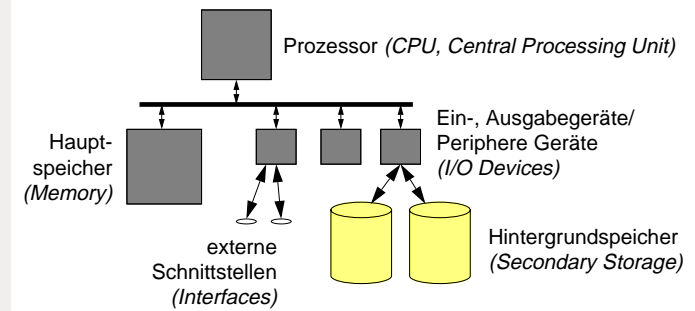


9 Zusammenfassung

- Freispeicherverwaltung
 - ◆ Speicherrepräsentation, Zuteilungsverfahren
- Mehrprogrammbetrieb
 - ◆ Relokation, Ein- und Auslagerung
 - ◆ Segmentierung
 - ◆ Seitenadressierung, Seitenadressierung und Segmentierung, TLB
 - ◆ gemeinsamer Speicher
- Virtueller Speicher
 - ◆ Demand paging
 - ◆ Seitenersetzungsstrategien: FIFO, B₀, LRU, 2nd chance (Clock)
- Seitenflattern
 - ◆ Super-Zustände, Arbeitsmengenmodell

F Implementierung von Dateien

■ Einordnung



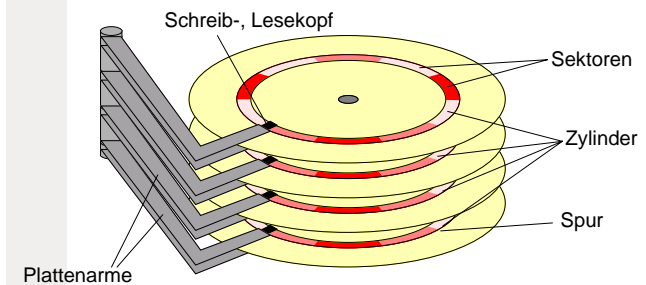
F Implementierung von Dateien

1 Medien

1.1 Festplatten

■ Häufigstes Medium zum Speichern von Dateien

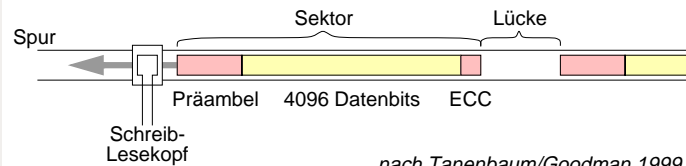
◆ Aufbau einer Festplatte



◆ Kopf schwebt auf Luftpolster

1.1 Festplatten (2)

■ Sektoraufbau



nach Tanenbaum/Goodman 1999

- ◆ Breite der Spur: 5–10 μm
 - ◆ Spuren pro Zentimeter: 800–2000
 - ◆ Breite einzelner Bits: 0,1–0,2 μm
- Zonen
- ◆ Mehrere Zylinder (10–30) bilden eine Zone mit gleicher Sektorenanzahl (bessere Plattenausnutzung)

1.1 Festplatten (4)

■ Zugriffsmerkmale

- ◆ blockorientierter und wahlfreier Zugriff
- ◆ Blockgröße zwischen 32 und 4096 Bytes (typisch 512 Bytes)
- ◆ Zugriff erfordert Positionierung des Schwenkarms auf den richtigen Zylinder und Warten auf den entsprechenden Sektor

■ Blöcke sind üblicherweise numeriert

- ◆ getrennte Numerierung: Zylindernummer, Sektornummer
- ◆ kombinierte Numerierung: durchgehende Nummern über alle Sektoren (Reihenfolge: aufsteigend innerhalb eines Zylinders, dann folgender Zylinder, etc.)

1.1 Festplatten (3)

■ Datenblätter zweier Beispielplatten

Plattentyp	Seagate Medialist	Seagate Cheetah
Kapazität	10,2 GB	36,4 GB
Platten/Köpfe	3/6	12/24
Zylinderzahl	CHS 16383/16/83	9772
Cache	512 kB	4 MB
Positionierzeiten	Spur zu Spur	0,6/0,9 ms
	mittlere	9,5 ms
	maximale	12/13 ms
Transferrate	8,5 MB/s	18,3–28 MB/s
Rotationsgeschw.	5.400 U/min	10.000 U/min
eine Plattenumdrehung	11 ms	6 ms
Stromaufnahme	4,5 W	14 W

1.2 Disketten

■ Ähnlicher Aufbau wie Festplatten

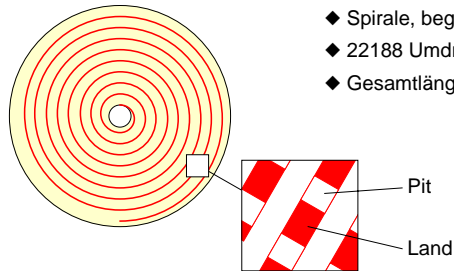
- ◆ maximal zwei Schreib-, Leseköpfe (oben, unten)
- ◆ Kopf berührt Diskettenoberfläche

■ Typische Daten

Diskettentyp	3,5" HD
Kapazität	1,44 MB
Köpfe	2
Spuren	80
Sektoren pro Spur	18
Transferrate	62,5 kB/s
Rotationsgeschw.	300 U/min
eine Umdrehung	200 ms

1.3 CD-ROM

■ Aufbau einer CD



- ◆ Spirale, beginnend im Inneren
- ◆ 22188 Umdrehungen (600 pro mm)
- ◆ Gesamtlänge 5,6 km

- ◆ **Pit:** Vertiefung, die von einem Laser abgetastet werden kann

1.3 CD-ROM (3)

- Kapazität
 - ◆ ca. 650 MB
- Varianten
 - ◆ **CD-R** (Recordable): einmal beschreibbar
 - ◆ **CD-RW** (Rewritable): mehrfach beschreibbar
- DVD (Digital Versatile Disk)
 - ◆ kleinere Pits, engere Spirale, andere Laserlichtfarbe
 - ◆ einseitig oder zweiseitig beschrieben
 - ◆ ein- oder zweischichtig beschrieben
 - ◆ Kapazität: 4,7 bis 17 GB

1.3 CD-ROM (2)

■ Kodierung

- ◆ **Symbol:** ein Byte wird mit 14 Bits kodiert (kann bereits bis zu zwei Bitfehler korrigieren)
- ◆ **Frame:** 42 Symbole werden zusammengefasst (192 Datenbits, 396 Fehlerkorrekturbits)
- ◆ **Sektor:** 98 Frames werden zusammengefasst (16 Bytes Präambel, 2048 Datenbytes, 288 Bytes Fehlerkorrektur)
- ◆ **Effizienz:** 7203 Bytes transportieren 2048 Nutzbytes

■ Transferrate

- ◆ **Single-Speed-Laufwerk:**
75 Sektoren pro Sekunde (153.600 Bytes pro Sekunde)
- ◆ **40-fach-Laufwerk:**
3000 Sektoren pro Sekunde (6.144.000 Bytes pro Sekunde)

2 Speicherung von Dateien

- Dateien benötigen oft mehr als einen Block auf der Festplatte
 - ◆ Welche Blöcke werden für die Speicherung einer Datei verwendet?

2.1 Kontinuierliche Speicherung

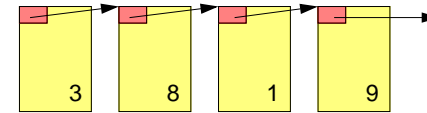
- Datei wird in Blöcken mit aufsteigenden Blocknummern gespeichert
 - ◆ Nummer des ersten Blocks und Anzahl der Folgeblöcke muss gespeichert werden
- ★ Vorteile
 - ◆ Zugriff auf alle Blöcke mit minimaler Positionierzeit des Schwenkarms
 - ◆ Schneller direkter Zugriff auf bestimmter Dateiposition
 - ◆ Einsatz z.B. bei Systemen mit Echtzeitanforderungen

2.1 Kontinuierliche Speicherung (2)

- ▲ Probleme
 - ◆ Finden des freien Platzes auf der Festplatte (Menge aufeinanderfolgender und freier Plattenblöcke)
 - ◆ Fragmentierungsproblem (Verschnitt: nicht nutzbare Plattenblöcke; siehe auch Speicherverwaltung)
 - ◆ Größe bei neuen Dateien oft nicht im Voraus bekannt
 - ◆ Erweitern ist problematisch
 - Umkopieren, falls kein freier angrenzender Block mehr verfügbar

2.2 Verkettete Speicherung

- Blöcke einer Datei sind verkettet



- ◆ z.B. Commodore Systeme (CBM 64 etc.)
 - Blockgröße 256 Bytes
 - die ersten zwei Bytes bezeichnen Spur- und Sektornummer des nächsten Blocks
 - wenn Spurnummer gleich Null: letzter Block
 - 254 Bytes Nutzdaten
- ★ File kann wachsen und verlängert werden

2.1 Kontinuierliche Speicherung (3)

- Variation
 - ◆ Unterteilen einer Datei in Folgen von Blocks (*Chunks*, *Extents*)
 - ◆ Blockfolgen werden kontinuierlich gespeichert
 - ◆ Pro Datei muss erster Block und Länge jedes einzelnen Chunks gespeichert werden
- ▲ Problem
 - ◆ Verschnitt innerhalb einer Folge (siehe auch Speicherverwaltung: interner Verschnitt bei Seitenadressierung)

2.2 Verkettete Speicherung (2)

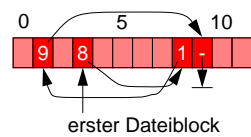
- ▲ Probleme
 - ◆ Speicher für Verzeigerung geht von den Nutzdaten im Block ab (ungünstig im Zusammenhang mit Paging: Seite würde immer aus Teilen von zwei Plattenblöcken bestehen)
 - ◆ Fehleranfälligkeit: Datei ist nicht restaurierbar, falls einmal Verzeigerung fehlerhaft
 - ◆ schlechter direkter Zugriff auf bestimmte Dateiposition
 - ◆ häufiges Positionieren des Schreib-, Lesekopfs bei verstreuten Datenblöcken

2.2 Verkettete Speicherung (3)

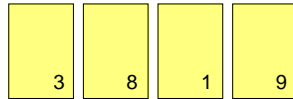
- Verkettung wird in speziellen Plattenblöcken gespeichert

- ◆ FAT-Ansatz (*FAT: File Allocation Table*), z.B. MS-DOS, Windows 95

FAT-Block



Blöcke der Datei: 3, 8, 1, 9



- ★ Vorteile

- ◆ kompletter Inhalt des Datenblocks ist nutzbar (günstig bei Paging)
- ◆ mehrfache Speicherung der FAT möglich: Einschränkung der Fehleranfälligkeit

Systemprogrammierung I

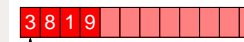
© 1997-2001, Franz J. Hauck, Inf 4, Univ. Erlangen-Nürnberg [F-File.fm, 2001-12-06 09:12]
Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F - 16

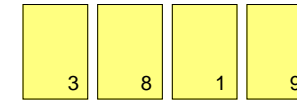
2.3 Indiziertes Speichern

- Spezieller Plattenblock enthält Blocknummern der Datenblöcke einer Datei

Indexblock



Blöcke der Datei: 3, 8, 1, 9



- ▲ Problem

- ◆ feste Anzahl von Blöcken im Indexblock
 - Verschnitt bei kleinen Dateien
 - Erweiterung nötig für große Dateien

Systemprogrammierung I

© 1997-2001, Franz J. Hauck, Inf 4, Univ. Erlangen-Nürnberg [F-File.fm, 2001-12-06 09:12]
Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F - 18

2.2 Verkettete Speicherung (4)

- ▲ Probleme

- ◆ mindestens ein zusätzlicher Block muss geladen werden (Caching der FAT zur Effizienzsteigerung nötig)
- ◆ FAT enthält Verkettungen für alle Dateien: das Laden der FAT-Blöcke lädt auch nicht benötigte Informationen
- ◆ aufwändige Suche nach dem zugehörigen Datenblock bei bekannter Position in der Datei
- ◆ häufiges Positionieren des Schreib-, Lesekopfs bei verstreuten Datenblöcken

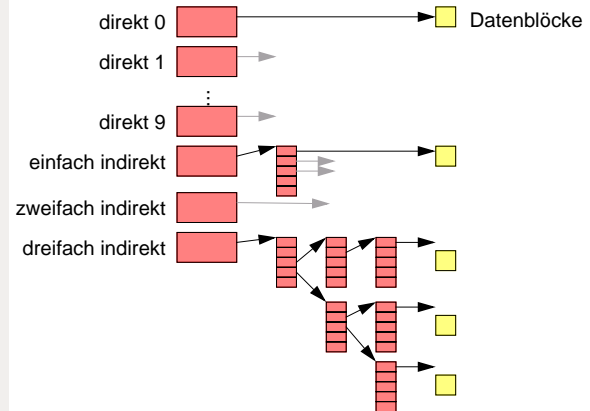
Systemprogrammierung I

© 1997-2001, Franz J. Hauck, Inf 4, Univ. Erlangen-Nürnberg [F-File.fm, 2001-12-06 09:12]
Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F - 17

2.3 Indiziertes Speichern (2)

- Beispiel UNIX Inode



Systemprogrammierung I

© 1997-2001, Franz J. Hauck, Inf 4, Univ. Erlangen-Nürnberg [F-File.fm, 2001-12-06 09:12]
Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

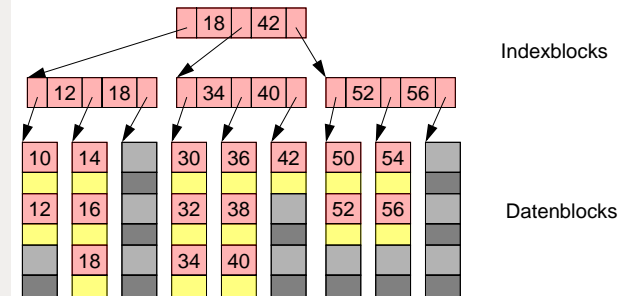
F - 19

2.3 Indiziertes Speichern (3)

- ★ Einsatz von mehreren Stufen der Indizierung
 - ◆ Inode benötigt sowieso einen Block auf der Platte (Verschnitt unproblematisch bei kleinen Dateien)
 - ◆ durch mehrere Stufen der Indizierung auch große Dateien adressierbar
- ▲ Nachteil
 - ◆ mehrere Blöcke müssen geladen werden (nur bei langen Dateien)

2.4 Baumsequentielle Speicherung (2)

- Beispiel eines B*-Baums: Schlüssel sind Integer-Zahlen



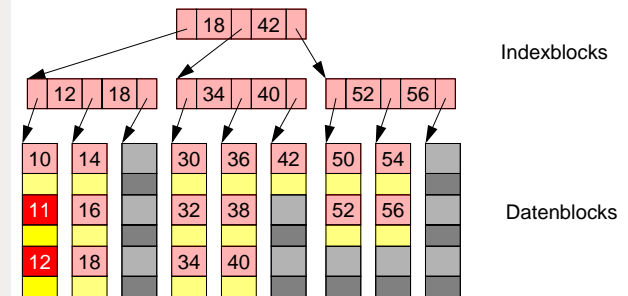
- ◆ Blöcke enthalten Verweis auf nächste Ebene und den höchsten Schlüssel der nächsten Ebene
- ◆ Blöcke der untersten Ebene enthalten Schlüssel und Sätze

2.4 Baumsequentielle Speicherung

- Satzorientierte Dateien
 - ◆ Schlüssel + Datensatz
 - ◆ effizientes Auffinden des Datensatz mit einem bekannten Schlüssel
 - ◆ Schlüsselmenge spärlich besetzt
 - ◆ häufiges Einfügungen und Löschen von Datensätzen
- Einsatz von B-Bäumen zur Satzspeicherung
 - ◆ innerhalb von Datenbanksystemen
 - ◆ als Implementierung spezieller Dateitypen kommerzieller Betriebssysteme
z.B. VSAM-Dateien in MVS (*Virtual Storage Access Method*)
z.B. NTFS Katalogimplementierung

2.4 Baumsequentielle Speicherung (3)

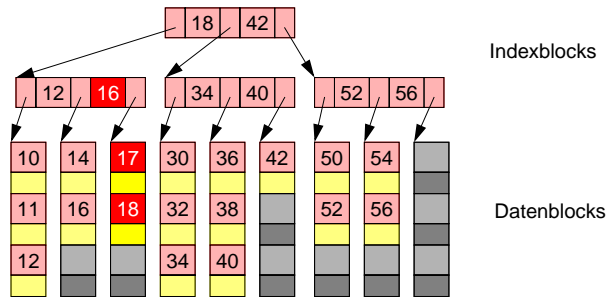
- Einfügen des Satzes mit Schlüssel „11“



- ◆ Satz mit Schlüssel „12“ wird verschoben
- ◆ Satz mit Schlüssel „11“ in freien Platz eingefügt

2.4 Baumsequentielle Speicherung (4)

- Einfügen des Satzes mit Schlüssel „17“



- ◆ Satz mit Schlüssel „18“ wird verschoben (Indexblock wird angepasst)
- ◆ Satz mit Schlüssel „17“ in freien Platz eingefügt

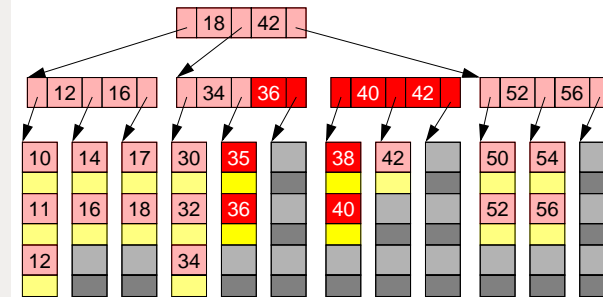
Systemprogrammierung I

© 1997-2001, Franz J. Hauck, Inf 4, Univ. Erlangen-Nürnberg [F-File.fm, 2001-12-06 09:12]
Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F - 24

2.4 Baumsequentielle Speicherung (6)

- Einfügen des Satzes mit Schlüssel „35“ (2. Schritt)



- ◆ Teilung bzw. Erzeugung eines neuen Indexblocks und dessen Verzeigerung

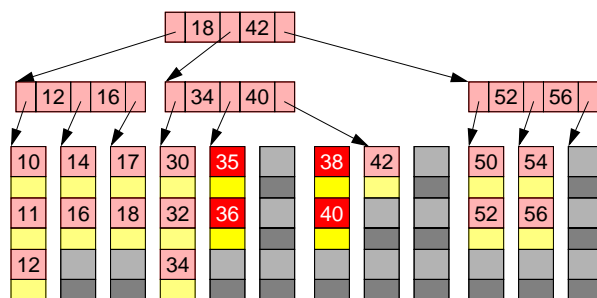
Systemprogrammierung I

© 1997-2001, Franz J. Hauck, Inf 4, Univ. Erlangen-Nürnberg [F-File.fm, 2001-12-06 09:12]
Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F - 26

2.4 Baumsequentielle Speicherung (5)

- Einfügen des Satzes mit Schlüssel „35“ (1. Schritt)



- ◆ Teilung des Blocks mit Satz „36“ und Einfügen des Satzes „35“
- ◆ Anfordern zweier weiterer, leerer Datenblöcke

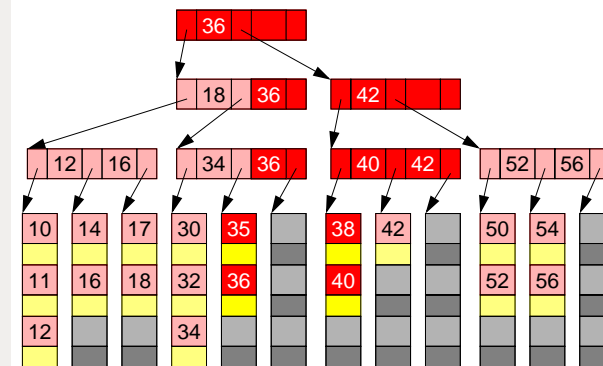
Systemprogrammierung I

© 1997-2001, Franz J. Hauck, Inf 4, Univ. Erlangen-Nürnberg [F-File.fm, 2001-12-06 09:12]
Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F - 25

2.4 Baumsequentielle Speicherung (7)

- Einfügen des Satzes mit Schlüssel „35“ (3. Schritt)



- ◆ Spaltung des alten Wurzelknotens, Erzeugen eines neuer neuen Wurzel

Systemprogrammierung I

© 1997-2001, Franz J. Hauck, Inf 4, Univ. Erlangen-Nürnberg [F-File.fm, 2001-12-06 09:12]
Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F - 27

2.4 Baumsequentielle Speicherung (8)

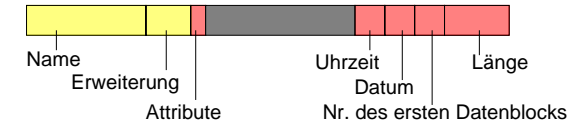
- ★ Effizientes Finden von Sätzen
 - ◆ Baum ist sehr niedrig im Vergleich zur Menge der Sätze
 - viele Schlüssel pro Indexblock vorhanden (je nach Schlüssellänge)
- ★ Gutes Verhalten im Zusammenhang mit Paging
 - ◆ jeder Block entspricht einer Seite
 - ◆ Demand paging sorgt für das automatische Anhäufen der oberen Indexblocks im Hauptspeicher
 - schneller Zugriff auf die Indexstrukturen
- ★ Erlaubt nebenläufige Operationen durch geeignetes Sperren von Indexblöcken
- Löschen erfolgt ähnlich wie Einfügen
 - ◆ Verschmelzen von schlecht belegten Datenblöcken nötig

4 Implementierung von Katalogen

4.1 Kataloge als Liste

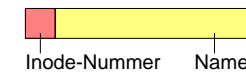
- Einträge gleicher Länge werden hintereinander in eine Liste gespeichert

◆ z.B. *FAT File systems*



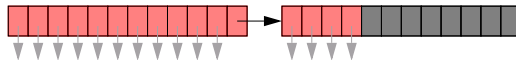
◆ für *VFAT* werden mehrere Einträge zusammen verwendet, um den langen Namen aufzunehmen

◆ z.B. *UNIX System V.3*



3 Freispeicherverwaltung

- Prinzipiell ähnlich wie Verwaltung von freiem Hauptspeicher
 - ◆ Bitvektoren zeigen für jeden Block Belegung an
 - ◆ verkettete Listen repräsentieren freie Blöcke
 - Verkettung kann in den freien Blöcken vorgenommen werden
 - Optimierung: aufeinanderfolgende Blöcke werden nicht einzeln aufgenommen, sondern als Stück verwaltet
 - Optimierung: ein freier Block enthält viele Blocknummern weiterer freier Blöcke und evtl. die Blocknummer eines weiteren Blocks mit den Nummern freier Blöcke



4.1 Kataloge als Liste (2)

- ▲ Problem
 - ◆ Lineare Suche durch die Liste nach bestimmtem Eintrag
 - ◆ Sortierte Liste: binäre Suche, aber Sortieraufwand

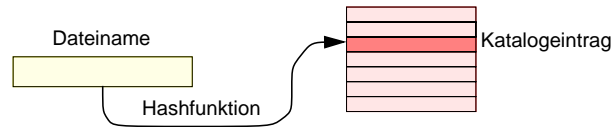
4.2 Einsatz von Hashfunktionen

- Hashing
 - ◆ Spärlich besetzter Schlüsselraum wird auf einen anderen, meist dichter besetzten Schlüsselraum abgebildet
 - ◆ Beispiel: Menge der möglichen Dateinamen wird nach $[0 - N-1]$ abgebildet ($N =$ Länge der Katalogliste)

4.2 Einsatz von Hashfunktionen (2)

■ Hashfunktion

- ◆ Funktion bildet Dateinamen auf einen Index in die Katalogliste ab
schnellerer Zugriff auf den Eintrag möglich (kein lineares Suchen)
- ◆ (einfaches aber schlechtes) Beispiel: $(\sum \text{Zeichen}) \bmod N$



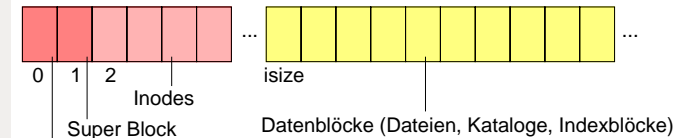
▲ Probleme

- ◆ Kollisionen (mehrere Dateinamen werden auf gleichen Eintrag abgebildet)
- ◆ Anpassung der Listengröße, wenn Liste voll

5 Beispiel: UNIX File Systems

5.1 System V File System

■ Blockorganisation

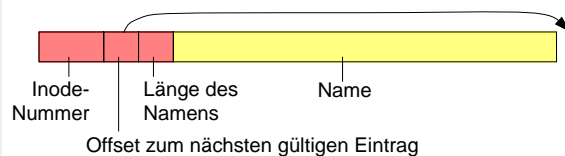


Boot Block

- ◆ Boot Block enthält Informationen zum Laden eines initialen Programms
- ◆ Super Block enthält Verwaltungsinformation für ein Dateisystem
 - Anzahl der Blöcke, Anzahl der Inodes
 - Anzahl und Liste freier Blöcke und freier Inodes
 - Attribute (z.B. *Modified flag*)

4.3 Variabel lange Listenelemente

■ Beispiel BSD 4.2, System V.4, u.a.

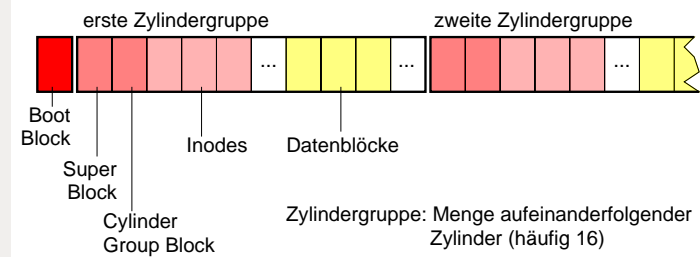


▲ Probleme

- ◆ Verwaltung von freien Einträgen in der Liste
- ◆ Speicherverschnitt (Kompaktifizieren, etc.)

5.2 BSD 4.2 (Berkeley Fast File System)

■ Blockorganisation

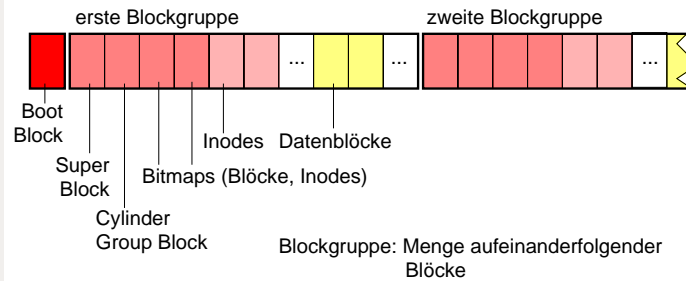


- ◆ Kopie des Super Blocks in jeder Zylindergruppe
- ◆ freie Inodes u. freie Datenblöcke werden im Cylinder group block gehalten
- ◆ eine Datei wird möglichst innerhalb einer Zylindergruppe gespeichert

★ Vorteil: kürzere Positionierungszeiten

5.3 Linux EXT2 File System

Blockorganisation



- ◆ Ähnliches Layout wie BSD FFS
- ◆ Blockgruppen unabhängig von Zylindern

5.4 Block Buffer Cache (2)

Schreiben erfolgt, wenn

- ◆ Datei geschlossen wird,
- ◆ keine freien Puffer mehr vorhanden sind,
- ◆ regelmäßig vom System (*fsflush* Prozess, *update* Prozess),
- ◆ beim Systemaufruf *sync()*,
- ◆ und nach jedem Schreibauftrag im Modus *O_SYNC*.

Adressierung

- ◆ Adressierung eines Blocks erfolgt über ein Tupel: (Gerätenummer, Blocknummer)
- ◆ Über die Adresse wird ein Hashwert gebildet, der eine der möglichen Pufferliste auswählt

5.4 Block Buffer Cache

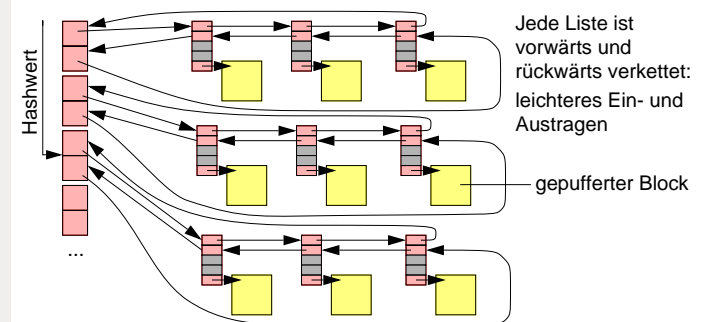
Pufferspeicher für alle benötigten Plattenblocks

- ◆ Verwaltung mit Algorithmen ähnlich wie bei Paging
- ◆ *Read ahead*: beim sequentiellen Lesen wird auch der Transfer des Folgeblocs angestoßen
- ◆ *Lazy write*: Block wird nicht sofort auf Platte geschrieben (erlaubt Optimierung der Schreibzugriffe und blockiert den Schreiber nicht)
- ◆ Verwaltung freier Blöcke in einer Freiliste
 - Kandidaten für Freiliste werden nach LRU Verfahren bestimmt
 - bereits freie aber noch nicht anderweitig benutzte Blöcke können reaktiviert werden (*Reclaim*)

5.4 Block Buffer Cache (3)

Aufbau des Block buffer cache

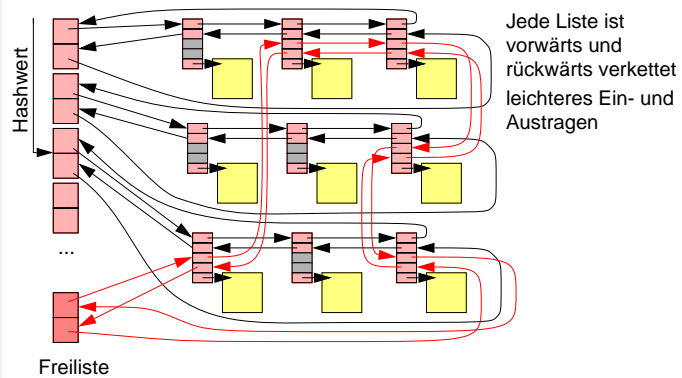
Pufferlisten (Queues)



5.4 Block Buffer Cache (4)

■ Aufbau des Block buffer cache

Pufferlisten (Queues)



5.5 Systemaufrufe

■ Bestimmen der Kachelgröße

```
int getpagesize( void );
```

■ Abbildung von Dateien in den virtuellen Adressraum

◆ Einblenden einer Datei

```
caddr_t mmap( caddr_t addr, size_t len, int prot, int flags,  
             int fd, off_t off );
```

- Einblenden an bestimmte oder beliebige Adresse
- lesbar, schreibbar, ausführbar

◆ Ausblenden einer Datei

```
int munmap( caddr_t addr, size_t len );
```

5.4 Block Buffer Cache (5)

■ Block Buffer Cache teilweise obsolet durch moderne Paging-Systeme

- ◆ Kacheln des Hauptspeichers ersetzen den Block Buffer Cache
- ◆ Kacheln können Seiten aus einem Adressraum und/oder Seiten aus einer Datei beherbergen

▲ Problem

- ◆ Kopieren großer Dateien führt zum Auslagern noch benötigter Adressraumseiten

5.5 Systemaufrufe (2)

◆ Kontrolloperation

```
int mctl( caddr_t addr, size_t len, int func, void *arg );
```

- zum Ausnehmen von Seiten aus dem Paging (Fixieren im Hauptspeicher)
- zum Synchronisieren mit der Datei

6 Beispiel: Windows NT (NTFS)

- File System für Windows NT
- Datei
 - ◆ einfache, unstrukturierte Folge von Bytes
 - ◆ beliebiger Inhalt; für das Betriebssystem ist der Inhalt transparent
 - ◆ dynamisch erweiterbare Dateien
 - ◆ Rechte verknüpft mit NT Benutzern und Gruppen
 - ◆ Datei kann automatisch komprimiert abgespeichert werden
 - ◆ große Dateien bis zu 8.589.934.592 Gigabytes lang
 - ◆ Hard links: mehrere Einträge derselben Datei in verschiedenen Katalogen möglich

6.1 Rechte

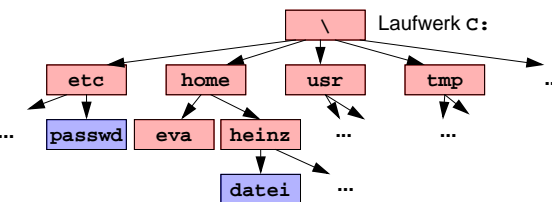
- Eines der folgenden Rechte pro Benutzer oder Benutzergruppe
 - ◆ *no access*: kein Zugriff
 - ◆ *list*: Anzeige von Dateien in Katalogen
 - ◆ *read*: Inhalt von Dateien lesen und *list*
 - ◆ *add*: Hinzufügen von Dateien zu einem Katalog und *list*
 - ◆ *read&add*: wie *read* und *add*
 - ◆ *change*: Ändern von Dateiinhalt, Löschen von Dateien und *read&add*
 - ◆ *full*: Ändern von Eigentümer und Zugriffsrechten und *change*

6 Beispiel: NTFS (2)

- Katalog
 - ◆ baumförmig strukturiert
 - Knoten des Baums sind Kataloge
 - Blätter des Baums sind Dateien
 - ◆ Rechte wie bei Dateien
 - ◆ alle Dateien des Katalogs automatisch komprimierbar
- Partitionen heißen Volumes
 - ◆ Volume wird (in der Regel) durch einen Laufwerksbuchstaben dargestellt
z.B. c:

6.2 Pfadnamen

- Baumstruktur



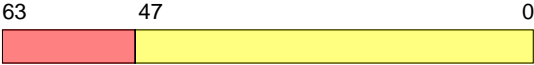
- Pfade

- ◆ wie unter FAT-Filesystem
- ◆ z.B. „C:\home\heinz\datei“, „\tmp“, „C:..\heinz\datei“

6.2 Pfadnamen (2)

- Namenskonvention
 - ◆ 255 Zeichen inklusive Sonderzeichen (z.B. „Eigene Programme“)
 - ◆ automatischer Kompatibilitätsmodus: 8 Zeichen Name, 3 Zeichen Erweiterung, falls „langer Name“ unter MS-DOS ungültig (z.B. `AUTOEXEC.BAT`)
- Kataloge
 - ◆ Jeder Katalog enthält einen Verweis auf sich selbst („.“) und einen Verweis auf den darüberliegenden Katalog im Baum („..“)
 - ◆ Hard links aber keine symbolischen Namen direkt im NTFS

6.3 Dateiverwaltung (2)

- *File-Reference*
 - ◆ Bezeichnet eindeutig eine Datei oder einen Katalog
- 
- 63 47 0
- Sequenz- Dateinummer
nummer
- Dateinummer ist Index in eine globale Tabelle (*MFT: Master File Table*)
 - Sequenznummer wird hochgezählt, für jede neue Datei mit gleicher Dateinummer

6.3 Dateiverwaltung

- Basiseinheit „Cluster“
 - ◆ 512 Bytes bis 4 Kilobytes (beim Formatieren festgelegt)
 - ◆ wird auf eine Menge von hintereinanderfolgenden Blöcken abgebildet
 - ◆ logische Cluster-Nummer als Adresse (LCN)
- Basiseinheit „Strom“
 - ◆ jede Datei kann mehrere (Daten-)Ströme speichern
 - ◆ einer der Ströme wird für die eigentlichen Daten verwendet
 - ◆ Dateiname, MS-DOS Dateiname, Zugriffsrechte, Attribute und Zeitstempel werden jeweils in eigenen Datenströmen gespeichert (leichte Erweiterbarkeit des Systems)

6.4 Master-File-Table

- Rückgrat des gesamten Systems
 - ◆ große Tabelle mit gleich langen Elementen (1KB, 2KB oder 4KB groß, je nach Clustergröße)

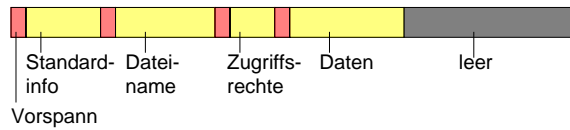
0	
1	
2	
3	
4	
5	
6	
7	
8	
...	

entsprechender Eintrag für eine *File-Reference* enthält Informationen über bzw. die Ströme der Datei

- ◆ Index in die Tabelle ist Teil der *File-Reference*

6.4 Master-File-Table (2)

Eintrag für eine kurze Datei



Ströme

- ◆ Standard Information (immer in der MFT)
 - enthält Länge, MS-DOS Attribute, Zeitstempel, Anzahl der Hard links, Sequenznummer der gültigen File-Reference
- ◆ Dateiname (immer in der MFT)
 - kann mehrfach vorkommen (Hard links, MS-DOS Name)
- ◆ Zugriffsrechte (*Security Descriptor*)
- ◆ Eigentliche Daten

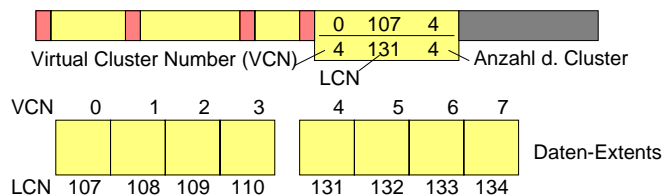
6.4 Master-File-Table (4)

Mögliche weitere Ströme (*Attributes*)

- ◆ Index
 - Index über einen Attributsschlüssel (z.B. Dateinamen) implementiert Katalog
- ◆ Indexbelegungstabelle
 - Belegung der Struktur eines Index
- ◆ Attributliste (immer in der MFT)
 - wird benötigt, falls nicht alle Ströme in einen MFT Eintrag passen
 - referenzieren weitere MFT Einträge und deren Inhalt

6.4 Master-File-Table (3)

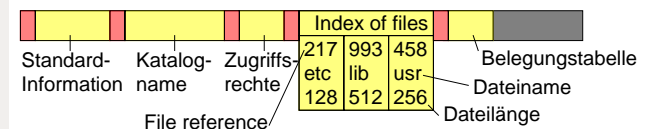
Eintrag für eine längere Datei



- ◆ Extents werden außerhalb der MFT in aufeinanderfolgenden Clustern gespeichert
- ◆ Lokalisierungsinformationen werden in einem eigenen Strom gespeichert

6.4 Master File Table (3)

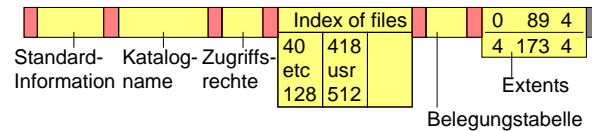
Eintrag für einen kurzen Katalog



- ◆ Dateien des Katalogs werden mit File-References benannt
- ◆ Name und Länge der im Katalog enthaltenen Dateien und Kataloge werden auch im Index gespeichert (doppelter Aufwand beim Update; schnellerer Zugriff beim Kataloglisten)

6.4 Master File Table (4)

Eintrag für einen längeren Katalog



Daten-Extents

VCN	0	1	2	3	4	5	6	7
	918	773	473		873	910		10
	cd	csh	doc		lib	news		tmp
	128	2781	128		512	1024		128
LCN	89	90	91	92	173	174	175	176

File reference
Dateiname
Dateilänge

- ◆ Speicherung als B⁺-Baum (sortiert, schneller Zugriff)
- ◆ in einen Cluster passen zwischen 3 und 15 Dateien (im Bild nur eine)

6.5 Metadaten (2)

Bedeutung der Metadateien

- ◆ MFT und MFT Kopie: MFT wird selbst als Datei gehalten (d.h. Cluster der MFT stehen im Eintrag 0)
MFT Kopie enthält die ersten 16 Einträge der MFT (Fehlertoleranz)
- ◆ Log File: enthält protokollierte Änderungen am Dateisystem
- ◆ Volume Information: Name, Größe und ähnliche Attribute des Volumes
- ◆ Attributtabelle: definiert mögliche Ströme in den Einträgen
- ◆ Wurzelkatalog
- ◆ Clusterbelegungstabelle: Bitmap für jeden Cluster des Volumes
- ◆ Boot File: enthält initiales Programm zum Laden, sowie ersten Cluster der MFT
- ◆ Bad Cluster File: enthält alle nicht lesbaren Cluster der Platte
NTFS markiert automatisch alle schlechten Cluster und versucht die Daten in einen anderen Cluster zu retten

6.5 Metadaten

Alle Metadaten werden in Dateien gehalten

Indexnummer	Metadatei	Bemerkung
0	MFT	Feste Dateien in der MFT
1	MFT Kopie (teilweise)	
2	Log File	
3	Volume Information	
4	Attributtabelle	
5	Wurzelkatalog	
6	Clusterbelegungstabelle	
7	Boot File	
8	Bad Cluster File	
...		
16	Benutzerdateien u. -kataloge	
17		
...		

6.6 Fehlererholung

NTFS ist ein Journaled-File-System

- ◆ Änderungen an der MFT und an Dateien werden protokolliert.
- ◆ Konsistenz der Daten und Metadaten kann nach einem Systemausfall durch Abgleich des Protokolls mit den Daten wieder hergestellt werden.
- ▲ Nachteile
 - ◆ etwas ineffizienter
 - ◆ nur für Volumes >400 MB geeignet

7 Dateisysteme mit Fehlererholung

- Mögliche Fehler
 - ◆ Stromausfall (dummer Benutzer schaltet einfach Rechner aus)
 - ◆ Systemabsturz
- Auswirkungen auf das Dateisystem
 - ◆ inkonsistente Metadaten
 - z.B. Katalogeintrag fehlt zur Datei oder umgekehrt
 - z.B. Block ist benutzt aber nicht als belegt markiert
- ★ Reparaturprogramme
 - ◆ Programme wie **chkdsk**, **scandisk** oder **fsck** können inkonsistente Metadaten reparieren
- ▲ Datenverluste bei Reparatur möglich
- ▲ Große Platten induzieren lange Laufzeiten der Reparaturprogramme

7.1 Journalled-File-Systems (2)

- Protokollierung
 - ◆ Für jeden Einzeltvorgang einer Transaktion wird zunächst ein Logeintrag erzeugt und
 - ◆ danach die Änderung am Dateisystem vorgenommen.
 - ◆ Dabei gilt:
 - Der Logeintrag wird immer **vor** der eigentlichen Änderung auf Platte geschrieben.
 - Wurde etwas auf Platte geändert, steht auch der Protokolleintrag dazu auf der Platte.

7.1 Journalled-File-Systems

- Zusätzlich zum Schreiben der Daten und Meta-Daten (z.B. Inodes) wird ein Protokoll der Änderungen geführt
 - ◆ Alle Änderungen treten als Teil von Transaktionen auf.
 - ◆ Beispiele für Transaktionen:
 - Erzeugen, löschen, erweitern, verkürzen von Dateien
 - Dateiattribute verändern
 - Datei umbenennen
 - ◆ Protokollieren aller Änderungen am Dateisystem zusätzlich in einer Protokolldatei (*Log File*)
 - ◆ Beim Bootvorgang wird Protokolldatei mit den aktuellen Änderungen abgeglichen und damit werden Inkonsistenzen vermieden.

7.1 Journalled-File-Systems (3)

- Fehlererholung
 - ◆ Beim Bootvorgang wird überprüft, ob die protokollierten Änderungen vorhanden sind:
 - Transaktion kann wiederholt bzw. abgeschlossen werden (*Redo*) falls alle Logeinträge vorhanden.
 - Angefangene aber nicht beendete Transaktionen werden rückgängig gemacht (*Undo*).

7.1 Journalled-File-Systems (4)

- Beispiel: Löschen einer Datei im NTFS
 - ◆ Vorgänge der Transaktion
 - Beginn der Transaktion
 - Freigeben der Extents durch Löschen der entsprechenden Bits in der Belegungstabelle (gesetzte Bits kennzeichnen belegten Cluster)
 - Freigeben des MFT Eintrags der Datei
 - Löschen des Katalogeintrags der Datei (evtl. Freigeben eines Extents aus dem Index)
 - Ende der Transaktion
 - ◆ Alle Vorgänge werden unter der File-Reference im Log-File protokolliert, danach jeweils durchgeführt.
 - Protokolleinträge enthalten Informationen zum *Redo* und zum *Undo*

7.1 Journalled-File-Systems (6)

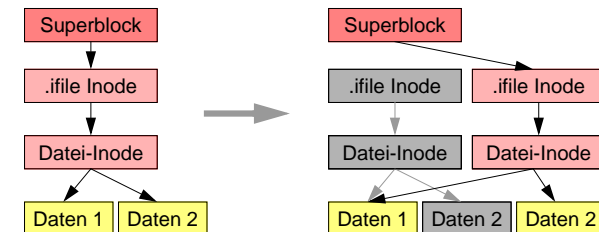
- ★ Ergebnis
 - ◆ eine Transaktion ist entweder vollständig durchgeführt oder gar nicht
 - ◆ Benutzer kann ebenfalls Transaktionen über mehrere Dateizugriffe definieren, wenn diese ebenfalls im Log erfasst werden.
 - ◆ keine inkonsistenten Metadaten möglich
 - ◆ Hochfahren eines abgestürzten Systems benötigt nur den relativ kurzen Durchgang durch das Log-File.
 - Alternative **chkdsk** benötigt viel Zeit bei großen Platten
- ▲ Nachteile
 - ◆ ineffizienter, da zusätzliches Log-File geschrieben wird
- Beispiele: NTFS, EXT3, ReiserFS

7.1 Journalled-File-Systems (5)

- ◆ Log vollständig (Ende der Transaktion wurde protokolliert und steht auf Platte):
 - *Redo* der Transaktion:
alle Operationen werden wiederholt, falls nötig
- ◆ Log unvollständig (Ende der Transaktion steht nicht auf Platte):
 - *Undo* der Transaktion:
in umgekehrter Reihenfolge werden alle Operation rückgängig gemacht
- Checkpoints
 - ◆ Log-File kann nicht beliebig groß werden
 - ◆ gelegentlich wird für einen konsistenten Zustand auf Platte gesorgt (*Checkpoint*) und dieser Zustand protokolliert (alle Protokolleinträge von vorher können gelöscht werden)
 - ◆ Ähnlich verfährt NTFS, wenn Ende des Log-Files erreicht wird.

7.2 Log-Structured-File-Systems

- Alle Änderungen im Dateisystem erfolgen auf Kopien
 - ◆ Der Inhalt veränderter Blöcke wird in einen neuen Block geschrieben



- ◆ Beispiel LinLogFS: Superblock einziger statischer Block (Anker im System)

7.2 Log-Structured-File-Systems (2)

- ★ Vorteile
 - ◆ Datenkonsistenz bei Systemausfällen
 - ein atomare Änderung macht alle zusammengehörigen Änderungen sichtbar
 - ◆ Schnappschüsse / Checkpoints einfach realisierbar
 - ◆ Gute Schreibeffizienz
 - Alle zu schreibenden Blöcke werden kontinuierlich geschrieben
- ▲ Nachteile
 - ◆ Gesamtperformanz geringer
- Beispiele: LinLogFS, BSD LFS, AIX XFS

9 Fehlerhafte Plattenblöcke

- Blöcke, die beim Lesen Fehlermeldungen erzeugen
 - ◆ z.B. Prüfsummenfehler
- Hardwarelösung
 - ◆ Platte und Plattencontroller bemerken selbst fehlerhafte Blöcke und maskieren diese aus
 - ◆ Zugriff auf den Block wird vom Controller automatisch auf einen „gesunden“ Block umgeleitet
- Softwarelösung
 - ◆ File-System bemerkt fehlerhafte Blöcke und markiert diese auch als belegt

8 Limitierung der Plattennutzung

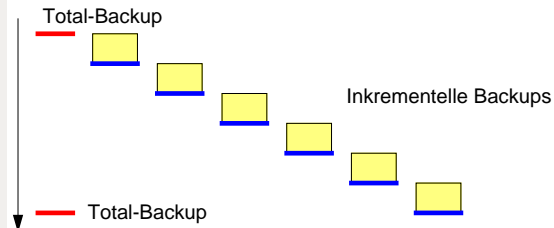
- Mehrbenutzersysteme
 - ◆ einzelnen Benutzern sollen verschieden große Kontingente zur Verfügung stehen
 - ◆ gegenseitige Beeinflussung soll vermieden werden (*Disk-full* Fehlermeldung)
- Quota-Systeme (Quantensysteme)
 - ◆ Tabelle enthält maximale und augenblickliche Anzahl von Blöcken für die Dateien und Kataloge eines Benutzers
 - ◆ Tabelle steht auf Platte und wird vom File-System fortgeschrieben
 - ◆ Benutzer erhält *Disk-full* Meldung, wenn sein Quota verbraucht ist
 - ◆ üblicherweise gibt es eine weiche und eine harte Grenze (weiche Grenze kann für eine bestimmte Zeit überschritten werden)

10 Datensicherung

- Schutz vor dem Totalausfall von Platten
 - ◆ z.B. durch Head-Crash oder andere Fehler
- Sichern der Daten auf Tertiärspeicher
 - ◆ Bänder
 - ◆ WORM Speicherplatten (*Write Once Read Many*)
- Sichern großer Datenbestände
 - ◆ Total-Backups benötigen lange Zeit
 - ◆ Inkrementelle Backups sichern nur Änderungen ab einem bestimmten Zeitpunkt
 - ◆ Mischen von Total-Backups mit inkrementellen Backups

10.1 Beispiele für Backup Scheduling

■ Gestaffelte inkrementelle Backups

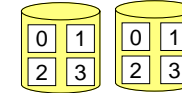


- ◆ z.B. alle Woche ein Total-Backup und jeden Tag ein inkrementelles Backup zum Vortag: maximal 7 Backups müssen eingespielt werden

10.2 Einsatz mehrere redundanter Platten

■ Gespiegelte Platten (*Mirroring*; RAID 0)

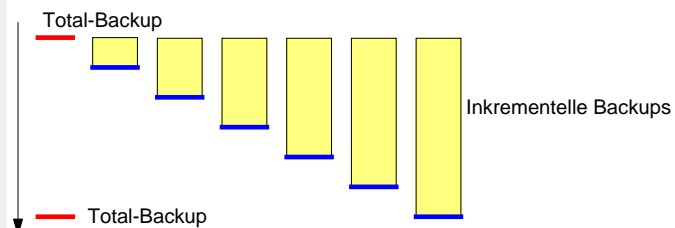
- ◆ Daten werden auf zwei Platten gleichzeitig gespeichert



- ◆ Implementierung durch Software (File-System, Plattentreiber) oder Hardware (spez. Controller)
 - ◆ eine Platte kann ausfallen
 - ◆ schnelleres Lesen (da zwei Platten unabhängig voneinander beauftragt werden können)
- ▲ **Nachteil**
- ◆ doppelter Speicherbedarf
 - ◆ wenig langsames Schreiben durch Warten auf zwei Plattentransfers

10.1 Beispiele für Backup Scheduling (2)

■ Gestaffelte inkrementelle Backups zum letzten Total-Backup



- ◆ z.B. alle Woche ein Total-Backup und jeden Tag ein inkrementelles Backup zum letzten Total-Backup: maximal 2 Backups müssen eingespielt werden

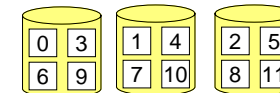
■ Hierarchie von Backup-Läufen

- ◆ mehrstufige inkrementelle Backups zum Backup der nächst höheren Stufe
- ◆ optimiert Archivmaterial und Restaurierungszeit

10.2 Einsatz mehrere redundanter Platten (2)

■ Gestreifte Platten (*Striping*; RAID 1)

- ◆ Daten werden über mehrere Platten gespeichert

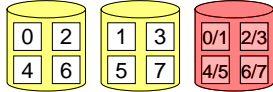


- ◆ Datentransfers sind nun schneller, da mehrere Platten gleichzeitig angesprochen werden können
- ▲ **Nachteil**
- ◆ keinerlei Datensicherung: Ausfall einer Platte lässt Gesamtsystem ausfallen
- Verknüpfung von RAID 0 und 1 möglich (RAID 0+1)

10.2 Einsatz mehrere redundanter Platten (3)

■ Paritätsplatte (RAID 4)

- ◆ Daten werden über mehrere Platten gespeichert, eine Platte enthält Parität

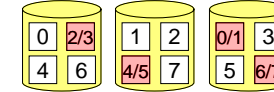


- ◆ Paritätsblock enthält byteweise XOR-Verknüpfungen von den zugehörigen Blöcken aus den anderen Streifen
- ◆ eine Platte kann ausfallen
- ◆ schnelles Lesen
- ◆ prinzipiell beliebige Plattenanzahl (ab drei)

10.2 Einsatz mehrere redundanter Platten (5)

■ Verstreuter Paritätsblock (RAID 5)

- ◆ Paritätsblock wird über alle Platten verstreut



- ◆ zusätzliche Belastung durch Schreiben des Paritätsblocks wird auf alle Platten verteilt
- ◆ heute gängigstes Verfahren redundanter Platten
- ◆ Vor- und Nachteile wie RAID 4

10.2 Einsatz mehrerer redundanter Platten (4)

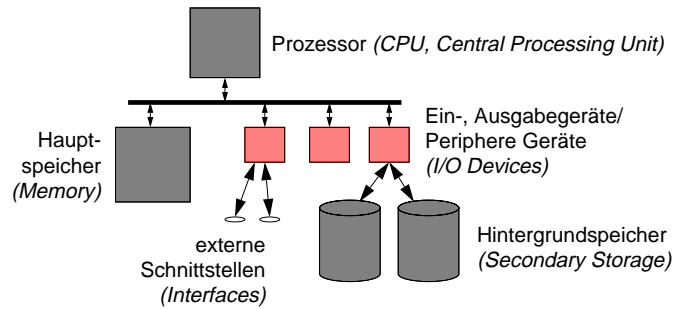
▲ Nachteil von RAID 4

- ◆ jeder Schreibvorgang erfordert auch das Schreiben des Paritätsblocks
- ◆ Erzeugung des Paritätsblocks durch Speichern des vorherigen Blockinhalts möglich: $P_{neu} = P_{alt} \oplus B_{alt} \oplus B_{neu}$ (P=Parity, B=Block)
- ◆ Schreiben eines kompletten Streifens benötigt nur einmaliges Schreiben des Paritätsblocks
- ◆ Paritätsplatte ist hoch belastet (meist nur sinnvoll mit SSD [Solid state disk])

G Ein-, Ausgabe

G Ein- und Ausgabe

■ Einordnung



Systemprogrammierung I

© 1997-2001, Franz J. Hauck, Inf 4, Univ. Erlangen-Nürnberg [G-InOut.fm, 2001-12-06 09:13]
Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

G - 2

1.1 Geräterepäsentation in UNIX

- Periphere Geräte werden als Spezialdateien repräsentiert
 - ◆ Geräte können wie Dateien mit Lese- und Schreiboperationen angesprochen werden
 - ◆ Öffnen der Spezialdateien schafft eine Verbindung zum Gerät, die durch einen Treiber hergestellt wird
 - ◆ direkter Durchgriff vom Anwender auf den Treiber
- Blockorientierte Spezialdateien
 - ◆ Plattenlaufwerke, Bandlaufwerke, Floppy Disks, CD-ROMs
- Zeichenorientierte Spezialdateien
 - ◆ Serielle Schnittstellen, Drucker, Audiokanäle etc.
 - ◆ blockorientierte Geräte haben meist auch eine zusätzliche zeichenorientierte Repräsentation

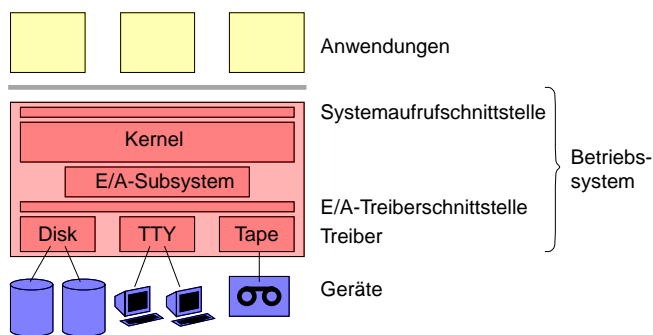
Systemprogrammierung I

© 1997-2001, Franz J. Hauck, Inf 4, Univ. Erlangen-Nürnberg [G-InOut.fm, 2001-12-06 09:13]
Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

G - 4

1 Gerätezugang und Treiber

■ Schichtung der Systemsoftware bis zum Gerät



Nach Vahalia, 1996

Systemprogrammierung I

© 1997-2001, Franz J. Hauck, Inf 4, Univ. Erlangen-Nürnberg [G-InOut.fm, 2001-12-06 09:13]
Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

G - 3

1.1 Geräterepäsentation in UNIX (2)

- Eindeutige Beschreibung der Geräte durch ein Tupel: (Gerätetyp, Major Number, Minor Number)
 - ◆ Gerätetyp: Block Device, Character Device
 - ◆ Major Number: Auswahlnummer für einen Treiber
 - ◆ Minor Number: Auswahl eines Gerätes innerhalb eines Treibers

Systemprogrammierung I

© 1997-2001, Franz J. Hauck, Inf 4, Univ. Erlangen-Nürnberg [G-InOut.fm, 2001-12-06 09:13]
Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

G - 5

1.1 Geräterepäsentation in UNIX (3)

■ Beispiel eines Kataloglisting von /dev (Ausschnitt)

```
crw----- 1 fzhauck 108, 0 Oct 16 1996 audio
crw----- 1 fzhauck 108,128 Oct 16 1996 audioc1
crw-rw-rw- 1 root 21, 0 May 3 1996 conslog
brw-rw-rw- 1 root 36, 2 Oct 16 1996 fd0
crw----- 1 fzhauck 17, 0 Oct 16 1996 mouse
crw-rw-rw- 1 root 13, 2 Jan 13 09:09 null
crw-rw-rw- 1 root 36, 2 Jul 2 1997 rfd0
crw-r----- 1 root 32, 0 Oct 16 1996 rsd3a
crw-r----- 1 root 32, 1 Oct 16 1996 rsd3b
crw-r----- 1 root 32, 2 Oct 16 1996 rsd3c
brw-r----- 1 root 32, 0 Oct 16 1996 sd3a
brw-r----- 1 root 32, 1 Oct 16 1996 sd3b
brw-r----- 1 root 32, 2 Oct 16 1996 sd3c
crw-rw-rw- 1 root 22, 0 Sep 19 09:11 tty
crw-rw-rw- 1 root 29, 0 Oct 16 1996 ttya
crw-rw-rw- 1 root 29, 1 Oct 16 1996 ttyb
```

1.1 Geräterepäsentation in UNIX (5)

■ Funktionen eines Block device-Treibers

- ◆ `d_open`: Öffnen des Gerätes
- ◆ `d_close`: Schließen des Gerätes
- ◆ `d_strategy`: Abgeben von Lese- und Schreibaufträgen auf Blockbasis
- ◆ `d_size`: Ermitteln der Gerätegröße (z.B. Partitions- oder Plattengröße)
- ◆ `d_xhalt`: Abschalten des Gerätes
- ◆ u.a.

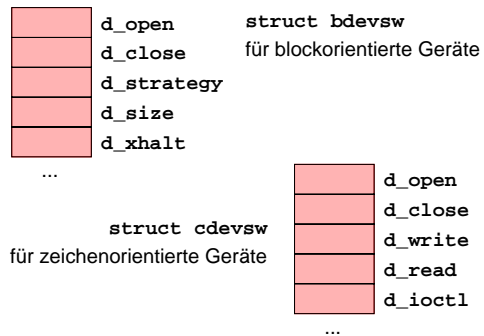
■ Funktionen eines Character device-Treibers

- ◆ `d_open`, `d_close`: Öffnen und Schließen des Gerätes
- ◆ `d_read`, `d_write`: Lesen und Schreiben von Zeichen
- ◆ `d_ioctl`: generische Kontrolloperation
- ◆ u.a.

1.1 Geräterepäsentation in UNIX (4)

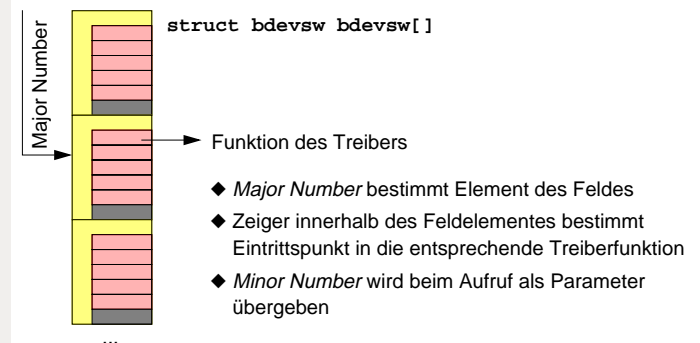
■ Interne Treiberschnittstelle

◆ Vektor von Funktionszeigern pro Treiber (Major Number):



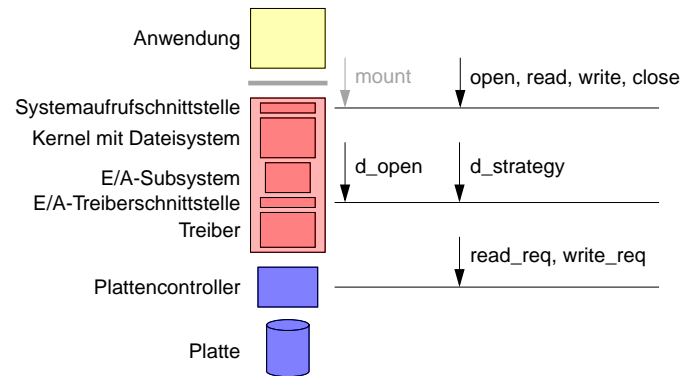
1.1 Geräterepäsentation in UNIX (6)

■ Felder für den Aufruf von Treibern (`bdevsw[]` und `cdevsw[]`)



2 Plattentreiber

- Software und Hardware zwischen Anwender und Platte

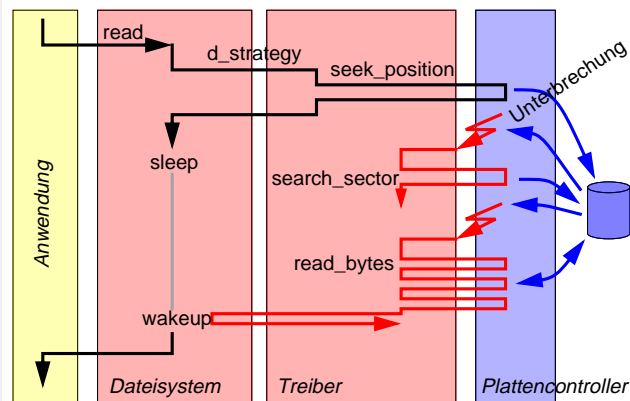


2.1 Einfacher Plattentreiber (2)

- ◆ Anwendung führt `read()` Systemaufruf aus.
- ◆ Dateisystem prüft, ob entsprechender Block im Speicher vorhanden.
- ◆ Falls der Block nicht vorhanden ist, wird ein Speicherplatz bereitgestellt und `d_strategy` im entsprechenden Treiber aufgerufen.
- ◆ Die Ausführung von `d_strategy` stößt Plattenpositionierung an.
- ◆ Die Anwendung blockiert sich im Kern. System kann andere Prozesse ablaufen lassen.
- ◆ Plattencontroller meldet sich bei erfolgter Positionierung durch eine Unterbrechung.
- ◆ Unterbrechungsbehandlung stößt Sektorsuche an.
- ◆ In erneuter Unterbrechung nach gefundenem Sektor werden die Daten im Pollingbetrieb eingelesen.
- ◆ Schließlich wird der Anwendungsprozess wieder aufgeweckt (in den Zustand bereit überführt).

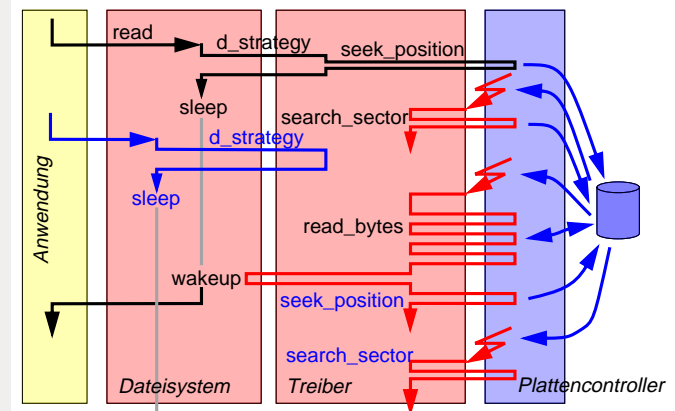
2.1 Einfacher Treiber

- Ablauf eines Leseaufrufs



2.1 Einfacher Plattentreiber (3)

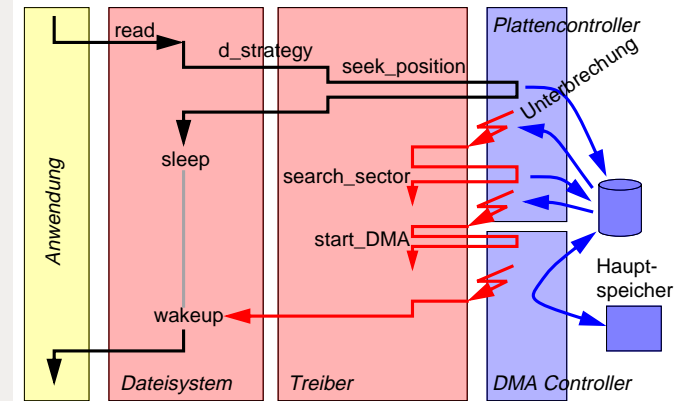
- Ablauf mehrerer Leseaufrufe



2.1 Einfacher Plattentreiber

- Unterbrechungsbehandlung ist auch für weitere Aufträge zuständig
 - ◆ Ist der Auftrag abgeschlossen, muss die Unterbrechungsbehandlung den nächsten Auftrag auswählen und aufsetzen, da der zugehörige Prozess bereits blockiert ist.
 - ◆ Die Unterbrechungen laufender Aufträge sorgen für die Abwicklung der folgenden Aufträge.

2.2 Treiber mit DMA (2)



2.2 Treiber mit DMA

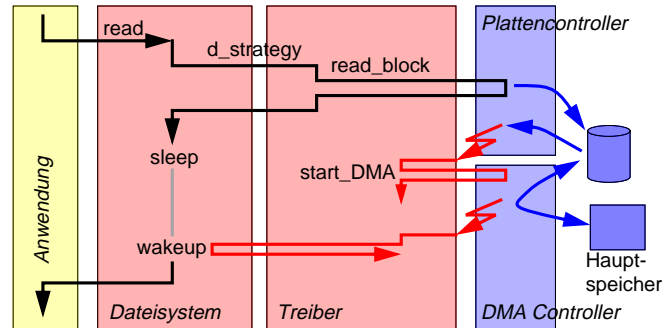
- DMA (*Direct Memory Access*) erlaubt Einlesen und Schreiben ohne Prozessorbeteiligung
 - ◆ DMA Controller erhält verschiedene Parameter:
 - die Hauptspeicheradresse zum Abspeichern bzw. Auslesen eines Plattenblocks
 - die Adresse des Plattencontrollers zum Abholen bzw. Abgeben der Daten
 - die Länge der zu transferierenden Daten
 - ◆ DMA Controller löst bei Fertigstellung eine Unterbrechung aus
- ★ Vorteile
 - ◆ Prozessor muss Zeichen eines Plattenblocks nicht selbst abnehmen (kein Polling sondern Interrupt)
 - ◆ Plattentransferzeit kann zum Ablauf anderer Prozesse genutzt werden

2.2 Treiber mit DMA (3)

- Große Systeme mit mehreren DMA-Kanälen und vielen Platten
 - ◆ es muss ein freier DMA-Kanal gesucht werden und evtl. auf einen freien gewartet werden bevor der Auftrag ausgeführt werden kann
 - ◆ Anforderung kann parallel zur Plattenpositionierung erfolgen
- Mainframe-Systeme
 - ◆ Steuereinheit fasst mehrere Platten zu einem Gerät zusammen
 - ◆ mehrere Steuereinheiten hängen an einem Kanal zum Hauptspeicher
 - ◆ zum Zugriff auf die eigentliche Platte muss erst die Steuereinheit und dann der Kanal belegt werden (Teilwegbelegung)
- DMA und Caching
 - ◆ heutige Prozessoren arbeiten mit Datencaches
 - ◆ DMA läuft am Cache vorbei: Betriebssystem muss vor dem Aufsetzen von DMA-Transfers Caches zurückschreiben und invalidieren

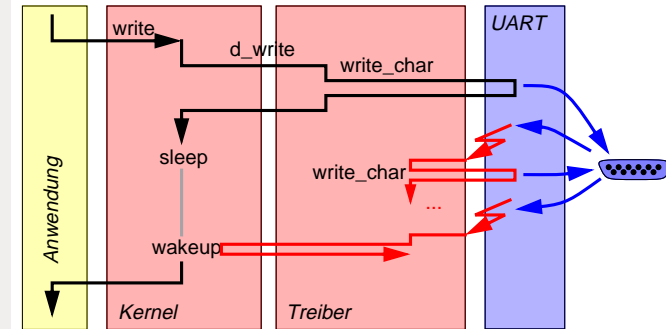
2.3 Treiber für intelligente Platte

- Intelligente Platten besitzen eigenen Prozessor für
 - ◆ das Umsortieren von Aufträgen (interne Plattenstrategie)
 - ◆ eigene Bad block-Erkennung, etc.



3.1 TTY-Treiber

- TTY-Treiber (*Teletype*, Fernschreiber) und der Ablauf eines Schreibaufrufs



- ◆ UART = Universal Asynchronous Receiver / Transmitter

3 Treiber für serielle Schnittstellen

- Einsatz serieller Schnittstellen (z.B. RS-232)
 - ◆ Terminals
 - ◆ Drucker
 - ◆ Modems
- Datenübertragung
 - ◆ zeichenweise seriell (z.B. Startbit, Datenbits, Stopbits)
 - ◆ getaktet in bestimmter Geschwindigkeit (Bitrate, z.B. 38.400 Bit/s), im Vergleich zu Platten relativ langsam
 - ◆ Flusskontrolle (d.h. Empfänger kann Datenfluss bremsen)
 - ◆ bidirektional
- Treiber
 - ◆ zeichenorientiertes Gerät
 - ◆ vom Prinzip her ähnlich dem Plattentreiber

3.1 TTY-Treiber (2)

- Enger Zusammenhang zwischen Ein- und Ausgabe
 - ◆ Echofunktion (getippte Zeichen werden angezeigt)
 - eingelesene Zeichen werden gleich wieder ausgegeben
 - ◆ Flusskontrolle (bestimmtes Zeichen in der Eingabe hält Ausgabe an: ^S)
 - wird ^S eingelesen wird Ausgabe angehalten bis ^Q eingelesen wird
- Zeilenorientierte Treiber
 - ◆ Anwendung will Zeichen zeilenweise, z.B. Shell
 - ◆ Treiber blockiert Prozess bis Zeilenende erkannt
 - ◆ Treiber erlaubt das Editieren der Zeile (Backspace, etc.)
- Signale
 - ◆ bestimmte Zeichen lösen Signale an korrespondierende Prozesse aus

3.2 TTY-Treiber in UNIX

- Konfigurierbar
 - ◆ Repräsentation einer seriellen Schnittstellen als zeichenorientiertes Gerät
 - ◆ durch Aufruf von `ioctl` kann Treiber konfiguriert werden

```
int ioctl( int fildes, int request, /* arg */ );
```
 - ◆ Kommando zum Lesen der Konfiguration: Übergabe einer Strukturadresse

```
struct termios t;
ioctl( fd, TCGETS, &t );
```
 - ◆ Kommando zum Schreiben einer Konfiguration:

```
ioctl( fd, TCSETS, &t );
```
 - ◆ Struktur enthält Bitfelder für verschiedene Einstellungen
 - ◆ Bitmasken sind als Makros verfügbar
 - ◆ näheres: „`man termios`“ und „`man ioctl`“

3.4 Einstellung der Ein-, Ausgabeverarbeitung

- Festlegen der Zeichen mit Sonderbedeutung
 - ◆ Erase-Character: löscht letztes Zeichen (Backspace)
 - ◆ Kill-Character: löscht ganze Zeile (^K)
- Eingabeverarbeitung
 - ◆ `ICRNL` CR-Zeichen wird als NL-Zeichen gelesen
 - ◆ `ICANON` kanonische Eingabeverarbeitung (Zeileneditierung)
 - ◆ `IXON` erlaube Flusskontrolle mit ^Q und ^S
- Ausgabeverarbeitung
 - ◆ `ECHO` schaltet Echofunktion ein
 - ◆ `ECHOE` Echo von Backspace als Backspace, Leerzeichen, Backspace
 - ◆ `ONLCR` NL-Zeichen wird als CR, NL ausgegeben

3.3 Einstellung der physikalischen Parameter

- Bitrate einer seriellen Schnittstelle
 - ◆ `B2400` 2400 Bit/s
 - ◆ `B4800` 4800 Bit/s
 - ◆ `B9600` 9600 Bit/s
 - ◆ `B19200` 19200 Bit/s
 - ◆ `B38400` 38400 Bit/s
 - ◆ `B57600` 57600 Bit/s
- Zeichengröße, Parität, Stoppbits
 - ◆ `CS7` 7 Bits
 - ◆ `CSTOPB` zwei Stoppbits sonst eins
 - ◆ `PARENB` Parität einschalten
 - ◆ `CRTSCTS` Hardware-basierte Flusskontrolle einschalten

3.5 Signalauslösung und Jobkontrolle

- Signalauslösung
 - ◆ `ISIG`: Schaltet Signale ein
 - ◆ `INTR`-Zeichen: sendet `SIGINT`-Signal (^C)
 - ◆ `QUIT`-Zeichen: sendet `SIGQUIT`-Signal (^|)
- Signal wird an ganze Prozessgruppe geschickt
 - ◆ alle Prozesse der Gruppe empfangen Signal
 - ◆ Beispiel: `cat /etc/passwd | grep Mueller | sort`
 - ◆ alle Prozesse erhalten `SIGINT` bei ^C
- Prozessgruppe
 - ◆ Prozessgruppen-ID wird wie eine Prozess-ID (PID) bezeichnet
 - ◆ Prozess mit gleicher PID und Prozessgruppen-ID ist Gruppenführer
 - ◆ Shell sorgt dafür, dass im Beispiel `cat`, `grep` und `sort` in der gleichen Prozessgruppe sind (`sort` wird Gruppenführer)

3.5 Signalauslösung und Jobkontrolle (2)

- Vordergrund- und Hintergrundprozesse
 - ◆ Hintergrundprozesse erhalten keine Signale.
 - ◆ Bei Shells mit Jobkontrolle kann zwischen Vorder- und Hintergrundprozessen umgeschaltet werden.
- Sessions
 - ◆ Shell öffnet eine Session, die mehrere Prozessgruppen enthalten kann (spezieller systemabhängiger Systemaufruf).
 - ◆ Shell wird Sessionführer.
 - ◆ Shell erzeugt Prozesse und Prozessgruppen.
 - ◆ Ein TTY wird Controlling-Terminal für alle Prozessgruppen der Session.
 - ◆ Unterbrechen der Terminalverbindung (**SIGTSTP**) wird dem Sessionführer zugestellt.

3.5 Signalzustellung und Jobkontrolle (4)

- Beispiel: Stoppen und wiederaufnehmen eines Vordergrundprozesses

```
prompt> cc -o test.c
^Z
Suspended
prompt> jobs
[1] Suspended cc -o test.c
prompt> fg %1
```

- ◆ Realisiert mit einem Signal namens **SIGTSTP**, das die Prozessgruppe stoppt
- ◆ Shell bekommt dies mit über ein `waitpid()`
- ◆ Shellkommando `fg` sendet ein Signal **SIGCONT** und die Prozesse fahren fort

3.5 Signalauslösung und Jobkontrolle (3)

- Vordergrundprozess
 - ◆ Eine Prozessgruppe der Session kann zur Vordergrundprozessgruppe gemacht werden.
 - ◆ **SIGINT** und **SIGQUIT** sowie die Eingabe vom Terminal werden nur der Vordergrundprozessgruppe zugestellt.
- Hintergrundprozesse
 - ◆ Alle Hintergrundprozesse bekommen keine Eingabe vom Terminal und werden gestoppt, wenn sie lesen wollen (Shell wird benachrichtigt).
- Jobkontrolle
 - ◆ Shell kann zwischen Vorder- und Hintergrundprozessgruppen umschalten
 - ◆ Benutzer kann Vordergrundprozesse stoppen und gelangt zur Shell zurück

3.5 Signalzustellung und Jobkontrolle (5)

- Beispiel: Stoppen eines Vordergrundprozesses, Umwandlung in einen Hintergrundprozess

```
prompt> cc -o test.c
^Z
Suspended
prompt> bg
[1] Running cc -o test.c
prompt>
```

- ◆ Wie auf vorheriger Folie, aber:
Shell schaltet die Prozessgruppe in den Hintergrund und wartet nicht mehr auf deren Beendigung.

3.5 Signalzustellung und Jobkontrolle (6)

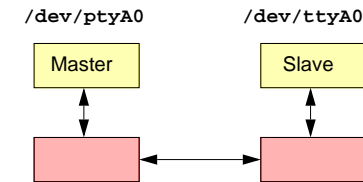
- Beispiel: Starten eines Hintergrundprozesses und Umwandlung in einen Vordergrundprozess

```
prompt> cc -o test.c &
prompt> jobs
[1] Running cc -o test.c
prompt> fg %1
```

- ◆ Shell startet eine Hintergrundprozessgruppe und nimmt Kommandos entgegen
- ◆ `fg` Kommando schaltet die Hintergrundgruppe in eine Vordergrundprozessgruppe um und wartet auf deren Beendigung mit `waitpid()`

3.6 Pseudo-Terminals (2)

- Master- und Slavesidee sehen wie ein normales TTY-Device aus



- ◆ Slavesidee besitzt Modul zur Flusskontrolle, Eingabeeditierung, Signalzustellung, Flusskontrolle etc.

3.6 Pseudo-Terminals

- Pseudo-TTY-Treiber (*PTY*)
 - ◆ keine echte serielle Schnittstelle vorhanden
 - ◆ Shell und andere Prozesse benötigen aber ein TTY für
 - Flusskontrolle,
 - Echofunktion,
 - Job-Kontrolle etc.
 - ◆ fungiert als gewohnte Schnittstelle von Anwendungsprozessen
 - ◆ Einsatz beispielsweise bei einem Fenstersystem (xterm-Programm)
 - xterm-Programm bedient die Masterseite eines PTY
 - Shell und Anwendungsprogramme sehen xterm-Fenster wie ein TTY (Slavesidee)

3.7 Warten auf mehrere Ereignisse

- Bisher: Lese- oder Schreibaufufe blockieren
 - ◆ Was tun beim Lesen von mehreren Quellen?
- Alternative 1: nichtblockierende Ein-, Ausgabe
 - ◆ `O_NDELAY` beim `open()`
 - ◆ Pollingbetrieb: Prozess muss immer wieder `read()` aufrufen, bis etwas vorliegt

3.7 Warten auf mehrere Ereignisse (2)

- Alternative 2: Blockieren an mehreren Filedeskriptoren
 - ◆ Systemaufruf:

```
int select( int nfd, fd_set *readfds, fd_set *writefds, fd_set *errorfds, struct timeval *timeout);
```
 - ◆ `nfd` legt fest, bis zu welchem Filedeskriptor `select` wirken soll.
 - ◆ `xxxfds` sind Filedeskriptoren, auf die gewartet werden soll:
 - `readfds` — bis etwas zum Lesen vorhanden ist
 - `writefds` — bis man schreiben kann
 - `errorfds` — bis ein Fehler aufgetreten ist
 - ◆ Timeout legt fest, wann der Aufruf spätestens deblockiert.
 - ◆ Makros zum Erzeugen der Filedeskriptormengen
 - ◆ Ergebnis: in den Filedeskriptormengen sind nur noch die Filedeskriptoren vorhanden, die zur Deblockade führten

5 Netzwerktreiber

- Beispiel: Ethernet
 - ◆ schneller serieller Bus mit CSMA/CD (*Carrier sense media access / Collision detect*)
zu deutsch: es wird dann gesendet, wenn nicht gerade jemand anderes sendet; Kollisionen werden erkannt und aufgelöst
 - ◆ spezieller Netzwerkchip
 - implementiert unterstes Kommunikationsprotokoll
 - erkennt eintreffende Pakete
- Netzwerktreiber
 - ◆ wird von höheren Protokollen innerhalb des Betriebssystems angesprochen, z.B. von der IP-Schicht

4 Bildschirmtreiber

- Bildspeicher
 - ◆ zeichenorientiert
 - ◆ pixelorientiert
- Aufgaben des Treibers
 - ◆ Bereitstellen von Graphikprimitiven (z.B. Ausgabe von Text, Zeichnen von Rechtecken, etc.)
 - ◆ Ansprechen von Graphikprozessoren (schnelle Verschiebeoperationen, komplexe Zeichenoperationen, 3D Rendering, Textures)
 - ◆ Einblenden des Bildspeichers in Anwendungsprogramme (z.B. X11-Server)
- Bildspeicher
 - ◆ spezieller Speicher, der den Bildschirminhalt repräsentiert
 - ◆ Dual ported RAM (Videochip und Prozessor können gleichzeitig zugreifen)

5 Netzwerktreiber (2)

- Senden
 - ◆ Treiber übergibt dem Netzwerkchip eine Datenstruktur mit den notwendigen Informationen: Sendeadresse, Adresse und Länge von Datenpuffern
 - ◆ Netzwerkchip löst Unterbrechung bei erfolgreichem Senden aus
- Empfangen
 - ◆ Treiber übergibt dem Netzwerkchip eine Datenstruktur mit Adressen von freien Arbeitspuffern
 - ◆ erkennt der Netzwerkchip ein Paket (für die eigene Adresse), füllt er das Paket in einen freien Puffer
 - ◆ der Puffer wird in eine Liste von empfangenen Paketen eingehängt und eine Unterbrechung ausgelöst
 - ◆ Treiber kann die empfangenen Pakete aushängen

5 Netzwerktreiber (3)

- Übertragung der Daten erfolgt durch DMA
 - ◆ evtl. direkt durch den Netzwerkchip
- Intelligente und nicht-intelligente Netzwerkhardware
 - ◆ intelligente Hardware: kann evtl. auch höhere Protokolle, Filterung etc.
 - ◆ nicht-intelligente Hardware: benötigt mehr Unterstützung durch den Treiber (Prozessor)

6 Andere Geräte (2)

- CD-ROM
 - ◆ wird wie Platte behandelt (eigener Treiber)
 - ◆ nicht beschreibbar
 - ◆ spezielle Treiber für Audio-Tracks möglich
- Maus und Tastatur
 - ◆ meist über serielle Schnittstellen und bestimmtes Protokoll implementiert
- Floppy-Disk
 - ◆ wird im Prinzip wie Platte behandelt (eigener Treiber)
 - ◆ spezielle Dateisysteme zur Realisierung von FAT-Dateisystemen unter UNIX

6 Andere Geräte

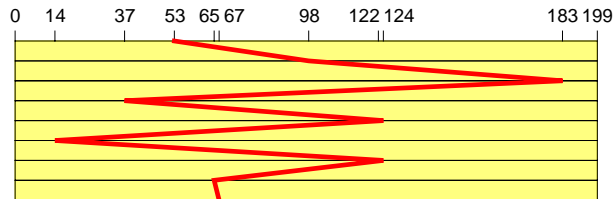
- Uhr
 - ◆ Hardwareuhren (z.B. DCF 77, GPS Empfänger)
 - ◆ Systemuhr fast immer in Software (wird mit Hardwareuhren synchronisiert)
 - ◆ UNIX: `getitimer`, `setitimer`
 - vier Intervalltimer pro Prozess: Signal `SIGALRM` nach Ablauf
 - Ablauf konfigurierbar:
Realzeit, Virtuelle Zeit, Virtuelle Zeit (einschl. Systemzeit des Prozesses)
- Bandlaufwerk
 - ◆ zeichenorientiertes Gerät
 - ◆ Spuloperationen durch `d_ioct1` realisiert

7 Disk-Scheduling

- Plattentreiber hat in der Regel mehrere Aufträge in seiner Warteschlange
 - ◆ Warteschlange wird z.B. in UNIX durch Aufruf der Funktion `d_strategy()` gefüllt
 - ◆ eine bestimmte Ordnung der Ausführung kann Effizienz steigern
 - ◆ Zusammensetzung der Bearbeitungszeit eines Auftrags:
 - Positionierzeit: abhängig von der aktuellen Stellung des Plattenarms
 - Latenzzeit: Zeit bis der Magnetkopf den Sektor bestreicht
 - Übertragungszeit: Zeit zur Übertragung der eigentlichen Daten
- ★ Ansatzpunkt: Positionierzeit

7.1 FCFS-Scheduling

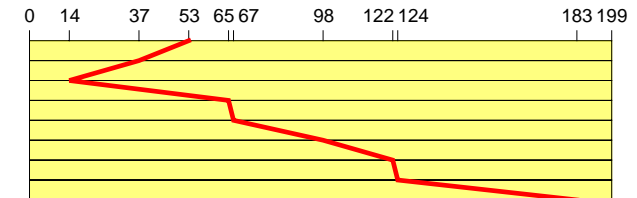
- Bearbeitung gemäß Ankunft des Auftrags
- ◆ Referenzfolge (Folge von Zylindernummern):
98, 183, 37, 122, 14, 124, 65, 67
- ◆ Aktueller Zylinder: 53



- ◆ Gesamtzahl der Spurwechsel: 640
- ◆ Weite Bewegungen des Schwenkarms: mittlere Bearbeitungsdauer lang

7.3 SCAN-Scheduling

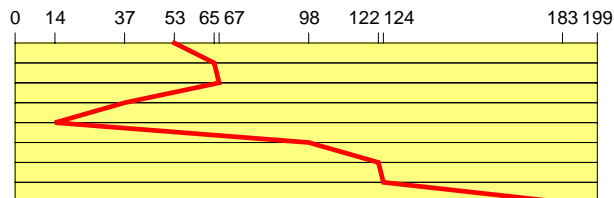
- Bewegung des Plattenarm in eine Richtung bis keine Aufträge mehr vorhanden sind (Fahrstuhlstrategie)
- ◆ Gleiche Referenzfolge (Annahme: bisherige Kopfbewegung Richtung 0)



- ◆ Gesamtzahl der Spurwechsel: 208
- ◆ Neue Aufträge werden miterledigt ohne zusätzliche Positionierzeit und ohne mögliche Aushungerung
- ◆ Variante C-SCAN (*Circular SCAN*): Bewegung nur in eine Richtung

7.2 SSTF-Scheduling

- Es wird der Auftrag mit der kürzesten Positionierzeit vorgezogen (*Shortest Seek Time First*)
- ◆ Gleiche Referenzfolge (Annahme: Positionierzeit proportional zum Zylinderabstand)

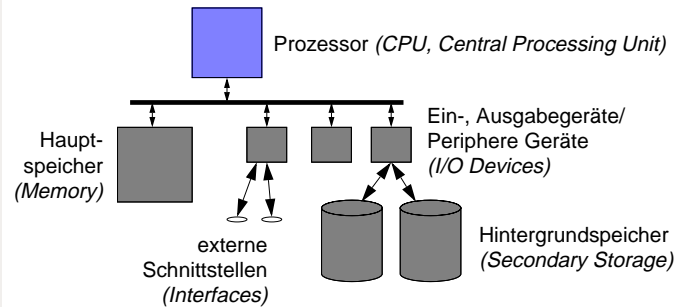


- ◆ Gesamtzahl von Spurwechseln: 236
- ◆ ähnlich wie SJF kann auch SSTF zur Aushungerung führen
- ◆ noch nicht optimal

H Verklemmungen

H Verklemmungen

■ Einordnung:

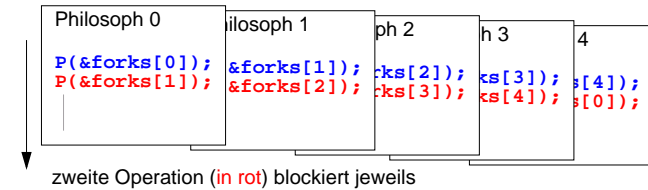


◆ Verhalten von Aktivitätsträgern / Prozessen

1 Motivation (2)

■ Problem der Verklemmung (Deadlock)

- ◆ alle Philosophen nehmen gleichzeitig die linke Gabel auf und versuchen dann die rechte Gabel aufzunehmen



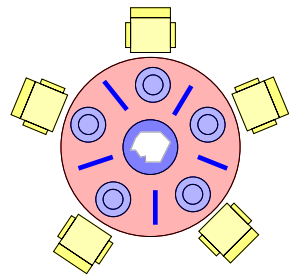
- ◆ System ist **verklemmt**: Philosophen warten alle auf ihre Nachbarn

■ Problemkreise:

- ◆ Vermeidung und Verhinderung von Verklemmungen
- ◆ Erkennung und Erholung von Verklemmungen

1 Motivation

■ Beispiel: die fünf Philosophen am runden Tisch



- ◆ Philosophen denken oder essen
"The life of a philosopher consists of an alternation of thinking and eating."
(Dijkstra, 1971)
- ◆ zum Essen benötigen sie zwei Gabeln, die jeweils zwischen zwei benachbarten Philosophen abgelegt sind

■ Philosophen können verhungern, wenn sie sich „dumm“ anstellen.

2 Betriebsmittelbelegung

■ Betriebsmittel

- ◆ CPU, Drucker, Geräte (Platten, CD-ROM, Floppy, Audio, usw.)
- ◆ nur elektronisch vorhandene Betriebsmittel der Anwendung oder des Betriebssystems, z.B. Gabeln der Philosophen

■ Unterscheidung von Typ und Instanz

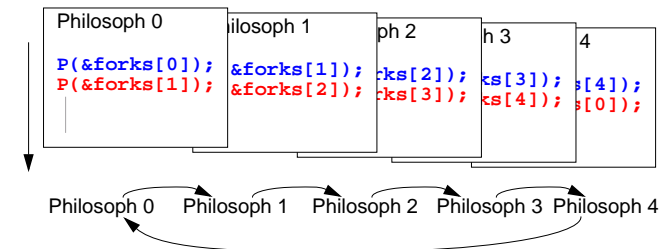
- ◆ Typ definiert ein Betriebsmittel eindeutig
- ◆ Instanz ist eine Ausprägung des Typs
(die Anwendung benötigt eine Instanz eines best. Typs, egal welche)
 - **CPU**: Anwendung benötigt eine von mehreren gleichartigen CPUs
 - **Drucker**: Anwendung benötigt einen von mehreren gleichen Druckern (falls Drucker nicht austauschbar und gleichwertig, so handelt es sich um verschiedene Typen)
 - **Gabeln**: jede Gabel ist ein eigener Betriebsmitteltyp

2.1 Belegung

- Belegung erfolgt in drei Schritten
 - ◆ Anfordern des Betriebsmittels
 - blockiert evtl. falls Betriebsmittel nur exklusiv benutzt werden kann
 - **Gabel:** nur exklusiv
 - **Bildschirmausgabe:** exklusiv oder nicht-exklusiv
 - ◆ Nutzen des Betriebsmittels
 - **Gabel:** Philosoph kann essen
 - **Drucker:** Anwendung kann drucken
 - ◆ Freigeben des Betriebsmittels
 - **Gabel:** Philosoph legt Gabel wieder zwischen die Teller

2.2 Voraussetzungen für Verklemmung (2)

- Beispiel: fünf Philosophen
 - ◆ Exklusive Belegung: **ja**
 - ◆ Nachforderungen von Betriebsmittel möglich: **ja**
 - ◆ Entzug von Betriebsmitteln: **nicht vorgesehen**
 - ◆ Zirkuläres Warten: **ja**

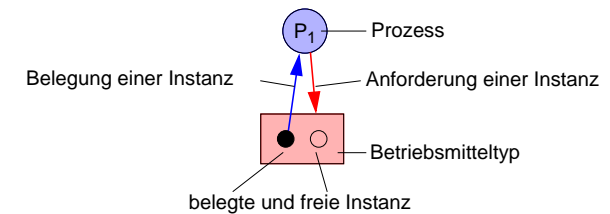


2.2 Voraussetzungen für Verklemmungen

- Vier notwendige Bedingungen
 - ◆ **Exklusive Belegung**
Mindestens ein Betriebsmitteltyp muss nur exklusiv belegbar sein.
 - ◆ **Nachforderungen von Betriebsmittel möglich**
Es muss einen Prozess geben, der bereits Betriebsmittel hält, und ein neues Betriebsmittel anfordert.
 - ◆ **Kein Entzug von Betriebsmitteln möglich**
Betriebsmittel können nicht zurückgefordert werden bis der Prozess sie wieder freigibt.
 - ◆ **Zirkuläres Warten**
Es gibt einen Ring von Prozessen, in dem jeder auf ein Betriebsmittel wartet, das der Nachfolger im Ring besitzt.

2.3 Betriebsmittelgraphen

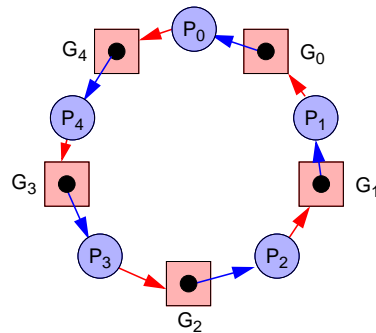
- Veranschaulichung der Belegung und Anforderung durch Graphen (nur exklusive Belegungen)



- Regeln:
 - ◆ kein Zyklus im Graph → keine Verklemmung
 - ◆ Zyklus im Graph → Verklemmung
 - ◆ nur jeweils eine Instanz pro Betriebsmitteltyp und Zyklus → **Verklemmung**

2.3 Betriebsmittelgraphen (2)

- Beispiel: fünf Philosophen



- ◆ Zyklus und jeder Betriebsmitteltyp hat nur eine Instanz → **Verklemmung**

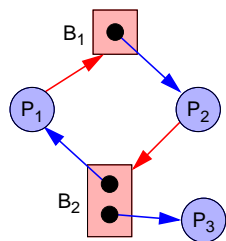
3 Vermeidung von Verklemmungen

- Ansatz: Vermeidung der notwendigen Bedingungen für Verklemmungen

- ◆ **Exklusive Belegung:**
oft nicht vermeidbar
- ◆ **Nachforderungen von Betriebsmittel möglich:**
alle Betriebsmittel müssen auf einmal angefordert werden
 - ungenutzte aber belegte Betriebsmittel vorhanden
 - Aushungerung möglich: ein anderer Prozess hält immer das nötige Betriebsmittel belegt

2.3 Betriebsmittelgraphen (3)

- Beispiel mit Zyklus und ohne Verklemmung



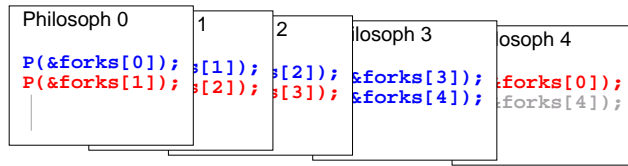
- ◆ Prozess 3 kann seine Instanz vom Betriebsmitteltyp B₂ wieder zurückgeben und den Zyklus damit auflösen

3 Vermeidung von Verklemmungen (2)

- ◆ **Kein Entzug von Betriebsmitteln möglich:**
Entzug von Betriebsmitteln erlauben
 - bei neuer Belegung werden alle gehaltenen Betriebsmittel freigegeben und mit der neuen Anforderung zusammen wieder angefordert
 - während ein Prozess wartet, werden seine bereits belegten Betriebsmittel anderen Prozessen zur Verfügung gestellt
 - möglich für CPU oder Speicher jedoch nicht für Drucker, Bandlaufwerke oder ähnliche
- ◆ **Zirkuläres Warten:** Vermeidung von Zyklen
 - Totale Ordnung auf Betriebsmitteltypen

3 Vermeidung von Verklemmungen (3)

- Anforderungen nur in der Ordnungsreihenfolge erlaubt



z.B. Gabeln: geordnet nach Gabelnummer

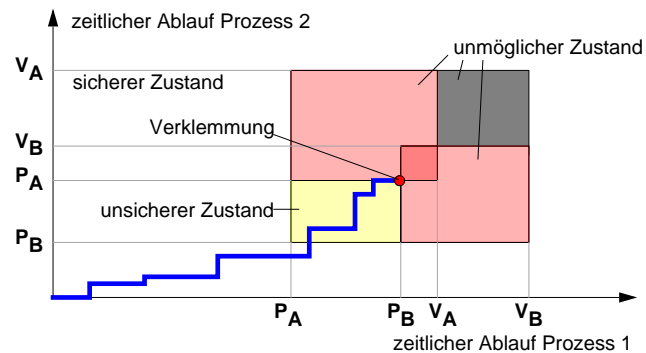
- Bei neuer Anforderung wird geprüft, ob letzte Anforderung kleiner bzgl. der totalen Ordnung war (Instanzen gleichen Typs müssen gleichzeitig angefordert werden); sonst: Abbruch mit Fehlermeldung
- Philosoph 4 bekäme eine Fehlermeldung, wenn er in der obigen Situation zuerst Gabel 4 und dann Gabel 0 anfordert: Rückgabe und neuer Versuch

4.1 Sichere und unsichere Zustände

- **Sicherer Zustand**
 - ◆ Es gibt eine Sequenz, in der die vorhandenen Prozesse abgearbeitet werden können, so dass ihre Anforderungen immer befriedigt werden können.
 - ◆ Sicherer Zustand erlaubt immer eine verklemmungsfreie Abarbeitung
- **Unsicherer Zustand**
 - ◆ Es gibt keine solche Sequenz.
 - ◆ Verklemmungszustand ist ein unsicherer Zustand
 - ◆ Ein unsicherer Zustände führt zwangsläufig zur Verklemmung, wenn die Prozesse ihre angenommenen Betriebsmittel wirklich anfordern bevor sie von anderen Prozessen wieder freigegeben werden.

4 Verhinderung von Verklemmungen

- Annahme: es ist bekannt, welche Betriebsmittel ein Prozess brauchen wird (hier je zwei binäre Semaphore A und B)
- ◆ Betriebssystem überprüft System auf unsichere Zustände



4.1 Sichere und unsichere Zustände (2)

- **Beispiel:**
 - ◆ 12 Magnetbandlaufwerke vorhanden
 - ◆ P₀ braucht (bis zu) 10 Laufwerke
 - ◆ P₁ braucht (bis zu) 4 Laufwerke
 - ◆ P₂ braucht (bis zu) 9 Laufwerke
- ◆ Aktuelle Situation: P₀ hat 5, P₁ hat 2 und P₂ hat 2 Laufwerke
- ◆ Zustand sicher?
- ◆ Aktuelle Situation: P₀ hat 5, P₁ hat 2 und P₂ hat 3 Laufwerke
- ◆ Zustand sicher?