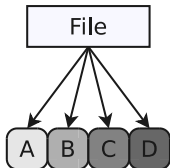
| Partition size | Cluster size |
|-----------------|--------------|
| < 16 TB | 4 kB |
| 16 TB - 32 TB | 8 kB |
| 32 TB - 64 TB | 16 kB |
| 64 TB - 128 TB | 32 kB |
| 128 TB - 256 TB | 64 kB |

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

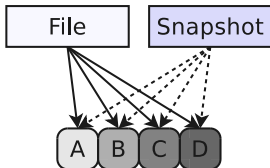
Source: <http://support.microsoft.com/kb/140365/de>

Most advanced Concept: Copy-on-write

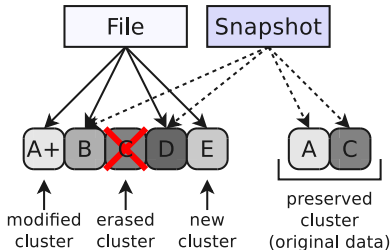
Before the snapshot



After the snapshot



After file modifications



- 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)

Accelerating Data Access with a Cache (1/2)

- Modern operating systems accelerate the access to stored data with a **Page Cache** (called *Buffer Cache*) in the main memory
 - If a file is requested for reading, the kernel first tries to allocate the file in the cache
 - If the file is not present in the cache, it is loaded into the cache
- The page cache is never as big as the amount of data on the system
 - That is why rarely requested files must be replaced
 - If data in the cache was modified, the modification must be passed down (written back) at some point in time
 - Optimal use of the cache is impossible because data accesses are non-deterministic (unpredictable)
- Most operating systems do not pass down write accesses immediately (\Rightarrow **write-back**)
 - Benefit: Better system performance
 - Drawback: System crashes may cause inconsistencies

Accelerating Data Access with a Cache (2/2)

- DOS and Windows up to version 3.11 use the *Smartdrive* utility to implement a page cache
 - All later versions of Windows also contain a *Cache Manager* that implements a page cache
- Linux automatically buffers as much data as there is free space in main memory
 - The command `free -m` returns an overview of the memory usage under Linux
 - It also informs in the `buffers` and `cached` columns how much main memory is currently used for the page cache

```
$ free -m
```

| | total | used | free | shared | buffers | cached |
|--------------------|-------|------|-------|--------|---------|--------|
| Mem: | 7713 | 6922 | 790 | 361 | 133 | 1353 |
| -/+ buffers/cache: | | 5436 | 2277 | | | |
| Swap: | 11548 | 247 | 11301 | | | |

Good sources regarding the page cache under Linux and how to empty it manually

http://www.thomas-krenn.com/de/wiki/Linux_Page_Cache

<http://unix.stackexchange.com/questions/87908/how-do-you-empty-the-buffers-and-cache-on-a-linux-system>

<http://serverfault.com/questions/85470/meaning-of-the-buffers-cache-line-in-the-output-of-free>

Fragmentation

- A cluster can only be assigned to a single file
 - If a file is bigger than a cluster, the file is split and stored in several clusters
 - **Fragmentation** means that logically related clusters are not located physically next to each other
- Objective: Avoid frequent movements of the HDDs arms
 - If the clusters of a file are distributed over the HDD, the heads need to perform more time-consuming position changes when accessing the file
 - For SSDs the position of the clusters is irrelevant for the latency

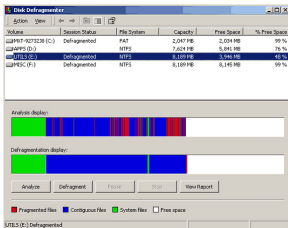


Image source: <http://windowsitpro.com>

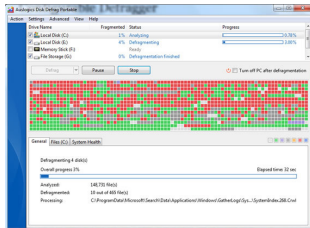


Image source: <http://www.teknobites.com>

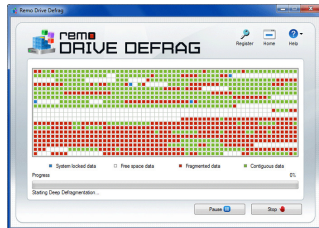


Image s.: <http://www.remOSOFTWARE.com>

Defragmentation (1/3)

- These questions are frequently asked:
 - Why is it for Linux/UNIX not common to defragment?
 - Does fragmentation occur with Linux/UNIX?
 - Is it possible to defragment with Linux/UNIX?
- First of all, we need to answer: What do we want to achieve with **defragmentation**?
 - Writing data to a drive, always leads to fragmentation
 - The data is no longer contiguously arranged
 - A continuous arrangement would maximum accelerate the **continuous forward reading** of the data because no more seek times occur
 - Only if the seek times are huge, defragmentation makes sense
 - With operating systems, which use only a little amount of main memory for caching HDD accesses, high seek times are very negative

Discovery 1: Defragmentation accelerates mainly the continuous forward reading

Defragmentation (2/3)

- Singletasking operating systems (e.g. MS-DOS)
 - Only a single application can be executed
 - If this application often hangs, because it waits for the results of read/write requests, this causes a poor system performance

Discovery 2: Defragmentation may be useful for singletasking operating systems. In practice, however, single-tasking operating systems are used seldom

- Multitasking operating systems
 - Multiple programs are executed at the same time
 - **Applications can almost never read large amounts of data in a row, without other applications in between, requesting r/w operations**
- In order to prevent that programs, which are executed at the same time, do interfere each other, operating systems read more data than requested
 - The system reads a stock of data into the **cache**, even if no requests for these data exist

Discovery 3: In multitasking operating systems, applications can almost never read large amounts of data in a row

Defragmentation (3/3)

- Linux systems automatically hold data in the cache, which is frequently accessed by the processes
 - **The impact of the cache greatly exceeds the short-term benefits, which can be achieved by defragmentation**

Discovery 4: Defragmenting has mainly a benchmark effect

Discovery 5: Enlarge the file system cache brings better results than defragmentation

- Defragmenting has mainly a **benchmark effect**
 - In practice, defragmentation (in Linux!) causes almost no positive impact
 - Tools like `defragfs` can be used for Linux file system defragmentation
 - Using these tools is often not recommended and useful

Helpful source of information: http://www.thomas-krenn.com/de/wiki/Linux_Page_Cache