

Vorlesung Betriebssysteme I

Thema 3: Dateisystem

Robert Baumgartl

28. Oktober 2015

Wozu ein Dateisystem?

Aufgaben von Dateisystemen:

- ▶ Verwirklichung sinnvoller Abstraktionen zum Strukturieren der abzulegenden Information (Datei, Verzeichnis)
- ▶ Management des Freispeichers

Herausforderungen:

- ▶ Langsamkeit der Medien, da meist mechanische Operationen notwendig
- ▶ Umfang der Informationen
- ▶ Fehlertoleranz

Heterogenität:

- ▶ magnetische Massenspeichermedien
- ▶ optische
- ▶ elektrische

Aufbau einer Festplatte

- ▶ Stapel von rotierenden Magnetplatten, konstante Rotationsgeschwindigkeit (CAV – *Constant Angular Velocity*)
- ▶ Rotationsgeschwindigkeit ca. $5400 - 15000 \text{ min}^{-1}$
- ▶ 2-16 Platten
- ▶ konzentrische Spuren (*Tracks*), ca. 10.000 pro Oberfläche
- ▶ übereinanderliegende Spuren bilden einen sog. *Zylinder*
- ▶ kleinste ansprechbare Einheit: physischer Block („Sektor“; 512 Byte), z.B. 150-300 Sektoren pro Spur
- ▶ 1 Schreib-Lesekopf pro Plattenoberfläche, radiale Bewegung aller Köpfe gemeinsam

Aufbau einer Festplatte

- ▶ historisch: Adressierung eines Sektors über {Zylinder, Kopf, Sektor}-Tripel (*Cylinder, Head, Sector* – *CHS*)
- ▶ heute: Logical Block Addressing (LBA), einfache Durchnumerierung aller Blöcke
- ▶ physisches Layout vor Nutzer verborgen: Abbildung Logischer Blocknummern auf Physische Blocknummern (LBN → PBN) durch Festplattenelektronik

Schematischer Aufbau einer Festplatte

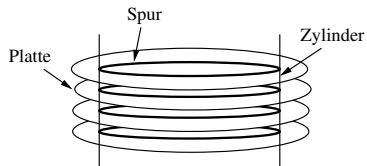


Abbildung: Seitenansicht

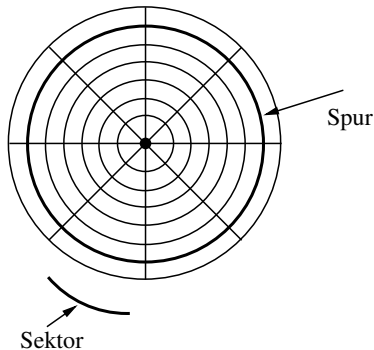


Abbildung: Draufsicht

Optische Medien: Compact Disc (CD)

Grundlage: *Red Book Standard* von Philips und Sony

<i>Standard</i>	<i>Kürzel</i>	<i>Bemerkungen</i>
Red Book	CD-DA	Audio-CD
Yellow Book	CD-ROM	
Green Book	CD-I	CD Interactive
Blue Book	CD-Extra	Audio+Daten-CD
Orange Book	CD-R[W]	Recordable CDs
White Book	Video-CD	
-	Photo CD	

Tabelle: Übersicht relevanter CD-Standards

Fakten zur CD

- ▶ Abtastung mittels eines Infrarotlasers, unterschiedliches Reflexionsverhalten von *Pits* und *Lands*
- ▶ eine (!) Spur (Breite: $0.5\ \mu\text{m}$), von innen nach außen gelesen, Abstand $1.6\ \mu\text{m}$
- ▶ konstante Speicherdichte \rightarrow *Constant Linear Velocity* (CLV) \rightarrow variable Umdrehungsgeschwindigkeit je nach Position auf Medium
- ▶ 2 (CD-Audio) bzw. 3 unabhängige Fehlerkorrektur-Schichten:
 - ▶ Symbol: 8 Bit Payload pro 14 Bit-Symbol (Eight-to-Fourteen Modulation, EFM)
 - ▶ Frame: SYNC + CTL + 32 Symbole (\sum 588 Bit, davon 24 Byte Nutzlast)
 - ▶ Sektor: 98 Frames á 24 Byte Nutzlast = 2352 Byte Länge
- ▶ kleinste adressierbare Einheit: Sektor (CD-ROM)

Welche Dateisysteme gibt es?

<i>BS</i>	<i>Dateisystem</i>
MS-DOS	FAT12, FAT16
Windows 9x	VFAT
Windows NT... Vista	NTFS
MacOS	HFS
Linux	ext2fs, ext3fs
OS/2	HPFS

Tabelle: Betriebssystemspezifische Dateisysteme

Gute BS lesen auch die Dateisysteme der „Konkurrenz“, sofern diese offen liegen.

Zusätzlich gibt es BS-übergreifende Dateisysteme, z. B. ISO9660 (Dateisystem der CD-ROM) oder CIFS.

Grundlegende Abstraktionen: Datei

Datei = „Ansammlung“ von Nutzdaten + Attribute

Beispiele typischer Attribute:

- ▶ *Schutz*: Wer darf welche Operation mit Datei ausführen?
- ▶ *Eigentümer* der Datei
- ▶ Beschränkungen der erlaubten Operationen (*Read-Only*)
- ▶ Beschränkungen der Sichtbarkeit der Datei (*Hidden Flag*, `.dateiname`)
- ▶ Dateiname
- ▶ Zeitstempel (letzter Zugriff, letzte Änderung, Kreation)
- ▶ Größe der Datei
- ▶ Stellung des Dateipositionszeigers

→ `stat`-Kommando unter Linux

Typen von Dateien

Unterscheidung von Dateitypen

- ▶ durch Attribute (Dateinamen, ASCII/binary-Flag),
- ▶ durch Dateinamen,
- ▶ durch *Magic Word*.

Ein *Magic Word* ist eine charakteristische Bytesequenz am Beginn der Datei, anhand derer ihr Typ identifiziert werden kann.

<i>Sequenz</i>	<i>Bedeutung</i>
JFIF	JPEG File Interchange Format
GIF89a	Graphics Interchange Format (V.89a)
#!/bin/bash	Shell-Skript
ELF	Executable and Linkable Format

Tabelle: Beispiele für Magic Words

Beispiele: JFIF, PDF

```
robge@ilpro121:~/txt/job/htw/bs1$ hexdump -C pic/tux2.jpg
00000000  ff d8 ff e0 00 10 4a 46  49 46 00 01 01 01 00 48  |ÿøÿà..JFIF.....H|
00000010  00 48 00 00 ff db 00 43  00 01 01 01 01 01 01 01  |.H..ÿÛ.C.....|
00000020  01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01  |.....|
*
00000050  01 01 01 01 01 01 01 01  01 ff db 00 43 01 01 01  |.....ÿÛ.C...|
00000060  01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01  |.....|
...
robge@ilpro121:~/txt/job/htw/bs1$ hexdump -C select-pages.pdf
00000000  25 50 44 46 2d 31 2e 34  0a 38 20 30 20 6f 62 6a  |%PDF-1.4.8 0 obj|
00000010  20 3c 3c 0a 2f 4c 65 6e  67 74 68 20 31 32 35 20  | <<./Length 125 |
00000020  20 20 20 20 20 20 0a 2f  46 69 6c 74 65 72 20 2f  |                |./Filter /|
00000030  46 6c 61 74 65 44 65 63  6f 64 65 0a 3e 3e 0a 73  |FlateDecode.>>.s|
00000040  74 72 65 61 6d 0a 78 da  8d 8e 31 0a c3 30 0c 45  |tream.xÚ..1.Ã0.E|
00000050  77 9f e2 5f c0 8a 24 47  ae bc 17 4a c6 9c a1 43  |w.â_À.$G...JË.¡C|
...
```

Dateinamenskonventionen

Jedes Dateisystem hat Regeln zum Aufbau eines Dateinamens:

FAT (File Allocation Table) – MS-DOS

- ▶ „berüchtigte“ 8.3-Konvention
- ▶ .COM, .EXE – ausführbare Dateien
- ▶ .BAT – Batchdateien (analog zu Shellskripten)

VFAT – ab Windows 95

- ▶ bis 255 Zeichen lang
- ▶ Unicode-kodiert
- ▶ keine Unterscheidung von Groß- und Kleinschreibung

Unix

- ▶ unterscheidet Groß- und Kleinschreibung
- ▶ `name.ext` eigentlich unüblich, aber trotzdem genutzt

Wurzelverzeichnis

VFAT: C:\, D:\, ..., Z:\


Unix: /

VMS: [000000]

<i>Betriebssystem</i>	<i>Trennsymbol</i>
Windows	\
Unix	/
Multics	>
VMS	:

Tabelle: Trennsymbole für Pfadangaben

(Abstrakte) Operationen über Dateien

<i>Operation</i>	Bemerkungen
Open	Vor eigentlichem Zugriff erforderlich
Read	(sequentiell)
Write	(sequentiell)
Seek	Verstellen des Dateipositionszeigers 
Close	nicht vergessen
Append	Anfügen von Daten an Dateiende
Truncate	Datei verkürzen (z. B. auf 0)
Rename	Datei umbenennen

- ▶ Dateien müssen vor Zugriff *geöffnet* werden.
- ▶ Lese- und Schreiboperationen nutzen *gemeinsam* den **Dateipositionszeiger**
- ▶ Dieser steht initial auf Position 0 und kann mittels *Seek*-Operation beliebig versetzt werden.
- ▶ Lesen und Schreiben versetzt den Dateipositionszeiger ebenfalls.
- ▶ Wird beim Zugriff das Ende der Datei erreicht, wird i. A. **EOF** (End of File) gemeldet
- ▶ Um mit dem Inhalt einer Datei zu arbeiten, muss diese in den Hauptspeicher transferiert oder eingeblendet werden.

Dateifunktionen der C-Bibliothek

<i>Funktion</i>	<i>Semantik</i>
<code>fopen()</code>	Eröffnen
<code>fclose()</code>	Schließen
<code>fread()</code>	Lesen
<code>fwrite()</code>	Schreiben
<code>fprintf()</code>	(formatiertes) Schreiben
<code>feof()</code>	Test auf Dateiende
<code>ferror()</code>	Test auf Fehler
<code>fseek()</code>	Versetzen des Positionszeigers
<code>ftell()</code>	Abfrage desselbigen
<code>flock()</code>	<i>Sperren</i> einer Datei

mehr: `man 3 stdio`

- ▶ gepuffert
- ▶ definiert in `stdio.h`
- ▶ geöffnete Datei wird durch `FILE*` identifiziert
- ▶ `stdin`, `stdout`, `stderr`
- ▶ betriebssystemeunabhängig (portabel)
- ▶ standardisiert nach ANSI C3.159-1989

Systemrufe zur Dateiarbeit (Unix)

<i>Operation</i>	<i>Semantik</i>
<code>open()</code>	Eröffnen der Datei
<code>read()</code>	Leseoperation
<code>write()</code>	Schreiboperation
<code>lseek()</code>	Verstellen des Dateipositionszeigers
<code>close()</code>	Schließen der Datei
<code>link()</code>	Verweis (<i>Hard Link</i>) auf Datei anlegen
<code>rename()</code>	Datei umbenennen
<code>mmap()</code>	Datei in Hauptspeicher einblenden

- ▶ definiert in `<unistd.h>`
- ▶ standardisiert in POSIX
- ▶ portabel nur in Unix-Betriebssystemen
- ▶ geöffnete Datei wird durch *Dateideskriptor* (integer) identifiziert

Funktionen zur Dateiarbeit (Win32) – kleine Auswahl

<i>Operation</i>	<i>Semantik</i>
CreateFile()	Öffnen (kein Witz!)
ReadFile()	Lesen
WriteFile()	Schreiben
SetFilePointer()	Dateipositionszeiger setzen
CloseFile()	Schließen der Datei
CreateHardLink()	(<i>Hard Link</i>) anlegen
MoveFile()	Datei umbenennen (u. a.)
CreateFileMapping()	Datei in Hauptspeicher einblenden

Systemrufe zur Dateiarbeit (Windows)

- ▶ MSDN listet 114 Funktionen zur Arbeit mit Dateien
- ▶ Identifikation geöffneter Objekte mit *Handles*

Beispiel:

```
HANDLE WINAPI CreateFile(  
    LPCTSTR    lpFileName,  
    DWORD      dwDesiredAccess,  
    DWORD      dwShareMode,  
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,  
    DWORD      dwCreationDisposition,  
    DWORD      dwFlagsAndAttributes,  
    HANDLE      hTemplateFile  
);
```

Einige typische Datei-Kommandos in Unix

<i>Kommando</i>	Semantik
cp	Kopieren (<i>copy</i>)
mv	Bewegen (<i>move</i>)
rm	Löschen (<i>remove</i>)
ln	Verweis anlegen (<i>link</i>)
chmod	Ändern der Zugriffsrechte (<i>change mode</i>)
chown	Ändern des Eigentümers (<i>change owner</i>)
dd	Umleitung von Strömen
shred	sicheres Löschen
stat	Anzeige der Dateiattribute

Kommandos über Massenspeicher und Dateisystem

<i>Kommando</i>	<i>Zweck</i>
du	Schätzen des Speicherbedarfs eines Verzeichnisses
df	Anzeige Belegungszustand
fdisk	Partitionierung
mount	Montieren des Datenträgers
mkfs	Anlegen eines Dateisystems
fsck	Prüfen (und Reparieren) der Integrität des Dateisystems
hdparm	Detailinformationen zum Massenspeicher

Verzeichnisse („Ordner“)

- Organisation der Dateien auf Massenspeicher
- üblich: Hierarchie von Verzeichnissen
- bevorzugte Datenstruktur: Baum, gerichteter Graph

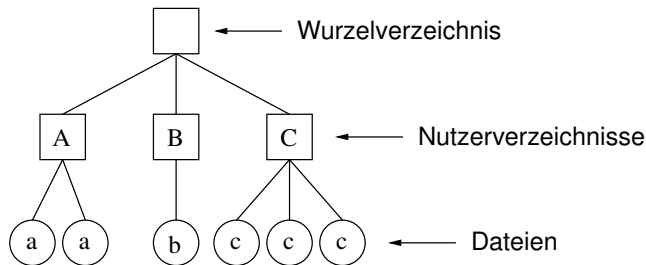


Abbildung: Zweistufiges Dateisystem

Hierarchien von Verzeichnissen

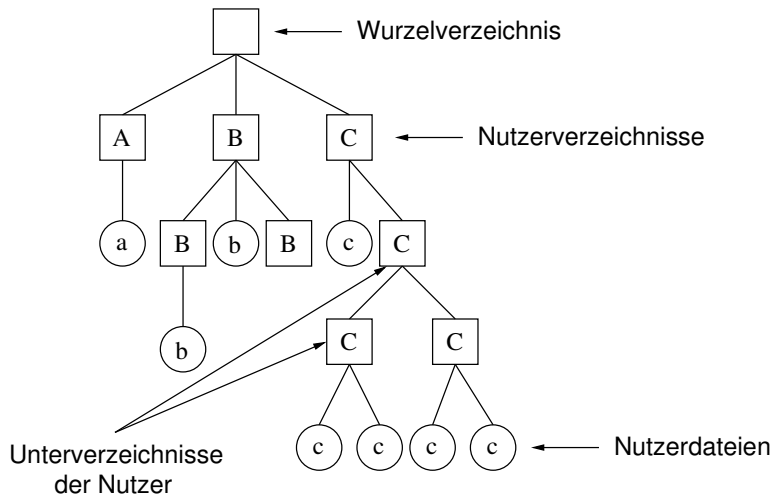


Abbildung: Typisches hierarchisches Dateisystem

Unix-Systemrufe über Verzeichnissen

`mkdir()` Anlegen eines neuen Verzeichnisses

`rmdir()` Löschen eines Verzeichnisses

`opendir()` Eröffnen

`closedir()` Schließen

`readdir()` Sequentielles Lesen der Einträge eines V.

`scandir()` Gezieltes Suchen von Einträgen innerhalb eines V.

`rewinddir()` Zurückstellen des Eintragszeigers

`symlink()` Anlegen eines *Soft Link*

Verweise (Links)

- ▶ zusätzliche Verweise auf Verzeichniseinträge
- ▶ Sinn: Vermeidung von Dateikopien, Vereinfachung der Aktualisierung, Erhöhung der Flexibilität
- ▶ UNIX: 2 Typen – *Soft Links*, *Hard Links*
- ▶ Systemrufe `link()`, `symlink()`
- ▶ Kommando `ln` zum Anlegen

Verweise auf Dateien und Verzeichnisse

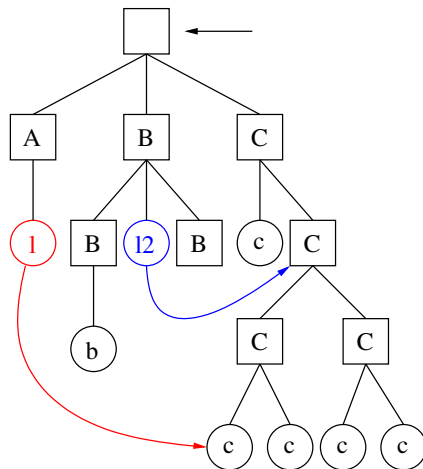


Abbildung: Beispiele für Verweis auf Datei (rot) und Verzeichnis (blau)

Abstraktion zur Beschränkung des Zugriffs
Wozu?

- ▶ Schaden durch
 - ▶ unkundige Nutzer,
 - ▶ bösartige Nutzer und
 - ▶ fehlerhafte Software

zu minimieren

Allgemeines Modell: **Zugriffsmatrix**

- ▶ Spalten: (passive) Objekte, z. B. Dateien, die Zugriffsbeschränkungen unterliegen
- ▶ Zeilen: (aktive) Subjekte, z. B. Nutzer oder Prozesse, deren Zugriff beschränkt werden soll
- ▶ Inhalt der Elemente: erlaubte Operationen, gewährte Rechte

Beispiel zur Zugriffsmatrix

	Datei 1	Datei 2	Datei 3	Datei 4
Nutzer A	Own/R/W		Own/R/W	
Nutzer B	R	Own/R/W	W	R
Nutzer C	R/W	R		Own/R/W

- ▶ Zugriffsmatrix i. a. spärlich besetzt
- ▶ → zwei Wege der Dekomposition

Access Control List (ACL)

- ▶ Dekomposition der Zugriffsmatrix nach Objekten
- ▶ für jedes Objekt wird gespeichert, welches Subjekt welche Operation mit ihm ausführen darf
- ▶ im Beispiel:
 - ▶ Datei 1: $A(\text{OWN/R/W}), B(\text{R}), C(\text{R/W})$
 - ▶ Datei 2: $B(\text{OWN/R/W}), C(\text{R})$
 - ▶ Datei 3: $A(\text{OWN/R/W}), B(\text{W})$
 - ▶ Datei 4: $B(\text{R}), C(\text{OWN/R/W})$

Capability List

- ▶ Dekomposition der Zugriffsmatrix nach Subjekten
- ▶ für jedes Subjekt wird gespeichert, auf welche Objekte es wie zugreifen darf

- ▶ Read-Only Flag im MS-DOS
- ▶ `rwxxrwxrwx`-Abstraktion im klassischen UNIX
- ▶ `rwlidka`-Rechte im Andrew File System (AFS); (für Verzeichnisse): read, write, lookup, insert, delete, lock, administer

Zugriffsrechte in Unix

- ▶ jede Datei hat 3 Rechte: Lesen, Schreiben, Ausführen
- ▶ Rechte werden für 3 Kategorien von Nutzern vergeben: den Eigentümer, die Gruppe, alle anderen Nutzer des Systems
- ▶ \Rightarrow 3x3 Bits, die gesetzt oder gelöscht sein können
- ▶ Ausführungsrecht für Verzeichnis: man darf *hineinwechseln*
- ▶ Änderung mittels `chmod`-Kommando

Beispiel:

```
~> chmod u+rwX g+r-wX o-rwX foo.sh
~> ls -l foo.sh
-rwxr----- 1 robge robge 4 2008-10-28 10:26 foo.sh
```

Ausgabe des Kommandos ls -l

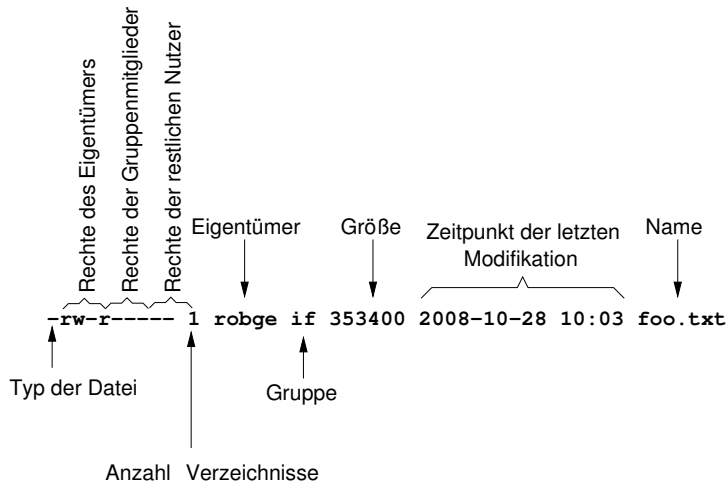


Abbildung: Ausgebenene Informationen bei `ls -l`

Zusammenfassung: Was haben wir gelernt?

- ▶ Was sind *Datei* und *Verzeichnis*?
- ▶ Dateityp, Namenskonventionen, Pfadsymbole
- ▶ Was versteht man unter Links?
- ▶ typische Kommandos, C-Funktionen und Systemrufe zur Dateiarbeit
- ▶ Wie werden Zugriffsrestriktionen für Dateien realisiert?