Grundlagen der Betriebssysteme

Praktische Einführung mit Virtualisierung

Stefan Bosse

Universität Koblenz - FB Informatik

Datein und Dateisysteme

1. Die Datei und ihr Speicherabbild, Struktur und die operationale Schnittstelle
2. Das Dateisystem als Abstraktionsmodell von Masenspeicherns und als Organisationsstruktur
3. Abbildung von logischen Dateisystemstrukturen auf physische Datenträger

> Dateisysteme erlauben die persistente, d.h. dauerhafte Ablage von Anwender und Systemdaten sowie ausführbaren Programmen. Wir betrachten zuerst die logische Organisation abgelegter Daten und die für den Zugriff benötigte Dateisystemschnittstelle.

- Dem folgen einführende Überlegungen zu Dateisystemimplementierungen und Details zu den Dateisystemen UFS, FAT, NTFS und ZFS. Dies wird ergänzt durch Betrachtungen zu Netzwerkdateisystemen und ausgewählten Dateisystemtechnologien, wie protokollierende Dateisysteme, Schattenkopien und Disk Scheduling.

Das Speichermodell eines Rechners

- Wir haben in einem Rechner i.A. verschiedene Schichten von Speichern die hierarchisch aufgebaut sind:

1. Prozessorregister
2. Cache

Primärspeicher

3. Hauptspeicher ⇒ **Kurzzeitspeicher**

Sekundärspeicher

4. Festplatten, CDROM, USB Flash EEPROM usw ⇒ **Langzeitspeicher**

Die Datei

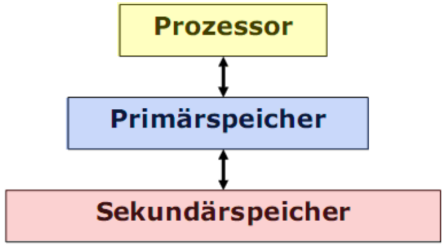


Abb. 1. Der Speicher eines Rechners ist unterteilt in Primärspeicher und Sekundärspeicher. In der Speicherhierarchie steht der Sekundärspeicher weiter vom Prozessor als Primärspeicher.

Die Datei

Eine Datei entspricht zunächst einem Datenkontainer mit einem linearen Speichermodell (wie ein Array).

- Die Datei ist in Speicherzellen unterteilt, i.A. ein Byte
- Die Speicherzellen werden indiziert, i.A. hat das erste Byte den Index Null, und allgemein $I=\{0,1,..,L-1\}$, mit L als Länge der Datei in Bytes.
- Die Bedeutung des Speicherinhalts und deren Struktur ist nicht nägher beschrieben.

Im Gegensatz zum RAM Speichermodell - Random Acesss ! - ist ein Zugriff in wahlloser Reihenfolge nicht "gewünscht" und i.A. ineffizient, und wird daher auch nicht direkt unetrstützt, d.h. es wird ein sequenzielles geordnetes oeprationales Zugriffsmodell bereitgestellt.

Die Datei

- Das sequenzille Zugriffsmodell wird über einen Dateizeiger realisiert.
  - Dieser wird aber nicht direkt verändert, sondern impliziz beim Lesen oder Schreiben einer Datei.
  - Der Zeiger kann aber neu gesetzt werden (typischerweise auf Anfang oder Ende eine Datei)
- Es sind folgende Operation auf Dateien möglich:

1. Lesen (*read*, ganz, teilweise, oder zellenweise) bis zum Ende (EOF, gegebn durch Länge).
   - Jedes Lesen setzt den Dateizeiger weiter.
2. Schreiben (*write*), i.A. inkrementell, d.h. das Ende einer Datei (Länge) kann verschoben (vergrößert) werden.
   - Jedes Schreiben setzt den Dateizeiger weiter.
3. Neupositionierung des Zeigers (*seek*)

Die Datei

- Diesen Basissatz kann man erweitern zu einer operationellen Schnittstelle (Programmier API):

1. Create und Delete
   - Erzeugen einer leeren Datei.
   - Löschen einer vorhandenen Datei.
2. Open und Close
   - Öffnen einer vorhandene Datei. Ein Dateiindex wird angelegt und positioniert. +Schließen der Datei. Der entsprechende Dateiindex wird gelöscht.
3. Write und Read
   - Schreiben in eine Datei ab einem Dateiindex.
   - Lesen aus einer Datei ab einem Dateiindex.
4. Seek
   - Positionieren des Dateiindexes.

Die Datei

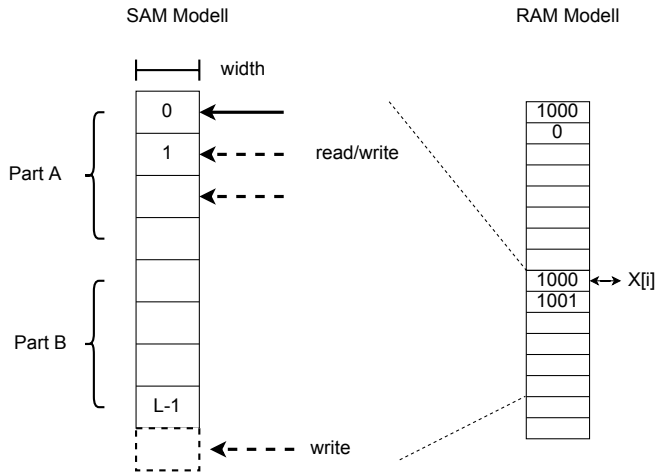
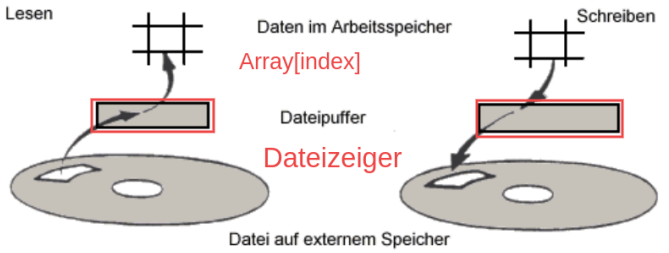


Abb. 2. Sequenzieller Zugriff (SAM) auf Dateien mit einem linearen Speichermodell. Die lineare Folge von Werten kann auch stukturiert sein, wie z.B. bei Programmdateien die einzlnen Segmente - aber betrifft die Operationen nicht! Struktur komtm später. Zum Vergleich rechts das RAM Modell mit Arrays.

Dateipufferung

- Teile einer Datei oder die ganze Datei werden vom SAM in das RAM Modell durch Leseoperationen abgebildet
  - Der Zugriff über Puffer ermöglich randomisiertes Lesen (und Schreiben) eines temporären Speicherabbilds der Datei (oder eines Teils davon)



[https://www.virtualuniversity.ch/telekom/algoda/18.html]

Abb. 3. Puffern von Dateiinhalten über Arrays

Dateipufferung

Die High-Level-Funktionen verwenden einen programminternen Dateipuffer. Die Funktionen greifen zum Lesen und Schreiben immer nur auf den Dateipuffer zu. Der Dateipuffer ermöglicht einen effektiver Datentransfer zwischen Primärspeicher und Sekundärspeicher.

Der Dateipuffer birgt jedoch das Risiko eines möglichen Datenverlustes in sich. Wenn eine Datei für die Ausgabe geöffnet wurde, werden Daten erst beschrieben, wenn der Puffer voll ist. Ist das Programm durch einen Fehler abgebrochen, gehen die Daten im Puffer verloren.

Die Funktion *fflush* schreibt sofort alle Daten des Ausgabestroms aus dem Dateipuffer in den Sekundärspeicher. Es wird empfohlen, nach der Ausgabe sehr wichtiger Daten die Funktion *fflush* aufzurufen. Das Schließen einer Datei bewirkt das gleiche.

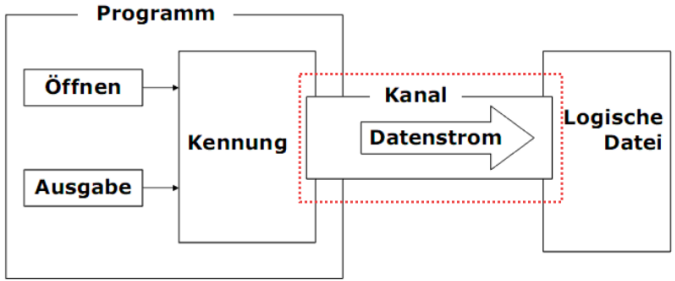
Datenströme

Jetzt befinden wir uns auf der operationalen und Betriebssystemebene. Die Dateimodell erlaubt sequenziellen Zugriff, daher kann man das Konzept des Kanals und Datenstroms einführen.

- Ein- und Ausgabe wird als Datenstrom (Byte-Strom, stream) behandelt. Streams sind immer unidirektional.

1. Beim Öffnen einer Datei wird ein Kanal (logischer Kanal, Zugriffskanal, Verbindung) vom Prozess zur vorhandenen logischen Datei angelegt und eine Übertragungsrichtung des Kanals festgelegt:
   - unidirektionaler Kanal für Ausgabe,
   - unidirektionaler Kanal für Eingabe,
   - bidirektionaler Kanal für abwechselnde Eingabe und Ausgabe.
2. Bei der Ausgabe wird eine Folge von Bytes durch den Kanal transportiert und in die Datei geschrieben.
3. Bei der Eingabe kann die gespeicherte Reihenfolge durch den Kanal exakt wieder abgerufen werden.

Datenströme



[https://hssm.hqedv.de/BSys08Dateisystem]

Abb. 4. Öffnen einer Datei und Lesen über einen sequenziellen Datenstrom

Dateizugriffsschnittstellen

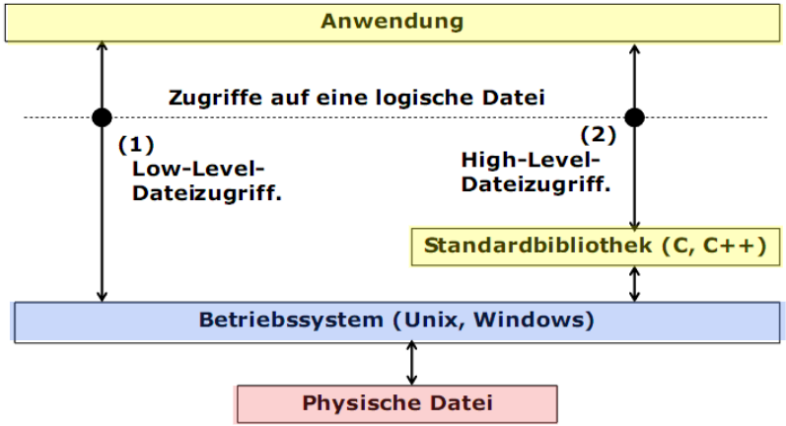
Eine Menge von Dateioperationen einer logischen Datei kann als eine Dateizugriffsschnittstelle bezeichnet werden. Die unterschiedlichen Systeme stellen konkrete Dateizugriffsschnittstellen zur Verfügung.

Zwei Abstraktionsniveaus sind bekannt:

1. Low-Level-Dateizugriff mittels der Operationen von Betriebssystemen. Diese Operationen sind hardwareunabhängig, gehören aber nicht zu dem ANSI C-Standard.

2. High-Level-Dateizugriff mittels der Operationen von Standardbibliotheken. Diese Operationen sind unabhängig von Betriebssystemen bzw. portabel.

Dateizugriffsschnittstellen



[https://hssm.hqedv.de/BSys08Dateisystem]

Abb. 5. Low- und High-level Dateizugriffsschnittstellen

High-Level-Dateizugriff

Es gibt zwei Arten von Zugriffsoperationen unter Unix / (C Standardbibliothek, POSIX):

1. Über einen Dateikanalindex (numerisch, Handle), Funktionen `open read write close`
2. Über einen Dateizeiger und Dateidiskriptor (Datenstruktur), Funktione `fopen fread fwrite fclose`

In beiden Fällen ist der Ablauf immer gleich:

```
OPEN(name, attributes)
REPEAT data=READ | WRITE(data)
CLOSE
```

C-Standardbibliothek	Unix-Dateisystem	Windows-Dateisystem	Beschreibung
fopen	open/creat	CreateFile	Neue Datei erzeugen
remove	unlink	DeleteFile	Datei löschen
fopen	open	CreateFile	Vorhandene Datei öffnen
fclose	close	CloseHandle	Datei schließen
fread/fwrite	read/write	ReadFile /WriteFile	Lesen von Datei und Schreiben in Datei
fseek	lseek	SetFilePointer	Positionieren des Dateizeigers
fflush	fsync	FlushFileBuffers	Sofortiges Schreiben des Puffers
feof	-	-	Prüft, ob Dateiende erreicht
FILE *	**int**	?	Datei Handle

Def. 1. Dateioperationen

Beispiel 1

```
#include <fcntl.h>
#include <sys/stat.h>
int buffer[1024];                                  // Zwischenpuffer
int nread;                                         // Anzahl gelesener Byte
int srcfd, dstfd;                                  // Dateideskriptoren
int main (int argc, char * argv[])
{
  srcfd = open ("srcFile", O_RDONLY);             // Datei srcFile öffnen
  dstfd = creat ("dstFile", S_IRWXU);             // Datei dstFile erzeugen
  do {                                             // Dateien umkopieren
   nread = read (srcfd, buffer, sizeof(buffer));
   if (nread > 0) {
     write (dstfd, buffer, nread);
   }
  } while (nread > 0);                            // Lese bis End-Of-File
  close (srcfd);
  close (dstfd);
  exit (0);
}
```

Bsp. 1. Elementares Beispiel: Kopieren einer Datei

Beispiel 2

High-Level-Dateizugriff

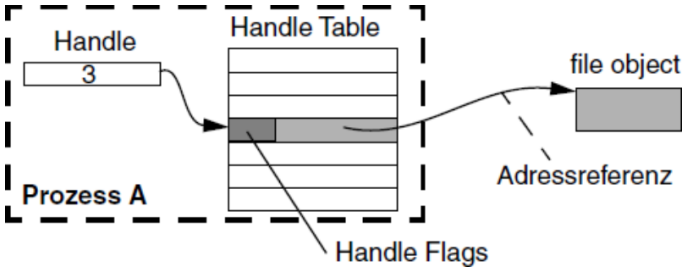


Abb. 6. Ein Filehandle ist ein numerische Index in einer Handletabelle, die dann die Zugrife auf Datein verwaltet.

High-Level-Dateizugriff

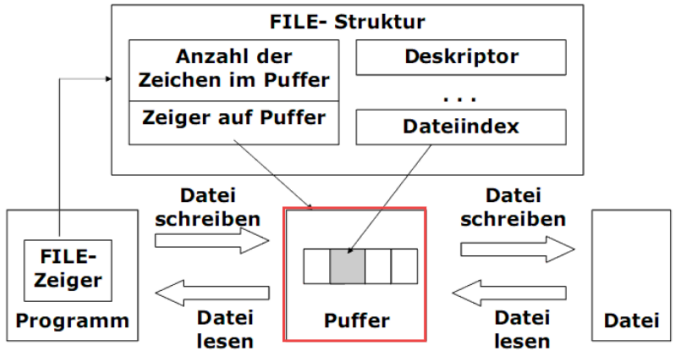


Abb. 7.

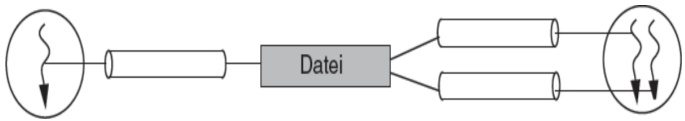


Abb. 8. Dateien können konkurrierend merhfach geöffnet werden (acuh vom gleichen Prozess)

High-Level-Dateizugriff

[bsdp]

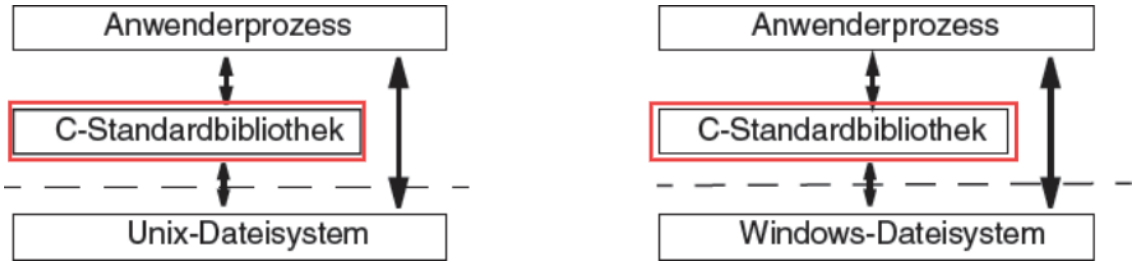
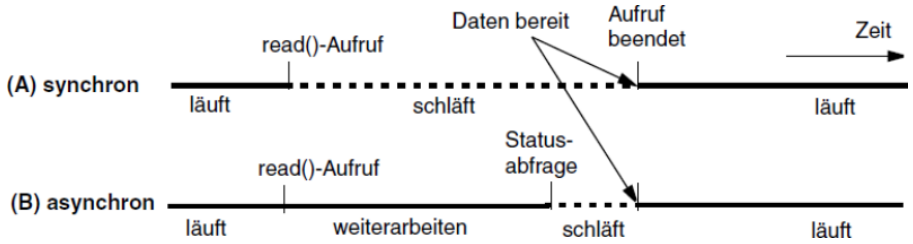


Abb. 9. Die C Bibliothek ist eine betriebssystemübergreifende Schnittstelle, nicht nur für den Dateizugriff

EA Multiplexing

- Nimmt man an adas mehrere Prozesse auf Dateien (quasi zeitgleich oder überlappend) zugreifen und das der Zugriff auf den Datenträger immer eine bestimmte Zeit τ in Anspruch nimmt, dann könnte man viele anstehende Operationen in einander verschachteln &rARr; EA Multiplexing

- Auch innerhalb eines Prozesses kann es mehrere ausstehende Dateioperationen (oder allgemein EA) geben. Wir unterscheiden dann **synchronen** und **asynchronen** Zugriff.



[bsdp]

Abb. 10. EA Multiplexing bei asynchronen Zugriff

Das Dateisystem (logische Schicht)

- Dateisysteme dienen der Verwaltung von Daten auf einem Sekundärspeicher.
- Dies beinhaltet Festplatten- und Halbleiterlaufwerke sowie die Wechselmedien CD-ROM und DVD.
- Die Verwaltung großer Datenbestände in Dateisystemen stellt erhöhte Anforderungen an die organisierte Ablage auf dem Medium.

Wie haben Datenstrukturen und Algorithmen:

- Dazu gehören effiziente Such-, Lese- und Schreibvorgänge.
- Die Dateiverwaltung stellt Möglichkeiten zur Verfügung, um Daten möglichst allgemein auf einem Datenträger abzulegen.
- Die Dateisystemschnittstelle bietet dazu Funktionen an, die von den implementationsabhängigen Details abstrahieren.

Ziel: Aus Anwendungssicht können alle Medien gleichartig behandelt werden mit gleicher Funktionsschnittstelle.

Das Dateisystem (logische Schicht)

- Die aus Anwendungssicht verwaltete Einheit in einem Dateisystem ist die logische Datei (logical file).
  - Sie stellt eine benannte Ablage für Anwenderdaten unterschiedlicher Art dar.
  - Mit einer Auswahl von Dateisystemfunktionen können Dateien angelegt, verwaltet, beschrieben/gelesen und wieder gelöscht werden.

Zusätzlich kann die Ablage auf einem Medium strukturiert werden, sodass auch bei sehr vielen Dateien die Übersicht gewahrt bleibt. Die übliche Lösung dieses Problems nutzt eine Verzeichnishierarchie, wenn auch alternative Ablageorganisationen denkbar wären.

- Eine logische Datei stellt eine geordnete Sammlung von Datensätzen (records) dar.
  - Ein Datensatz ist die kleinste adressierbare Informationseinheit innerhalb einer Datei.
  - Typischerweise entspricht ein Datensatz auf der Ebene der Dateisystemschnittstelle einem Bytewert beliebigen Inhalts, analog zur Organisation des Hauptspeichers.

Das Dateisystem (logische Schicht)

- Legt man Dateien auf einem Massenspeicher ab, so ist es üblich, eine Reihe damit verbundener Informationen ebenfalls dort zu speichern.
  - Dateiname, Ablageort, Zugriffsrechte und Ablagedatum

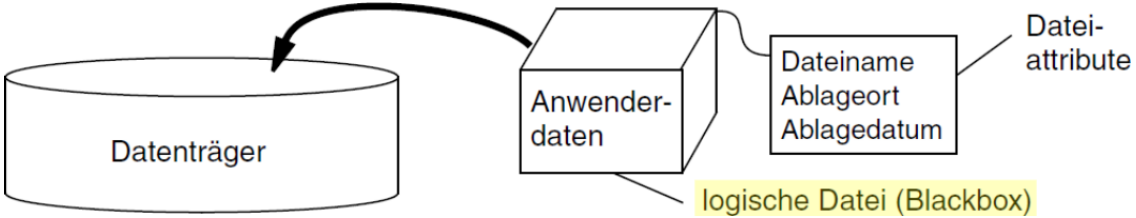


Abb. 11. Logische Datei: Datenablage mit Hilfsinformationen (Blackbox-Modell)

Taxonomie der Dateisysteme

Eine **Datei** (file) ist ein logisches Betriebsmittel, welches eine endliche Menge zusammengehöriger Daten beinhaltet. Dateien existieren, um Daten zu speichern und später wieder abzurufen.

Ein **Dateisystem** (file system, Dateiverwaltung) ist ein Teil des Betriebssystems, der der Verwaltung von Daten auf einem Sekundärspeicher dient.

Alle Dateisysteme zielen auf Geräteunabhängigkeit ab, d. h ein Programm muss sich nicht darum kümmern, ob eine Datei sich auf Platte, Magnetband oder einem anderen Massenspeicher befindet.

Ein **Dateideskriptor** ist eine Tabelle mit Dateiattributen. Dateideskriptoren werden vom Dateisystem auf dem Datenträger abgelegt.

Dateidiskriptoren

Dateiattribute sind dateispezifische Informationen für der Verwaltung [https://hssm.hqedv.de/BSys08Dateisystem] von Dateien. Einige Dateiattribute sind;

1. Dateiname
2. Dateityp (Text, binäre Daten, Programmdatei)
3. Ablageort (Verzeichnis, Ordner, URL usw.)
4. Datum der Erstellung, Veränderung, letzer Zugriff usw.
5. Zugriffsrechte

Als **Metadaten** bezeichnet man allgemein Daten, die Informationen über andere Daten enthalten.

Dateitypen

Jedes Betriebssystem muss zwei Varianten von **Dateitypen** unterscheiden:

1. ausführbare Programme, die in Form von Dateien (executable files) gespeichert werden. Bei der Auswahl wird dieses Programm gestartet, z.B.: „prog.exe“.

2. nicht ausführbare Dateien (Daten, Dokumente). Diese Dateien werden vom Betriebssystem nicht interpretiert, sondern Anwendungsprogrammen zugeordnet, die die Dateien interpretieren können.

Das Dateisystem, die Datenstrukturen und Algorithmen unterscheiden diese Dateitypen nicht. Sie unterscheiden u.U. andere Dateiarten: Swap Datei, Protokolldatei (des Dateisystems), I-Node Tabellendatei, usw..

- Es müssen aber Verkettungen, Verknüpfungspunkte mit anderen Dateisystemen und Gerätedateien von echten Datein mit Daten unterschieden werden.

Dateitypen

Nach der Art des Inhaltes werden unterschieden:

1. Binärdateien enthalten beliebige Bytewerte (auch nicht-alphabetische Zeichen) und können in der Regel nur durch entsprechende Anwendungen gelesen werden. Dazu gehören auch ausführbare Dateien. Beispiele: a.xls, a.exe, a.obj.

2. Textdateien (ASCII-Dateien) enthalten nur reinen Text und einfache Steuerzeichen, z.B. Zeilenumbruch. Viele Steuerdateien sind Textdateien. Beispiele: a.htm , a.txt, a.bpr.

Das Dateiformat kann auf drei verschiedene Arten automatisch ermittelt werden:

1. Erweiterung eines Dateinamens (`.exe`,`.png`,`.html`), ist aber unsicher und nicht eindeutig
2. Die Ermittlung durch Dateiinhalte. Dabei wird auf bekannte Muster untersucht, z.B.: UNIX identifiziert den Dateityp durch die so genannten „magischen Zahlen“.
3. Die Ermittlung durch Metadaten ist die zuverlässigste Methode, weil das Dateiformat zusammen mit der Datei abgelegt ist.

Dateitypen

```
kernel.elf, shell.exe
=====================
00000000   7F 45 4C 46  01 01 01 00  00 00 00 00  00 00 00 00  .ELF............
00000010   02 00 03 00  01 00 00 00  00 00 01 00  34 00 00 00  ............4...

basekernel.png
======================
00000000   89 50 4E 47  0D 0A 1A 0A  00 00 00 0D  49 48 44 52  .PNG........IHDR
00000010   00 00 03 F5  00 00 00 EE  08 06 00 00  00 78 42 27  .............xB'
```

Bsp. 2. Beispiele von Dateitypen und dme "Header"

Die logische Datei

Eine logische Datei wird vom Dateisystem wie ein langer char-Vektor behandelt:

1. Eine logische Datei ist ein Behälter für eine Reihenfolge von Bytes.
2. Der Dateiinhalt ist beliebig, da er vom Dateisystem nicht interpretiert wird.
3. Die Dateilänge ist nicht im Voraus abzusehen (zumindest beim Schreiben).

Zu der Spezifikation einer logischen Datei gehört ein Dateiindex (Schreib-/Lese-Zeiger, Dateizeiger, Byte-Position, Seek-Zeiger, Zeichenposition, aktuelle Position, Dateiposition), der von Dateioperationen (read, write) benutzt wird.

Verzeichnishierarchie

- Dateisysteme sind heute meist **hierarchisch organisiert**, indem in einem Wurzelverzeichnis (root directory) sich sowohl Dateien (files) als auch Unterverzeichnisse (sub directories) anlegen lassen.
- Innerhalb von Unterverzeichnissen, manchmal auch als Ordner (folder) bezeichnet, können weitere Dateien und Unterverzeichnisse vorhanden sein.
- Ein Verzeichnis, als Oberbegriff sowohl für das Wurzelverzeichnis als auch für Unterverzeichnisse verstanden, stellt im Grunde genommen nichts anderes dar als eine Liste von Dateinamen.
- Diese Dateinamen können normale Dateien (regular files) oder Spezialdateien (special files) bezeichnen. Verzeichnislisten gelten sowohl unter Windows als auch unter Unix als Spezialdateien der Art Verzeichnisdatei (directory file).
- Verzeichnisse (Ordner) sind also Tabellen die Namen mit anderen Dateien oder Verzeichnissen verknüpfen!

Verzeichnishierarchie

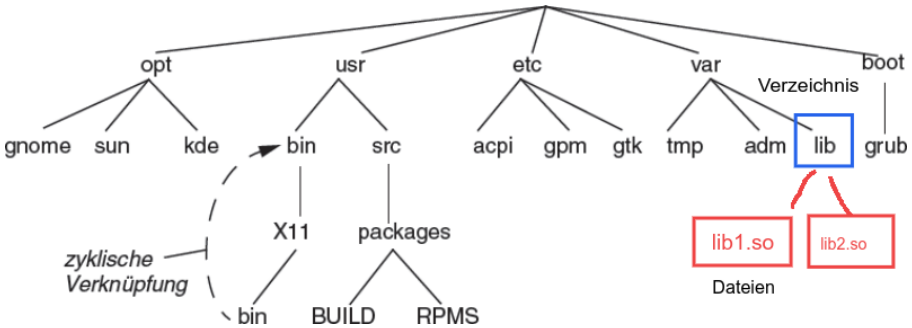


Abb. 12. Verzeichnishierarchie unter Unix

- Die logische Organisation eines Dateisystems ist somit ein Verzeichnisbaum (top-down)
- Da es aber auch Verkettungen (Verlinkung, smybolisch) von Dateien und vor allem Verzeichnis geben kann, können auch Zyklen enthalten sein.
- Die logische Organisation eines Dateisystems ist somit ein zyklischer Verzeichnisgraph

Verzeichnis

Ein Verzeichnis kann Einträge von verschiedenen Dateiarten enthalten:

1. normale Dateien (regular files),
2. wiederum Verzeichnisse (directory file),
3. Verknüpfungen (links)
4. weitere Arten von Spezialdateien, z.B. Gerätedatei (device files).

Verzeichnisfunktionen

- Genauso wie Dateien können Verzeichnisse gelesen und verändert werden!

UNIX,POSIX	Windows	Beschreibung
mkdir	CreateDirectory	Verzeichnis erzeugen
rmdir	RemoveDirectory	Verzeichnis löschen
opendir	-	Verzeichnis öffnen
closedir	FindClose	Verzeichnis schliessen
readdir	FindFirstFile	Ersten Eintrag suchen
readdir	FindNextFile	Nächsten Eintrag suchen
getcwd	GetCurrentDirectory	Arbeitsverzeichnis abfragen
chdir	SetCurrentDirectory	Arbeitsverzeichnis ändern
link, symlink	CreateHardLink CreateSymbolicLink	Dateiverknüpfung anlegen

Tab. 1. Verzeichnisfunktionen (C Standardbibliothek)

Pfade

Verzeichnisse sind Tabellen die auf andere Dateien oder Verzeichnisse verweisen. Verzeichnisse sind selbst Dateien. Einzelne Verzeichnisse können nicht direkt abgerufen werden sondern erfordern eine "Suche".

- Verzeichnisbäume werden durch Suchpfade als Aneinanderkettung der Namen entlang eines Pfades dargestellt (hyperlogische Struktur durch Pfade)
- Das Trennzeichen ist festgelegt:
  - UNIX: `/`
  - MS Windows: `\\`
- Es gibt immer ein Wurzelverzeichnis:
  - UNIX : `/` (alle Laufwerke werden von dort aus erreicht)
  - MS Windows: `X:\\`, wobei *X* ein Laufwerksbezeichner ist

Pfade

Relativer Pfadname: Dateien können relativ zu dem Arbeitsverzeichnis adressiert werden.

Dafür wird zusätzliche Symbolik verwendet:

- Der Name "." verweist auf das Arbeitsverzeichnis.
- Der Name ".." verweist auf das Elternverzeichnis.

Vorteil: Pfadnamen werden kürzer, Nachteil: relative Pfade müssen vor dem Zugriff immer in absolute aufgelöst werden!

Das Dateisystem (physische Schicht)

- Bisher hatten wir nur eine vereinfachte logische Sicht auf Dateien und Dateisysteme:
  - Die Datei war ein linearer Datenbereich auf den über Zeiger und Puffer zugegriffen wurde
  - Das Dateisystem war ein Verzeichnisbaum (oder zyklischer Graph).
- Jetzt kommt die "Implementierung" dieser abstrakten Konzepte auf physische Datenträger.

Realisierung von Dateisystemen

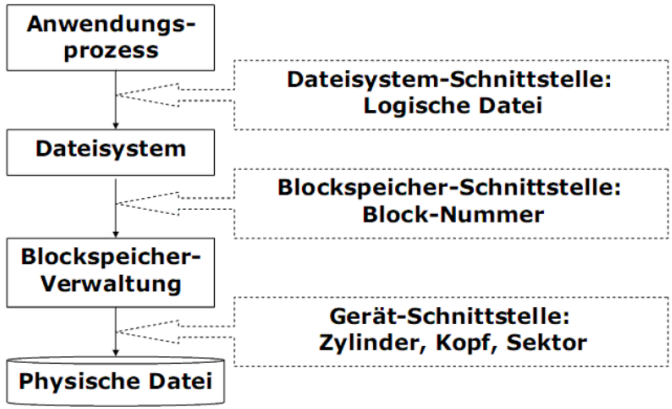


Abb. 13. Schichtenmodell eins Dateisystems: Von logisch bis physische und Gerätetreiber

Blockspeicher als Grundlage

- Bisher war die elementare Strukturgröße der Datei ein Byte. Jetzt führen wir größere Einheiten für vereinfachte und effizientere Verarbeitung ein: **Blöcke**

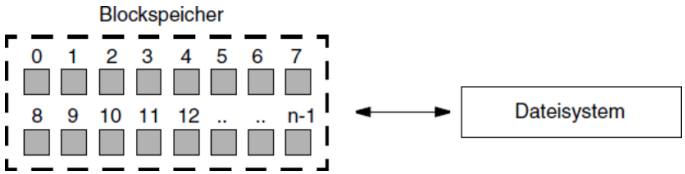


Abb. 14. Blockspeichermodell als Realisierung eins Dateisystems

Ein Datenträger stellt im Blockspeichermodell somit einen Behälter mit n eindeutig adressierbaren Blöcken dar, die entsprechend ihrer Nummerierung auch auf dem Datenträger benachbart angelegt sind.

Blockspeicher als Grundlage

- Eine Datei ist logisch immer zusammenhängend, im Blockspeichermodell wie beim Hauptspeicher und VMM nicht mehr notwendigerweise.

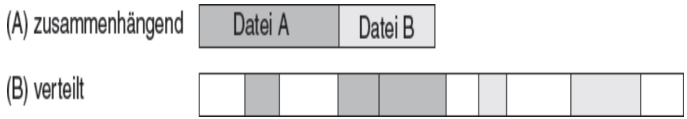


Abb. 15. Zusammenhängende und verteilte Belegung

Die zusammenhängende Belegung führt zu den gleichen Problemen, wie wir sie bereits bei der dynamischen Speicherverwaltung kennengelernt haben. Im Vordergrund steht dabei die unter Umständen erhebliche externe Fragmentierung. Dem steht dafür eine im sequenziellen Zugriff auf die Festplatte hohe Geschwindigkeit gegenüber, da keine oder nur minimale mechanische Kopfbewegungen nötig sind.

Physisch vers Logisch

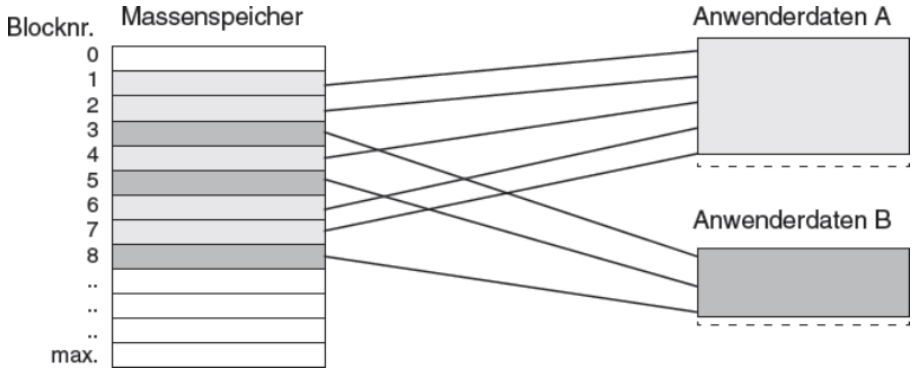


Abb. 16. Beispiel einer Dateispeicherung: Links physische Sicht, Rechts logische Sicht

Wie verwaltet man nun "verstreute" Dateien und Verzeichnisse auf Blockebene?

Verwaltungsformen

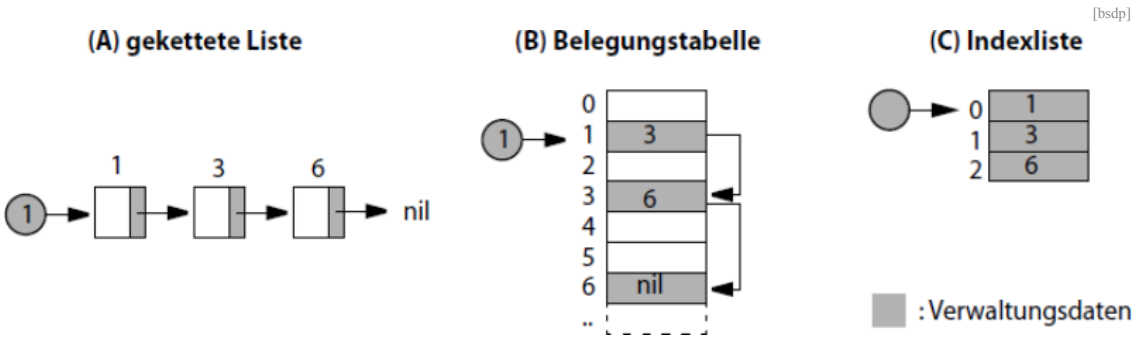


Abb. 17. Typische Datenstrukturen für die Verwaltung von Blockgruppen die zu einer Datei gehören

Belegungsformen

Zusammenhängende Belegung	Verteilte Belegung
+ Kleine Zugriffszeiten infolge minimaler Kopfbewegungen auf der Festplatte (da alle Blöcke benachbart) + Problemlos, wenn Platzbedarf im Voraus bekannt (z.B. CD-ROM, DVD-ROM) + Dateispeicherung durch erste Blocknummer und Anzahl Blöcke ausreichend beschrieben - Externe Fragmentierung kann groß sein - Bei Dateivergrößerung evtl. zeitfressendes Umkopieren ganzer Datei nötig	+ Keine externe Fragmentierung + Einfache Dateivergrößerung möglich - Unter Umständen lange Suchzeiten, da Datei über ganze Festplatte verteilt sein kann - Komplizierte Beschreibung der Dateispeicherung infolge Verteilung

Tab. 2. Vor- und Nachteile zusammenhängender bzw. verteilter Belegung

Methoden (Formen) der Dateiallokation:

1. Kontinuierliche Allokation.
2. Allokation durch verkettete Liste.
3. Allokation durch eine Belegungstabelle.
4. Allokation durch I-Nodes (wird noch eingeführt).

Datenträgeraufteilung

Man unterscheidet grundlegend zwei Bereiche (die aber nicht immer getrennt werden können), die eine Aufteilung des Datenträgers bedeuten:

1. Verwaltungsbereiche
2. Datenbereiche

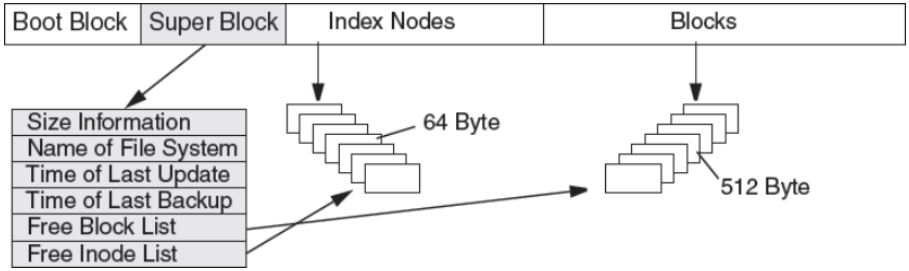


Abb. 18. Datenträgeraufteilung des UFS. Index Nodes und die ersten Blöcke gehören zur Verwaltung (physisch), Blocks gehören zur logischen Dateisystemschicht

Datenträgeraufteilung

Der Datenträger wird in vier Bezirke aufgeteilt:

- Der Boot-Block wird für das Laden des Betriebssystems benötigt.
- Im Super-Block sind Basisinformationen über den Datenträger abgelegt, wie Größe, Name des Dateisystems und des Datenträgers, Zeiteinträge des letzten Zugriffs und der Datensicherung. Daneben enthält er die Anzahl aller Index Nodes und den Beginn der Liste freier Blöcke.
- Die Index Nodes, abgekürzt auch Inodes genannt, dienen der Verwaltung einzelner Dateien. Man kann sie als Verwaltungsblöcke bezeichnen. Sie sind im dritten Bereich als Liste enthalten (i-list).
- Die Datenblöcke (Blocks) enthalten sowohl Verzeichnisdaten als auch die eigentlichen Nutzdaten. Mit Nutzdaten sind diejenigen Inhalte gemeint, die Applikationsprogramme persistent (d.h. dauerhaft) ablegen. Im Unterschied dazu werden alle Daten, die der Verwaltung der gespeicherten Nutzdaten dienen, als Metadaten bezeichnet.

Index Nodes

- Internal oder Index Nodes (I-Nodes) sind Referenzen und in einfache Inode Tabellen zusammengefasst.
- Zu jeder Datei wird im Verzeichniseintrag die Inode-Nummer und der Dateiname abgelegt.
- Die Inode-Nummer ist ein Index in eine Liste der verfügbaren Verwaltungsblöcke (i-list).

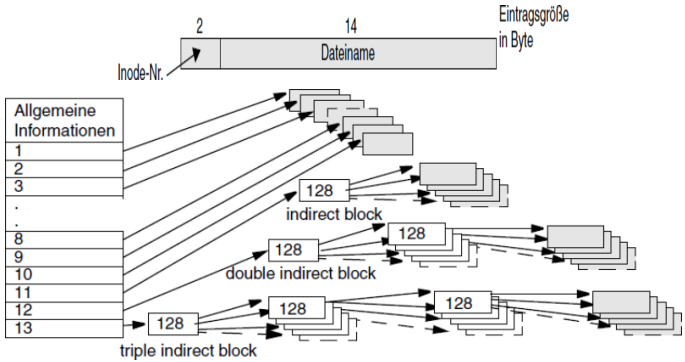


Abb. 19. Struktur und Vernetzung eines Inodes. Die Tabelle links enthält zwei Spalten: Inode Nummer und Name

Index Node Tabellen und Verkettung

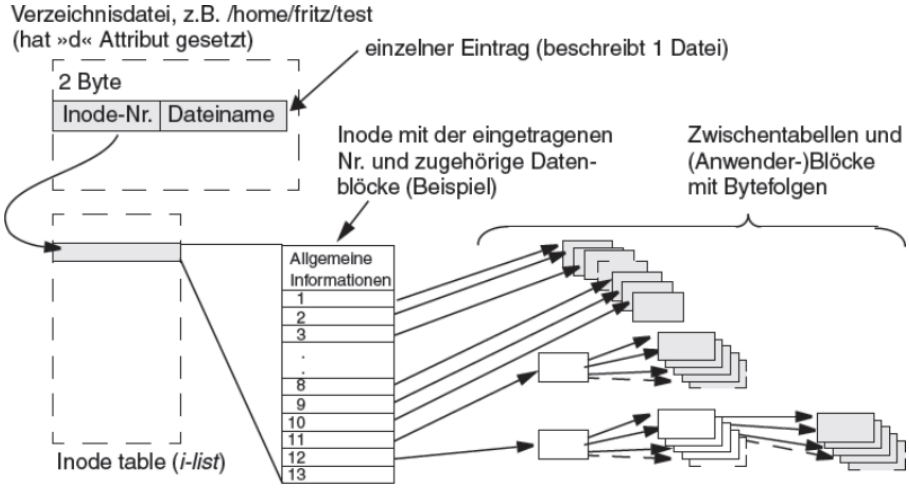


Abb. 20. Zusammenhang zwischen Verzeichniseintrag, Verzeichnisblock und Datenblöcke

Index Node Tabellen und Verkettung

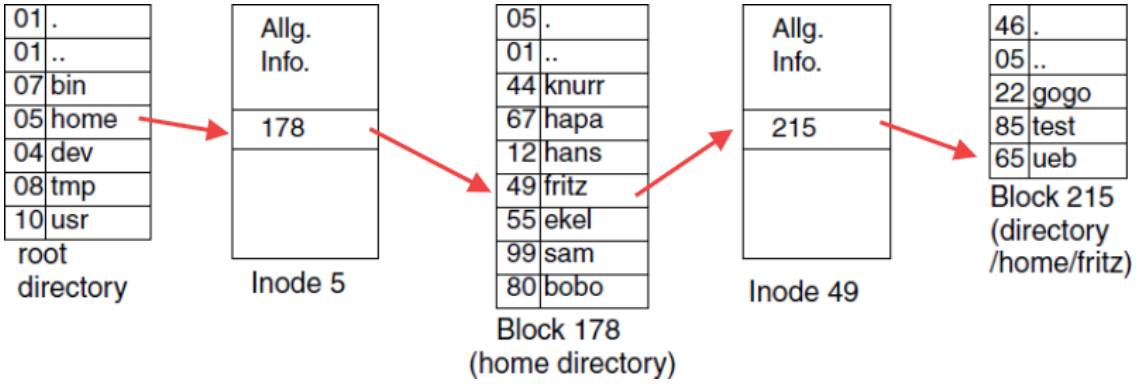


Abb. 21. Beispiel zur Dateilokalisierung im klassischen Unix-Dateisystem

Hierarchische Tabellenverkettung

- Eine I-Nodes Tabelle reicht nicht aus um große Dateien mit allen Blöcken zu erfassen.
- Daher werden die Tabellen selber wieder verkettet

[https://hssm.hqedv.de/BSys08Dateisystem]

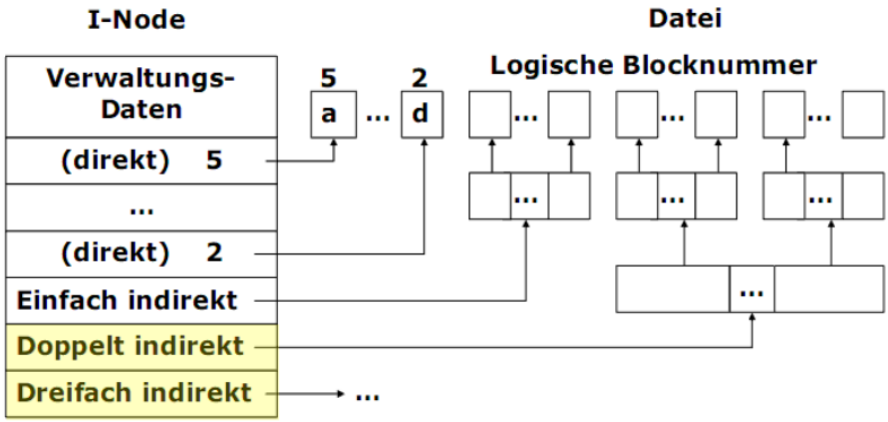


Abb. 22. Prinzipbeispiel für große Datei mit Verkettung von I-Nodes Tabellen

Storage Pools und Logging

- Was passiert wenn während der Veränderung des Dateisystems ein Fehler auftritt? Z.B. Stromausfall oder Softwarefehler oder Datenträgerfehler?
- Nur ein Protokollierung der Operationen auf Dateisystemen ermöglich eine Rekosntruktion und Wiederherstellung der Konsitenz der veretilten Datenstrukuren (I-Nodes Tabellen usw.)
- Was machen wir wenn wir mehr als einen Datenträger für ein Dateisystem verwenden wollen?
  - Storage Pools können helfen

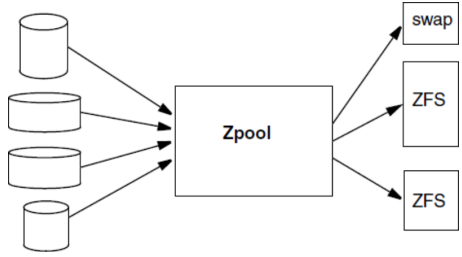


Abb. 23. Unter ZFS (Zettabyte File System) lassen sich Laufwerke zu einem Storage Pool (Zpool) zusammenfassen, aus dem sich mehrere ZFS-Dateisysteminstanzen und Swap-Laufwerke erzeugen lassen.

Zusammenfassung

1. Datein sind abstrakt und durch ein lienares Zellenmodell beschrieben.

2. Bei der physische Realisierung einer Datei (Speicherung auf Datenträger) wird diese in Blöcke unterteilt die eindeutig adressierbar sind und über den Datenträger verteilt sein können.

3. Dateisysteme haben verschiedene Schichten: Logisch ⇒ Verzeichnisgraph, Physisch ⇒ Speicherung auf Datenträger z.B. via I-Nodes Tabellen

4. Verzeichnisse sind Datenstrukturen und Tabellen und auch Dateien und verknüpfen Daten mit Namen (Dateien haben eigentlich keine Namen!) und bauen die Verzeichnishierarchie auf.