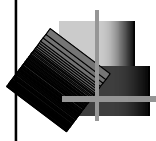
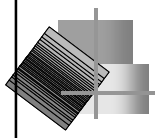


# Systemsoftware I

## 8. Dateisysteme

1

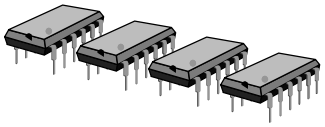


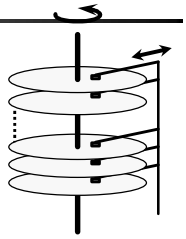
## Dominanz der Dateisysteme

- Wesentliche Leistungen eines Betriebssystems sind für den Anwender unsichtbar
  - Virtuelle Adreßräume
  - Kontrollflüsse
  - Synchronisation
  - Kommunikation
  - …
- Qualität dieser Betriebssystem-funktionen bestimmt Leistung und Einsatzgebiet des Gesamtsystems
- Dateisystem ist für den Anwender exponiert
  - Alle dauerhaft gespeicherten Daten sind dort gespeichert
  - Nur eine Facette des Gesamtsystems

2

## Interner vs. Externer Speicher





- Eigenschaften
  - Flüchtig
  - Byte- und Wort-Zugriffe
  - Verhältnismäßig schnell (60 ns)
- Hohe Kosten
  - 1 MB ca. 14 DM (aktuelle Tiefpreise)
- Geringe Kapazität
  - Aktuell 16 bis 256 MByte

- Eigenschaften
  - Nicht-Flüchtig
  - Blockzugriffe (512 Byte bis 8 Kbyte)
  - Sehr langsam (10-15 ms)
- Geringe Kosten
  - 1 MB ca. 0.34 DM
- Hohe Kapazität
  - Aktuell 1 bis 4 GByte

3

## Dateisysteme

- Dauerhafte Speicherung von Programmen und Daten
- Datei
  - Benannte Menge zusammengehörender Information auf externem Speicher
- Dateiattribute
  - Name
  - Typ (in manchen Dateisystemen)
  - Ort
  - Größe
  - Zugriffsrechte
  - Zeitinformationen
  - Eigentümer
- Verzeichnisse speichern Attribute mehrerer Dateien
  - Verzeichnisse sind selbst Dateien

4

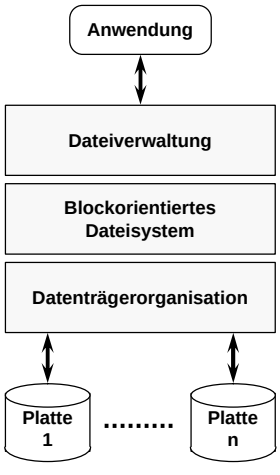
## Nutzungscharakteristika bei Dateien

- Empirisch ermittelte Eigenschaften:
  - Dateien sind meist klein (wenige Kbyte)
  - Dateien werden häufiger gelesen, seltener geschrieben (und noch seltener gelöscht)
  - Sequentieller Zugriff ist dominant
  - Manche, wenige Dateien werden von vielen benutzt
  - Großteil der Dateien wird nur von einer Person benutzt
- Nutzungscharakteristik bestimmt in wesentlichen Teilen Aufbau und Funktion des Dateisystems
- Typische Dateinutzung optimieren
  - Sequentieller Lesezugriff auf kleine Dateien
  - Caching sinnvoll (Lesen häufiger, 1 Benutzer häufig)

5

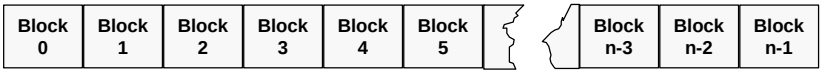
## Schichten eines Dateisystems

- Datenträgerorganisation
  - Hardware-unabhängige Schnittstelle zu allen externen Speichern
  - Verwaltung und Ansteuerung der verschiedenen Speichersysteme
- Blockorientiertes Dateisystem
  - Unterteilung eines Datenträgers in mehrere Einheiten
  - Blockbasierter Zugriff
  - Logische Blocknummern
- Dateiverwaltung
  - Realisierung der Anwendungs-schnittstelle
  - Verzeichnisstruktur
  - Dateizugriff




6

# Schicht 1: Datenträgerorganisation

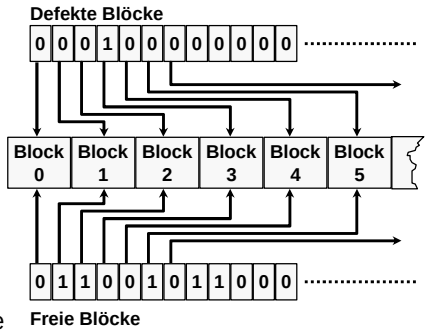



- Logische Durchnummerierung jedes Datenträgers
  - n sehr groß (1 Gbyte Platte und Blockgröße 4 Kbyte: n = 262144 Blöcke)
- Ansteuerung u.U. aufwendig
  - Physische Adresse: vereinzelt noch Zylinder, Spur und Sektor
  - Abbildung logische Blocknummer auf physische Adresse
- Weitere Aufgaben
  - Formatierung durchführen oder koordinieren
  - Verwaltung defekter Blöcke
  - Verwaltung freier Blöcke
  - Lese- und Schreiboperationen über logische Blocknummern

7

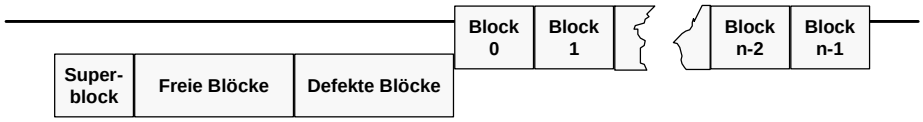
# Verwaltung defekter und freier Blöcke

- Keine Platte ist perfekt
  - Von Anfang an defekte Blöcke
  - Zusätzliche Blöcke zum Ausbessern
  - Blöcke können defekt werden
    - `Add_Bad_Block (logische Blocknummer)`
  - Löcher in der logischen Nummerierung vermeiden
- Verwaltung freier Blöcke
- Gängige Realisierung
  - Bitvektoren für defekte und freie Blöcke
  - Vektoren werden auf dem Datenträger selbst gespeichert
  - 8, 16 oder 32 gleichzeitige Tests durch logische Operationen




8

# Organisation




- Superblock
  - Speicherung der Plattenaufteilung
  - Größe und Position der Bitvektoren
  - ...
- Superblock wird typischerweise mehrfach repliziert

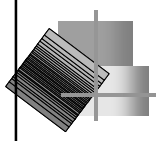
9

# Kleinste Einheiten

?

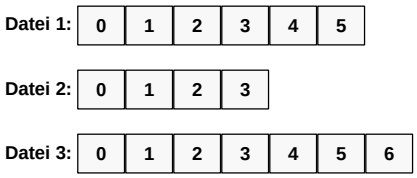
- Granulat der Plattennutzung wird durch Betriebssystem und MMU bestimmt
  - MMU´s unterstützen nur bestimmte Seitengrößen
  - Adreßraumverwaltung und andere Komponenten für bestimmte Seitengröße ausgelegt
- Betriebssystem legt Größe logischer Blöcke fest
  - Blockgröße = Vielfaches der minimalen Zugriffseinheit eines externen Speichers
  - Beispiel:
    - 512 Byte kleinste Zugriffs-einheit einer Platte
    - 8 Kbyte Seitengröße
    - 1 logischer Block = 16 physische Blöcke
- Microsoft definiert Größe physischer Blöcke über Plattengröße
  - Je größer die Platte
    - desto größer die minimale Zugriffseinheit
    - desto größer die interne Fragmentierung
  - Beachte: die meisten Dateien sind klein
- Beispiel
  - Platten bis 250 Kbyte: 2 Kbyte
  - Platten bis 500 Kbyte: 4 Kbyte
  - Platten bis 1 Gbyte: 8 Kbyte
  - ...

10

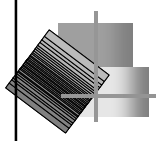


# Schicht 2: Blockorientiertes Dateisystem

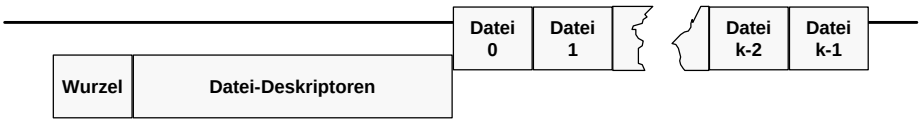
- Aufteilung des Plattenplatzes auf mehrere Dateien
- Eigenschaften
  - Dateien besitzen interne Namen (noch nicht symbolisch)
  - Dateien bestehen aus einer Menge von Blöcken
  - Blöcke einer Datei werden relativ zum Dateianfang nummeriert
  - Verwaltung der meisten Dateiattribute
- Funktionen
  - Erzeugen und Löschen
  - Vergrößern und Verkleinern
  - Öffnen und Schließen
  - Lesen und Schreiben von Blöcken




11



# Organisation




- Wurzel
  - Anzahl der reservierten Datei-Deskriptoren = Maximale Anzahl Dateien auf Datenträger
  - Position und Größe der Deskriptoren
  - Häufig in den Superblock integriert
- Aufbau einer Datei
  - Welche Blöcke gehören zu welcher Datei?
  - Zuteilungsverfahren physische zu logische Blöcke

12

# Dateiaufbau

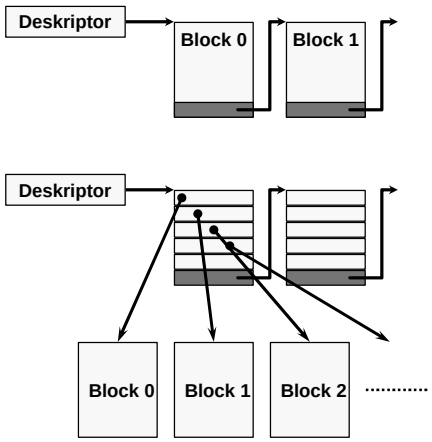

- Zusammenhängende Belegung
- Vorteile
  - Schneller sequentieller und wahlfreier Zugriff
  - Verwaltung einfach
- Nachteile
  - Externe Fragmentierung
  - Verkleinerung von Dateien erzeugt schlecht nutzbare Löcher
  - Dateiwachstum unmöglich oder aufwendig


- Verteilte Belegung
- Vorteile
  - Minimale interne Fragmentierung
  - Vergrößerung und Verkleinerung von Dateien einfach
- Nachteile
  - Unterschiedliche Zugrffszeiten auf verschiedene Blöcke
  - Verwaltung aufwendiger
- Gängiges Verfahren
  - Clustering, d.h. zusammenhängende Belegung, auf Wunsch meist möglich

13

# Verteilte Belegung: Realisierungsvarianten

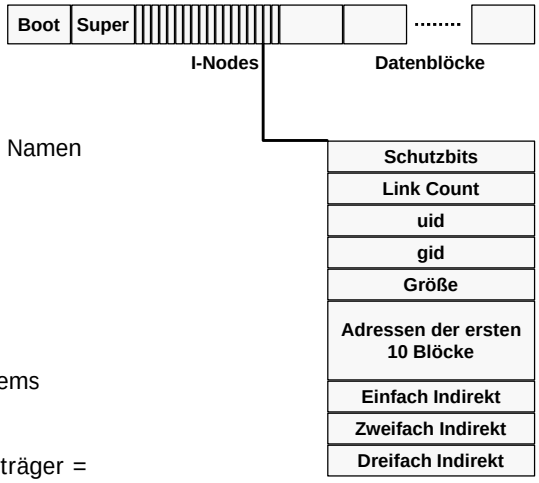
- Speicherung der Blockreihenfolge
- Intern: Verkettung innerhalb der Blöcke
  - Exotische Blcokgrößen für Anwendungen
  - Wahlfreier Zugriff teuer
- Extern: Spezielle Blöcke speichern Blockreihenfolge
  - Einfach und doppelt verkettete Listen
  - Bäume
  - Mischformen




14

## Beispiel UNIX

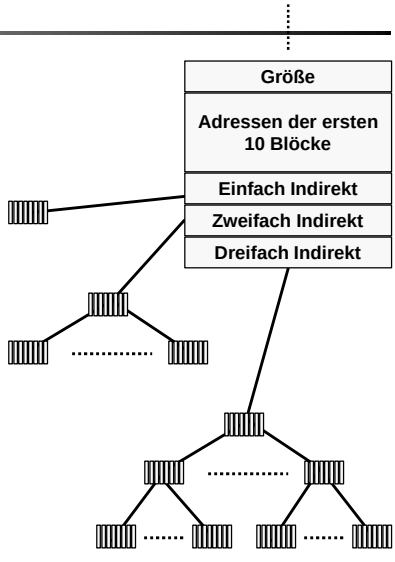
- I-Node
  - Größe 64 Byte
  - Zugriffsrechte
  - Anzahl Referenzen = Anzahl verschiedener Namen
  - Eigentümer
  - Größe
- I-Node 1
  - Bad Block List
- I-Node 2
  - Wurzel des Dateisystems
- Terminologie:
  - 1 organisierter Datenträger = Dateisystem



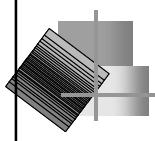

15

## Inode: Blockverkettung

- Bevorzugung kurzer Dateien
- Indirektionsstufen verweisen auf spezielle Indexblöcke (keine Daten)
- Beispiel
  - Blockgröße 1 Kbyte
  - Blockverweis 4 Byte
- Anzahl Zugriffe in Abhängigkeit der Dateigröße:
  - 0: 1 Byte bis 10 Kbyte
  - 1: 10 Kbyte bis 266 Kbyte
  - 2: 266 Kbyte bis 65.5 Mbyte
  - 3: 65.5 Mbyte bis 16 Gbyte



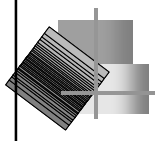

16



## Schicht 3: Dateiverwaltung

- Organisation der blockorientierten Dateien
- Unterstützung verschiedener Dateitypen
  - Normale Dateien
  - Verzeichnisse
  - Spezialdateien, z.B. Zugriff auf Geräte über Dateischnittstelle
- Normale Dateien
  - Lesenden und schreibenden Zugriff von Bytes
  - Sequentielle und wahlfreie Zugangsformen
  - Schutz
- Verzeichnisse
  - Flache und hierarchische Verzeichnisstrukturen
  - Azyklische Verzeichnisstrukturen
  - Symbolische Namen für Dateien
  - Zusammenfassen mehrerer Datenträger

17



## Einfache Dateien

- Aufbau einer Datei
  - Byte-Sequenz
    - Dateisystem sieht und unterstützt keine höheren Organisationsformen
    - Häufige Sichtweise: z.B. UNIX, Windows, ...
  - Sequenz von Sätzen gleicher Länge
    - Veraltete Organisationsform
    - in Zeiten von 80-Zeichen-Terminals und Lochkarten populär
  - Über Indexstrukturen organisierte Dateien
    - ISAM, Invertierte Indextabellen, Bäume, ...
    - Veraltete Organisationsform
- Status Quo
  - Dateisysteme unterstützen meist “nur” Byte-Sequenzen
  - Höhere Organisationsformen werden heute von Datenbanksystemen erbracht
  - Übergang Dateisystem-Datenbanksystem wird fließend

18

## Elementare Dateioperationen

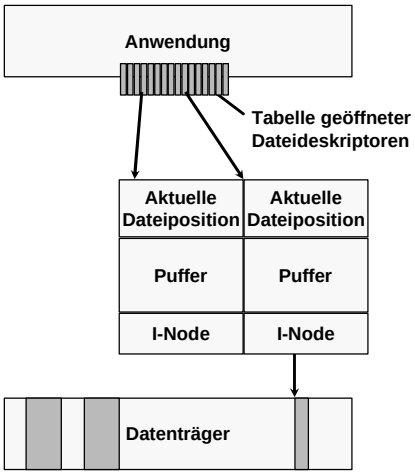
```
char b[1000];
int fd,ret;

fd = open("X",O_RDONLY);
ret = read(fd,b,sizeof b);
while (ret > 0) {
   ...
   ret = read(fd,b,sizeof b);
   }
close(fd);
```

- Datei erzeugen
  - Initial 0 Byte Länge
  - Setzen einzelner Atrribute
- Datei öffnen
  - Anwendung plant Zugriff auf Datei
  - Zugriffsart festlegen
- Von Datei lesen
  - n Bytes ab aktueller Position lesen
- Datei löschen
  - Freigabe der belegten Blöcke
  - Freigabe des Deskriptors
- Datei schließen
  - Keine weiteren Zugriffe durch Anwendung
  - Freigabe dynamisch belegter Ressourcen
- Auf Datei schreiben
  - n Bytes ab aktueller Position schreiben

19

## Sequentieller Zugriff

- System-interner Zeiger auf die aktuelle Dateiposition
  - Initialer Wert 0
- Lese- und Schreiboperationen arbeiten relativ zur aktuellen Dateiposition
- System-interner Puffer hebt blockorientierten Dateizugriff auf
  - Nicht jeder Lesezugriff bedeutet auch Plattenzugriff



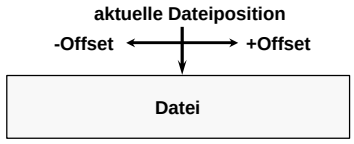

20

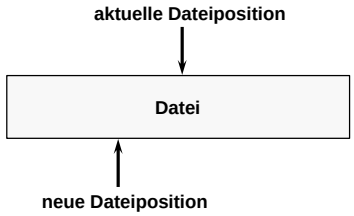
# Wahlfreier Zugriff

- Explizite Änderung der aktuellen Dateiposition
- Beispiel UNIX:

```
int lseek (
     int Datei,
     int Offset,
     bool Relative )
```

  - Relativ: Dateiposition wird um angegebene Anzahl Bytes nach vorne oder hinten verändert
  - Absolut: Offset ist die neue aktuelle Dateiposition
  - Rückgabewert: Neue Dateiposition (Absolut)
- Problem:
  - Geringe Effizienz, da Pufferung nicht greift



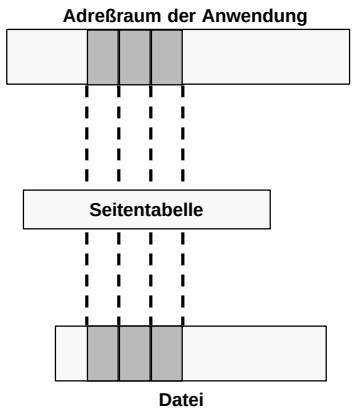

21

# Einblenden von Dateien

- Nutzung virtueller Speichertechniken
  - Seitenersetzung lagert Seiten zwischen Hauptspeicher und Platte ein und aus
  - Anwenden auf Dateien: Ein- und Auslagern von Dateiblöcken
- Beispiel: UNIX

```
Adresse mmap (
     Adressvorschlag,
     Länge,
     Schutz,
     Shared/Private,
     Dateideskriptor,
     Offset innerhalb der Datei)
```

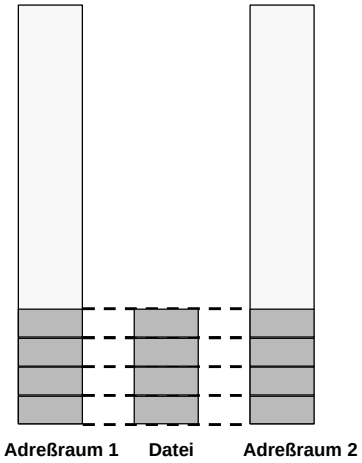
- Änderung der Zugriffsrechte für einzelne Seiten jederzeit möglich




22

## Vorteile

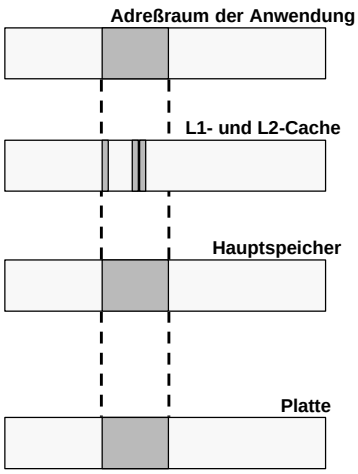
- Keine Benachteiligung eines bestimmten Zugriffsverfahrens
  - Sequentieller Zugriff: Seiten werden der Reihe nach eingeblendet
  - Wahlfreier Zugriff: Benötigte Seiten werden nach Bedarf nachgeladen
  - Auslagern eingeblendeter Seiten bei Speichermangel
  - Erneuter Zugriff bei ausreichendem Speicher direkt über Hauptspeicher
- Dateizugriff über Lese- und Schreiboperationen
- Nurlesedateien besonders geeignet
  - Programmcode
  - Bibliotheken (Dynamische Bibliotheken)




23

## Inkonsistenzen durch Pufferung

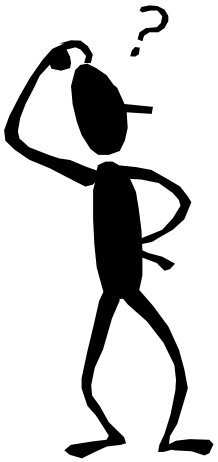
- Original der Seite auf Platte
- Replikate der Seite (oder von Teilen)
  - Teile in L1- und L2-Cache
  - Kopie im Hauptspeicher
  - Kopie im Platten-Controller
  - Kopie im Platten-Cache
- Inkonsistenzfenster
  - Änderung bis Aktualisierung der Platte
  - Lesen unterschiedlicher Inhalte durch mehrere Kontrollflüße
  - Datenverlust und Inkonsistenz im Fehlerfall
- Lösung
  - Write-Through (bei Hardware)
  - Deferred-Write (Dateisystem)
  - Sperren




24

## Quizfrage

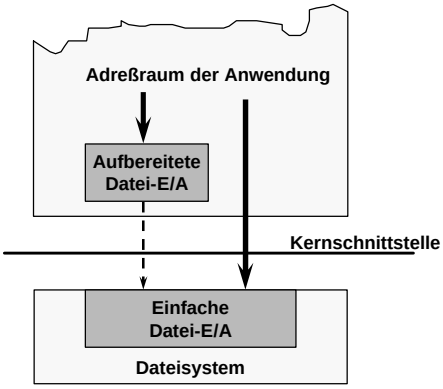
- Wie kann ein Dateisystem Write-Through auch auf Software-Ebene erreichen?
- Ist es sinnvoll?



25

## Aufrufschnittstelle

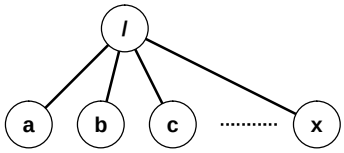
- Einfache Schnittstelle
  - Öffnen, Schließen
  - Dateideskriptoren des Dateisystems
  - Lesen- und Schreiben von Bytes
    - `Read`
- Aufbereitete Schnittstelle
    - `Write`
  - Öffnen, Schließen
  - Spezielle Dateideskriptoren
  - Lesen- und Schreiben von Daten
    - Integer
    - Real
    - Zeichenkette
    - ...
  - Textformatierung
    - `printf`
    - `scanf`

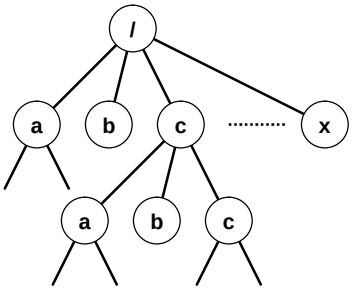



26

# Verzeichnisse

- Heutige Dateisysteme verwalten viele 1000 Dateien
  - z.B. allein 2000 Dateien in meinem Heimverzeichnis
- Verzeichnisse organisieren Dateien
  - Flache Verzeichnisstrukturen
    - Kein Name doppelt
    - Explosion der Namenslänge
    - Keine sinnvolle Ordnung bei vielen Dateien
    - Veraltet
  - Hierarchische Verzeichnisstrukturen
    - Inhalt sind Dateien und Unterverzeichnisse
    - Pfadname für Datei

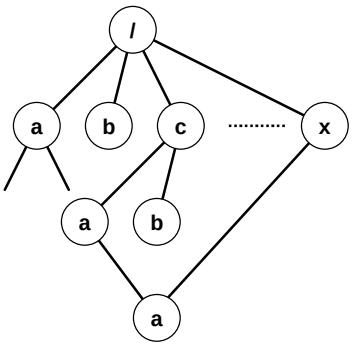


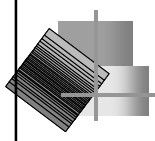

27

# Azyklische Verzeichnisstrukturen

- Zugriff auf eine Datei über mehrere Pfade
  - Datei (Verzeichnis) besitzt mehrere Namen
  - Gemeinsame Nutzung der Datei
  - Aufbau spezifischer Umgebungen
- Links
  - Pfad (I-Node) verweist auf eine Datei an anderer Stelle
  - Referenzen werden gezählt
  - Löschen einer Datei: `unlink`
  - Physisches Löschen erst bei einem Referenzzähler 0
- Link-Varianten
  - Symbolische Links
  - Hardlinks



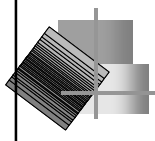

28



## Aufrufschnittstelle

- Pfadangaben relativ zu ausgezeichneten Verzeichnissen
  - Wurzel (absolute Adressierung)
  - Heimverzeichnis eines Benutzers (Home Directory)
  - Aktuelles Arbeitsverzeichnis (Working Directory)
- Verzeichnisoperationen
  - Abfrage des aktuellen Arbeitsverzeichnisses (`getwd`)
  - Arbeitsverzeichnis wechseln (`chdir`)
  - Verzeichnis anlegen (`mkdir`)
  - Leeres Verzeichnis löschen (`rmdir`)
- Lesen und Suchen in Verzeichnissen
  - Öffnen als einfache Datei nicht möglich
    - Verzeichnisstruktur für Anwendungen unzugänglich
  - Sequentielles Lesen eines Verzeichnisses
    - Öffnen: `DIR *opendir (Pfad)`
    - Nächsten Eintrag lesen: `struct dirent *readdir (DIR *)`
    - Wahlfreies Positionieren (`seekdir`)

29



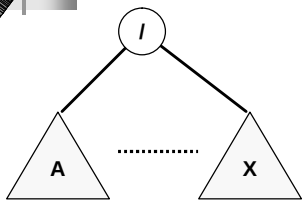
## Zugriff auf Dateiattribute in UNIX

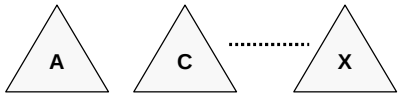
- Verzeichniseintrag (`dirent`) enthält primär Dateinamen (relativ zum Verzeichnis)
- Dateiattribute durch Aufruf von `stat (Pfad,struct stat *)`:

```
struct stat {
     dev_t st_dev;          // Datenträger
     ino_t st_ino;          // I-Node der Datei
     mode_t st_mode;        // Schutzmodus und Dateityp
     nlink_t st_nlink;      // Anzahl Referenzen
     uid_t st_uid;          // Eigentümer
     gid_t st_gid;          // Gruppenzuordnung
     off_t st_size;         // Dateigröße
     time_t st_atime;       // Letzte Zugriffszeit
     time_t st_mtime;       // Letzte Änderung
     time_t st_ctime;       // Änderung am Schutzmodus
                            // oder Dateinamen
     }
```

30

## Ein oder mehrere Verzeichniswurzeln



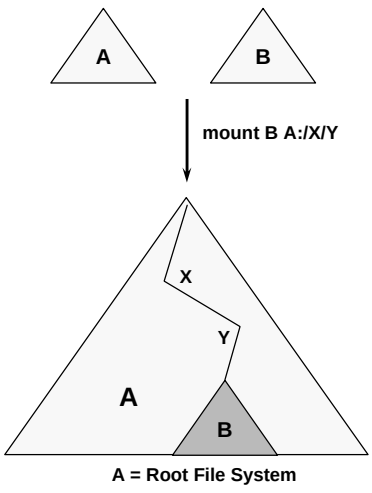

- Datenträger werden in eine übergeordnete Verzeichnis-struktur integriert
- Ortstransparenz
- Dateien vollständig auf einem Datenträger

- Jeder Datenträger besitzt eine eigene Verzeichnisstruktur
- Keine Ortstransparenz
- Links häufig nicht über Datenträgergrenzen
- Dynamische Namenszuordnung zu Datenträgern problematisch

31

## Mounting

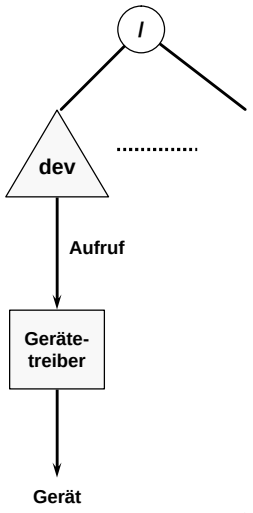
- Jeder Datenträger beginnt lokal mit einem Wurzelverzeichnis (/)
- Ausgezeichnetes Dateisystem (Root File System)
- Weitere Dateisysteme werden an einen bestimmten Pfadnamen gebunden (mounting)
- Betriebssystem verwaltet Mounting-Tabelle
  - Pfadname, Gerät
  - Zugriffsrechte
- Erweiterbar auf entfernte Platten
  - Platte an anderen Rechnern
  - Zugriff über Nachrichten
  - Netzwerk-Dateisysteme




32

# Spezialdateien

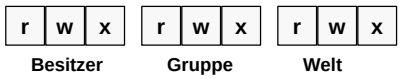
- Bekannte Beispiele
  - Verankerung von Namen für andere Betriebsmittel
    - Named Pipes
    - Message Queues
- Zugriff auf Geräte über Dateisystem
  - E/A-Geräte wie Dateien benutzen
  - Aufruf der jeweiligen Gerätetreiber
  - Beispiele
    - Terminal: `/dev/tty`
    - Platte (mit Struktur): `/dev/sd0a`
    - Platte (Raw): `/dev/rsd0a`
    - Netzwerk, ...
- Unterscheidung zwischen
  - Blockorientiertem Zugriff (Block Special Files)
  - Zeichenorientiertem Zugriff (Character Special Files)




33

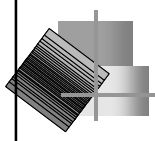
# Schutz von Dateien

- Wer darf auf eine Datei zugreifen?
- Wie darf man auf eine Datei zugreifen?
- Subjekte = Prozesse, ...
- Objekte = Dateien
- Zwei Varianten
  - Subjekt-basiert
  - Objekt-basiert




- Beispiel UNIX
  - 11 Schutzbits
- Dateien
  - R: Lesender Zugriff
  - W: Schreibender Zugriff
  - X: Ausführen
- Verzeichnisse
  - R: Verzeichnis lesen
  - W: Änderungen erlaubt, z.B. Datei löschen oder hinzufügen
  - X: Zugriffsrecht auf alle Dateien im Verzeichnis

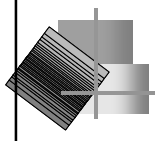
34



# Caching

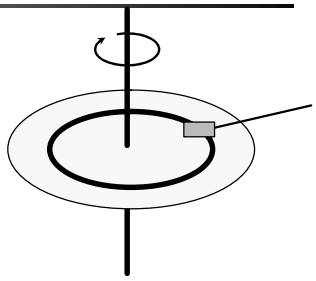
- Zwischenspeichern von Seiten (Blöcken) gerade bei Dateisystemen reizvoll
  - im Vergleich zur restlichen Hardware hohe Zugriffszeiten
  - Gutartige Zugriffscharakteristik
    - Lesen überwiegt: geringe Konfliktwahrscheinlichkeit
    - Sequentieller Zugriff häufig: Information über die Zukunft
- Dateisysteme nutzen meist großen Zwischenspeicher
  - u.U. restlichen freien Hauptspeicher
  - Bei großem Zwischenspeicher ist schnelle Trefferermittlung wichtig
  - Hash-Tabellen, ...
- Inkonsistenzen durch Datenreplikate
  - Explizites Rausschreiben (Flush) durch Anwendung
  - Implizites, periodisches Rausschreiben durch Dateisystem (UNIX alle 30 Sekunden; geht meist noch nach Rechnerabsturz)

35



# Read-Ahead

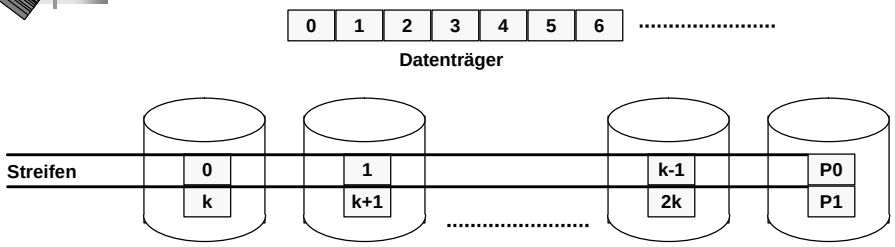
- Sequentieller Zugriff überwiegt
  - Hohe Wahrscheinlichkeit, daß nachfolgende Daten ebenfalls gelesen werden
- Besondere Eigenschaften aller Platten
  - Positionierungszeit des Kopfes lang (8 ms und mehr)
  - Keine Positionierungszeit für nachfolgende Sektoren
  - Lesegeschwindigkeit aufeinander-folgender Sektoren eines Zylinders hängt nur von Umdrehungszahl ab
  - Elektronik beschränkender Faktor
- Voraussetzungen
  - Zylinderstruktur der Platte sichtbar
  - Dateisystem unterstützt Clustering



- Beispiel
  - 1 Gbyte verteilt auf 1500 Zylinder
  - 700 Kbyte pro Zylinder
  - 5400 upm: 62 MB/sec
  - 7200 upm: 82 MB/sec

36

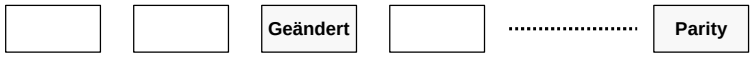
## Striping




- Leistungsfähiger Datei-Server
  - 1 Server mit mehreren, unabhängigen Platten (RAID)
  - Mehrere Server mit jeweils eigenen Platten
- Aufteilen eines Dateisystems auf mehrere Platten
  - Zugriff auf mehrere Platten/Rechner verteilen
    - Bessere Ausnutzung schneller Busse und Netze
  - Fehlertoleranz
    - zusätzliche Platte für Parity

37

## Problem: Änderungen




- Änderungsgranulat größer gleich Streifengröße
  - Zusätzliche Kosten für eine Änderung 1/(N-1)
- Änderungsgranulat kleiner als Streifengröße
  - Aktualisierung der Parity-Information
    - Alte Parity-Information lesen
    - Neue Parity berechnen
    - Zusätzliche Kosten: 1 Block lesen und 2 Blöcke schreiben
    - Geänderten Datenblock und neue Parity zurückschreiben
- Lösungsansatz
  - Log-based File Systems (Rosenblum und Ousterhout, 1991)
  - Änderungslogs sammeln und “Anhängen”
  - Änderungslogs meist größer als Streifen

38