BETRIEBSSYSTEME DATEISYSTEM

Prof. Dr. Jörg Mielebacher
mail@mielebacher.de

Die folgenden Folien führen in das Dateisystem und seine Verwaltung durch das Betriebssystem ein. Zu jeder Folie sind Notizenseiten erfasst.

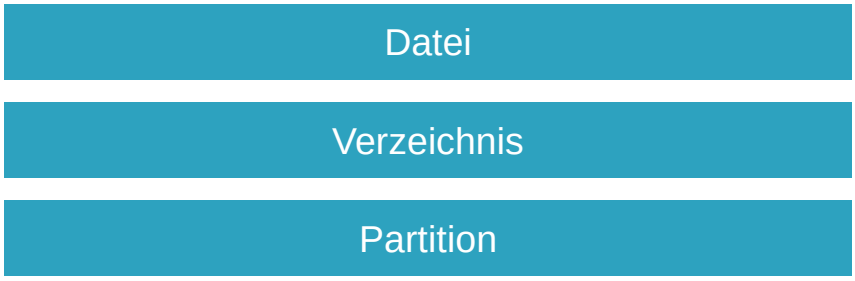
Verbesserungsvorschläge und Fehlerhinweise können Sie gerne an die Adresse mail@mielebacher.de senden.

Wiederholung

- I.d.R. gibt es die Standard-Datenströme stdin (Eingabe), stdout (Ausgabe) und stderr (Fehlerausgabe). Diese lassen sich umleiten.
- Das Ein- und Ausgabe-Subsystem ist ein besonders wichtiger Teil des Betriebssystems und sorgt für Hardwareunabhängigkeit, einheitliche Benennung, effiziente Nutzung, hardwarenahe Fehlerbehandlung sowie Schutz der Geräte.
- Meist nutzt man Memory-Mapped-IO.
- Anwendungen können nur über das Betriebssystem auf die Hardware zugreifen; dafür werden Treiber genutzt, die mit den Gerätecontrollern kommunizieren.
- Puffer wirken gegen Geschwindigkeitsunterschiede.
- Die Anbindung der Geräte erfolgt durch programmierte E/A, Interrupts oder DMA.

Dateien in der Praxis

Ein erster Überblick



Wenn man sich im Dateisystem eines Computers bewegt, trifft man auf Dateien, Verzeichnisse und Partitionen. Dateien speichern Daten oder Anweisungen (d.h. Programme). Verzeichnisse fassen Dateien und andere Verzeichnisse zusammen (es entsteht ein Verzeichnisbaum). Teilweise werden Verzeichnissen auch selbst als Dateien repräsentiert (z.B. unter Linux). Partitionen wiederum trennen Bereiche von Datenträgern und enthalten Verzeichnisse und Dateien.

Dateien

Skripte

Verknüpfungen

Ausführbare Dateien

Bildddateien

Verzeichnisse

Bytecode

Videodateien

Textdateien

Audiodateien

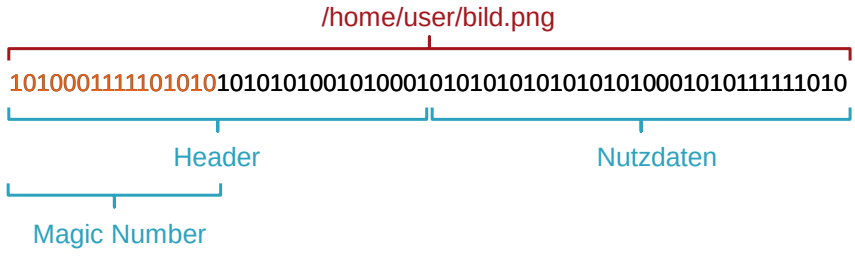
Pseudodateien

Binärdateien

u.v.m.

Unter einer Datei versteht man einen zusammenhängenden Block von Daten, der auf einem Datenträger gespeichert ist. Man unterscheidet eine Vielzahl von Dateien – ausführbare Dateien (im Sinn von: Dateien mit Maschinencode), Skripte, Bytecode (z.B. Java) , menschenlesbare Textdateien (Quellcode, XML, JSON, HTML, meist Log-Dateien usw.), Binärdateien, darunter z.B. Audiodateien (MP3, WAV usw.), Bilddateien (JPG, PNG, GIF usw.), Videodateien (MPG, MP4 usw.), Verknüpfungen zu anderen Dateien oder Verzeichnissen, Verzeichnisse (plattformabhängig – unter Linux werden sie als Datei gespeichert) und Pseudodateien (ebenfalls sehr verbreitete unter Linux, z.B. die Inhalte von /proc). Die Liste ließe sich noch weiter fortsetzen.

Dateiformat



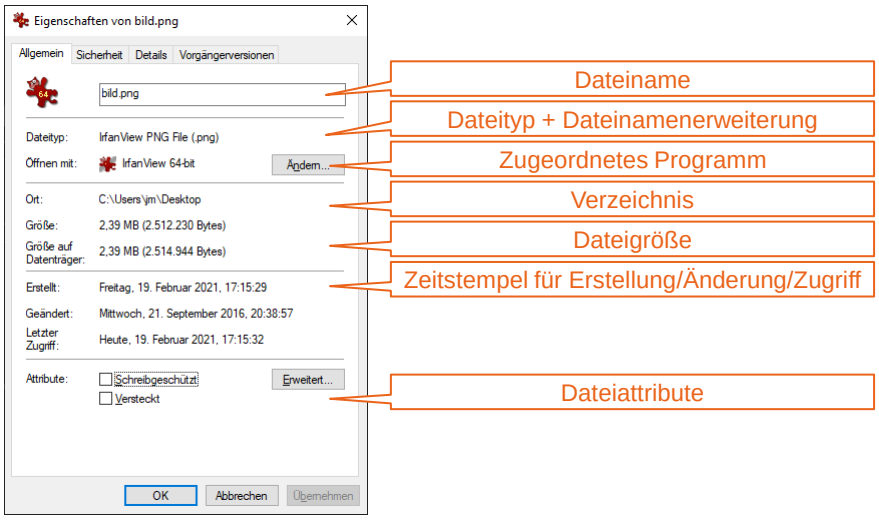

```
root@herkules:/home/jm# file bild.png
bild.png: PNG image data, 2953 x 1973, 8-bit colormap, non-interlaced
```

Auf dem Datenträger sind Dateien ist – egal was darin gespeichert ist – zunächst ein langer Strom von Bytes. Üblicherweise werden Dateien durch einen bestimmten Dateinamen in einem Verzeichnis angesprochen. Heute ist man bei der Vergabe von Datei- und Verzeichnisnamen recht frei, meist ist ihre Länge auf 255 Zeichen begrenzt, teilweise sind einzelne Zeichen nicht erlaubt (z.B. *, ?, / usw.), da diese andere Bedeutungen haben. Das war nicht immer so: In CP/M und MS DOS beispielsweise waren bis in die 1990er-Jahre nur die sog. 8.3-Namen erlaubt, d.h. eine maximale Länge von 8 Zeichen vor dem Punkt und maximal 3 Zeichen nach dem Punkt.

Wie Informationen in einer Datei codiert sind, legt das Dateiformat fest. Das ist aber nicht Aufgabe des Betriebssystems. Dateiformate werden von den Herstellern oder Standardisierungsgremien festgelegt. Das Dateiformat beinhaltet meist die Unterteilung der Datei in einen Header und die Nutzdaten. Welches Dateiformat eine Datei verwendet, wird auf unterschiedliche Weise bestimmt: Windows setzt seit langer Zeit vor allem auf die Dateiendung (hier: .png) und erlaubt es, Dateiendungen mit bestimmten Programmen zu verknüpfen. Wichtig: Eine Änderung der Dateiendung hat keinerlei Auswirkungen auf den Dateiinhalt – eine Textdatei mit der Endung .mp3 bleibt immer noch eine Textdatei; der Text wird also nicht in Audio umgewandelt.

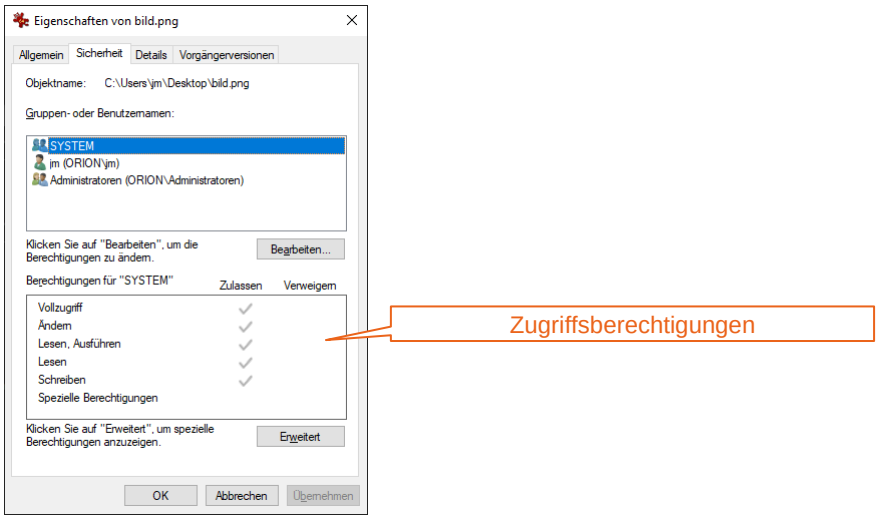
Unter Linux wird vor allem ausgenutzt, dass sich das Dateiformat in den Dateien selbst erkennen lässt, meist durch sog. Magic Numbers, also Zeichen am Anfang der Datei, die sich einem bestimmten Dateityp zuordnen lassen (hier: 0x89504e470d0a1a0a). Teilweise legt auch der Speicherort einer Datei ihr Dateiformat fest (z.B. bei Pseudodateien). Untersuchen lässt sich das unter Linux z.B. mit dem file Befehl.

Informationen und Attribute unter Windows



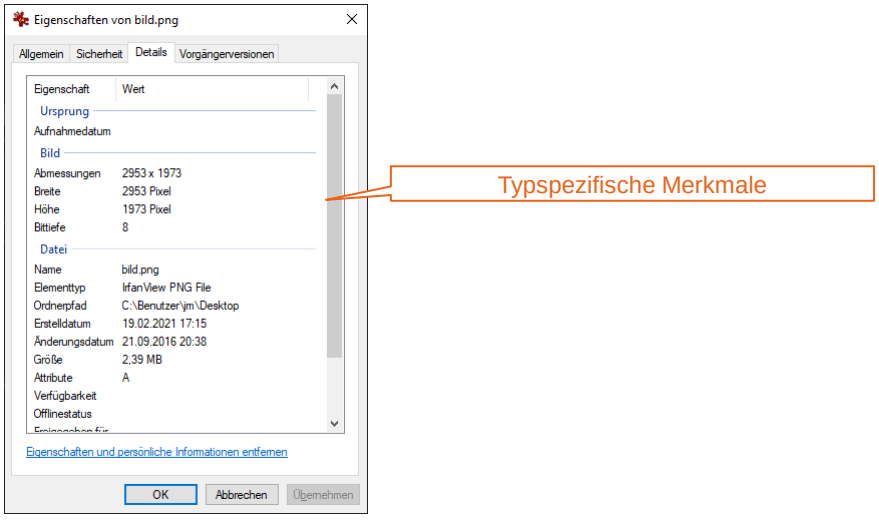
Durch Rechtsklick und „Eigenschaften“ kann man unter Windows, detaillierte Informationen zu einer Datei abrufen. Neben Dateiname, Dateityp und Verzeichnis gehören hierzu das zugeordnete Programm, die Größe der Datei in Bytes und die davon abweichende Größe auf dem Datenträger (da immer ganze Zuordnungseinheiten belegt werden, mehr dazu später), außerdem Zeitstempel der Erstellung, der letzten Änderung und des letzten Zugriffs. Für jede Datei sind außerdem Attribute festgelegt – vor allem, ob die Datei gegen Änderungen geschützt ist („Schreibgeschützt“) und ob sie bei Standardeinstellungen in Verzeichnissen ausgeblendet ist („Versteckt“).

Dateiattribute unter Windows



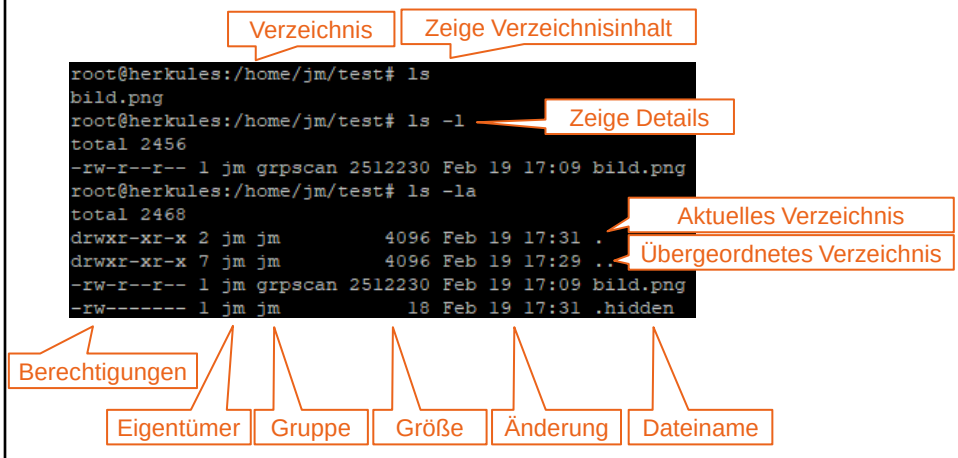
Der Tab Sicherheit erlaubt das Festlegen von Berechtigungen auf Benutzer- oder Gruppenebene, auch lässt sich hierüber der Besitzer festlegen.

Dateiattribute unter Windows



Je nach Dateityp werden unter Details typspezifische Merkmale angezeigt, z.B. Größe oder Farbtiefe bei Bildern.

Dateiattribute unter Linux



Auf der Linuxkommandozeile lassen sich mit dem ls-Befehl Inhalte eines Verzeichnisses anzeigen, standardmäßig zeigt es nur die Namen der Dateien und Verzeichnisse und blendet versteckte Dateien aus. Mit ls –l lassen sich auch Dateils anzeigen, mit der Option –a werden auch versteckte Dateien angezeigt. Unter Linux werden Dateien, deren Name mit einem Punkt beginnt, als versteckt behandelt.

ls –l zeigt unter anderem die Berechtigungen an sowie den Namen des Eigentümers, den Namen der Gruppe, die Dateigröße und den Zeitpunkt der letzten Änderung. Zu erkennen sind außerdem die beiden besonderen Verzeichnis . (das auf das aktuelle Verzeichnis verweist) und .. (das auf übergeordnete Verzeichnis verweist).

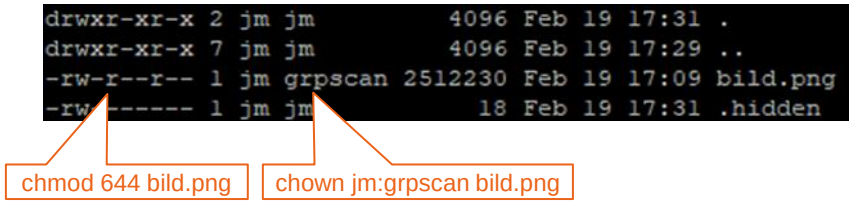
Dateiattribute unter Linux

```
root@herkules:/home/jm/test# stat bild.png
  File: bild.png
  Size: 2512230         Blocks: 4912       IO Block: 4096   regular file
Device: 801h/2049d      Inode: 7340574     Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/      jm)   Gid: ( 1001/ grpscan)
Access: 2021-02-19 17:10:00.923532045 +0100
Modify: 2021-02-19 17:09:38.863658747 +0100
Change: 2021-02-19 17:32:01.600005838 +0100
 Birth: -
```

Detailliertere Informationen lassen sich mit dem Befehl stat anzeigen. Hier werden dann neben gerätespezifischen Angaben auch die vollständigen Zeitstempel von letztem Zugriff, letzter Änderung des Inhalts (Modify) und der letzten Änderung der Metadaten (Change).

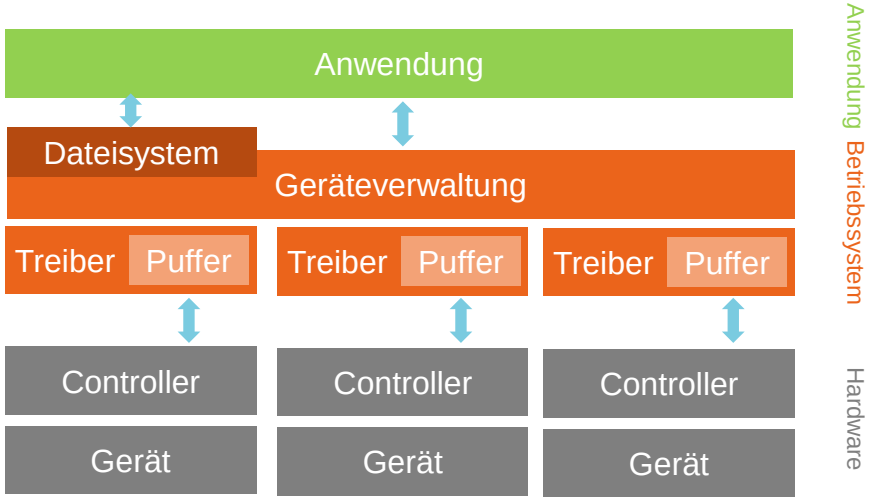
Dateiberechtigungen unter Linux

Benutzerklasse	User			Group			Other		
Berechtigung	r	w	x	r	w	x	r	w	x
Wert	4	2	1	4	2	1	4	2	1



Linux verwendet meist eine dreistellige Oktalzahl (d.h. je 3x3bit), um die Berechtigungen für eine Datei oder ein Verzeichnis festzulegen. Dabei werden die drei Benutzerklassen User (Eigentümer), Group (Gruppe) und Other (alle Anderen) unterschieden. Für jede Klasse werden die drei Berechtigungen read (Lesen), write (Schreiben) und execute (Ausführen, d.h. Datei kann als Programm ausgeführt werden, bedeutet bei Verzeichnissen aber auch, in das Verzeichnis wechseln zu können) erfasst. Daneben werden noch andere Bits erfasst, dazu zählen Setuid (SUID), mit dem sich die ID des ausführenden Benutzers bestimmen lässt (gilt aber nur auf manchen Systemen), gleiches gilt für Setgid (SGID) für die ID der ausführenden Gruppe. Wichtig ist das Sticky bit, das festgelegt, ob man in Verzeichnissen die Dateien anderer Benutzer löschen darf.
Für das Anpassen von Berechtigungen setzt man unter Linux üblicherweise chmod und chown (ggf. auch chgrp) ein. Im hier gezeigten Beispiel kann bild.png von Eigentümer, anderen Gruppenmitgliedern und allen anderen gelesen, aber nur vom Eigentümer verändert werden. Niemand kann die Datei ausführen – daher chmod 644 bild.png; das Ergebnis kann man gut in der Ausgabe von ls –l erkennen. Die Datei gehört dem Benutzer jm und der Gruppe grpscan, was mit chown jm:grpscan bild.png erreicht wurde. Noch ein Tipp: chown und chmod unterstützen jeweils die Option –R, mit dem sie rekursiv, d.h. für alle Elemente und Unterordner eines Verzeichnisses, angewandt werden.

Das Dateisystem



Das Betriebssystem übernimmt im Hinblick auf Dateien wichtige Aufgaben: Es überseits einerseits die Anbindung der Speichergeräte (z.B. Festplatten). Andererseits stellt es auf diesen Speichergeräten das Dateisystem bereit, durch das das Anwendungen Dateien und Verzeichnisse auf den Speichergeräten verwenden können.

Dateien in C++

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
    ofstream file( "helloworld.txt" );

    if( !file )
    {
        cout << "File not ready." << endl;
        return -1;
    }

    file << "Hello world" << endl;

    file.close();

    return 0;
}
```

Datei zum Schreiben öffnen

Datei nicht geöffnet

Ausgabe in Datei

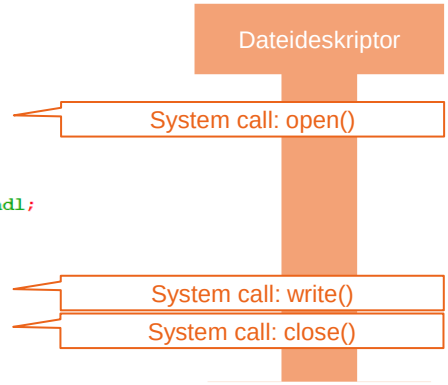
Datei schließen

Das gezeigte C++-Programm öffnet die Datei helloworld.txt im aktuellen Verzeichnis und schreibt in sie den Text „Hello world“. Anschließend wird die Datei geschlossen. Dieses Vorgehen ist typische für viele Programme – Dateien werden geöffnet, entweder zum Lesen oder zum Schreiben oder auch für beides; sie werden für Datei-Ein/Ausgabe verwendet und anschließend wieder geschlossen, um z.B. für andere Prozesse verwendbar zu werden.

In C++ werden ifstream (Lesen) und ofstream (Schreiben) für Dateien verwendet; die Zugriffe erfolgen über die jeweiligen Streams. In C würde man Dateien mit fopen öffnen, was einen FILE*-Zeiger liefert, mit fprintf bzw. fscanf Inhalte schreiben bzw. lesen und mit fclose wieder schließen.

Dateien in C++

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
    ofstream file( "helloworld.txt" );

    if( !file )
    {
        cout << "File not ready." << endl;
        return -1;
    }

    file << "Hello world" << endl;

    file.close();

    return 0;
}
```



Wie üblich greift der Prozess nicht selbst auf die Festplatte zu, sondern muss im User Mode entsprechende Systemaufrufe starten. Dateien werden unter Linux beispielsweise mit open geöffnet, was auch die Prüfung der Dateiberechtigungen mit einschließt. Inhalte werden mit write geschrieben und die Datei wird mit close geschlossen. Die C- bzw. C++-Standardbibliothek verbirgt diese plattformspezifischen Aufrufe, sodass z.B. unter Windows und Linux gleichermaßen mit ofstream gearbeitet werden kann.

Ein wichtiger Aspekt der Systemaufrufe besteht in der Nutzung sog. Dateideskriptoren. Das Öffnen einer Datei liefert einen solchen Dateideskriptor in Form einer Integer-Zahl, die bei allen nachfolgenden Aufrufen verwendet wird, um die betroffene Datei zu identifizieren. Dies erfolgt also nur beim Öffnen anhand des Dateinamens, anschließend immer anhand des Dateideskriptors. Dateideskriptoren nennt man je nach Plattform auch Datei-Handle oder einfach nur Handle (auch wenn Handle sich eigentlich nicht nur auf Dateien bezieht).

Drei Dateideskriptoren kennen Sie bereits: 0 für stdin, 1 für stdout und 2 für stderr.

Typische Systemaufrufe

Dateien

- Öffnen (open)
- Schließen (close)
- Lesen (read)
- Schreiben (write)
- Positionieren (seek)
- Erzeugen (creat)
- Löschen (unlink)

Verzeichnisse

- Erzeugen (mkdir)
- Löschen (rmdir)
- Einträge lesen (readdir)
- Wechseln (chdir)

Das Betriebssystem stellt für die Nutzung von Dateien und Verzeichnissen eine Reihe von Systemaufrufen zur Verfügung – drei von ihnen haben sie bereits gesehen. Die hier genannten Aufrufe beziehen sich auf Linux (unistd.h); sie sind plattformspezifisch, weshalb man immer die Funktionen der C bzw. C++-Standardbibliothek verwenden sollte.

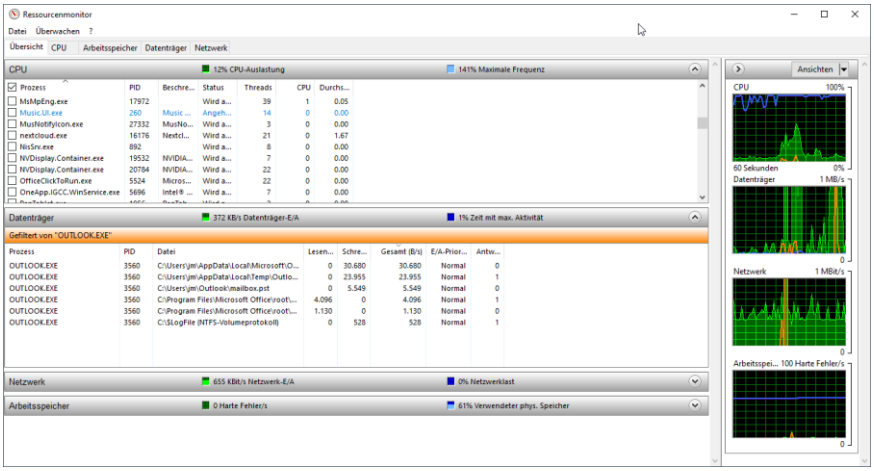
Typisch auf allen Plattformen sind das Öffnen und Schließen von Dateien, das Lesen und Schreiben, das Positionieren (d.h. das Verschieben des Schreib-/Lesezeigers, also des Offsets innerhalb der Datei, ab dem geschrieben bzw. gelesen wird) sowie das Erzeugen und Löschen von Dateien.

Verzeichnissen lassen sich erzeugen und löschen, man kann die Einträge des Verzeichnisses (d.h. die enthaltenen Elemente) auslesen sowie in ein Verzeichnis wechseln.

Prozesskontext

Das Betriebssystem verwaltet die von einem Prozess geöffneten Dateien als Teil des sog. Prozess-Kontexts, d.h. das Betriebssystem „weiß“, welche Dateien von welchem Prozesse auf welche Weise genutzt werden.

Dateien im Windows-Ressourcenmonitor



Unter Windows kann man z.B. den Ressourcenmonitor nutzen, um die von einem Prozess verwendeten Dateien anzuzeigen. Diese werden unter Datenträger für den ausgewählten Prozess angezeigt. Da die Daten laufend aktualisiert werden, kann man sehr gut die Lese/Schreib-Aktivität des Prozesses untersuchen und ggf. auch Engpässe des Datenträgers identifizieren.

Eine Alternative auf der Kommandozeile ist das Tool handle aus der SysinternalsSuite, mit dem sich geöffnete Dateien je Prozess anzeigen lassen bzw. welcher Prozess eine bestimmte Datei verwendet.

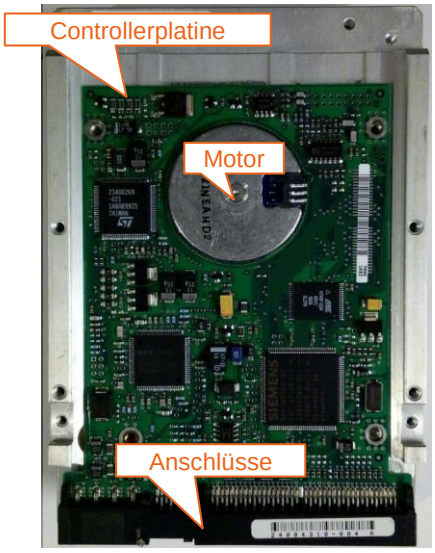
/proc/…/fd und lsof unter Linux

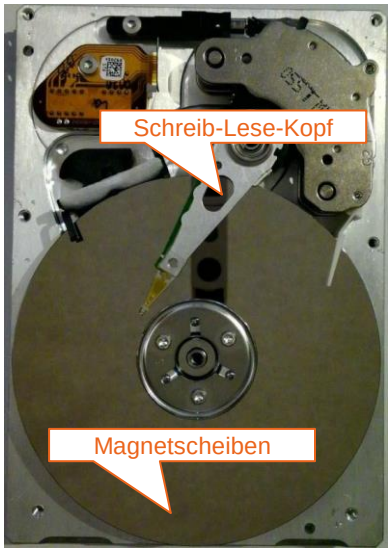
```
root@herkules:/proc/9269/fd# ls -la
total 0
dr-x------ 2 root root  0 Dec 25 15:52 .
dr-xr-xr-x 9 root root  0 Dec 25 15:52 ..
lr-x------ 1 root root 64 Dec 25 15:52 0 -> /dev/null
lrwx------ 1 root root 64 Dec 25 15:52 1 -> /dev/null
lrwx------ 1 root root 64 Dec 25 15:52 10 -> /dev/null
l-wx------ 1 root root 64 Dec 25 15:52 11 -> /run/samba/smbd.pid
```

```
root@herkules:/# lsof -p 9269
COMMAND  PID USER   FD      TYPE             DEVICE SIZE/OFF      NODE NAME
smbd    9269 root  cwd       DIR               8,1     4096         2 /
smbd    9269 root  rtd       DIR               8,1     4096         2 /
smbd    9269 root  txt       REG               8,1    92256   6566946 /usr/sbin/smbd
smbd    9269 root  mem       REG               8,1   421888   9045890 /var/lib/samba/private/passdb.tdb
smbd    9269 root  mem       REG               8,1   421888   8917500 /var/lib/samba/account_policy.tdb
smbd    9269 root  mem-r     REG              0,21   454656     14266 /run/samba/gencache_notrans.tdb
smbd    9269 root  mem       REG               8,1  1609728   8784678 /var/cache/samba/gencache.tdb
```

Unter Linux enthält das /proc-Verzeichnis für jeden Prozess im Unterverzeichnis fd (file descriptor) Verweise von den numerischen Dateideskriptoren auf die geöffnete Datei. Alternativ kann man lsof nutzen, um die geöffneten Dateien eines Prozesses anzuzeigen.

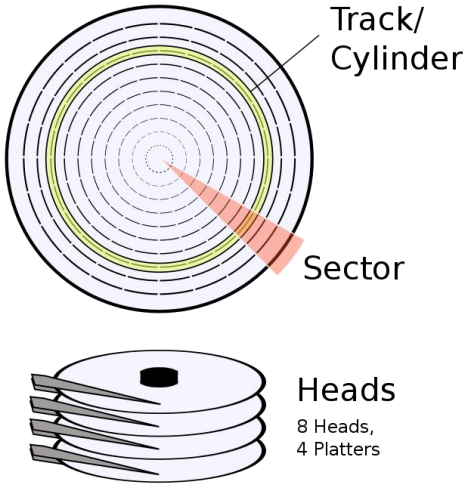
Aufbau einer konventionellen Festplatte





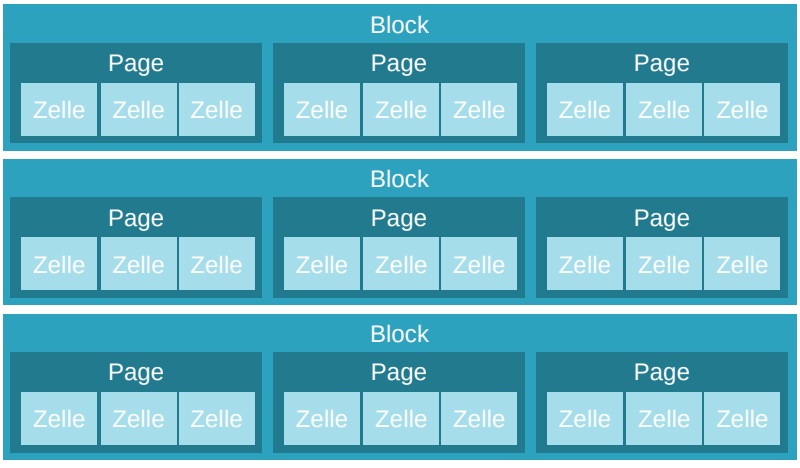
Wie bereits beim letzten Mal gesehen basiert das Prinzip einer konventionellen Festplatte auf dem Beschreiben und Auslesen magnetischer Oberflächen. Die Magnetscheiben (Platter) drehen sich, und der Schreib-Lese-Kopf (bzw. die Köpfe – einer für jede Oberfläche) bewegt sich auf einer kreisförmigen Bahn zwischen dem Scheibeninneren und dem Scheibenäußeren. Hierdurch lässt sich jeder Punkt auf den Oberflächen erreichen. Diesen Punkt zu erreichen, führt jedoch zu einer Suchzeit, die u.a. von der verwendeten Mechanik und der Art der Datenspeicherung abhängt. Deshalb ist sequentielles Lesen i.d.R. effizienter als verteiltes Lesen.

Cylinder-Head-Sector (CHS)



Die kleinste Einheit der Festplatte ist der sog. Sektor (meist 512 Byte groß) – Sektoren werden stets im Ganzen gelesen bzw. geschrieben. Die konzentrischen Bahnen einer Oberfläche bezeichnet man als Track oder Zylinder. Sektoren sind dabei kurze Abschnitte eines Zylinders auf einer bestimmten Oberfläche. Da es für jede Scheibe zwei Schreib-Lese-Köpfe gibt, nämlich einen für die Ober- und einen für die Unterseite, kann man unter Angabe von Zylinder (Cylinder), Kopf (Head) und Sektor (Sector) – also CHS – jeden Sektor einer Festplatte adressieren. Diese Art der Adressierung war über viele Jahre typisch für den Zugriff auf Festplatten. Man musste die entsprechenden Festplattenmerkmale im BIOS eintragen, um die Festplatte nutzen zu können.

Pages und Blocks

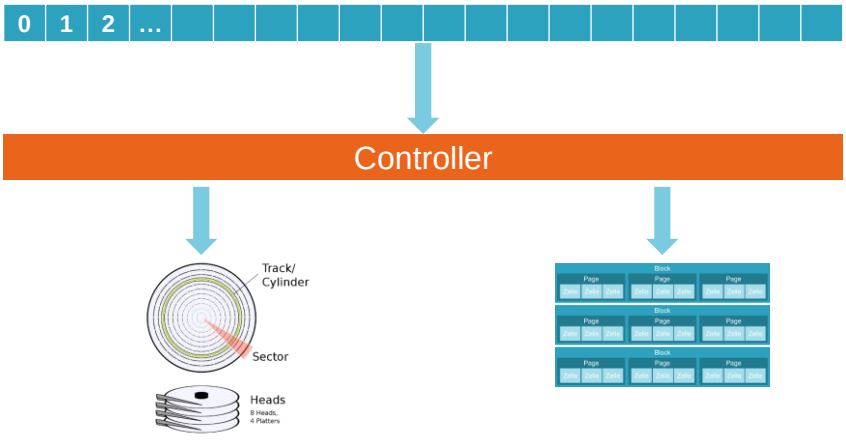


CHS eignet sich verständlicherweise nicht für die heute verbreiteten SSD-Laufwerke (aber auch nicht für optische Speichermedien, Bandlaufwerke usw.). SSDs besitzen als kleinste Einheit Speicherzellen – je nach Technologie enthalten sie 1 (SLC), 2 (MLC) oder 3 bit (TLC). Mehrere Zellen werden zu einer Page zusammengefasst; meist sind diese 4kB groß und sind die kleinste Einheit die gelesen oder geschrieben werden kann, d.h. einzelne Speicherzellen lassen sich nicht verändern, nur die gesamte Page.

128 oder 256 Pages werden wiederum zu einem Block zusammengefasst; während aber Pages sich einzeln lesen oder schreiben lassen, kann man immer nur blockweise löschen. Das Löschen ist aber wiederum die Voraussetzung für ein erneutes Schreiben.

Die Adressierung bei SSD müsste folglich auf Pages und Blocks basieren. Allerdings übernimmt das nicht das Betriebssystem, sondern der SSD-Controller selbst, der auch viele weitere Aufgaben übernimmt, z.B. das Verteilen von Zugriffen, um die begrenzte Zahl von Schreibzugriffen gleichmäßig über die gesamte SSD zu verteilen.

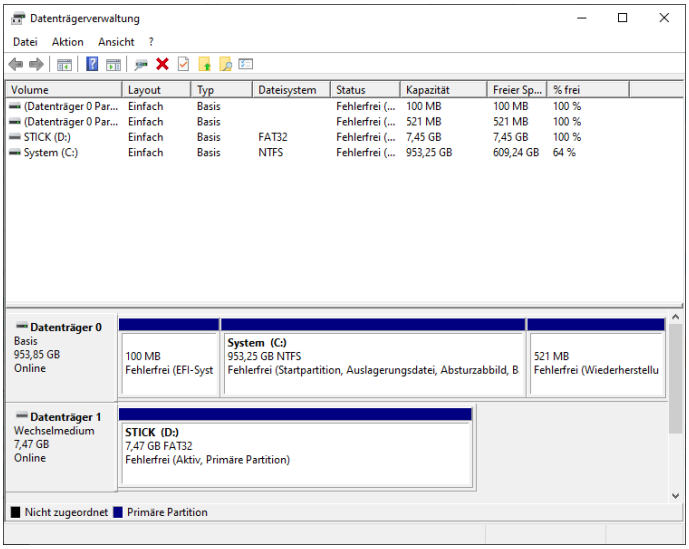
Logical Block Addressing (LBA)



Um von der hardwarespezifischen Adressierung zu abstrahieren, hat man das sog. Logical Block Addressing (LBA) eingeführt. Hierbei wird die einzelnen Blöcke der Festplatte fortlaufend durchnummeriert. Soll nun auf einen von ihnen zugegriffen werden, muss der (meist) 48bit-LBA an den Festplatten-Controller gegeben werden, der ihn dann in CHS oder einen Zugriff auf die gewünschte Page umsetzt.

Es ist also ein weiteres Beispiel, wie durch Abstraktion die Komplexität durch die vielfältigen Geräte verringert werden kann.

Datenträgerverwaltung unter Windows



Informationen über die angeschlossenen Datenträger erhält man unter Windows mit der sog. Datenträgerverwaltung. Hierüber lassen sich auch Partitionen auslesen bzw. bearbeiten (dazu später mehr).

fdisk –l unter Linux

```
root@herkules:/# fdisk -l
Disk /dev/sdc: 3.7 TiB, 4000787030016 bytes, 7814037168 sectors
Disk model: WDC WD40EFRX-68N
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disklabel type: gpt
Disk identifier: ECE3083F-4394-499E-AA2C-EA4157B16B9E

Device     Start        End    Sectors  Size Type
/dev/sdc1   2048 7814035455 7814033408  3.7T Linux RAID


Disk /dev/sda: 223.6 GiB, 240065183744 bytes, 468877312 sectors
Disk model: WDC WDS240G2G0B-
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x8004e2c0
```

Unter Linux kann man mit fdisk –l Informationen zu den Festplatten und darauf befindlichen Partitionen abrufen. Hier werden beispielsweise auch die Gesamtgröße, die Zahl der Sektoren und die Sektorengröße angezeigt.

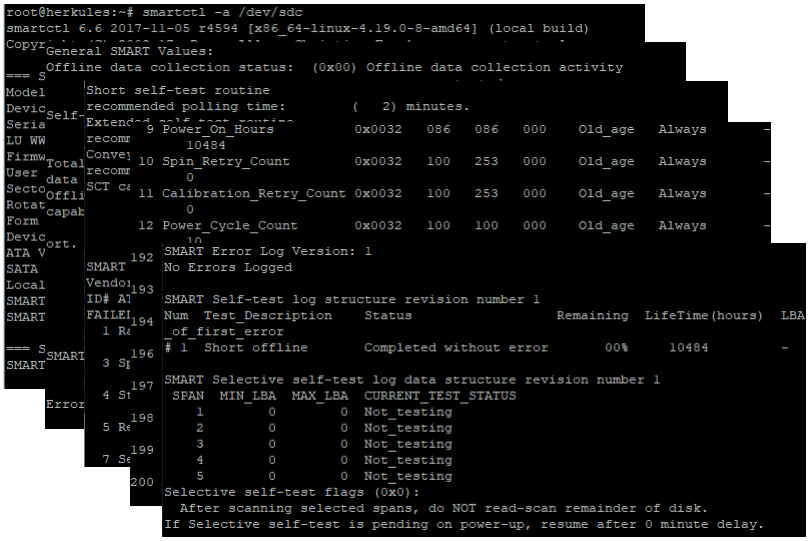
fdisk als solches erlaubt es unter anderem, Partitionen anzulegen und zu bearbeiten.

hdparm -I unter Linux

```
root@herkules:~# hdparm -I /dev/sdc

/dev/sdc:
           Logical Sector-0 offset:                  0 bytes
ATA de     device size with M = 1024*1024:     3815447 MBytes
                   *    READ_BUFFER command
                   *    NOP cmd
                   *    DOWNLOAD MICROCODE
                        *    Phy event counters
 SATA Capabil:          *    Idle-Unload when NCQ is active
Standa                  *    NCQ priority information
                        *    READ_LOG_DMA_EXT equivalent to READ_LOG_EXT
                        *    DMA Setup Auto-Activate optimization
                             Device-initiated interface power management
Config                  *    Software settings preservation
                        *    SMART Command Transport (SCT) feature set
                        *    SCT Write Same (AC2)
                        *    SCT Error Recovery Control (AC3)
     Command:           *    SCT Features Control (AC4)
                        *    SCT Data Tables (AC5)
                             unknown 206[12] (vendor specific)
                             unknown 206[13] (vendor specific)
                        *    DOWNLOAD MICROCODE DMA command
                        *    WRITE BUFFER DMA command
                        *    READ BUFFER DMA command
               Security:
                       Master password revision code = 65534
                                supported: enhanced erase
                              488min for SECURITY ERASE UNIT. 488min for ENHANCED SECURITY ERASE UNIT.
                     Logical Unit WWN Device Identifier: 50014ee2111a36bb
                              NAA             : 5
                              IEEE OUI        : 0014ee
                              Unique ID       : 2111a36bb
                     Checksum: correct
```

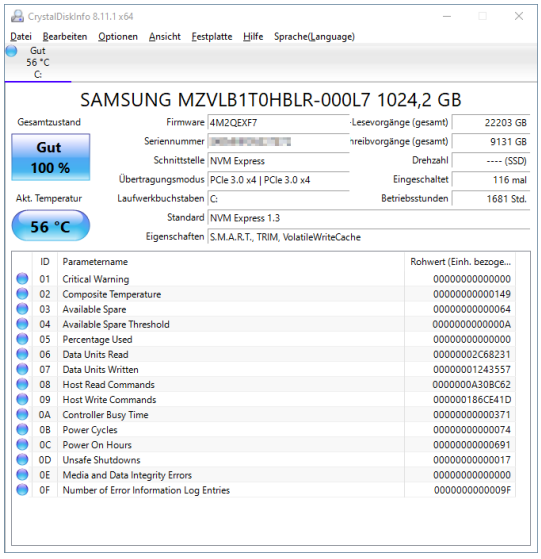
Möchte man sehr detaillierte Angaben zu der Festplatte, kann man mit hdparm –I arbeiten. Angezeigt werden dann nicht nur Angaben zu Größe, Plattengeometrie usw., sondern auch zu unterstützten Befehlen und zur Integrität.

smartctl unter Linux



Die von Festplatten erfassten „Gesundheitsdaten“ werden über SMART zur Verfügung gestellt. Hierüber sind auch kurze und lange Tests möglich. Möchte man diese anzeigen kann man mit smartctl aus dem Paket smartmontools arbeiten. smart –a zeigt dann z.B. die Testergebnisse an.

SMART Status unter Windows



Unter Windows gibt es verschiedene Tools, mit denen sich der Festplattenzustand anhand der SMART-Daten untersuchen lässt. Ein Beispiel ist das hier gezeigte CrystalDiskInfo.

Es gibt zwar mit wmic diskdrive get status auch eine in Windows integrierte Lösung; deren Aussagekraft ist jedoch sehr gering.

RAID

RAID 0: Beschleunigung ohne Redundanz

Disk 1	Disk 2
A1	A2
A3	A4
A5	A6
A7	A8

RAID 1: Spiegelung

Disk 1	Disk 2
A1	A1
A2	A2
A3	A3
A4	A4

RAID 5: Beschleunigung und Redundanz

Disk 1	Disk 2	Disk 3	Disk 4
A1	A2	A3	parA
B1	B2	parB	B3
C1	parC	C2	C3
parD	D1	D2	D3

RAID steht für Redundant Array of Independent Disks. Darunter versteht man das Zusammenfügen mehrerer Festplatten in einen RAID-Verbund, der einzelnes logisches Laufwerk repräsentiert. Dabei unterscheidet man Software- und Hardware-RAID. Bei einem Software-RAID wird der Verbund von der Software (d.h. dem Betriebssystem) verwaltet – unter Linux lässt es sich z.B. mit mdadm anlegen. Bei einem Hardware-RAID wird ein eigener RAID-Controller eingebaut, der den RAID-Verbund verwaltet; diese sind teurer, können aber Vorteile bei der Performance bringen.

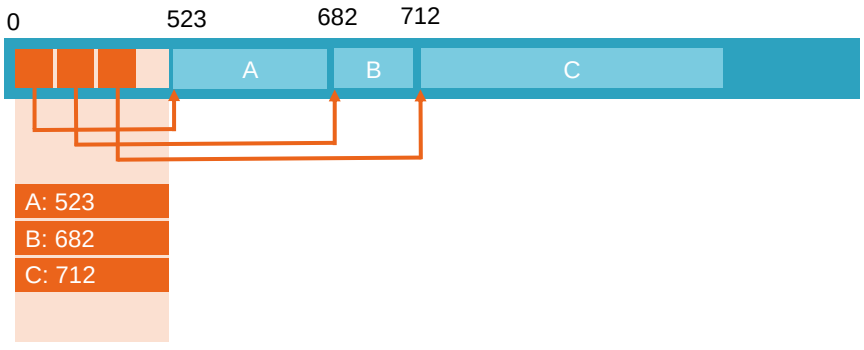
RAID ist besonders im Serverumfeld sehr verbreitet, um die Ausfallsicherheit und die Perfomance des Festplattenspeichers zu verbessern. Man unterscheidet dabei verschiedene RAID-Level; die wichtigsten unter ihnen wollen wir hier näher betrachten:

RAID 0 verteilt die Daten über mehrere Festplatten; dies kann Zugriffe beschleunigen, gerade bei konventionellen, mechanischen Festplatten, bei denen die Suchzeiten bei verteilten Zugriffen die Leistungsfähigkeit reduzieren können. Allerdings bietet ein RAID 0 keinerlei Redundanz. Der Ausfall einer Festplatte kann zu erheblichen Datenverlusten führen.

RAID 1 spiegelt die Daten zwischen den verbundenen Festplatten – die Festplatten enthalten also dieselben Daten. Eine Änderung der Daten wird immer auf alle enthaltenen Festplatten übertragen. Hierdurch schützt ein RAID 1 gegen Hardwareausfall – fällt eine Festplatte aus, kann sie durch eine andere ersetzt werden. Werden Daten aber z.B. versehentlich gelöscht, bietet ein RAID 1 keinen Schutz; das Löschen würde sich auch auf die übrigen Festplatten auswirken.

RAID 5 verwendet mehrere Festplatten und verbindet Beschleunigung durch das verteilte Speichern mit Redundanz durch Paritätsinformation, die auf den beteiligten Festplatten verteilt werden. Anhand dieser Paritätsinformationen lässt sich eine ausgefallene Festplatte rekonstruieren. Teilweise wird statt RAID 5 auch eine RAID 10 (RAID 1+0) eingesetzt, bei dem mehrere RAID 1 zu einem RAID 0 verbunden werden.

Die Idee eines Dateisystems



Ein Datenträger ist zunächst eine lineare Anordnung von adressierbaren Datenblöcken. Möchte man auf ihm Daten in Form von Dateien speichern, muss man Metadaten verwalten, die eine Zuordnung schaffen zwischen dem Dateinamen und der Anfangsadresse auf dem Datenträger. Diese Zuordnung muss selbst natürlich auch auf dem Datenträger gespeichert werden, z.B. in Form einer Tabelle.

Diese Art der Verwaltung bringt uns jedoch zurück zu Problemen, wie wir sie bereits beim Speichermanagement gesehen haben.

Aufgabe

- Welche Nachteile besitzt das gerade gezeigte Dateisystem?
- Wie könnte man das Löschen von Dateien behandeln?
- Was passiert, wenn die Zuordnungstabelle beschädigt wird? Wie könnte man das verhindern?

FAT32

- FAT: File Allocation Table
- Cluster: Zusammenfassung mehrerer Sektoren
- FAT enthält einen Eintrag pro Cluster
  (32 Bit, davon 4 Bit reserviert → ca. 270.000.000 Cluster)

➔ Jede Datei belegt mindestens ein Cluster

- Maximale Dateigröße 4GB
- 8.3-Dateinamen, längere Namen (LFN) durch VFAT
- Verkettung der Cluster durch Verweis auf Nachfolger

➔ Datei muss nicht in fortlaufenden Clustern liegen

- Löschen durch Kennzeichnung freier Cluster
- Partitionen sind Laufwerke (z.B. C:)
- Immer noch verbreitet (z.B. als exFAT für USB Sticks)

Unter Windows war lange Zeit das Dateisystem FAT das Standarddateisystem. Seine Vorläufer sind in den späten 1970er Jahren entstanden. Seit Windows 95 wird FAT32 eingesetzt. Der Name FAT geht auf ein wichtiges Element zurück, die File Allocation Table (FAT). Hierfür fasst man mehrere Festplattensektoren zu sog. Clustern (auch teilweise als Zuordnungseinheit bezeichnet) zusammen. Für jedes Cluster enthält die FAT einen Eintrag. Da 4 Bit der namensgebenden 32 Bit reserviert sind, stehen also 2^28 Cluster-Adressen zur Verfügung. Abhängig von der Sektorgröße lassen sich damit Dateisytemgrößen von 2-16TB erreichen; da jede Datei mindestens ein Cluster belegt, sind also maximal ca. 270.000.000 Dateien möglich. Die maximale Dateigröße beträgt bei FAT32 4GB.

Häufig ist man überrascht, dass FAT32 an sich keine langen Dateinamen (LFN) unterstützt. Tatsächlich erlaubt FAT32 nur 8.3-Dateinamen; die langen Dateinamen wurden durch die Erweiterung VFAT (Virtual FAT) möglich. Dabei werden aber weiterhin der vollständige Dateiname und der 8.3. Dateiname gespeichert.

Da eine Datei meist nicht nur einen Cluster erfordert, werden die Cluster einer Datei durch Verkettung zusammengefügt. Hierfür verweist jeder FAT-Eintrag auf das Nachfolgercluster. Dateien müssen hierdurch auch nicht in fortlaufenden Clustern liegen, sondern können über den Datenträger verteilt sein.

Sollen Dateien gelöscht werden, werden die zugehörigen Cluster als frei gekennzeichnet. Auch wird der erste Buchstabe des Dateinamens überschrieben. Gelöschte Dateien lassen sich daher vergleichsweise leicht wiederherstellen.

FAT32 kennt Partitionen. Windows bezeichnet diese als Laufwerke und weist ihnen einen Laufwerksbuchstaben zu (z.B. C:).

FAT ist immer noch sehr verbreitet, auch wenn unter Windows mittlerweile NTFS das Standard-Dateisystem ist. FAT (meist in der Form exFAT) findet man z.B. auf Wechseldatenträgern (USB-Sticks usw.).

NTFS

- NTFS: New Technology File System
- Journaling: Änderungen zunächst in reservierten Speicherbereich schreiben
- Berechtigungen auf Datei- und Verzeichnisebene
- File Reference: eindeutiger Verweis auf Datei/Verzeichnis
- MFT: Master File Table
- MFT enthält pro Datei einen Eintrag mit Größe, Zeitstempel usw., kleine Dateien sind eingebettet
- 255 Zeichen pro Dateiname, Sonderzeichen erlaubt
- Erlaubt transparente Kompression und Verschlüsselung
- Zusätzliche Dateien zur Fehlersicherheit

NTFS (New Technology File System) wurde erstmals 1993 in Windows NT 3.1 eingeführt. Gegenüber FAT bietet es verfeinerte Berechtigungen und beseitigt auch sonst eine Einschränkungen (z.B. die maximale Dateigröße). Ein wichtiger Unterschied liegt darin, dass es sich um ein Journaling-Dateisystem handelt. Dabei werden Änderungen zunächst in einen speziellen Speicherbereich geschrieben, um Fehler besser kompensieren zu können.
Jede Datei (und auch jedes Verzeichnis) wird durch ein Dateinummer eindeutig referenziert; sie ist zugleich Index des Herzstücks von NTFS, der sog. Master File Table (MFT). Dabei handelt es sich um eine spezielle Datei, die einen Index aller Dateien und Verzeichnisse enthält, wobei jeder Eintrag eine fest Größe hat und neben Dateigröße, Zeitstempel usw. sogar für kleine Dateien den Dateiinhalt enthält, sonst sind es Verweise auf die Cluster.
Für Dateinamen sind 255 Zeichen erlaubt, auch Sonderzeichen.
NTFS erlaubt auch die transparente Kompression und Verschlüsselung von Dateien, d.h. beim Öffnen einer Datei wird diese unbemerkt dekomprimiert bzw. entschlüsselt.
Für NTFS werden eine Reihe von Dateien auf dem Datenträger abgelegt: Neben dem MFT wird auch eine Kopie davon gespeichert, um robust gegenüber Fehlern zu sein. Ein Log File protokolliert Änderungen am Dateisystem. Ein Boot File enthält die Anweisungen, die anfänglich ausgeführt werden sollen. Auch gibt es z.B. ein Bitmap, das die Belegung aller Cluster wiedergibt und ein Bad Cluster File, in dem defekte Cluster erfasst werden.

ext4

- ext4: Fourth extended filesystem
- Journaling-Dateisystem
- 48bit-Blocknummern
- Basierend auf Inodes
- Extents für zusammenhängende Blöcke
- Superblock als Anfangspunkt

ext4 (fourth extended filesystem) wird unter Linux verwendet; wie NTFS ist es ein Journaling-Dateisystem. Es verwendet 48bit-Blocknummern (der Vorgänger ext3 konnte mit 32bit nur 16TB addressieren).

Die ext-Dateisystem basieren auf sog. Inodes. Alle benannten Objekte besitzen einen solchen Inode, der die Art der Datei, Eigentümer (UID), Gruppe (GID), die Zugriffsrechte, Zeitstempel, Dateigröße, den sog. Linkzähler (d.h. die Anzahl der Verweise auf diesen Inode) sowie Verweise auf die Blöcke, in denen die Daten gespeichert sind, enthält. Die Inodes selbst enthalten keinen Dateinamen – dieser ist in der sog. Inode-Liste des Verzeichnisses abgelegt. Die Inode-Nummer kann man entweder mit ls –i oder mit stat anzeigen.

Eine Besonderheit von ext4 ist die Verwendung sog. Extents: Da bei großen Dateien die Liste der einzelnen Datenblöcke sehr groß werden kann, erlauben es Extents, im Inode die Nummer der ersten und des letzten Blocks eines zusammenhängenden Bereichs von Blöcken zu erfassen.

In jeder Partition ist neben der Liste der Inodes und den eigentlichen Datenblöcken vor allem der sog. Superblock angelegt. Soll eine Datei geöffnet werden, beginnt man bei diesem Superblock und folgt seinen Verweisen bis zum Verzeichnis, in dem die Datei liegt, um von dort auf die einzelnen Datenblöcke zugreifen zu können.

Weitere Dateisysteme

- ISO9660
- UDF
- APFS
- btrfs
- u.v.m.

Neben den genannten Dateisystemen gibt es eine Vielzahl weiterer Dateisysteme, z.B. ISO9660 für CD-ROMs und UDF für DVDs und Blu-Ray. APFS (Apple File System) ist der Nachfolger des dort über viele Jahre eingesetzten HFS (Hierarchical File System). btrfs (Btree File System) wird teilweise als Nachfolger von ext4 unter Linux gesehen und bringt interessante Features (z.B. Snapshots) mit sich.

Typische Verwaltungsaufgaben

- Partitionieren: Unterteilen des Datenträgers
- Formatieren: Erstellen des Dateisystems
- Einhängen/Mounten: Partitionen verwendbar machen
- Prüfen: Suchen und Beseitigen von Fehlern
- Überwachung: Auditing und Belegung

Rund um Dateisysteme gibt es eine Reihe wiederkehrender Verwaltungsaufgaben:

Am Anfang steht das sog. Partitionieren, bei dem ein Datenträger in eine oder mehrere Partitionen unterteilt wird. Jede Partition könnte dabei ein unterschiedliches Dateisystem erhalten. Unter Windows ist das Partitionieren mit der Datenträgerverwaltung, dem Tool fdisk oder auch mit Drittanbieterlösungen möglich. Unter Linux stehen z.B. parted oder fdisk zur Verfügung. Plattformübergreifend kann z.B. das grafische Werkzeug GParted verwendet werden. Unter Linux ist außerdem der Einsatz von LVM (Logical Volume Manager) verbreitet; LVM erlaubt es Partitionen zu erzeugen, die sich dynamisch anpassen lassen und sich über mehrere Festplatten erstrecken.

Das Formatieren hat zum Ziel, das Dateisystem auf einer Partition zu erstellen. Unter Windows kann dies ebenfalls mit der Datenträgerverwaltung oder dem Windows Explorer erfolgen. Linux bietet mit den mkfs-Befehlen die Möglichkeit, eine Partition zu formatieren, z.B. mit mkfs.ext4 oder mkfs.exfat.

Das Einhängen oder Mounten betrifft nicht alle Betriebssysteme, unter Linux ist es notwendig. Dateisysteme müssen gemountet werden. Die Grundidee besteht darin, Geräte in /dev so einzuhängen, dass sie z.B. in /mnt als Verzeichnis zur Verfügung stehen. Dauerhaft kann das z.B. über /etc/fstab erfolgen. Dynamisch steht hierfür der mount-Befehl zur Verfügung.

Die Betriebssysteme bieten verschiedenste Möglichkeiten, Fehler im Dateisystem zu erkennen. Dies kann z.B. passieren, wenn die Verwaltungsstrukturen durch Abstürze oder

auch Bitfehler inkonsistent werden. Unter Windows gibt es die über den Explorer erreichbare Fehlerüberprüfung oder das Kommandozeilenwerkzeug chkdsk. Unter Linux erreicht man dies z.B. mit fsck, was bei eingehängten Partitionen aber nicht funktioniert. Diese Prüfung lässt sich aber beim Neustart erzwingen oder sich per Live-System durchführen.

Bei der Überwachung von Datenträgern gibt es vielfältige Ansätze, z.B. das Auditing, bei dem Zugriffe auf Dateien protokolliert werden. Alltäglich sind jedoch Fragen der Belegung von Partitionen, z.B. unter Linux mit df, das den freien Speicher anzeigt, oder mit du, das den Speicherbedarf eines Verzeichnisses anzeigt. Im Rahmen der Überwachung war früher auch die Fragmentierung ein wichtiger Aspekt – bei Fragmentierung werden Dateien nicht in benachbarten Blöcken gespeichert, wodurch der Zugriff verlangsamt wird. Bei SSD-Festplatten ist dies jedoch unwichtig geworden.

NFS und SMB

- NFS: Network File System

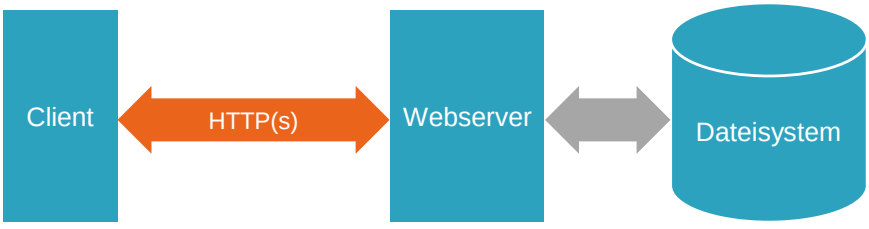
➔ Dateizugriff über Netzwerke (UDP oder TCP)

- SMB: Server Message Block

➔ Dateizugriff über Netzwerke, aber auch Druck usw.

NFS (Network File System) und SMB (Server Message Block) erlauben beide den Zugriff auf Dateien über eine Netzwerkverbindung. NFS ist vor allem im Unix-Umfeld anzutreffen, SMB vor allem unter Windows. Unter Linux ist das Paket Samba sehr populär, da es Windows-Rechnern via SMB die Möglichkeit gibt, auf unter Linux gespeicherte Dateien über das Netzwerk zuzugreifen, z.B. in der Art eines Dateiservers.

WebDAV



WebDAV (Web-based Distributed Authoring and Versioning) ist kein eigenes Dateisystem, sondern ein Netzwerkprotokoll, über das Dateien über das Netzwerk (speziell auch das Internet) zur Verfügung gestellt werden; hierfür wird das Protokoll HTTP genutzt, üblicherweise auch verschlüsselt.

Ein Webserver übernimmt dabei die Aufgabe, die Anfragen über HTTP entgegenzunehmen, die Dateien lokal im Dateisystem zu öffnen und per HTTP bereitzustellen.

Zusammenfassung

- Das Betriebssystem ermöglicht Anwendungen den Zugriff auf das Dateisystem.
- Das Dateiformat beschreibt, wie Dateien aufgebaut und codiert sind; dies ist keine Aufgabe des Betriebssystems.
- Das Betriebssystem verwaltet für jeden Prozess die Datei-Handles im Prozesskontext.
- Konventionelle Festplatten adressieren durch CHS, SSD durch Pages und Blocks; verallgemeinert wird dies durch LBA.
- RAID fasst mehrere Festplatten zu einem logischen Laufwerk zusammen und bietet Redundanz.
- FAT32 bzw. exFAT, NTFS und ext4 sind verbreitete Dateisysteme. Es gibt noch viele weitere.
- Partitionierung, Formatierung, Einhängen, Prüfung und Überwachung sind typische Verwaltungsaufgaben.
- NFS, SMB und WebDAV erlauben den Zugriffe auf Dateien über Netzwerke.

Aufgaben

1. Welche Arten von Dateien gibt es?
2. Welche Systemaufrufe gibt es üblicherweise für Dateien und Verzeichnisse?
3. Sie wollen eine externe Festplatte unter Linux verwenden, welche Schritte sind notwendig?
4. Was geschieht, wenn Sie in einem C-Programm fopen einsetzen?
5. Was ist der Unterschied zwischen fopen und open?
6. Was versteht man unter Datei-Handle?
7. Sie geben nacheinander chown a:b x und chmod 740 x ein. Was darf der Benutzer a mit x machen? Was müsste erfüllt sein, damit ein Nutzer c den Inhalt von x lesen kann? Was dürfen andere Benutzer?
8. Warum kann sich die Dateigröße von der Größe der Datei auf dem Datenträger unterscheiden?

Aufgaben

9. Wie können Sie unter Windows bzw. unter Linux herausfinden, welche Datei-Handles einem Prozess zugeordnet sind?
10. Was versteht man unter einem Inode? Wofür wird er eingesetzt?
11. Wo wird der Dateiname in ext4 gespeichert?
12. Nennen Sie Unterschiede von FAT32 und NTFS?
13. Worin unterscheiden sich FAT und MFT?
14. Wofür setzt man WebDAV ein? Was braucht man hierfür?
15. Auf einem Linux-Server sei Samba installiert und eingerichtet. Wofür lässt sich Samba einsetzen?