Vorlesung

Ausgewählte Kapitel der praktischen Betriebsprogrammierung I

Wintersemester 1998/99 (10306)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Ausgewählte Kapitel der praktischen Betriebsprogrammierung I (WS 1998/99)

Inhalt

A Organisatorisches A.1
A.1 Vorlesung A.1
A.2 Übungen A.5

B Überblick B.1

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

C Betriebssystemarchitekturen . . . C.1

C.1 Entwicklung der Betriebssystemstrukturierung . . . C.1
C.2 Monolithische Betriebssystemkerne . . . C.2
1 Vorteile . . . C.2
2 Nachteile . . . C.3
3 Struktur . . . C.4
C.3 Minimalkerne . . . C.5
1 Vorteile . . . C.5
2 Nachteile . . . C.6
3 Struktur . . . C.7
C.4 Objektbasierte Systeme . . . C.8
1 Sandboxing . . . C.9
2 Vorteile . . . C.10
C.5 Betriebssysteme für Embedded Systems . . . C.11
C.6 Anforderungen an zukünftige Betriebssysteme . . . C.12

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D UNIX-Überblick D.1

D.1 Historische Entwicklung D.2
D.2 Grundkonzepte D.4
1 Dateien und Dateisystem D.4
2 Programme und Prozesse. D.7
3 Prozeßkommunikation. D.8
D.3 UNIX Systemstruktur D.10
1 Benutzermodus D.11
2 Systemkern D.11
3 Wechsel zwischen Benutzermodus und Systemkern D.13
D.4 Struktur des Systemkerns D.14
D.5 Dateisystem D.15
D.6 Prozeßverwaltung D.19
1 Prozeß D.19
2 Datenstrukturen zur Ausführung eines Programms D.19
3 Datenstrukturen zur Abwicklung eines Prozesses D.21
4 Multiprogramming, Scheduling D.22
5 Prozeßzustände D.23
D.7 Treiber D.24
1 raw- oder character-device driver D.24
2 block-device driver. D.24

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

E MACH

E MACH E.1
E.1 Überblick E.1
1 Motivation E.1
2 wesentliche Eigenschaften E.3
E.2 Architektur E.4
E.3 Basis-Abstraktionen des MACH-Kerns E.5
1 Task E.5
2 Thread E.6
3 Memory Object E.7
4 Port E.8
5 Message E.8
E.4 Systemschnittstelle E.9

F Prozesse F.1
F.1 Überblick F.1
F.2 UNIX — Prozeßbild und Speicherorganisation F.4
1 Speicherorganisation eines Programms F.4
2 Speicherorganisation eines Prozesses F.5
3 Stackaufbau eines Prozesses F.6
F.3 UNIX — Verwaltungsstrukturen des Systemkerns F.12
1 Verwaltungsdaten pro Prozeß F.12
2 Globale Prozeßverwaltungsdaten F.14
3 Kommando ps F.21

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.4 UNIX - Erzeugen eines neuen Prozesses F.22
1 Überblick F.22
2 Ablauf F.24
F.5 UNIX — Prozeßumschaltungen F.25
1 Überblick F.25
2 Ablauf (Beispiel aus XELOS) F.26
3 Ablauf (2) F.27
4 Wechsel zwischen zwei Prozessen F.30
5 Erzeugen eines neuen Prozesses F.31
F.6 UNIX — Ausführen eines Programms F.32
F.7 UNIX — Terminieren von Prozessen F.34
1 Überblick F.34
2 Ablauf von exit im Kern F.35
3 Systemaufruf wait( ) F.37
4 typische wait( )-Anwendung F.39
F.8 MACH — Tasks und Threads F.41
1 Motivation F.41
2 Vergleich von Prozeß- und Thread-Konzepten F.42
3 Abstraktionen in MACH: Tasks und Threads F.48
F.9 UNIX — Prozesse, LWPs & Threads F.49
1 pthread-Benutzerschnittstelle F.50
2 Threads & UNIX-Semantik F.53

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.10 Koordinierungsmechanismen F.57
1 UNIX — Semaphore F.57
2 Pthreads-Koordinierung F.64
3 Thread-Cancellation F.72
4 Koordinierung im UNIX-Kern (sleep/wakeup). F.73
F.11 UNIX — Signale. F.79
1 Ablauf der Signalverarbeitung im Systemkern F.80
2 SystemV ↔ BSD/POSIX-Semantik. F.82
F.12 UNIX — Scheduling F.83
1 Scheduling in UNIX System V F.85
2 Scheduling in BSD UNIX F.87
3 Vergleich: SystemV ´ BSD. F.90
F.13 MACH — Scheduling F.96
1 Architektur F.96
2 Schedulingstrategie & Prioritäten F.97
3 Scheduling-Unterstützung für nebenläufige Anwendungen F.99

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

G Interprozeßkommunikation G.1
G.1 Überblick G.1
G.2 Sockets G.2
1 Communication Domain und Protokoll G.3
2 Socket Typen. G.5
3 Client-Server Modell G.7
4 Generieren eines Sockets G.8
5 Namensgebung G.8
6 Verbindungsanforderungen annehmen G.9
7 Verbindungsaufbau und Kommunikation (Stream-Sockets). G.9
8 Stream-Sockets — Systemschnittstelle. G.11
9 Verbindungslose Sockets G.12
10 weitere Systemaufrufe der Socketschnittstelle G.12

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

H Rechnerkommunikation (TCP/IP) H.1

H.1 Überblick H.1
H.2 Einordnung in das ISO/OSI Referenzmodell H.4
H.3 Internet Protocol - IP H.5
1 Adressierung H.6
2 Routing H.9
H.4 User Datagram Protocol - UDP H.16
1 Motivation H.16
2 Protocol-Port Konzept H.17
H.5 Transmission Control Protocol - TCP H.19
H.6 Systemschnittstelle H.20
1 Adressierung H.20
2 Rechnernamen / IP-Adressen H.21
3 weitere Verwaltungsdaten H.23
H.7 spezielle Funktionalität. H.24

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

I Sicherheit in heutigen Betriebssystemen I.1

- I.1 Überblick I.1
- I.2 UNIX-Sicherheitsmechanismen I.2
- I.3 Angriffe auf UNIX-Systeme I.4
  - 1 Angriffspunkte I.4
  - 2 Vorgehensweise I.5
  - 3 Gegenmaßnahmen I.6
  - 4 Risikoabschätzung I.7
- I.4 Netzdienste unter UNIX I.8
- I.5 Verschlüsselungsverfahren I.9
  - 1 Anwendung von Verschlüsselung I.10
  - 2 IP Security I.11
- I.6 Firewalls I.12

J UNIX-Dateisystem J.1

- J.1 Funktionalität J.1
- J.2 Directories (Kataloge) J.2
- J.3 Inodes J.2
  - 1 Umwandlung: Pfad : Inode J.3
  - 2 Datei öffnen J.5
  - 3 Kontrollstrukturen und Systemaufruf dup( ) J.10
  - 4 Kontrollstrukturen und Systemaufruf fork( ) J.11
- J.4 Vnodes J.12
  - 1 Motivation J.12
  - 2 Kontrollstrukturen (SystemV.4) J.14

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

J.5 Dateisysteme J.15
1 SystemV-Dateisystem J.15
2 Berkeley Fast File System (UNIX File System — UFS) J.17
J.6 Virtuelles Dateisystem J.19
1 Überblick J.19
2 Mount-Mechanismus J.20
J.7 Network File System (NFS) J.24
1 Überblick J.24
2 Beispiel J.27
3 Sicherheitsaspekte J.28
4 Implementierung J.29
J.8 Block-Puffer Cache J.31
1 Verwaltungsstrategien. J.31
2 Integration von Buffer Cache und Speicherverwaltung J.32
3 Kontrollstrukturen J.33
J.9 Dateisystem - Buffer Cache J.35
1 Kontrollfluß J.35
J.10 Treiber J.36
1 Überblick J.36
2 Treiber-Funktionsvektoren. J.38
J.11 Hintergrundspeicher. J.42
1 Organisation einer Disk - Laufwerk J.42
2 Organisation einer Disk (SCSI) J.45
3 Organisation einer Disk — Betriebssystem J.45
4 Disk-Zugriffsstrategien. J.48
J.12 Disk-Treiber: Datenstrukturen und Kontrollfluß J.50

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

A Organisatorisches

- Dozent
  - Dr.-Ing. Jürgen Kleinöder, IMMD IV (Lehrstuhl für Betriebssysteme)
  - kleinoeder@informatik.uni-erlangen.de

A.1 Vorlesung

- „Ausgewählte Kapitel der praktischen Betriebsprogrammierung I“
  - Architektur und interner Aufbau von Betriebssystemen und Ausführungsplattformen
    (am Beispiel UNIX, MACH und Java / JavaOS)
    (Prozeßverwaltung, Kommunikation, Speicherverwaltung, Dateisystem)
  - für Studierende der Fachrichtung Informatik ab 5. Semester
- Termin: Mi. von 10.15 bis 11.45 im H4 (2 SWS)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

A.1 Vorlesung (2)

- Skript
  - kein Skript im eigentlichen Sinne
  - Folien
    - werden im WWW zur Verfügung gestellt und können selbst ausgedruckt werden
    - wir verteilen Kopien vor der Vorlesung
      (Gutschein für komplettes Skript: 6 DM)
  - weitergehende Informationen zum Nachlesen findet man am besten in der Literatur
- URL zur Vorlesung
  - http://www4.informatik.uni-erlangen.de/Lehre/WS98/V_AKBP1/
  - hier findet man Termine, Folien zum Ausdrucken und evt. Zusatzinformationen

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

A.1 Vorlesung (3)

Literatur

- Uresh Vahalia: UNIX Internals: The New Frontiers, Prentice Hall, 1994.
- Berny Goodheart, James Cox: The magic garden explained, Prentice Hall, 1994.
- A. Silberschatz, P. B. Galvin: Operating Systems Concepts. 5th Edition, Addison-Wesley, 1998.
- W. Richard Stevens: Advanced Programming in the UNIX Environment. Addison-Wesley, 1995.
- Andreas Hieronymus: UNIX - Systemarchitektur und Programmierung, Viehweg, 1993.
- Maurice J. Bach: The Design of the UNIX Operating System, Prentice Hall, 1986.
- Samuel J. Leffler et al.: The design and implementation of the 4.3BSD UNIX operating system, Addison-Wesley, 1989. (die deutsche Version dieses Buchs ist sehr schlecht übersetzt - Vorsicht!)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

A.1 Vorlesung (4)

Schein

- 4-stündig:
  Bearbeitung der Übungsaufgaben
- 2-stündig:
  Kolloquium über die Vorlesung
- benotet:
  Bearbeitung der Übungsaufgaben
  + Kolloquium über Vorlesung und Übungen

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

A.2 Übungen

- Verantwortlich für die Übung
  - Dipl.-Inf. Erich Meier
- Inhalt
  - Programmierung von systemnahen Anwendungen
  - Themenbereiche:
    - Dateisystem
    - Nebenläufigkeit & Koordinierung
    - Rechnerkommunikation
- Anmeldung zur Übung
  - „login: akbpan“ an allen CIP-Workstations der Informatik

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Übungen (2)

- Übungsbesprechung
  - Di. 14 - 16 Raum 0.031
  - Do. 10 - 12 Raum 0.031

  - Intensive Diskussion über Probleme bei den Aufgaben
  - Zusätzliche Hintergrundinformationen

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

B Überblick

- Betriebssystemarchitekturen
  - Monoloithische Betriebssytemkerne
  - Minimalkerne
  - Objektbasierte Systeme & Sandboxing
- UNIX Überblick
  - Geschichte,
  - Grundkonzepte (Datei, Prozess, Interrupts, Exceptions)
- MACH Überblick
  - Motivation, Unterschiede zu UNIX
  - Architektur
  - Abstraktionen
    (Adreßräume, leichtgewichtige Prozesse, Interprozeßkommunikation)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

B Überblick (2)

- Prozesse + Programme
  (Verwaltung, Koordinierung und Scheduling)
  - ◆ UNIX-Prozesse
  - ◆ MACH Tasks & Threads
  - ◆ Koordinierung
  - ◆ Scheduling
- Interprozeßkommunikation & Rechnerkommunikation
  - ◆ UNIX IPC-Mechanismen (Kurzüberblick)
  - ◆ UNIX-Sockets
  - ◆ MACH-IPC (Ports, Messages und Capabilities)
- TCP/IP und Internet — Netzkonzepte, Schnittstellen

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

B Überblick (3)

- UNIX-Dateisystem, Buffer Cache, E/A-System
  - Benutzerschnittstelle (grober Überblick),
  - Mechanismen im UNIX-Systemkern
    (Konzepte, Strategien, Datenstrukturen),
  - Virtuelle Dateisystemschnittstelle (UFS, NFS, Automounter),
  - Treiber,
  - Zusammenhänge
    (etwa: was hat ein open und ein write mit der Bewegung des Disk-Kopfes zu tun)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

B Überblick (4)

- Java — Virtual Machine
  - Architektur
  - Zusammenhang JVM und Betriebssystem
- JavaOS
  - Realisierung von Betriebssystemmechanismen auf einer virtuellen Maschine
- Windows NT
  - Architektur
  - Konmzepte im Vergleich zu UNIX

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

C Betriebssystemarchitekturen

C.1 Entwicklung der Betriebssystemstrukturierung

- ➜ Monolithische Kerne
- ➜ Minimalkerne
- ➜ Objektbasierte Systeme & Sandboxing

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

C.2 Monolithische Betriebssystemkerne

- Alle Betriebssystemkomponenten zu einem Kern zusammengefaßt
- Ausführung generell im privilegierten Modus

1 Vorteile

+ Effiziente Kommunikation zwischen den Komponenten des Kerns durch Prozeduraufrufe
+ Direkter Zugriff auf alle Datenstrukturen des Kerns jederzeit möglich
+ Privilegierte Befehle jederzeit aufrufbar

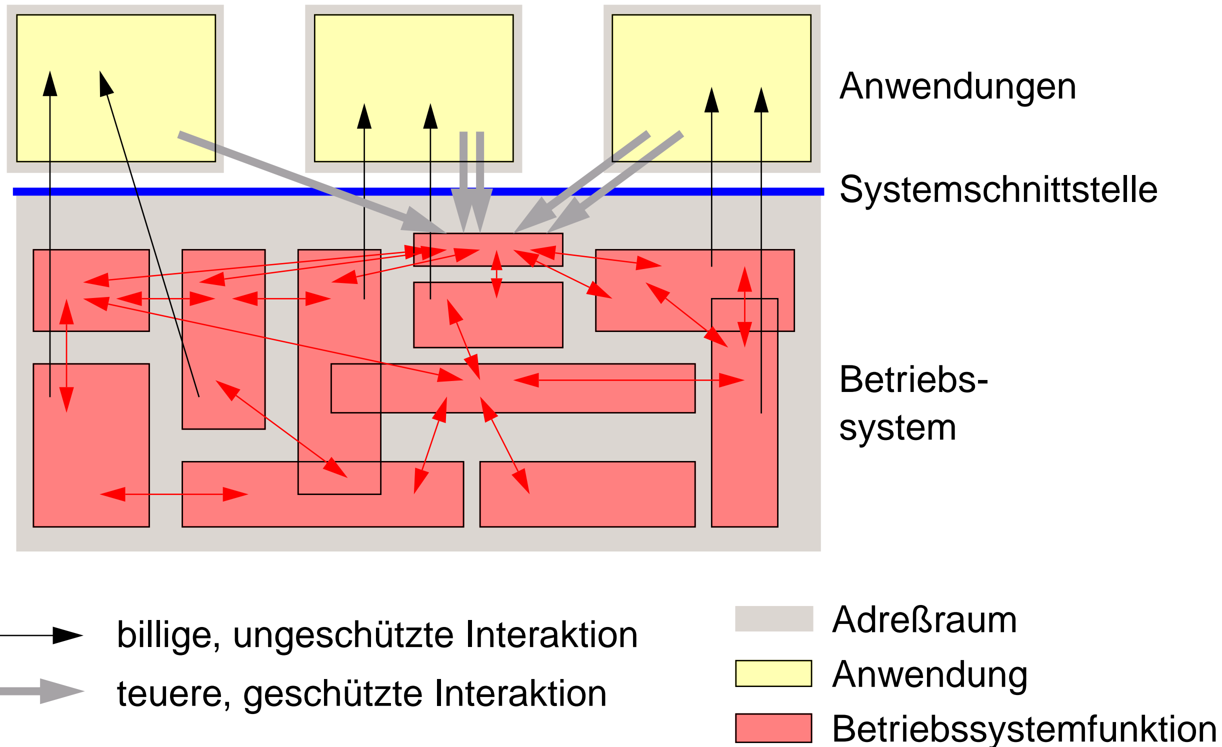
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Nachteile

- Mechanismen strikt vorgegeben
  - individuelle Anpassung an Anwendungen nicht möglich
  - Erweiterung oder Reduzierung in modernen Systemen teilweise durch dynamisch ladbare/entfernbare Module (vor allem Treiber, Dateisysteme)
- Strukturierung meist schlecht
  → änderungsunfreundlich und fehleranfällig
- Kein Schutz gegenüber Kernkomponenten möglich
  (kritisch z. B. bei zugekauften Treibern!)
- Niedriges Abstraktionsniveau
  Programmierung "direkt auf der nackten Hardware"

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Struktur



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

C.3 Minimalkerne

- Auslagerung von Betriebssystemkomponenten
  - ➜ Plazierung der Komponenten in eigenen Adreßräumen
  - ➜ Ausführung im Benutzermodus
- Reduktion des Kerns auf Basisfunktionalität und die Mechanismen, für die privilegierter Modus notwendig ist

1 Vorteile

+ Bessere Modularisierung

+ Ausgelagerte Teile leichter individuell zu gestalten
  - ➜ zusätzlich angepaßte Module erzeugbar
  - ➜ nicht benötigte Module können einfach weggelassen werden

+ Ausgelagerte Teile werden nicht privilegiert ausgeführt → + Sicherheit

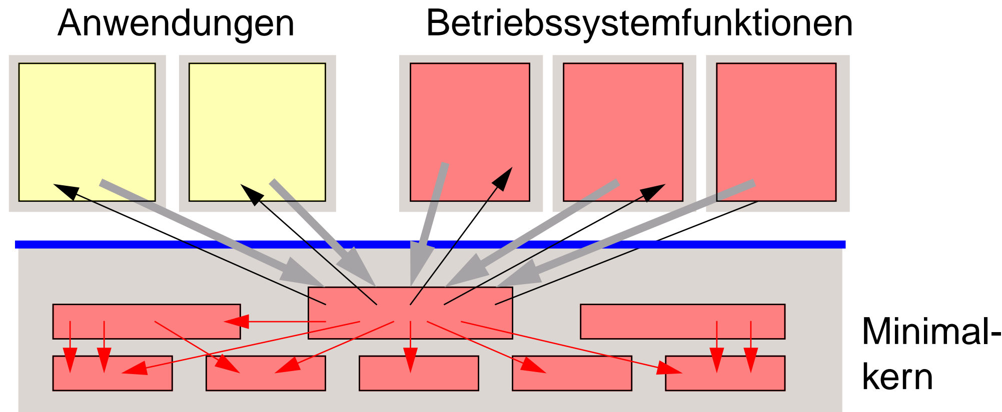
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Nachteile

– Kommunikation zwischen Modulen teuer (RPC, 20- bis 70-facher Aufwand eines Prozeduraufrufs)
  - ➔ Module werden aus Effizienzgründen möglichst groß gehalten
  - ➔ innerhalb der Module Strukturierung beliebig
  - ➔ aus Effizienzgründen verbleiben Komponenten im Minimalkern, die nicht dorthin gehören

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Struktur



billige, ungeschützte Interaktion

teuere, gesicherte Interaktion

Adreßraum

Anwendung

Betriebssystemfunktion

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

C.4 Objektbasierte Systeme

- Fortführung des Wegs 'monolithischer Kern → Minimalkern'

- Weitere Modularisierung nur möglich, wenn Inter-Modul-Kommunikationskosten reduziert werden
  - ➜ Adreßraumkapselung muß aufgegeben werden
  - ➜ Verlust an Sicherheit
  - ➜ Sicherheit muß anders gewährleistet werden

↪ Sicherung von Modulgrenzen

- ◆ auf Sprachebene
- ◆ beim Laden von Software
- ◆ durch spezielle Laufzeit-Überprüfung

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Sandboxing

= Garantieren von Modulgrenzen auf Softwareebene

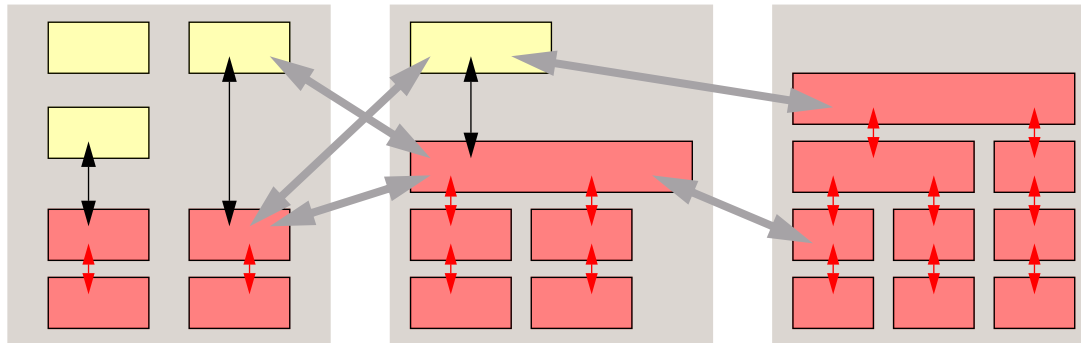
- Sprachunterstützung
  - ➜ objektorientierte bzw. objektbasierte Programmierung
- Kontrolle beim Laden von Software
  - ◆ Byte-Code-Verifier (z. B. bei Java Byte Code)
  - ◆ Einfügen von prüfendem Code (bei Maschinencode)
- "Betriebssystem-artige" Komponenten prüfen zusätzlich, ob Anforderungen eines Moduls zulässig sind
- Wenn Sicherheit auf Sprachebene oder beim Laden garantiert wird, kann auf Adreßraumgrenzen verzichtet werden, sonst nicht!
  - ➜ gleiche Modul-Interaktion adreßraum-lokal und adreßraum-übergreifend wünschenswert
  - ◆ nicht-vertrauenswürdige Module werden mit MMU-Hilfe gekapselt

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
C-BSArch.doc 1999-02-24 18.31
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Vorteile

+ Strukturierungsmöglichkeiten von Systemen mit ausgelagerten Komponenten mit Effizienz eines monolithischen Kerns kombinierbar

+ Individuelle Anpaßbarkeit wie bei einem Minimalkern möglich
$\rightarrow$ durch feinere Granularität der Module jedoch weit flexibler



billige, durch Objektkapselung geschützte Interaktion

teuere, durch Adreßraumgrenze geschützte Interaktion

Adreßraum

Anwendungsobjekte

Betriebssystemobjekte

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

C.5 Betriebssysteme für Embedded Systems

- Grundsätzlicher Unterschied zu "general-purpose"-Systemen
  - ◆ Anforderungen sind vor Systemstart bekannt
  - ◆ auszuführende Applikationen sind bekannt
  - ◆ keine oder nur wenig Dynamik im laufenden System
  - ◆ kein Schutz gegen "böswillige" Programme notwendig
    - ➥ keine Adreßraumgrenzen

- Realisierung
  - ◆ meist Bibliothek mit Betriebssystemfunktionen
    - ➤ Betriebssystemfunktionen werden statisch mit Anwendung gebunden
    - ➤ Betriebssystemfunktionen als "dynamic link library"

- Beispiele
  - ◆ Windows CE (breites Anwendungsspektrum)
  - ◆ OSEK-Architektur (Zielanwendung: KFZ-Elektronik)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

C.6 Anforderungen an zukünftige Betriebssysteme

- Betriebssysteme müssen an die Anforderungen von Anwendungen anpaßbar sein
  - Effizienz
  - Schutzanforderungen
  - Zugriffs- oder Cache-Strategien

  ➔ Anwendungen dürfen nicht gezwungen sein, Betriebssystemmechanismen zu umgehen

- Höheres Abstraktionsniveau bei der Programmierung von Betriebssystemkomponenten

  ➔ Anpassung, Erweiterung und Wartung von Betriebssystemen einfacher

- Betriebssysteme für verteilte Systeme, Mehrprozessorsysteme, Parallelrechner

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D UNIX-Überblick

- Historische Entwicklung
- Grundkonzepte
  - Dateien und Dateisystem
  - Prozesse, Prozeßkommunikation
  - Betriebsmodi
  - Interrupts, Traps, Exceptions
- Systemkomponenten
  - Dateisystem
  - Prozeßverwaltung
  - Treiber

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.1 Historische Entwicklung

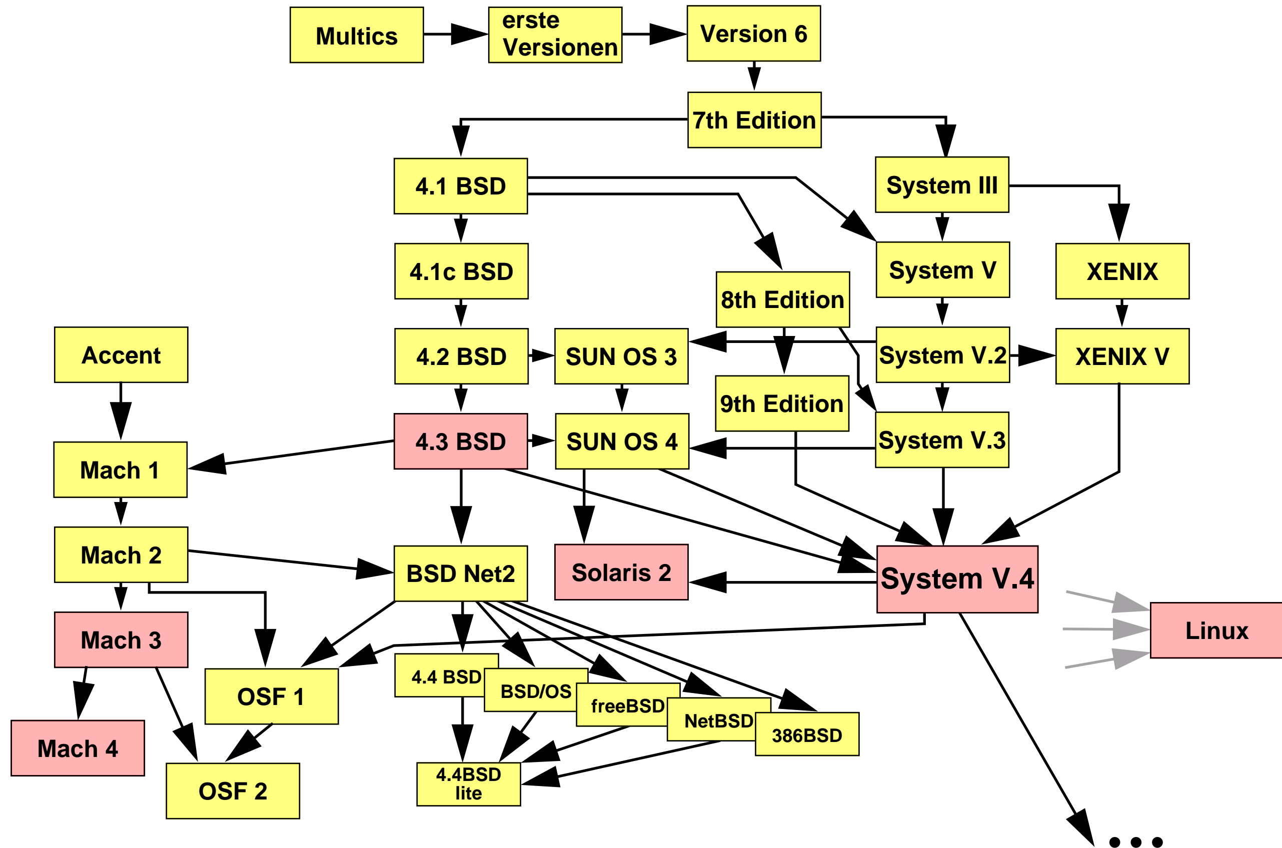
1969	erste Implementierung auf einer PDP7 *(Ken Thompson)*
1971	Reimplementierung in der Sprache B
1973	Erste Version in C
1976	UNIX Version 6
1978	UNIX Version 7 - erste portable Systemversion
ab 1980	Entstehung der Berkeley-Linie

1981	4.1BSD	1992	BSD Net2
1983	4.2 BSD	1994	4.4 BSD
1986	4.3 BSD		

1982	erste kommerziell vertriebene UNIX-Version von AT&T (System III)
1984	UNIX SystemV / Release 2
1986	UNIX SystemV / Release 3
1992	UNIX SystemV / Release 4

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.1 Historische Entwicklung (2)



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.2 Grundkonzepte

1 Dateien und Dateisystem

Verwaltung persistenter Daten — Abstraktion vom Plattenspeicher

- Datei
  - einfacher, unstrukturierter Bytestrom
  - kann beliebige Text- oder Binärinformation enthalten
  - kann dynamisch erweitert werden

- Dateiattribute
  - Name
  - Typ
  - Ortsinformation
  - Größe
  - Zeitstempel
  - Rechte
  - Eigentümer, Gruppe

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
D-UNIX.doc 1999-02-24 18.31
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Dateien und Dateisystem (2)

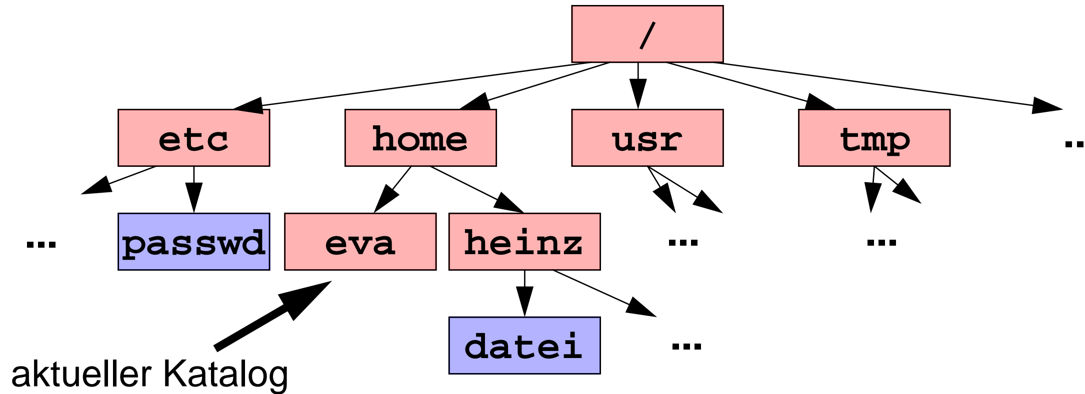
Dateisystem

- hierarchisch, baumförmig strukturiert
- Knoten sind **Dateikataloge *(Directories)***,
  "Blätter" sind Verweise ***(Links)*** auf Dateien ***(Files)***
- der oberste Dateikatalog ist die **Wurzel *(Root)*** des Dateibaums
- jedem Benutzer ist zu jeder Zeit eine **aktuelle Wurzel *(current root)*** und ein **aktueller Dateikatalog *(current working directory)*** zugeordnet
- Peripheriegeräte (Terminals, Magnetbänder, ...) ***(special files)*** werden analog zu normalen Dateien ***(regular files)*** behandelt
- Dateien und Kataloge sind mit Zugriffsrechten versehen

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
D-UNIX.doc 1999-02-24 18.31
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Dateien und Dateisystem (3)

- Baumstruktur & Pfade im Dateibaum



- z.B. „**/home/heinz/datei**“, „**/tmp**“, „**../heinz/datei**“
- jeder Katalog enthält Verweise auf sich selbst **(.)** und auf den darüberliegenden Katalog **(..)**
- mehrere Dateisysteme können zu einem homogenen Baum zusammenmontiert werden

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Programme und Prozesse

Programm

- eine Reihe von Maschinenbefehlen, die in einer Datei abgespeichert sind

Prozeß

- die Ausführung eines Programms in der durch die Prozeßdatenstrukturen definierten Umgebung
  - Speicherbereiche
  - Zugriffsrechte
  - Prioritäten
- mehrere Prozesse können (quasi-)gleichzeitig ablaufen ***(Multitasking)***
- mehrere Prozesse können das gleiche Programm ausführen, ohne sich gegenseitig zu beeinflussen
- Prozesse können neue Prozesse (identische Abbilder ihrer selbst) erzeugen ***(fork)***
- Prozesse können das ablaufende Programm durch ein anderes ersetzen ***(exec)***

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Prozeßkommunikation

- Prozesse können mit anderen Prozessen Daten austauschen
- UNIX stellt dazu folgende Kommunikationsmechanismen zur Verfügung:

- ursprüngliche Mechanismen (UNIX V7)
  - **Signale**
    - Auslösung einer Unterbrechung bei einem anderen Prozeß
    - Empfänger kann auf die einzelnen Signale gezielt reagieren
  - **Pipes**
    - gerichtete FIFO-Puffer zur Übertragung beliebiger Daten
    - nur zwischen dem Erzeuger-Prozeß der Pipe und von ihm erzeugten Prozessen, bzw. zwischen seinen Sohn-Prozessen
  - exit-Status
    - ein Byte, das durch den Vater-Prozeß abgefragt werden kann
  - Dateien

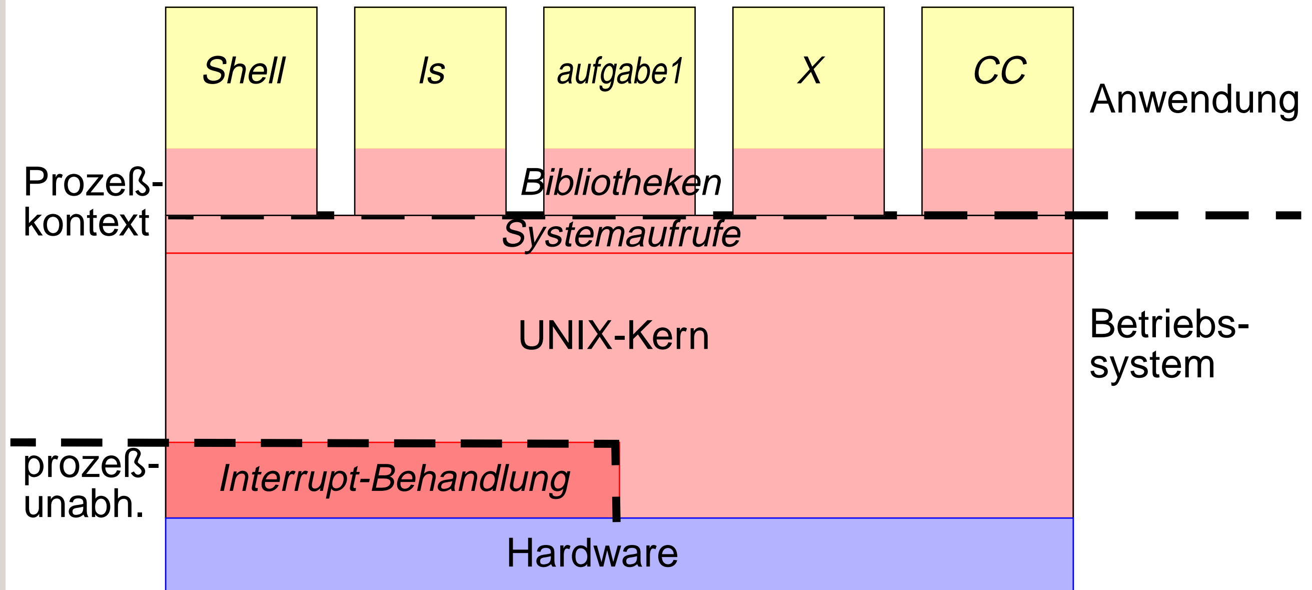
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Prozeßkommunikation (2)

- System V - Erweiterungen
  - ➥ **named Pipes**
  - ➥ **Messages**
  - ➥ **shared Memory**
  - ➥ **Semaphore**
- Berkeley-Erweiterungen
  - ➥ **Sockets**
    - Endpunkte von FIFO-Speicher-Paaren —
      im Gegensatz zu Pipes bidirektionale Kommunikation möglich

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.3 UNIX Systemstruktur



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

D-UNIX.doc 1999-02-24 18.31

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Benutzermodus

- Programmcode wird von einem Prozeß im Benutzermodus *(user mode)* ausgeführt (nicht privilegiert!)

2 Systemkern

- Alle Funktionen des Betriebssystems sind in einem **monolithischen Kern** zusammengefaßt
- Der Systemkern wird im **privilegierten Modus** (supervisor) ausgeführt
- in der Kern-Umgebung gibt es für jeden Prozeß einen eigenen Stack (lokale Daten), jedoch nur ein Datensegment (globale Daten) für alle Prozesse gemeinsam

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Systemkern (2)

- **Prozeßwechsel** im Systemkern sind nur an fest definierten Stellen möglich, an denen die globalen Datenstrukturen in konsistentem Zustand sind

  ➥ Probleme bei Multiprozessorsystemen!!!

- Hardware-Interrupts bewirken den Aufruf einer *Interrupt Service Routine*, die die Daten von der Hardware annimmt und ggf. Prozesse aufweckt

- Interrupt Service Routinen laufen in der Kern-Umgebung ab, haben aber keinen Prozeßkontext

  ◆ Interrupt Service Routinen können auf die Datenstrukturen des Kerns zugreifen

    ➥ während Manipulationen an Kern-Datenstrukturen, die nicht unterbrochen werden dürfen, müssen Interrupts verhindert werden

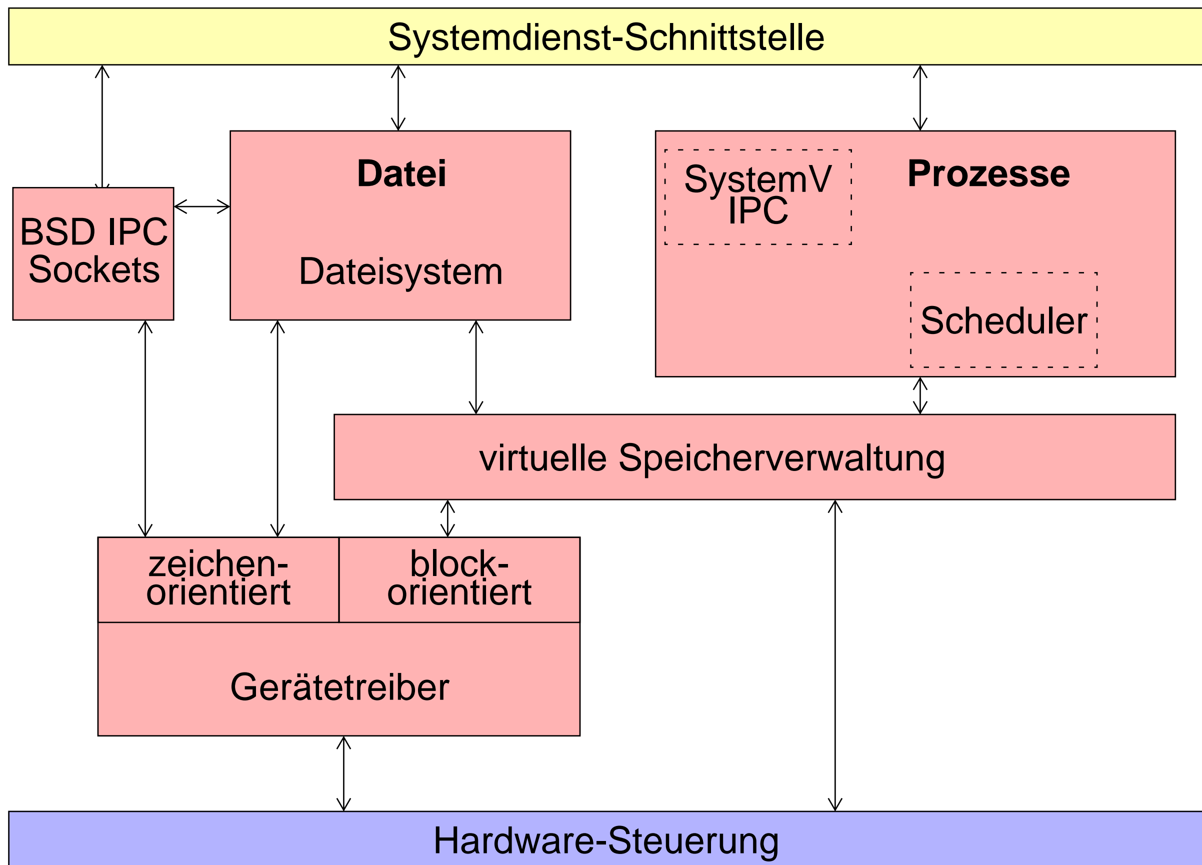
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Wechsel zwischen Benutzermodus und Systemkern

- Betriebssystemdienste führt ein Prozeß durch
  - Wechsel in den **System-Modus *(kernel mode)*** (privilegiert!)
  - und Aufruf von Funktionen des UNIX-Kerns aus
    - ➥ ein Prozeß fordert nicht einen Betriebssystemdienst von einem Kern-Prozeß an, sondern sein Aktivitätsträger wechselt in den Systemkern und führt dort den Systemdienst selbst aus!
- der Wechsel zwischen Benutzer- und Kernelmodus geschieht durch ***Traps*** ***(system calls**, SVCs**)***
- ***Exceptions*** (z. B. Division durch 0) bewirken ebenfalls einen Wechsel in den Systemkern, wo die Fehlerbehandlung veranlaßt wird

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.4 Struktur des Systemkerns



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
D-UNIX.doc 1999-02-24 18.31
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.5 Dateisystem

- jede Datei (*Directory*, *special file*, *named pipe*, ...) wird durch einen **Inode** vollständig beschrieben

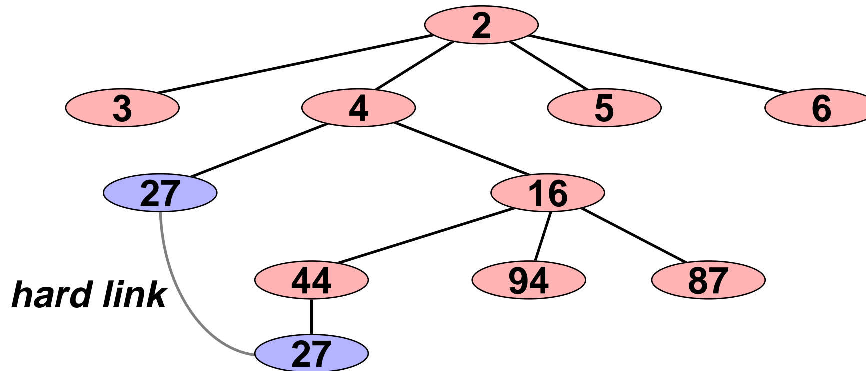
- die **Directories** enthalten nur die Zuordnungen
Dateiname $\Rightarrow$ ***Inode (Link)***

- es können mehrere Dateinamen auf den gleichen Inode verweisen (nur innerhalb eines Dateisystems!): ***Hard-Links***

- die Zuordnung ***Pfadname*** $\Rightarrow$ ***Inode*** erfolgt durch Absuchen des Pfads:
  - *Directory einlesen*
  - *Pfadkomponente suchen und Inodenummer feststellen, ...*
  - Kernel-Funktion ***lookup***

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

- Directory-Baum als Namensverzeichnis für das (eigentlich flache) Dateisystem



Inode 2

2	`.`
2	`..`
3	`bin`
4	`dir1`
5	`etc`
6	`lib`

Inode 3

3	`.`
2	`..`
0	
0	

Inode 4

4	`.`
2	`..`
16	`dir2`
27	`xxx`
0	

...

Inode 16

16	`.`
4	`..`
44	`dir3`
94	`xxx`
87	`yyy`

...

Inode 27

```
dies ist
eine
Testdatei
```

...

Inode 44

44	`.`
16	`..`
27	`dat1`
0	

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.5 Dateisystem (3)

- der UNIX-Dateibaum kann aus mehreren Dateisystemen - auch auf verschiedenen physikalischen Platten - bestehen

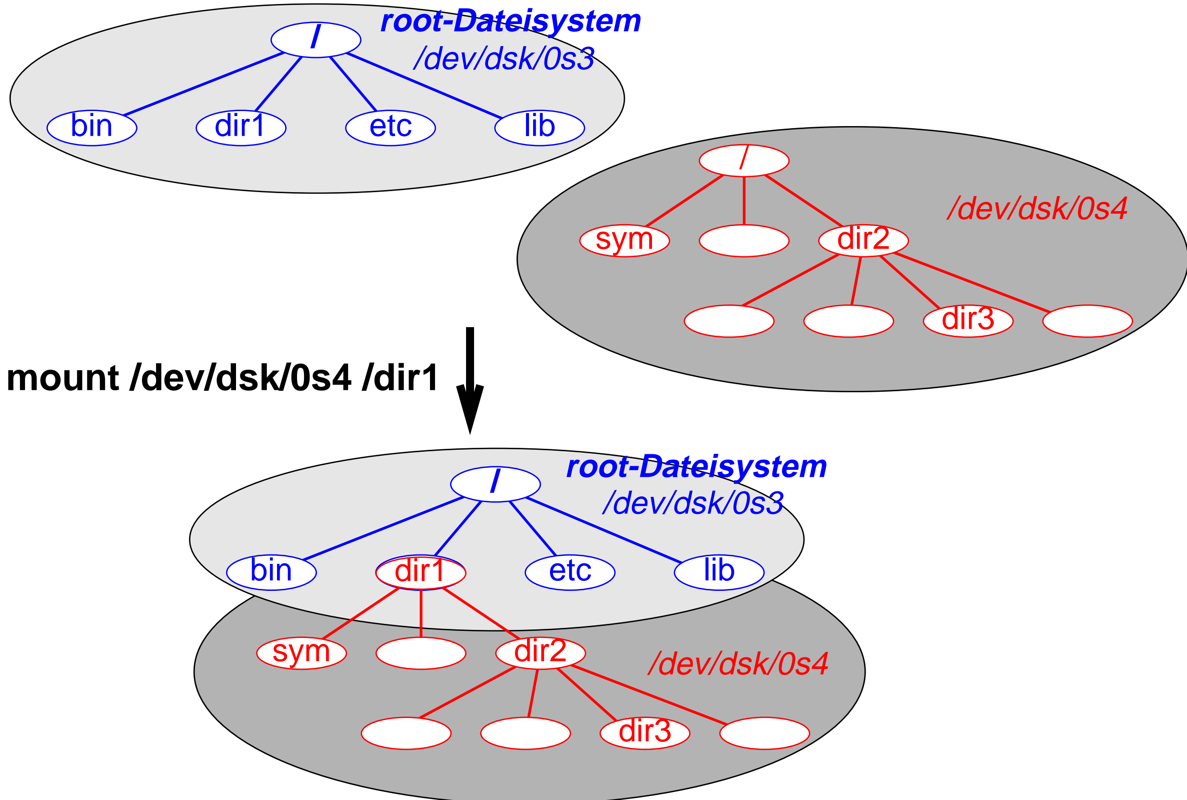
- jedes Dateisystem besteht aus einem Dateibaum mit einer ***Root***

- durch ***Mounten*** kann die Root eines Dateibaums an einen beliebigen Knoten eines anderen (bereits gemounteten) Dateibaums angebunden werden

- ***Network File System (NFS)*** erlaubt auch das Mounten von beliebigen Directories anderer Rechner

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.6 Prozeßverwaltung

1 Prozeß

- Ausführung eines Programms in der durch die Prozeßdatenstrukturen definierten Umgebung

2 Datenstrukturen zur Ausführung eines Programms

- Programmkode und Programmdaten werden in **Segmenten *(Regions)*** organisiert
- abhängig vom gerade auszuführenden Programm, können diese während der Lebenszeit eines Prozesses gewechselt werden

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Datenstrukturen zur Ausführung eines Programms (2)

- **Text-Segment**
  Maschineninstruktionen des Programms
  (in der Regel schreibgeschützt, wird von mehreren Prozessen gemeinsam benutzt)

- **Daten-Segment**
  Daten des Programms (global und *static*), dynamisch erweiterbar (*malloc(3), sbrk(2)*)

- **Stack-Segment**
  lokale Daten und Aufrufparameter von Funktionen
  sowie Sicherungsbereiche für Registerinhalte und Rücksprungadressen
  — wächst bei Bedarf

- **shared Daten-Segment (optional)**
  gemeinsamer Datenbereich für mehrere Prozesse

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Datenstrukturen zur Abwicklung eines Prozesses

weitgehend unabhängig vom gerade abzuwickelnden Programm

- ***proc*-Struktur**

  Prozeßdaten, die während der Lebenszeit eines Prozesses ständig im Hauptspeicher benötigt werden

  z. B.
  - Hauptspeicher/Disk-Adressen der Segmente
  - Prioritäten
  - Signale

- ***user*-area**

  Daten, die zur Ausführung eines Prozesses benötigt werden

  z. B.
  - Segment-/Seiten-Kachel -Tabellen
  - *signal-handler*
  - Zugriffsrechte
  - Laufzeiten

  außerdem Stack für Abwicklung von Funktionen des Systemkerns *(kernel-stack)*

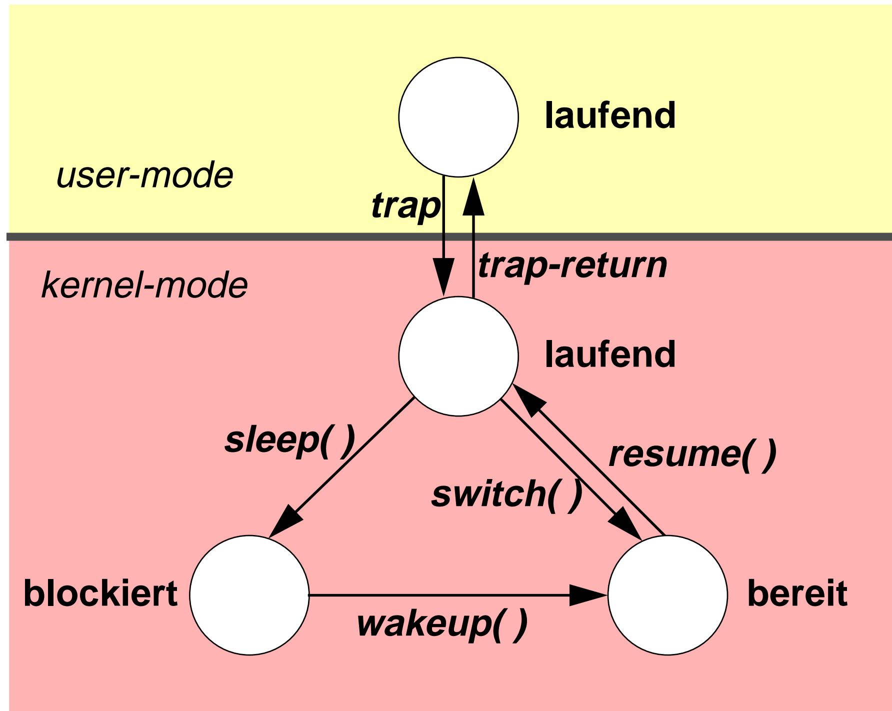
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Multiprogramming, Scheduling

- UNIX erlaubt die (quasi-)gleichzeitige Abwicklung mehrerer Programme (Prozesse)
- Umschaltung zwischen Prozessen durch **Prozeßwechsel *(context switching)***
- Prozeßwechsel erfolgen
  - wenn Prozesse warten müssen (z. B. auf E/A), oder
  - nach einer bestimmten Laufzeit — **Zeitscheibe *(time slice)***
- die Entscheidung, welcher Prozeß als nächstes den Prozessor zugeteilt bekommt ***(Scheduling)*** erfolgt auf der Basis **dynamischer Prioritäten *(multi-level feedback)***

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Prozeßzustände



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.7 Treiber

- Treiber erzeugen die Verbindung zwischen dem Systemkern und Peripheriegeräten (Terminals, Drucker, Platten, Magnetbänder, ...)

1 *raw-* oder *character-device driver*

- E/A erfolgt zeichenweise oder in beliebigen Blöcken
- Kommunikation mit der Peripherie erfolgt in der gleichen Reihenfolge, in der die Daten angefordert/geliefert werden

2 block-device driver

- E/A erfolgt in Blöcken (512 Byte, 1K, 4K oder 8K).
- die Blöcke werden im System gepuffert und in möglichst günstiger Reihenfolge von/zur Peripherie übertragen

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

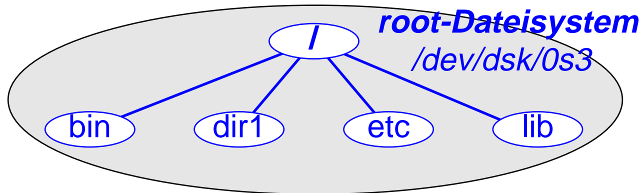
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

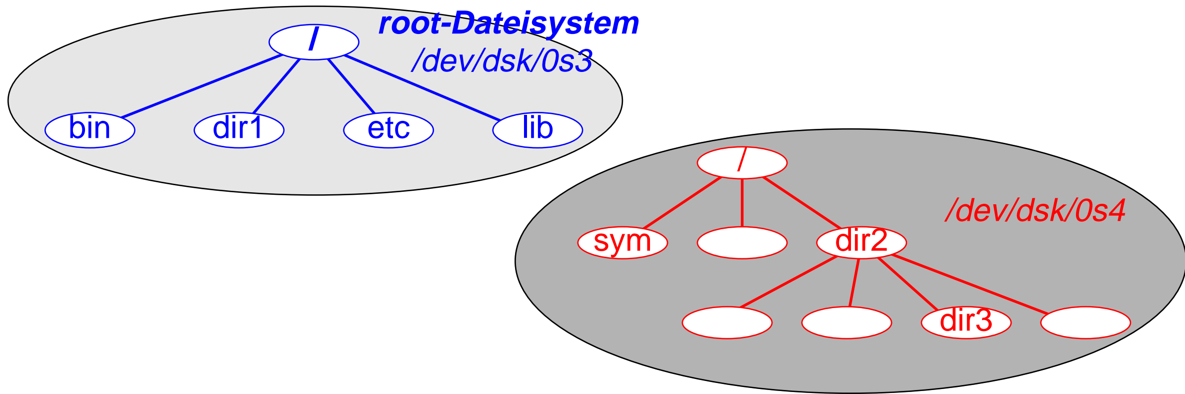
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.5 Dateisystem (4)



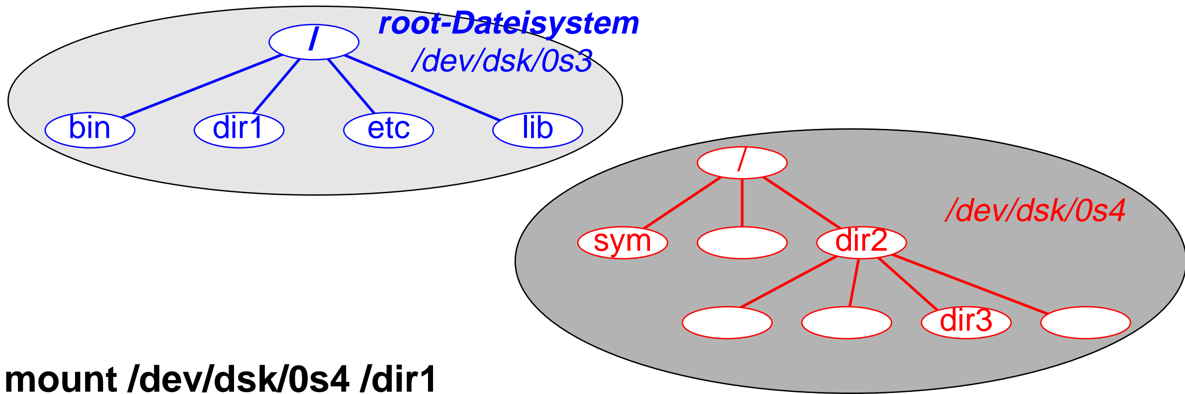
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.5 Dateisystem (4)



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

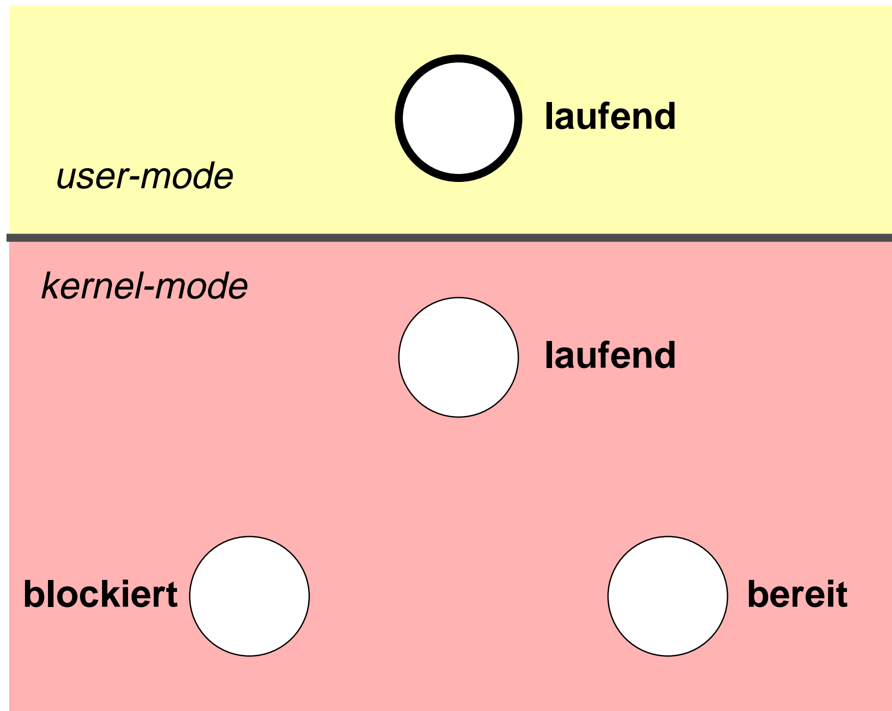
D.5 Dateisystem (4)



mount /dev/dsk/0s4 /dir1

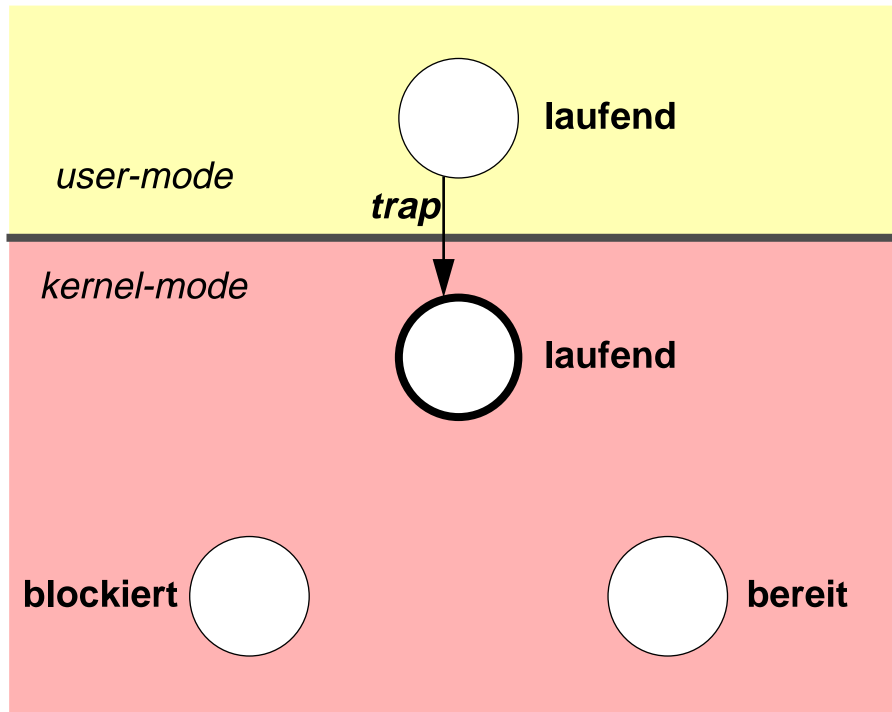
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Prozeßzustände



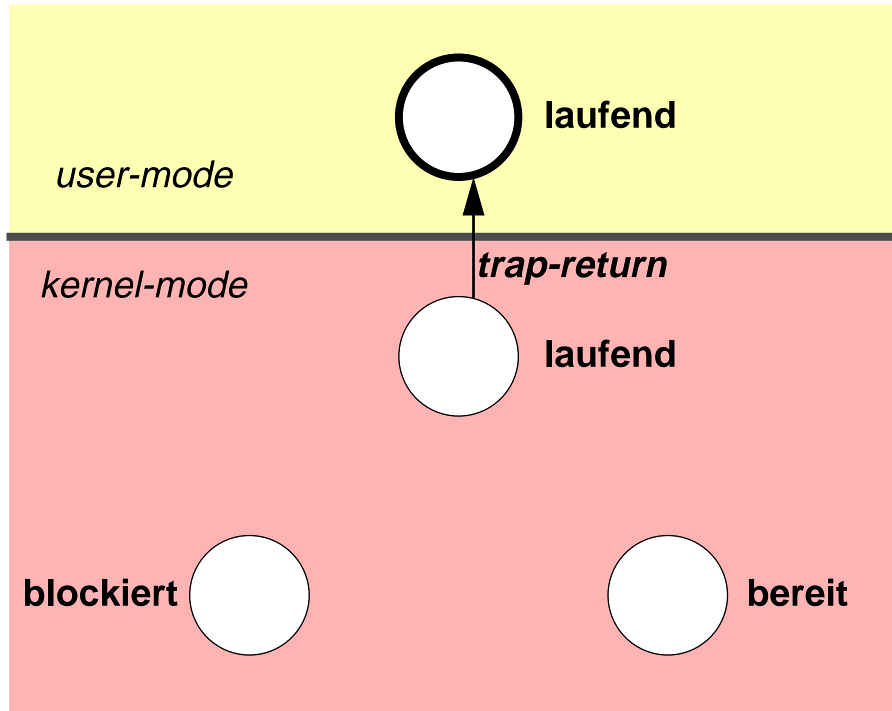
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Prozeßzustände



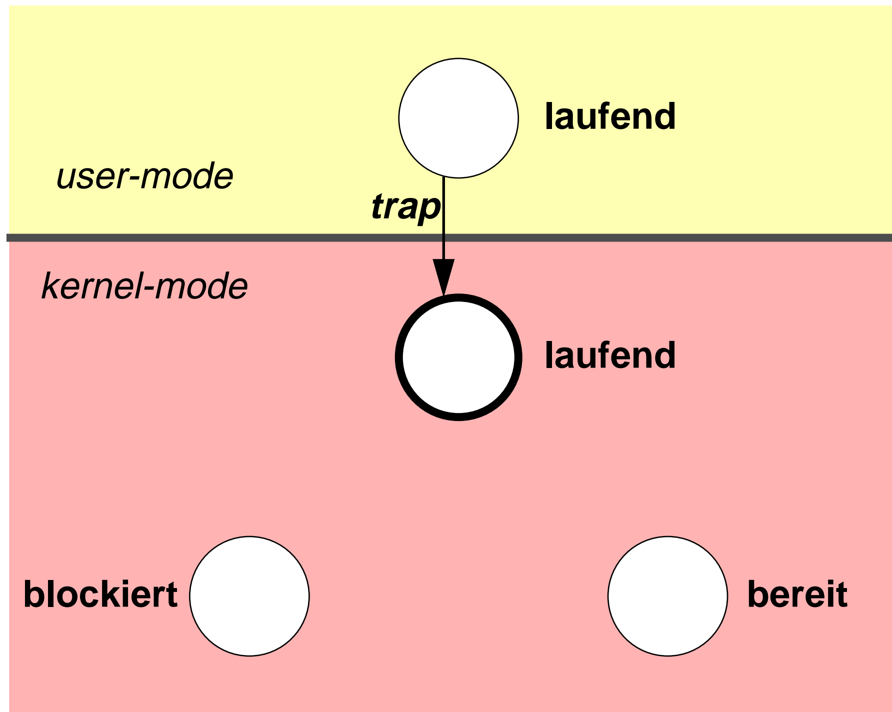
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Prozeßzustände



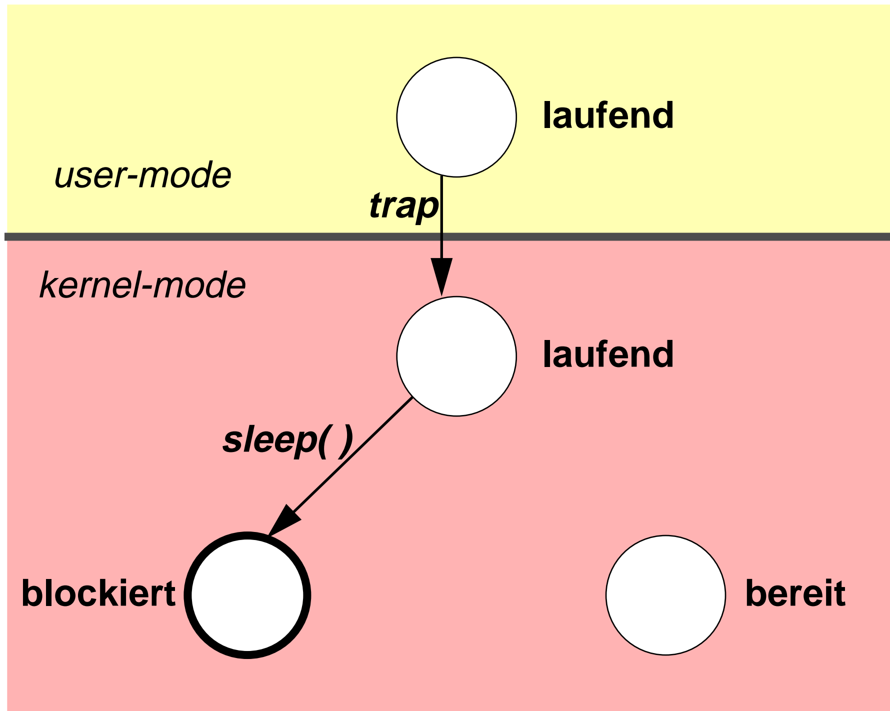
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Prozeßzustände



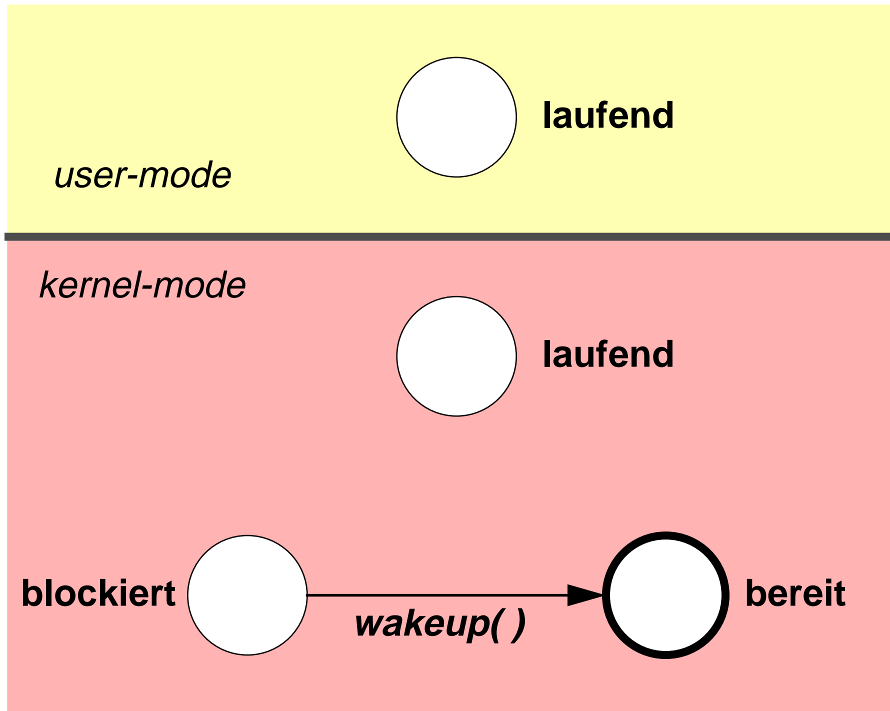
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Prozeßzustände



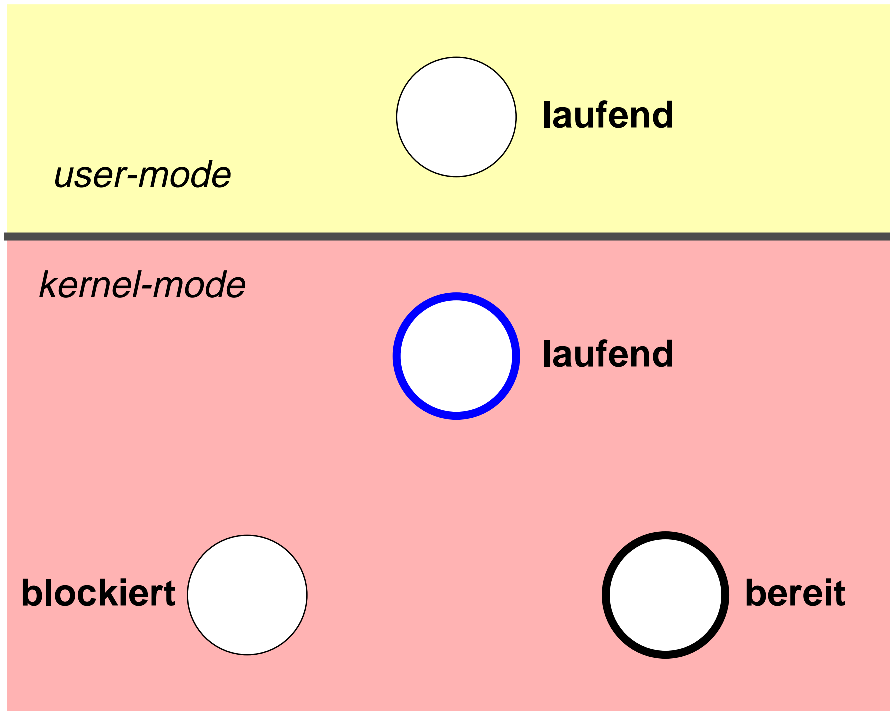
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Prozeßzustände



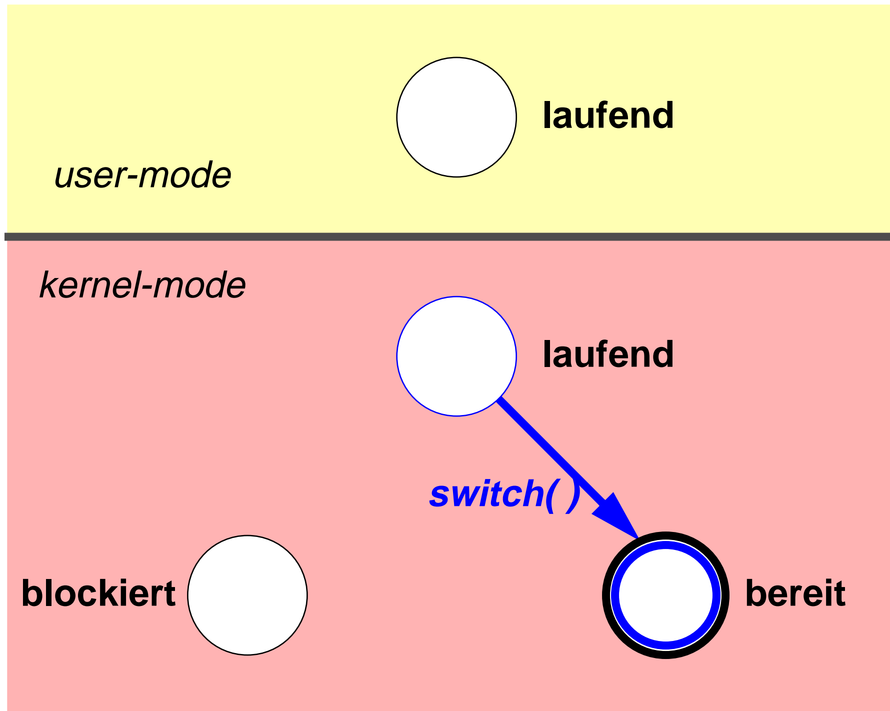
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Prozeßzustände



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

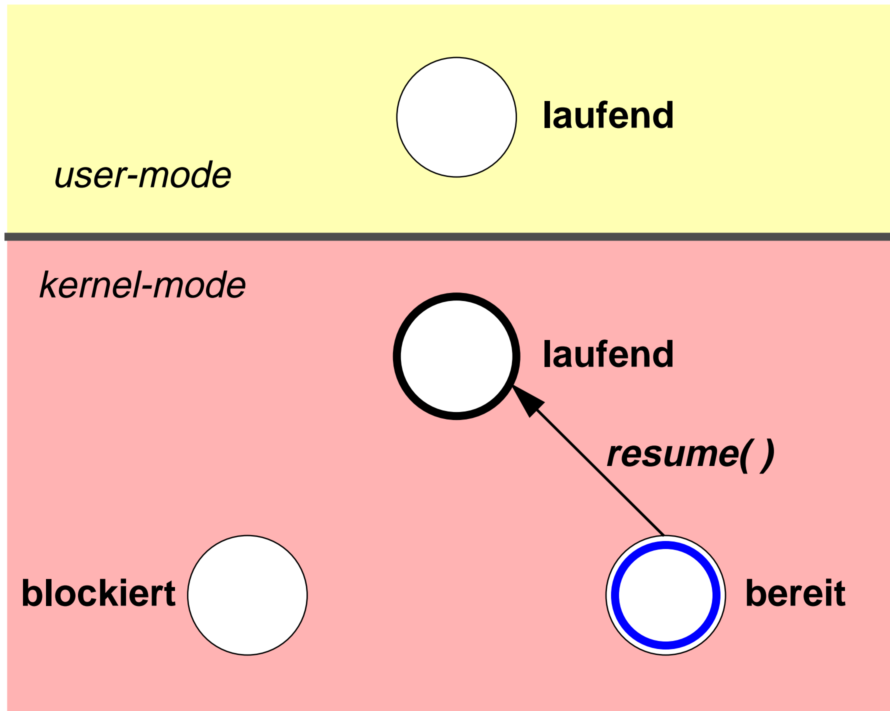
5 Prozeßzustände



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

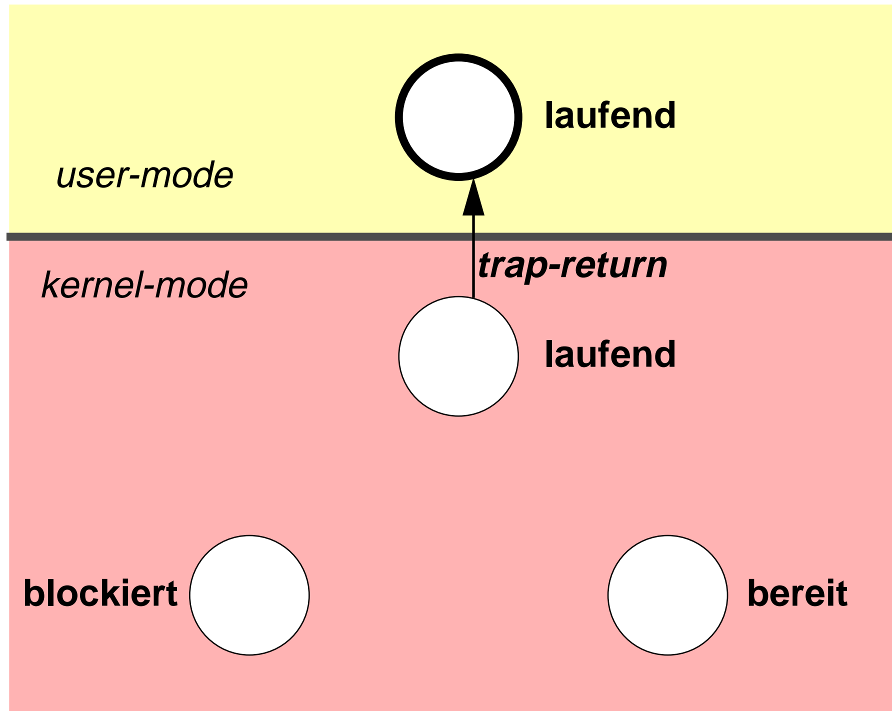
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Prozeßzustände



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

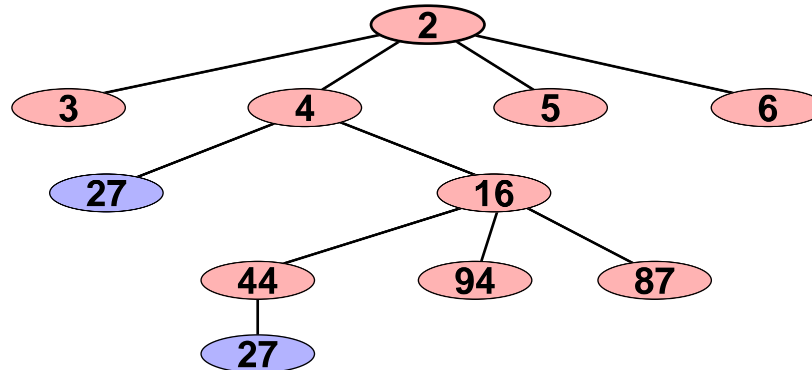
5 Prozeßzustände



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.5 Dateisystem (2)

- Directory-Baum als Namensverzeichnis für das (eigentlich flache) Dateisystem



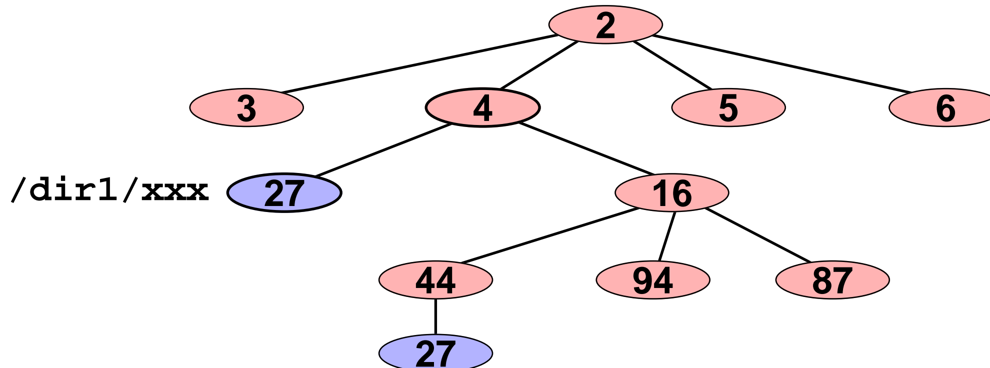
Inode 2

2	`.`
2	`..`
3	`bin`
4	`dir1`
5	`etc`
6	`lib`

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.5 Dateisystem (2)

- Directory-Baum als Namensverzeichnis für das (eigentlich flache) Dateisystem



Inode 2

2	.
2	..
3	bin
4	dir1
5	etc
6	lib

Inode 4

4	.
2	..
16	dir2
27	xxx
0	

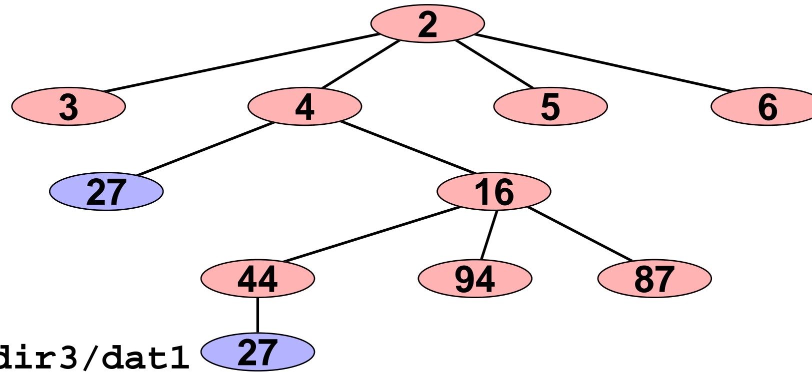
Inode 27

```
dies ist
eine
Testdatei
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.5 Dateisystem (2)

- Directory-Baum als Namensverzeichnis für das (eigentlich flache) Dateisystem



`/dir1/dir2/dir3/dat1`

Inode 2

2	`.`
2	`..`
3	`bin`
4	`dir1`
5	`etc`
6	`lib`

Inode 4

4	`.`
2	`..`
16	`dir2`
27	`xxx`
0	

. . .

Inode 16

16	`.`
4	`..`
44	`dir3`
94	`xxx`
87	`yyy`

. . .

Inode 27

```
dies ist
eine
Testdatei
```

. . .

Inode 44

44	`.`
16	`..`
27	`dat1`
0	

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

E MACH

E.1 Überblick

- Entwicklung seit 1986 an der Carnegie Mellon Univ. (CMU)
- später Basis für OSF

1 Motivation

- UNIX-Systemkern heute nicht mehr so einfach aufgebaut und leicht modifizierbar wie früher
  - früher auf PDP11 mit 32k HSP lauffähig,
    heute 1,5 - 2 MB Speicher (inkl. Puffer etc.) notwendig
  - viele neue Funktionen wurden kritiklos in den Kern eingebaut,
    weil dort die notwendigen Information leicht zugänglich sind

➥ Aufblähung des Kerns, Abstraktionen und Strukturierung werden mehr und mehr zerstört

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
E-MACH.doc 1999-02-24 18.31
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Motivation (2)

- für Einsatz von UNIX auf Multiprozessorsystemen sind aufwendige Modifikationen notwendig
  - Kern-Monitor muß in Teile zerlegt werden oder ganz aufgegeben werden
  - an Stellen, an denen die Monitor-Eigenschaft aufgegeben wird, ist aufwendige Koordinierung erforderlich
- Benutzer- oder anwendungsspezifische Betriebssystemmechanismen werden immer häufiger gefordert
  - spezielle Pufferstrategien für Datenbanken
  - spezielle Blockgrößen in Dateisystemen für Bildverarbeitung
  - spezielle Zugriffsschutzmechanismen im Dateisystem
  - …

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 wesentliche Eigenschaften

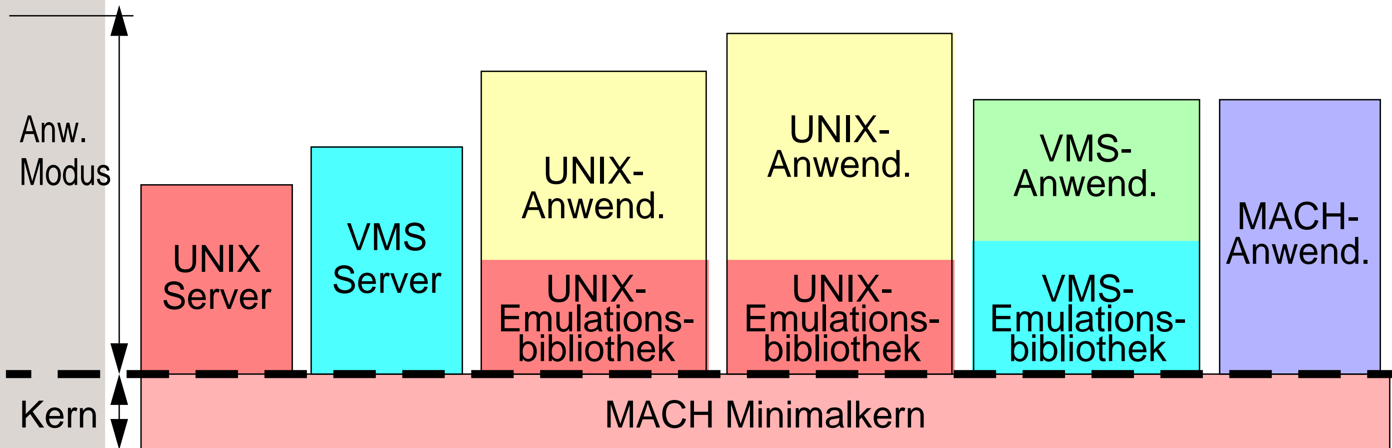
- Unterstützung für Multiprozessoren
- neues - hardwareunabhängiges - Konzept der virtuellen Speicherverwaltung
- *Capability*-basierte Interprozeßkommunikation - transparent erweiterbar für Netzwerk-Kommunikation
- Modularer Aufbau, einfache Erweiterbarkeit, nur die wichtigsten Funktionen sind im MACH-Kern realisiert
  ➥ **policy - mechanism separation**
- die Betriebssystemumgebung wird durch ***Server*** realisiert, die über die Basis-Mechanismen des MACH-Kerns angesprochen werden können
  - Dateisystem
  - Netzwerk-Kommunikation (z. B. TCP/IP)
  - Scheduling

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

E.2 Architektur

- Minimalkern als Emulationsbasis für UNIX und andere Betriebssysteme
- eine Betriebssystememulation besteht aus zwei Teilen:
  - eine Komponente im virt. Adreßraum der Anwendung
  - einer Menge von Server-Prozessen
- Verschiedene Betriebssysteme können gleichzeitig auf MACH residieren
  - z. B. BSD UNIX, SystemV UNIX, HP-UX, VMS



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

E.3 Basis-Abstraktionen des MACH-Kerns

1 Task

Umgebung für Programmausführungen

- virtueller Adreßraum
- Verwaltungseinheit für den Zugriff zu Systemressourcen
    - Prozessoren
    - Port-Capabilities (= Zugriffsrechte)
    - Speicherbereiche
- passiv

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 *Thread*

- **ein Aktivitätsträger innerhalb einer *Task***
  - besteht aus
    - einem unabhängigen Programmzähler
    - einem Registersatz
    - eigenem Stack-Bereich im virtuellen Adreßraums der *Task*
- alle *Threads* einer *Task* haben gemeinsamen Zugriff zu allen *Task*-Ressourcen

➥ **Aufspaltung des traditionellen Prozeß-Paradigmas in zwei orthogonale Konzepte**
- **Aktivitätsträger**
- **Ausführungsumgebung**

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Memory Object

Basiselement der Speicherverwaltung

- Segment in einem virtuellen Adreßraum
- kann gleichzeitig in mehrere Adreßräume abgebildet sein (= shared memory)
- Ein-/Auslagerung der Seiten erfolgt über den ***Pager*** des Memory Objects (kann auch eine normale Anwendung sein)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 *Port*

Kommunikationskanal - Warteschlange für Mitteilungen

- wird vom MACH-Kern verwaltet
- geschütztes Objekt
- Port-Referenzen haben **Capability**-Eigenschaften

5 *Message*

Menge von Daten-Objekten zur Kommunikation zwischen *Threads*

- werden über Ports verschickt
- können max. den gesamten virt. Adreßraum umfassen
- besitzen einen Typ, können Zeiger auf Daten und *Capabilities* für *Ports* enthalten (wird vom Kern interpretiert!)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

E.4 Systemschnittstelle

- Abstraktionen des MACH-Kerns (Tasks, Threads, …)
  und externer Betriebssystem-Server (Dateien, TCP-Ports, …)
  werden durch Softwareobjekte (Tasks, Threads, etc.) repräsentiert

- Zur Kommunikation mit diesen Objekten werden Ports eingerichtet

- Operationen werden durch Messages an diese Ports ausgeführt

Beispiel: Beim Generieren einer *Task* erhält man die Zugriffsrechte auf einen *Port* dieser *Task*.
Durch *Messages* an diesen *Port* kann die *Task* beeinflußt werden.

- Jede Task hat initial Zugriff auf einen **Task-Port** (Port des Kerns!) für Systemaufrufe und einen **Bootstrap-Port** für andere Interaktionen
  - ◆ UNIX-Anwendungen benutzen den Bootstrap-Port zur Kommunikation mit dem UNIX-Server

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

E.4 Systemschnittstelle (2)

- Ausnahmesituationen (Adressierungsfehler, Division durch 0, …) werden vom Kern durch Messages an einen speziellen **Exception-Port** der auslösenden Task mitgeteilt

- Server-Anwendungen erhalten über zwei spezielle, priviligierte Ports direkten Zugriff auf die Systemressourcen:
  - **Device-Port** generische Geräteschnittstelle
  - **Host-Port** priv. Systemschnittstelle (Systemzeit setzen, Prozessormanipulationen, booten, etc.)

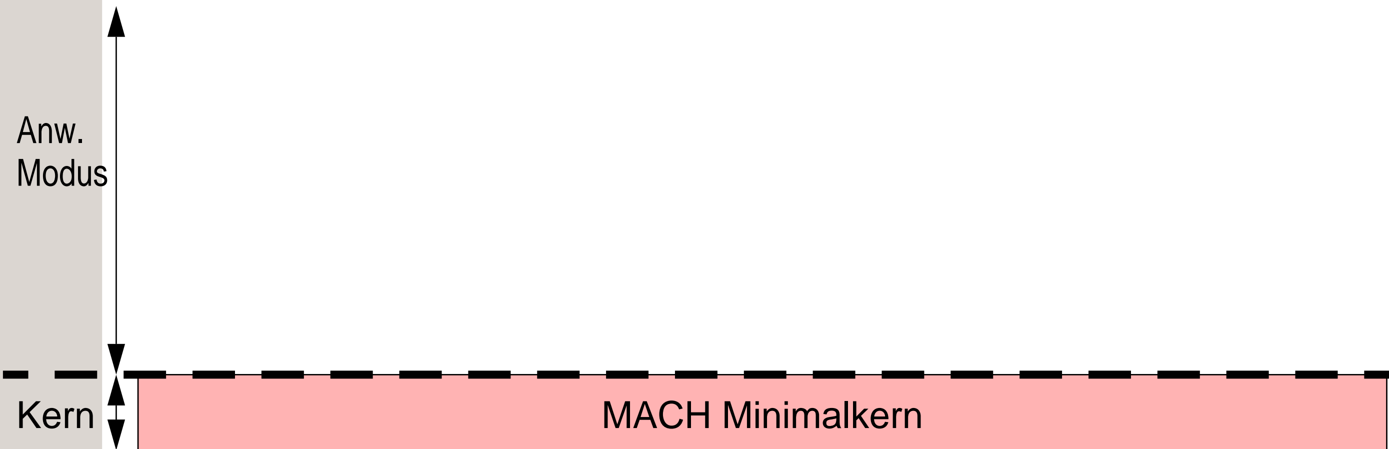
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.2 Architektur

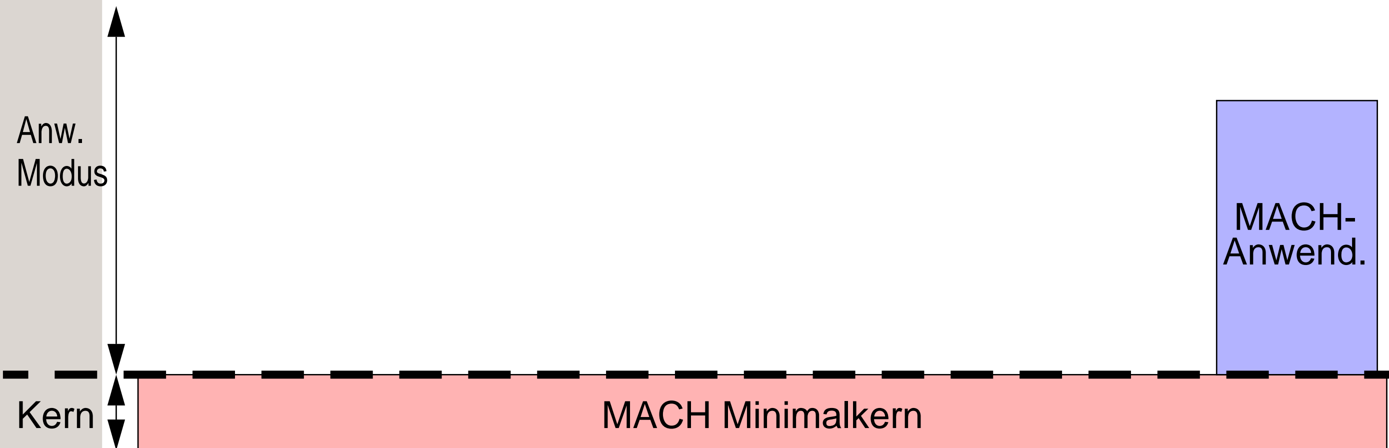
- Minimalkern als Emulationsbasis für UNIX und andere Betriebssysteme
- eine Betriebssystememulation besteht aus zwei Teilen:
  - eine Komponente im virt. Adreßraum der Anwendung
  - einer Menge von Server-Prozessen
- Verschiedene Betriebssysteme können gleichzeitig auf MACH residieren
  - z. B. BSD UNIX, SystemV UNIX, HP-UX, VMS



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.2 Architektur

- Minimalkern als Emulationsbasis für UNIX und andere Betriebssysteme
- eine Betriebssystememulation besteht aus zwei Teilen:
  - eine Komponente im virt. Adreßraum der Anwendung
  - einer Menge von Server-Prozessen
- Verschiedene Betriebssysteme können gleichzeitig auf MACH residieren
  - z. B. BSD UNIX, SystemV UNIX, HP-UX, VMS

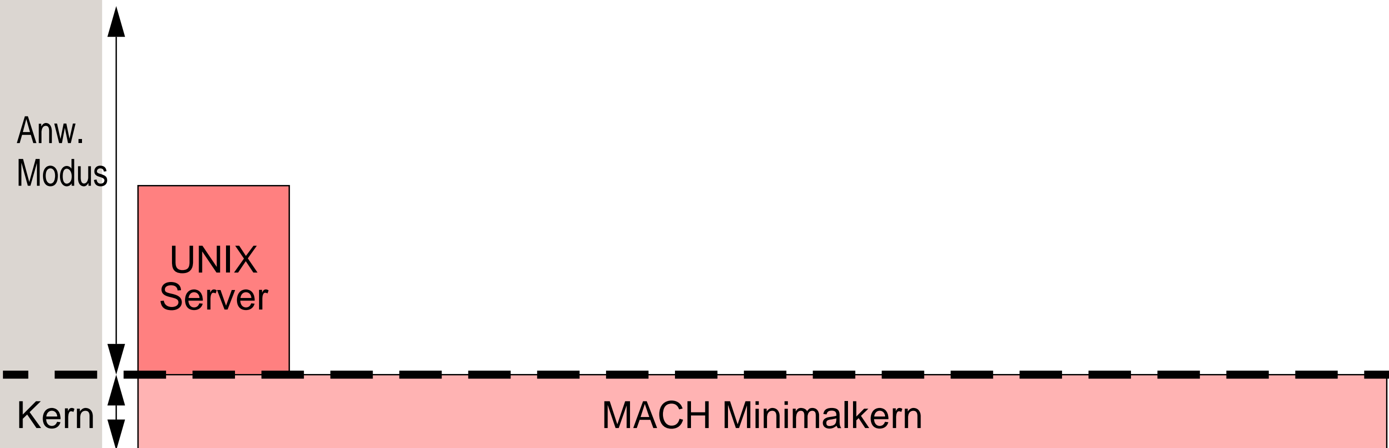


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.2 Architektur

- Minimalkern als Emulationsbasis für UNIX und andere Betriebssysteme
- eine Betriebssystememulation besteht aus zwei Teilen:
  - eine Komponente im virt. Adreßraum der Anwendung
  - einer Menge von Server-Prozessen
- Verschiedene Betriebssysteme können gleichzeitig auf MACH residieren
  - z. B. BSD UNIX, SystemV UNIX, HP-UX, VMS



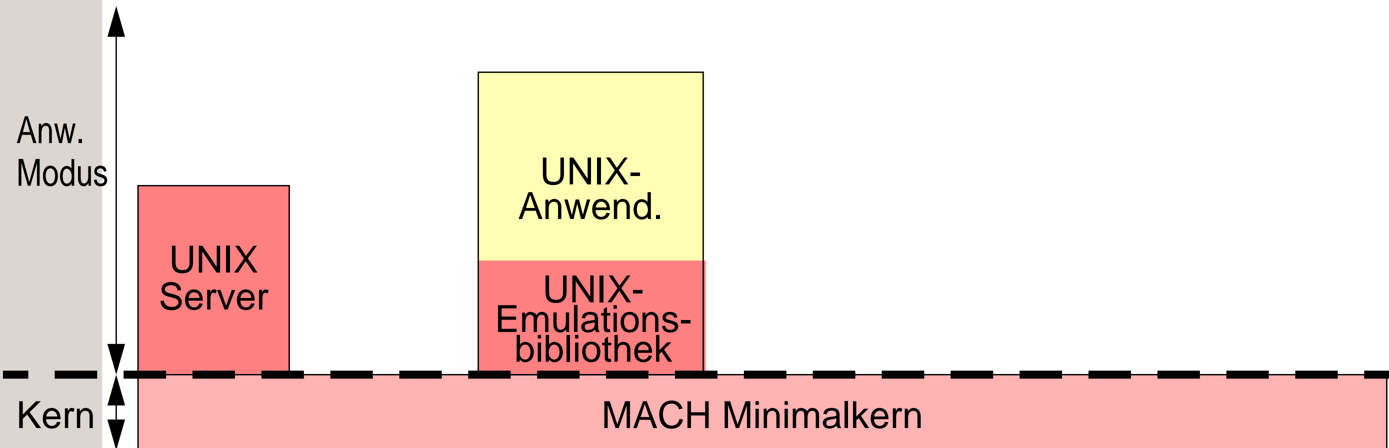
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.2 Architektur

- Minimalkern als Emulationsbasis für UNIX und andere Betriebssysteme

- eine Betriebssystememulation besteht aus zwei Teilen:
  - ◆ eine Komponente im virt. Adreßraum der Anwendung
  - ◆ einer Menge von Server-Prozessen

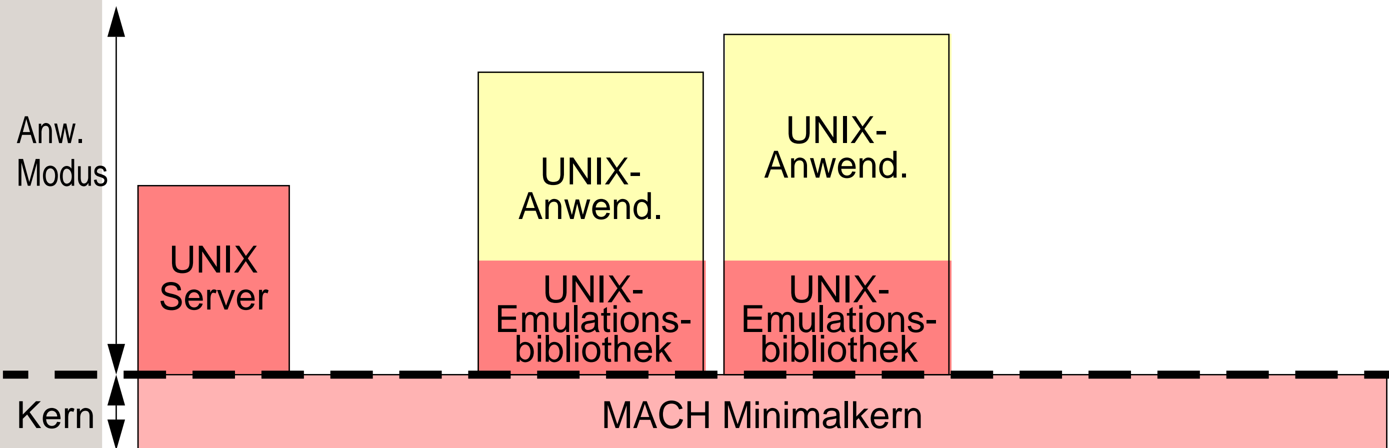
- Verschiedene Betriebssysteme können gleichzeitig auf MACH residieren
  - ◆ z. B. BSD UNIX, SystemV UNIX, HP-UX, VMS



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.2 Architektur

- Minimalkern als Emulationsbasis für UNIX und andere Betriebssysteme
- eine Betriebssystememulation besteht aus zwei Teilen:
  - eine Komponente im virt. Adreßraum der Anwendung
  - einer Menge von Server-Prozessen
- Verschiedene Betriebssysteme können gleichzeitig auf MACH residieren
  - z. B. BSD UNIX, SystemV UNIX, HP-UX, VMS



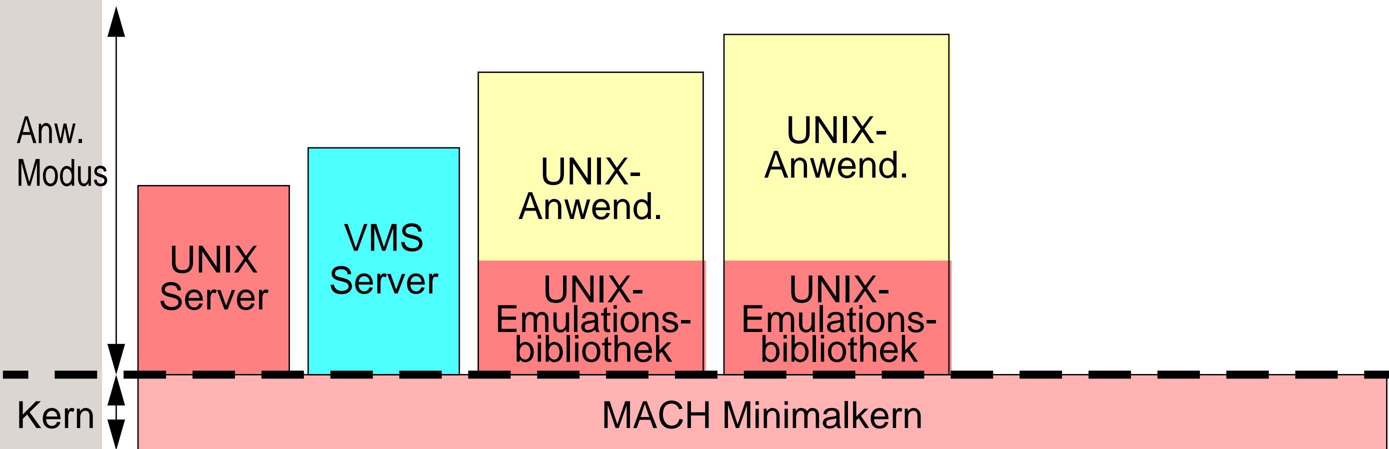
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.2 Architektur

- Minimalkern als Emulationsbasis für UNIX und andere Betriebssysteme

- eine Betriebssystememulation besteht aus zwei Teilen:
  - eine Komponente im virt. Adreßraum der Anwendung
  - einer Menge von Server-Prozessen

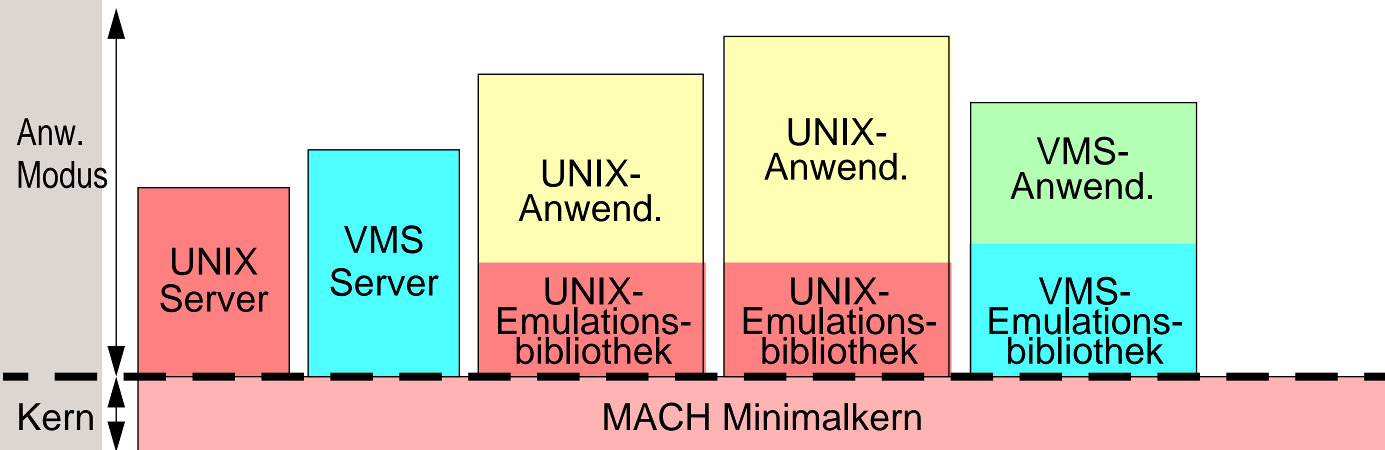
- Verschiedene Betriebssysteme können gleichzeitig auf MACH residieren
  - z. B. BSD UNIX, SystemV UNIX, HP-UX, VMS



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.2 Architektur

- Minimalkern als Emulationsbasis für UNIX und andere Betriebssysteme
- eine Betriebssystememulation besteht aus zwei Teilen:
  - eine Komponente im virt. Adreßraum der Anwendung
  - einer Menge von Server-Prozessen
- Verschiedene Betriebssysteme können gleichzeitig auf MACH residieren
  - z. B. BSD UNIX, SystemV UNIX, HP-UX, VMS



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F Prozesse

F.1 Überblick

- UNIX-Prozesse
  - Prozeßbild, Speicherorganisation
  - Verwaltungsstrukturen, Identifikatoren
  - Prozeßzustände
  - Erzeugen von Prozessen
  - Prozeßumschaltungen
  - Ausführen von Programmen
  - Beenden von Prozessen

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.1 Überblick (2)

- MACH — Tasks und Threads

  - Motivation
  - Tasks
    - Konzept
    - Anwendungsschnittstelle
  - Threads
    - User-level-Threads / Kernel-Threads
    - P-Threads
    - Anwendungsschnittstelle

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Überblick (3)

- Koordinierung
  - ◆ UNIX-Koordinierungsmechanismen
    - ➤ Semaphore
    - ➤ Kern-Koordinierung (sleep/wakeup)
  - ◆ Pthreads-Koordinierung
    - ➤ Mutexes
    - ➤ Condition Variables
- Scheduling
  - ◆ UNIX
    - ➤ SystemV
    - ➤ BSD
  - ◆ MACH

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

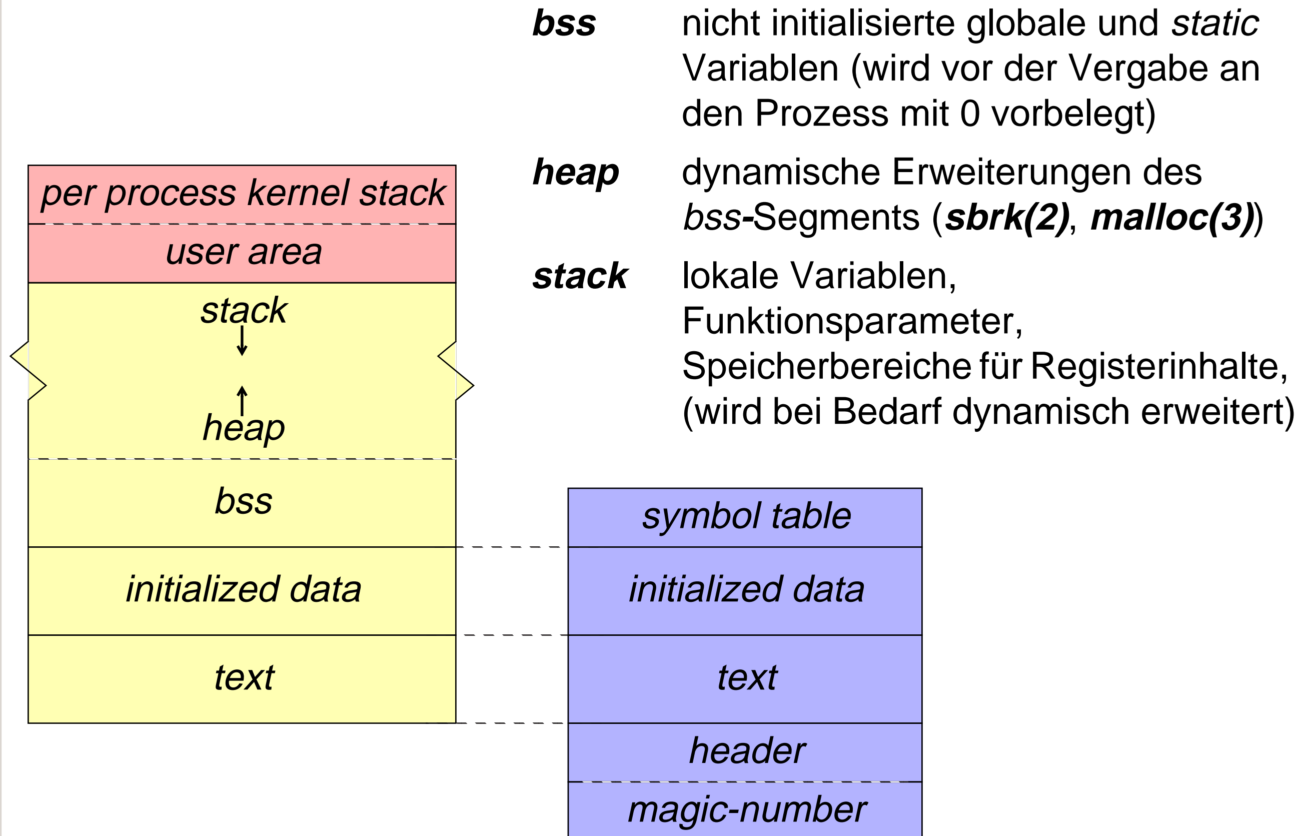
1 Speicherorganisation eines Programms

- Format einer ausführbaren Datei (***a.out-*** oder ***executable*-Format)**

magic number	zur Identifizierung des Dateiformats (z. B. verschiedene *executable* Formate möglich)
header	Verwaltungsinformationen (z. B. Größen der einzelnen Segmente)
text	Programmkode
initialized data	initialisierte globale und *static* Variablen
symbol table	Zuordnung der im Programm verwendeten symbolischen Namen von Funktionen und globalen Variablen zu Adressen (z. B. für Debugger)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Speicherorganisation eines Prozesses



bss nicht initialisierte globale und *static* Variablen (wird vor der Vergabe an den Prozess mit 0 vorbelegt)

heap dynamische Erweiterungen des *bss*-Segments (***sbrk(2)***, ***malloc(3)***)

stack lokale Variablen, Funktionsparameter, Speicherbereiche für Registerinhalte, (wird bei Bedarf dynamisch erweitert)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Stackaufbau eines Prozesses

- für jede Funktion wird ein ***Stack-Frame*** angelegt, in dem
  – lokale Variablen der Funktion
  – Aufrufparameter an weitere Funktionen
  – Registerbelegung der Funktion während des Aufrufs weiterer Funktionen

  gespeichert werden

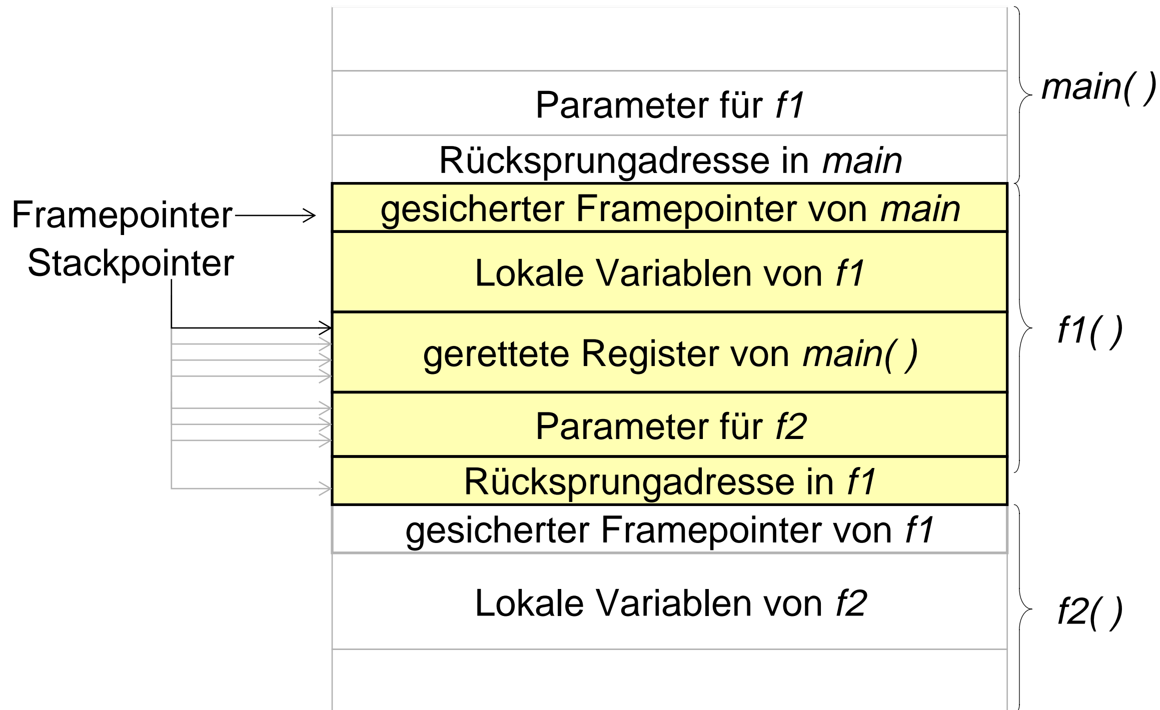
- Stackorganisation ist abhängig von
  – Prozessor
  – Compiler und
  – Betriebssystem

- Beispiele aus einem UNIX auf Motorola 68k-Prozessor
  – typisch für CISC-Prozessoren
  – RISC-Prozessoren mit Registerfiles gehen anders vor!

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Stackaufbau eines Prozesses (2)

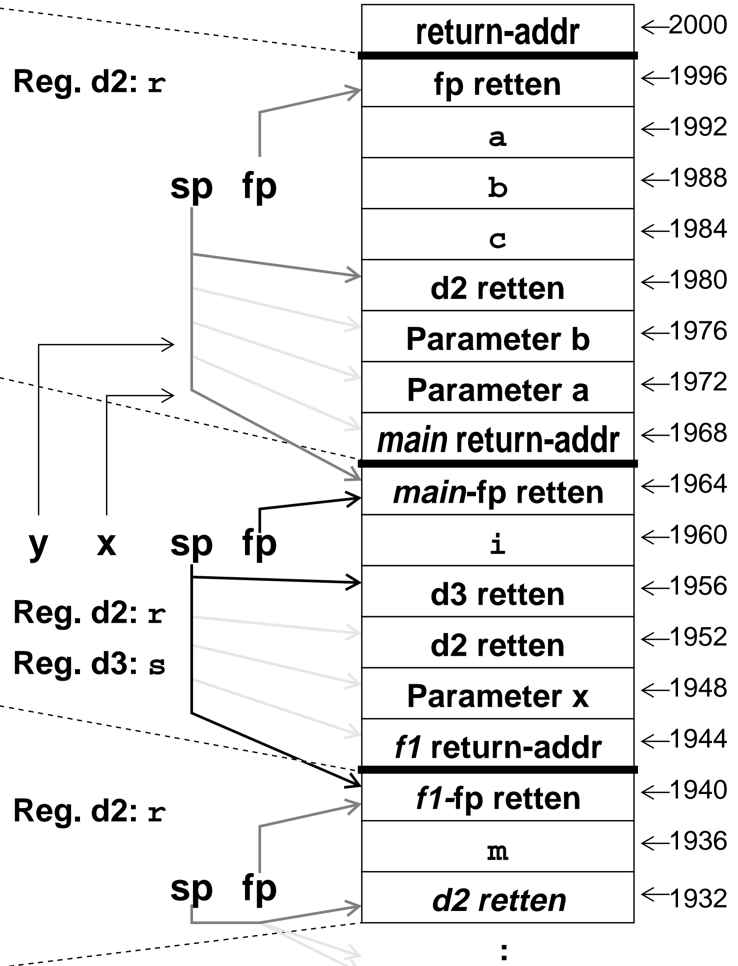
- Aufbau eines **Stack-Frames** (Funktionen *main( ), f1( ), f2( )*)



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Stack mehrerer Funktionsaufrufe

```
main() {
        int a, b, c;
        register r;

        a = 10;
        b = 20;
        r = 40;

        f1(a, b);

        return(a);
}
f1(x,y) {
        int i;
        register r;
        register s;

        x++;

        i = f2(x);

        return(i);
}
f2(z) {
        int m;
        register r;

        r = z;
        m = 100;

        return(z+1);
}
```



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Stackaufbau (4)

Motorola 68000-Assembler — main( )

```
main() {
        int a, b, c;
        register r;

        a = 10;
        b = 20;
        r = 40;

        f1(a, b);

        return(a);
}
```

```
#NO_APP
gcc2_compiled.:
.text
     .even
.globl _main

_main:
     link a6,#-12       # alten Framepointer retten und Stackbereich fuer lokale Var. reservieren (12 Byte)
     movel d2,sp@-      # Register d2 fuer Variable r retten - auf Stack legen und Stackpointer dekr.
     jbsr ___main       # (C++ - Schnittstelle - hier ihne Bedeutung)
     moveq #10,d1       # 10 in Hilfsregister d1 laden
     movel d1,a6@(-4)   # a (Framepointer - 4) = 10 (aus Register d1)
     moveq #20,d1       # 20 in Hilfsregister d1 laden
     movel d1,a6@(-8)   # b (Framepointer - 8) = 20 (aus Register d1)
     moveq #40,d2       # r (Register d2) = 40
     movel a6@(-8),sp@- # b (Framepointer - 8) auf Stack legen, Stackpointer um Wortlaenge (4 Byte) dekr.
     movel a6@(-4),sp@- # a (Framepointer - 4) auf Stack legen, Stackpointer um Wortlaenge (4 Byte) dekr.
     jbsr _f1           # Funkt. f1 aufrufen, dabei Ruecksprungaddr. auf Stack legen und Stackpointer dekr.

                        # ... hier Fortsetzung nach return aus f1
     addqw #8,sp        # Stackpointer um 8 erhoehen - damit entfernen der Aufrufpar. a und b (je 4 Byte)
     movel a6@(-4),d0   # a (Framepointer - 4) in Register d0 laden (Rueckgabeparameter)
     jra L1             # jump auf return-code

L1:                     # return-code fuer main
     movel a6@(-16),d2  # Register d2 restaurieren (war fuer Var. r auf Stack gerettet worden)
     unlk a6            # geretten alten Framepointer restaurieren, Stackpointer = Framepointer
     rts                # return - Ruecksprungadresse von Stack nehmen - Stackpointer inkrementieren
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Motorola 68000-Assembler — Funktion f1( )

```
f1(x, y)
{
        int i;
        register r, s;
        x++;
        i = f2(x);
        return(i);
}
```

```
.even
.globl _f1

_f1:
    link a6,#-4          # Framepointer der Funktion main retten und
                           Stackbereich fuer lokale Var. reservieren (4 Byte)
    movel d3,sp@-        # Register d3 fuer Var. s retten - auf Stack legen und Stackptr. dekr.
    movel d2,sp@-        # Register d2 fuer Var. r retten - auf Stack legen und Stackptr. dekr.
    addql #1,a6@(8)      # 1 auf Parameter x addieren (x liegt auf Adresse Stackpointer + 8)
    movel a6@(8),sp@-    # x auf Stack legen, Stackpointer um Wortlaenge (4 Byte) dekrementieren
    jbsr _f2             # f2 aufrufen, dabei Ruecksprungaddr. auf Stack legen und Stackpointer dekr.

                         #  ... hier Fortsetzung nach return aus f2
    addqw #4,sp          # Stackpointer um 4 erhoehen - damit entfernen des Aufrufparameters x (4 Byte)
    movel d0,a6@(-4)     # Rueckgabewert von f2 (steht in Reg. d0) in Var. i (Framepointer - 4) speichern
    movel a6@(-4),d0     # Variable i in Register d0 laden (Rueckgabewert an main)
    jra L2               # jump auf return-code

L2:                           # return-code fuer f1
    movel a6@(-12),d2    # Register d2 restaurieren (war fuer Var. r auf Stack gerettet worden)
    movel a6@(-8),d3     # Register d3 restaurieren (war fuer Var. s auf Stack gerettet worden)
    unlk a6              # geretten Framepointer von main restaurieren, Stackpointer = Framepointer
    rts                  # return - Ruecksprungadresse von Stack nehmen - Stackpointer inkrementieren
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Stackaufbau (6)

Motorola 68000-Assembler — Funktion f2( )

```
f2(z)
{
        int m;
        register r;
        r = z;
        m = 100;
        return(z+1);
}
```

```
.even
.globl _f2

_f2:
     link a6,#-4        # Framepointer der Funktion f1 retten und Stackbereich fuer
                        lokale Var. reservieren (4 Byte)
    movel d2,sp@-       # Register d2 retten - auf Stack legen und Stackptr. dekr.
    movel a6@(8),d1     # Parameter z (liegt auf Addr. Framepointer +8) an Var. r (Register d1) zuweisen
    moveq #100,d2       # 100 in Hilfsregister d2 schreiben
    movel d2,a6@(-4     # m (Framepointer - 4) = 100 (aus Register d2)
    movel a6@(8),d0     # Uebergabeparameter z in Register d0 (Rueckgabewert) kopieren
    addql #1,d0         # 1 auf Rueckgabewert addieren
    jra L3              # jump auf return-code

L3:                     # return-code fuer f1
    movel a6@(-8),d2    # Register d2 restaurieren
    unlk a6             # geretten Framepointer von f1 restaurieren, Stackpointer = Framepointer
    rts                 # return - Ruecksprungadresse von Stack nehmen - Stackpointer inkrementieren
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Verwaltungsdaten pro Prozeß

User area

- Daten zur Verwaltung des Programmablaufs durch den Systemkern, z. B.
  - Segmenttabelle bzw. Verweis auf Seiten-Kachel-Tabelle
  - Verweis auf Eintrag in die Prozeßtabelle
  - Statistik-Daten (Laufzeiten, ...)
  - aktuelle Directory *(current working directory)*
  - aktuelle Root
  - user file descriptor table
  - Adressen der *signal-handler*-Funktionen
  - Bereich zum Sichern des Prozeßzustands (Register) bei Prozeßwechsel
  - Puffer für Systemaufruf-Parameter und -Rückgabewerte

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Verwaltungsdaten pro Prozeß (2)

- der Speicherbereich der *user area* des gerade aktiven Prozesses
  - wird an eine feste Adresse des virtuellen Adreßraums des Systemkerns abgebildet
    - Zugriff auf die *user area* erfolgt in den Kern-Funktionen über die Struktur **u** (z. B. ***u.u_ofile***)
    - bei einem Prozeßwechsel wird die *user area* des neuen Prozesses auf den Speicherbereich der Struktur **u** abgebildet
    - bei jedem Prozeßwechsel muß die Abbildung virtuelle Adressen → physikalische Adressen geändert werden
  - oder wird über einen festen Zeiger im Systemkern adressiert
    - Zugriff auf die *user area* erfolgt in den Kern-Funktionen über den Zeiger **uunix** (z. B. ***uunix->u_ofile***)
    - bei einem Prozeßwechsel wird der Zeiger **uunix** auf die neue *user area* gelegt
    - jeder Zugriff auf Elemente der *u-area* erfolgt indirekt

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Verwaltungsdaten pro Prozeß (3)

- Kernel stack
  - Stack für Funktionen des Systemkerns, die im Kontext des Prozesses ablaufen (zur Abwicklung von Systemaufrufen)
  - bei Prozeßwechsel wird der Kernel Stack des neuen Prozesses entsprechend in den virt. Adreßraum des Kerns gemapped

2 Globale Prozeßverwaltungsdaten

- Prozeßtabelle
  - alle wichtigen Verwaltungsdaten über einen Prozeß, die immer verfügbar sein müssen, werden in

    ***proc*-Strukturen**

    gehalten
  - die *proc*-Strukturen bilden die **Prozeßtabelle**
    (bei Booten statisch als Feld angelegt oder dynamisch als Liste verwaltet)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Globale Prozeßverwaltungsdaten (2)

- … Prozeßtabelle
  - eine *proc*-Struktur enthält u. a. folgende Daten:
    - Prozeßzustand (laufend/bereit, blockiert, ausgelagert, ...)
    - Verweise auf die Segmente des Prozesses (meist über *page table*)
    - Verweis auf die *user area*
    - Verweis auf die *credential*-Struktur mit den Zugriffsrechten (*User-Id*s, Gruppen-Rechte)
    - Speicher für Daten zur Signalverarbeitung
    - Scheduling-Parameter (z. B. Priorität)
    - Information über Ereignis, auf das der Prozeß wartet
  - das Kommando ***ps(1)***, gibt Auszüge der Prozeßtabelle aus

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Globale Prozeßverwaltungsdaten (3)

- Prozeßidentifikatoren und Prozeßgruppen

pid ***process id*** - eindeutiger Prozeßidentifikator
($0 \leq$ *pid* ≤ 30000) ***getpid(2)***

ppid ***parent process id*** = *pid* des ”Vaterprozesses”
getppid(2)

pgrp ***process group id*** = *pid* des Prozeßgruppenführers
setpgrp(2), getpgrp(2), ***setpgid(2)*** (POSIX)
(siehe auch Abschnitt "Sessions und Prozeßgruppen")

real user id (ruid) und ***real group id (rgid)***
uid und *gid* des Benutzers, der den Prozeß erzeugt hat
(festgelegt durch *login* des Benutzers)
setuid( ), getuid( ), setgid( ), getgid( )

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Globale Prozeßverwaltungsdaten (4)

- … Prozeßidentifikatoren und Prozeßgruppen

effective uid (euid), effective gid (egid), access groups

uid und *group ids,* die während des Programmablaufs zum Prüfen der Zugriffsrechte auf Systemressourcen verwendet werden

– *euid* und *egid* entsprechen normalerweise *ruid* und *rgid*

– *access groups* ist eine Menge zusätzlicher *group ids*, deren Rechte der Prozeß zusätzlich besitzt (BSD)

– beim Laden eines Programms mit *user-s-Bit* wird die *euid* auf die *uid* des Dateibesitzers gesetzt,
analog für *egid*

– ***seteuid( ), geteuid( ), setegid( ), getegid( )***

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Globale Prozeßverwaltungsdaten (5)

- Sessions und Prozeßgruppen
(POSIX 1003.1 legt Standard für Prozeßgruppen und Job-Control fest)
  - Prozesse werden in **Sessions** und **Prozeßgruppen** gruppiert
  - Eine **Session** entspricht i. a. einer Login-Sitzung oder einem Fester in einer Window-Umgebung
    - eine neue *session* wird mit ***setsid(2)*** erzeugt, der aufrufende Prozeß wird dadurch ***session leader***
    - der *session leader* wird gleichzeitig Führer einer eigenen Prozeßgruppe
    - ein *session leader* kann ein (noch keiner *session* zugeordnetes) Terminal als ***controlling terminal*** anfordern,
    er wird dadurch ***controlling process*** des Terminals
    - *session*- und Prozeßgruppen-Mitgliedschaft und damit auch das *controlling terminal* werden an Sohnprozesse vererbt
    - durch *setsid(2)* verliert ein Prozeß sein bisheriges *controlling terminal*

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

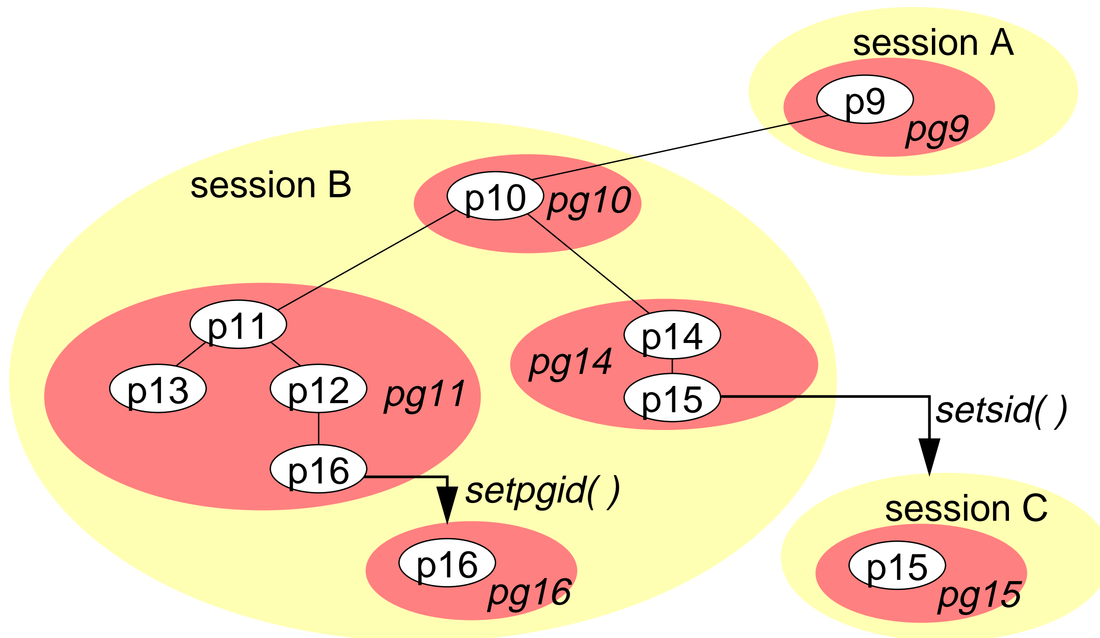
2 Globale Prozeßverwaltungsdaten (6)

- … Sessions und Prozeßgruppen
  - ◆ Eine **Prozeßgruppe** ist eine Menge von Prozessen einer *session*
    - ➤ mit ***setpgid(2)*** kann ein Prozeß die Zuordnung zu einer Prozeßgruppe für sich oder einen seiner Sohnprozesse verändern
      - – Prozeßgruppe, Aufrufer und betroffener Prozeß müssen zu der selben *session* gehören
    - ➤ dem *controlling tty* einer *session* kann eine Prozeßgruppe der *session* zugeordnet werden (***tcsetpgrp( )***)
      - – die Prozesse dieser Prozeßgruppe laufen im **Vordergrund**
      - – die Prozesse aller anderen Prozeßgruppen der *session* laufen im **Hintergrund**
      - – Signale (*Interrupt, Stop, Hangup*) vom Terminal-Treiber werden allen Prozessen dieser Prozeßgruppe zugestellt

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Globale Prozeßverwaltungsdaten (7)

- … Sessions und Prozeßgruppen



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Kommando ps

```
 F S   UID   PID  PPID C PRI NI     ADDR     SZ    WCHAN TTY      TIME CMD
19 T     0     0     0 0   0 SY 10416f88      0          ?        0:00 sched
 8 S     0     1     0 0  41 20 604df608     85 604df800 ?        0:03 init
19 S     0     2     0 0   0 SY 604def48      0 10432c34 ?        0:00 pageout
19 S     0     3     0 0   0 SY 604de888      0 10435cdc ?       34:37 fsflush
 8 S     0   239     1 0  51 20 604dd448    253 604dd640 ?        0:11 sendmail
 8 S 10402  3705     1 1  51 20 608c3620   4808 6098490e console 28:16 maker5X.
 8 S     0   122     1 0  40 20 60693610    257 600a8b8e ?        0:15 rpcbind
 8 S     0   124     1 0  94 20 60692f50    249 600a8aee ?        0:00 keyserv
 8 S     0   135     1 0  41 20 60691b10    231 600a8bb6 ?        0:00 ypbind
 8 S     0   179     1 0  41 20 60691450    479 609848e6 ?        8:06 automoun
 8 S     0   238     1 0  40  8 606906d0    230 606908c8 ?        1:01 xntpd
 8 S     0   201     1 0  51 20 60759b18    206 6002deb8 ?        0:02 cron
 8 S 10402  1776     1 0  50 20 609ea028    396 60985ade console  0:01 xterm
 8 S     0   269     1 0  41 20 60758018    210 604fe060 ?        0:12 sshd
 8 S 10402  3579     1 0  58 20 608c1b20    159 608c1d18 console  0:00 csh
 8 S     0   313     1 0  40 20 608c1460    726 600a8206 ?        0:00 dtlogin
 8 S 10402  3638  3629 0  40 20 60aac8b8    180 60a680be pts/3    0:01 elm
 8 S 10402  3597  3579 0  77 20 608c21e0    121 608c2250 console  0:00 openwin
 8 O     0  8150  3633 0  41 20 609ec1e8    114          pts/2    0:00 ps.exe
 8 S 10402  3601  3597 0  51 20 60aaadb8    331 60aaae28 console  0:00 xinit
 8 S 10402  3693  3668 0  51 20 60a921f0   1963 609853fe console  0:00 netscape
 8 S 10402  1777  1776 0  40 20 609ea6e8    188 60ade4c6 pts/4    0:00 rlogin
 8 S 10402  3613  3603 0  40 20 60a90db0    381 600a813e console  0:45 tvtwm
 8 S 10402 27875     1 0  51 20 60758d98    399 60985ba6 console  0:18 xterm
 8 S 10402  3603  3601 0  49 20 60aab478    121 60aab4e8 console  0:00 .xinitrc
 8 S 10402  3669     1 0  50 20 608c2f60    708 6098585e console  0:04 cm
 8 Z 10402  3610  3603 0   0                                      0:00 <defunct>
 8 R 10402  3602  3601 2  43 20 60a92f70  23544          console 63:44 Xsun
 8 S 10402  3633  3622 0  41 20 609ecf68    276 609ed160 pts/2    0:02 tcsh-6.0
 8 S 10402  3623  3618 0  40 20 608c28a0    258 60a689be pts/0    0:01 tcsh-6.0
 8 S 10402  3668     1 2  51 20 60b5edc0   4165 600a809e console 125:56 netscape
 8 S 10402  3622     1 0  40 20 60a928b0    401 600a804e console  0:03 xterm
 8 S 10402  3680     1 0  40 20 60b5fb40    373 600a818e console  0:00 xclipboa
 8 S 10402 27876 27875 0  40 20 60b60200    286 60b905de pts/1    3:15 elm
 8 T 10402  3799  3633 0  52 20 60b5e700    188          pts/2    0:00 rlogin
 8 Z 10402  3614  3613 0   0                                      0:00 <defunct>
 8 S 10402  3019     1 0  50 20 60a91b30    397 6098463e console  0:01 xterm
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.4 UNIX - Erzeugen eines neuen Prozesses

1 Überblick

- jeder UNIX-Prozeß kann mit dem Systemaufruf ***fork(2)*** einen neuen Prozeß erzeugen
- *fork( )* erzeugt eine nahezu identische Kopie des aufrufenden Prozesses
- der *fork( )* aufrufende Prozeß wird **Vaterprozeß *(parent process)*** genannt
- der durch *fork( )* neu erzeugte Prozeß wird als **Sohnprozeß *(child process)*** bezeichnet
- der Sohnprozeß erbt alle Rechte und Einschränkungen vom Vaterprozeß (einschließlich aller Daten der *user area*, z. B. offene Dateien)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Überblick (2)

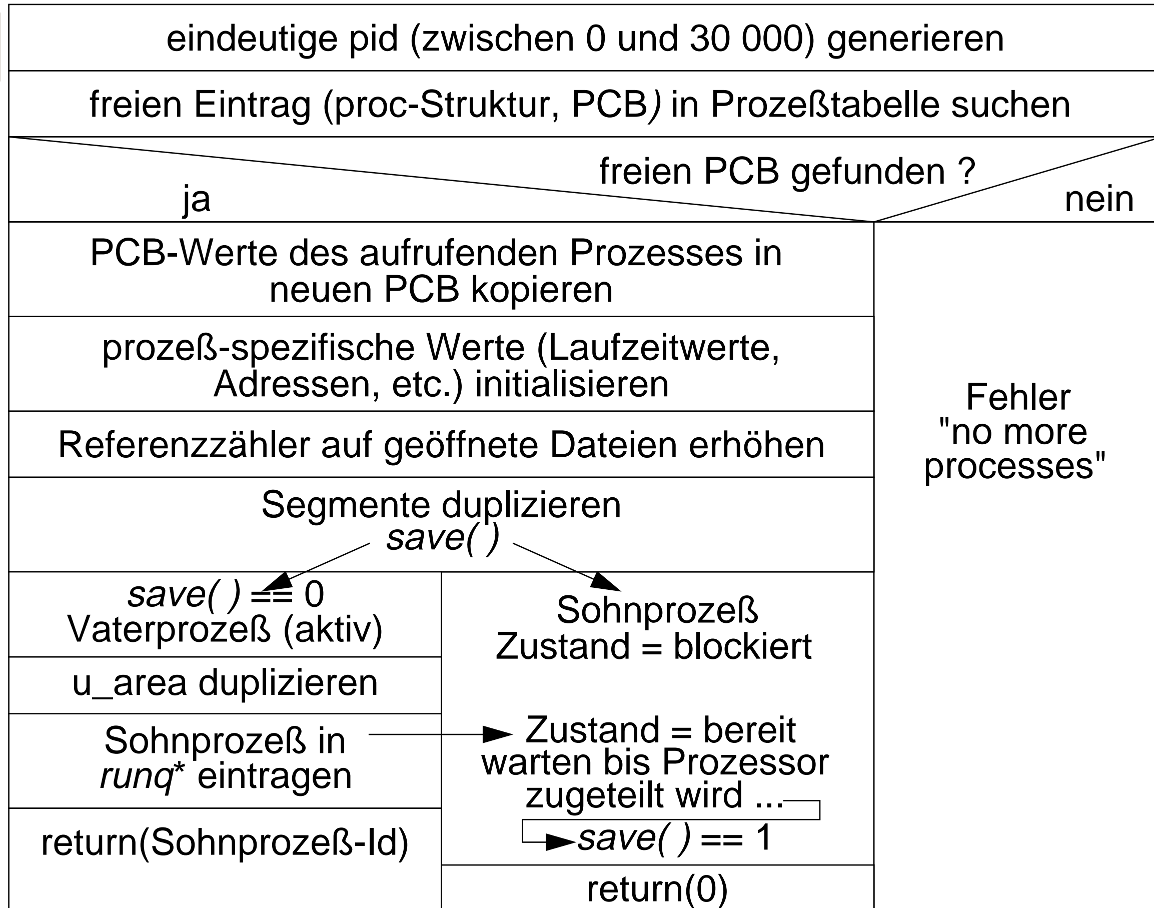
- wesentliche Unterschiede zwischen Vater- und Sohnprozeß:

fork-Ergebnis:	Vaterprozeß: *pid* des Sohnes, Sohnprozeß: 0
PID:	Sohnprozeß erhält nächste freie *pid*
Eintrag in Prozeßtabelle:	Sohnprozeß erhält eigene proc-Structur, Inhalt wird weitgehend vom Vaterprozeß kopiert

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Ablauf

newproc( )



*) *runq*: Warteschlange von bereiten Prozesse in UNIX

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
F-Prozesse.doc 1999-02-24 18.31
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.5 UNIX — Prozeßumschaltungen

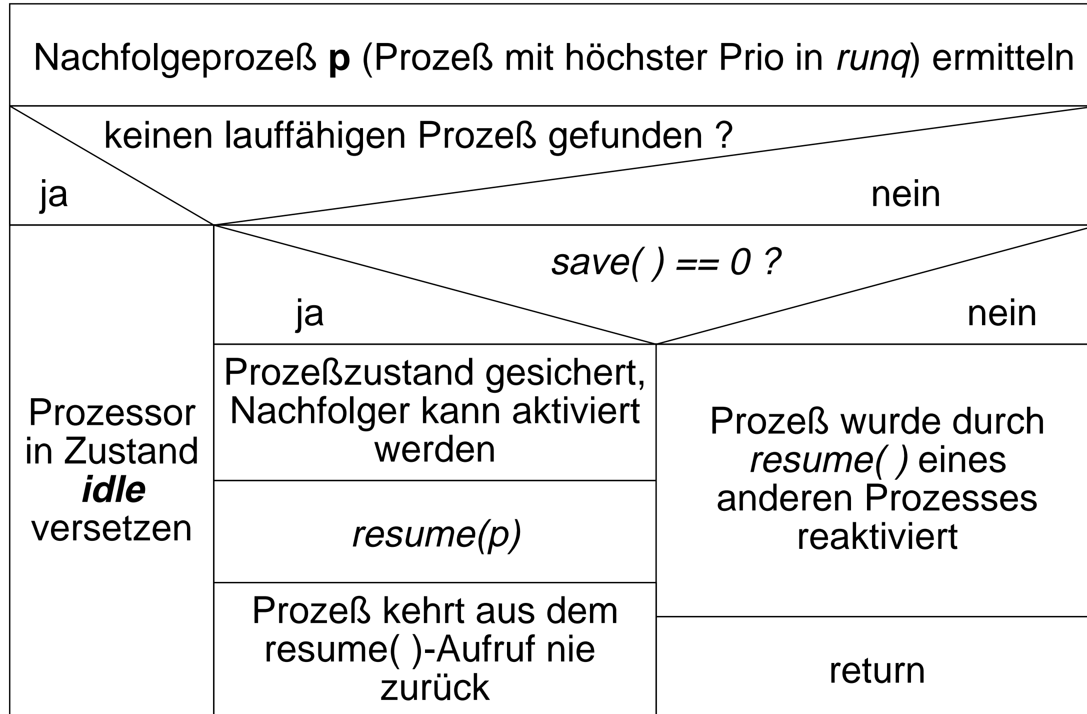
1 Überblick

- Prozeßumschaltungen erfolgen nur an wenigen Stellen im Systemkern:
  - ➥ wenn sich ein Prozeß blockiert (***sleep( )***)
    - – Prozeß aktiviert höchstprioren, laufähigen Prozeß
    - – wenn das Ereignis, auf das gewartet wird, eintritt, wird der Prozeß durch ***wakeup( )*** in die Warteschlange der bereiten Prozesse (***runq***) eingetragen und kann reaktiviert werden
  - ➥ vor Rückkehr aus einem Systemaufruf, wenn ein höherpriorer Prozeß bereitsteht
    - – Prozeß trägt sich in die ***runq*** ein
    - – Prozeß aktiviert höchstprioren, laufähigen Prozeß
- die Prozeßumschaltung erfolgt in den Funktionen ***swtch( )***, ***save( )*** und ***resume( )*** des Systemkerns (stark abhängig von Hardware und UNIX-Version!)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Ablauf (Beispiel aus XELOS)

swtch( )



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Ablauf (2)

save( )

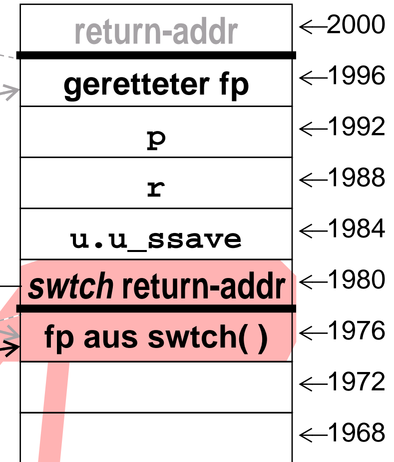
Register, Frame-Pointer, gesicherten Frame-Pointer von *swtch( )* und Rücksprungadresse nach *swtch( )* in ***u-area*** sichern
return(0)

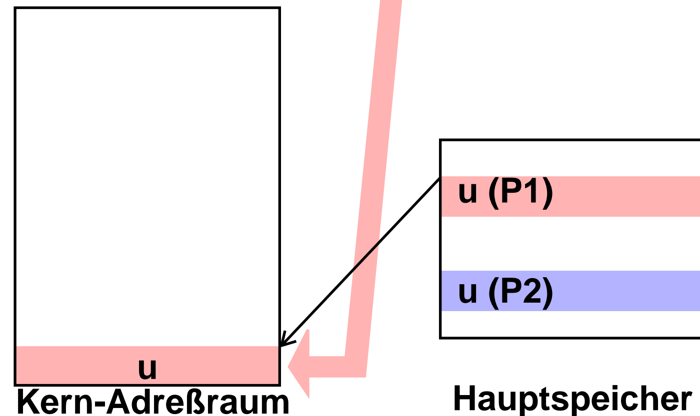
resume( )

u-area des Nachfolgeprozesses auf **u** mappen Kernel-Stack des Nachfolgeprozesses mappen (= eigentliche Prozeßumschaltung)
Register, Frame-Pointer, gesicherten Frame-Pointer von *swtch( )* und Rücksprungadresse nach *swtch( )* aus ***u-area*** restaurieren
return(1) = return aus früherem *save( )*-Aufruf

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

```
swtch() {
     struct proc *p;
     int r;
     p = getrq();

     r = save(u.u_ssave);

     if (r == 0) {
        resume(p);
     } else {
        return(1);
     }
}
save(savearea *s) {
     /* Register,
        sp, fp,
        fp aus swtch und
        swtch return-addr
        -> s sichern */

     return(0);
}
```

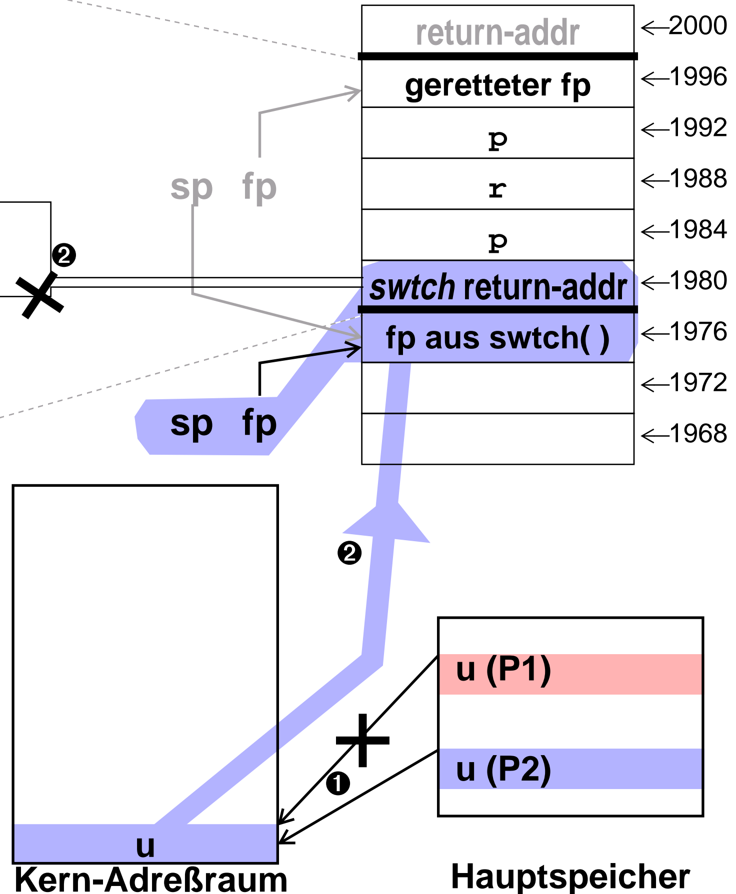




Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

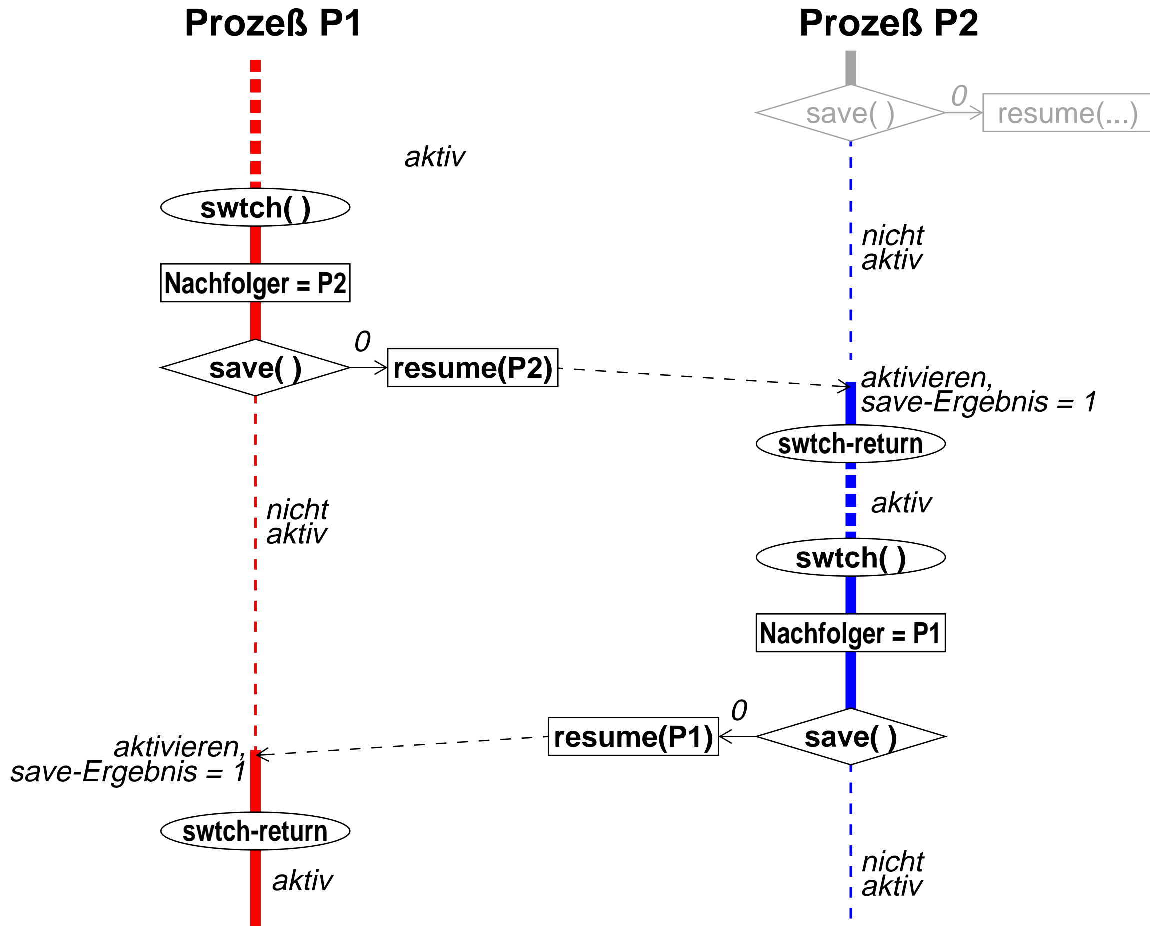
3 ■ Ablauf (4)

```
swtch() {
     struct proc *p;
     int r;
     p = getrq();

     r = save(u.u_ssave);

     if (r == 0) {
          resume(p);
     } else {
          return(1);
     }
}
resume(struct proc *p) {
     /* u-area und
❶         kernel-stack
          von p mappen */

     /* gesicherte Register,
❷         fp und return-addr
          restaurieren */

     return(1);
}
```



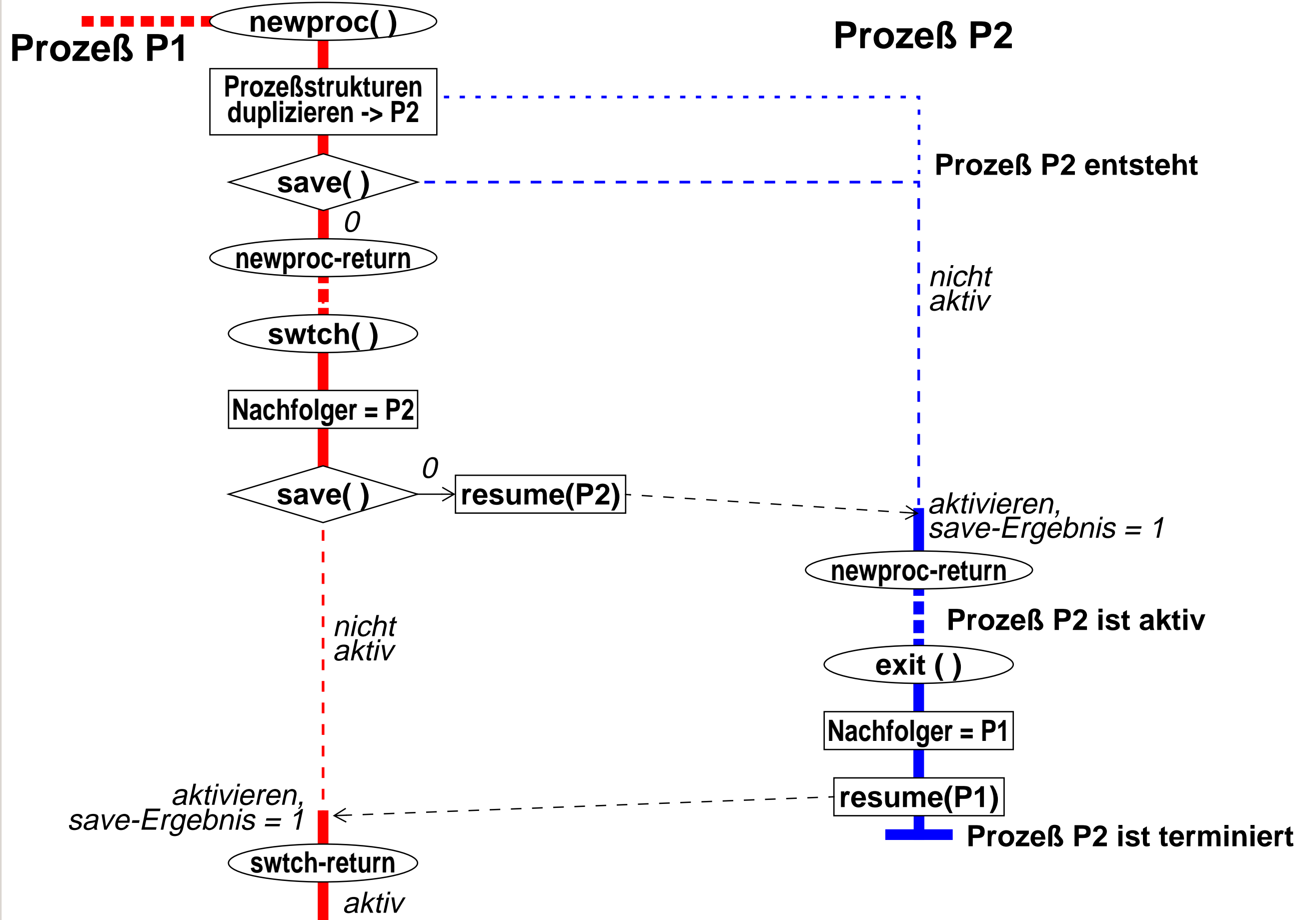
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Erzeugen eines neuen Prozesses



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.6 UNIX — Ausführen eines Programms

- ein UNIX-Prozeß kann die Ausführung eines Programms durch ein anderes Programm ersetzen — ***exec(2), execve(2)***
  - durch Laden eines neuen Programms werden die zuvor von dem Prozeß bearbeiteten Programm-Datenstrukturen zerstört
    ➥ **im Erfolgsfall gibt es kein return aus *exec( )***
  - durch Laden eines neuen Programms entsteht **kein neuer Prozeß**

- *execve( )* werden die Argumente ***(argv)*** und ein Environment ***(envp)*** für das neue Programm mitgegeben

- Details:
  - programmspezifische Daten des Prozesses (Segmente, *signal-handler*-Funktionen) werden initialisiert
  - ***effective uid / effective gid*** des Prozesses werden auf Eigentümer/Gruppe der Programm-Datei geändert, wenn bei dieser ein ***s-bit*** gesetzt ist

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.6 UNIX — Ausführen eines Programms (2)

- … Details:
  - alle anderen Prozeßparameter (in *proc*-Struktur und *user area*) bleiben unverändert (*pid, ppid, pgrp, session, controlling tty, current working directory, current root directory, ...*)
  - **offene Dateideskriptoren bleiben erhalten**, es sei denn, sie wurden als *close-on-exec* (siehe ***fcntl(2)***) markiert

- Programmier-Beispiel:

```
if ( (pid = fork()) < 0 ) {
    perror("fork");
    exit(1)
} else if (pid == 0) {                    /* child process */
    execl("/bin/cp", "cp",    "/tmp/a", "/tmp/b", (char *)0);
    perror("exec");
    exit(1);
} else {                                  /* parent process */
                ...
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.7 UNIX — Terminieren von Prozessen

1 Überblick

- Prozesse terminieren, wenn
  - sie den Systemdienst ***exit(2)*** aufrufen
  - ein Signal an den Prozeß zugestellt wurde, für das keine Signalbearbeitungsfunktion vorgesehen ist

- die Startumgebung für C-Programme ruft nach einem *return* aus der Funktion *main* automatisch *exit(0)* auf

- dem *exit*-Aufruf kann ein Status-Wert (1 Byte) mitgegeben werden, der durch den Vaterprozeß abgefragt werden kann (→ *wait(2)*)

- ein Vaterprozeß kann auf das terminieren von Sohnprozessen warten un deren exit-Status abfragen — ***wait(2)***

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Ablauf von exit im Kern

- System V:
  wenn der Prozeß ein Prozeßgruppenführer mit zugeordnetem Terminal war (typischerweise *login-shells*), wird ein SIGHUP-Signal an alle Prozesse dieser Prozeßgruppe gesendet

- POSIX:
  wenn der Prozeß ein *controlling process* (*session leader* mit zugeordnetem *controlling tty*) war, wird allen Prozessen in der Vordergrund-Prozeßgruppe ein SIGHUP-Signal gesendet, das *controlling tty* wird von der *session* abgekoppelt

- alle offenen Dateideskriptoren werden geschlossen
- alle Segmente (Text, Daten, Stack) werden freigegeben; *shared memory* wird abgekoppelt, aber nicht gelöscht
- *user area* (enthält Kernel-Stack auf dem exit operiert!) und *proc*-Struktur bleiben noch erhalten
- alle noch existierenden Sohnprozesse erhalten den *init*-Prozeß (*pid* = 1) als neuen Vaterprozeß
- Statistikdaten und exit-Status werden in der *proc*-Struktur notiert

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Ablauf von exit im Kern (2)

- neuer Prozeßzustand: **ZOMBIE**
- Vaterprozeß wird informiert
  (*wakeup( ),* falls in *wait( )* blockiert, und Signal SIGCLD/SIGCHLD)
- Nachfolgeprozeß bestimmen und Prozeßwechsel

■ ZOMBIE-Prozesse bleiben so lange bestehen, bis der Vaterprozeß mit dem Systemaufruf ***wait(2)*** den exit-Status abfragt

- mit *wait( )* werden Statistikdaten an den Vaterprozeß übertragen und können danach mit ***times(2)*** abgefragt werden
  - ***times(2)*** liefert die Summe der Laufzeiten aller Sohnprozesse, auf die mit ***wait(2)*** gewartet wurde
- alternativ können die Statistikdaten des Sohnprozesses mit ***wait3(2)*** oder ***wait4(2)*** direkt abgefragt werden
- der Eintrag in Prozeßtabelle wird freigegeben
- die *user area* des ZOMBIE-Prozesses ist jetzt passiv (Kernel-Stack wird nicht mehr genutzt) und kann freigegeben werden

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Systemaufruf *wait( )*

- ein Prozeß kann warten, bis ein Sohnprozeß terminiert oder gestoppt wird und dabei den Status des Sohnprozesses abfragen
  ➥ übliche Arbeitsweise einer Shell bei Vordergrundprozessen

- ***wait(2)*** blockiert den aufrufenden Prozeß so lange, bis ein Sohnprozeß im Zustand ZOMBIE existiert oder ein Sohnprozeß gestoppt wird
  - ◆ *pid* dieses Sohnprozesses wird als Ergebnis geliefert
  - ◆ als Parameter kann ein Zeiger auf einen *int*-Wert mitgegeben werden, in dem der Status (16 Bit) des Sohnprozesses abgelegt wird

Sohnprozeß ist	Status & `0xff00`	Status & `0x00ff`
gestoppt	Signalnummer	`0xff`
terminiert durch *exit( )*	exit-code	`0x00`
terminiert durch Signal	`0x00`	Signalnummer
terminiert durch Signal + *coredump*	`0x00`	Signalnummer + `0x80`

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
F-Prozesse.doc 1999-02-24 18.31
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Systemaufruf *wait ( )* (2)

- existiert kein Sohnprozeß, liefert *wait( )* eine Fehlermeldung
- in BSD und POSIX gibt es zusätzlich die Möglichkeit, das Blockieren beim *wait*-Aufruf zu verhindern, wenn keine terminierten oder gestoppten Sohnprozesse existieren
  - ***wait3(2)***, ***waitpid(2)*** oder ***wait4(2)*** mit Option *WNOHANG*
- in POSIX kann man mit *wait4( )* oder *waitpid( )* die Menge der Prozesse, auf die gewartet wird einschränken

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 typische *wait( )*-Anwendung

- ein Prozeß führt eine Liste aller existierenden Sohnprozesse und interessiert sich für deren exit-Status, darf selbst aber nicht durch *wait( )* blockiert werden

  → Lösung: Signal SIGCLD (SystemV) bzw. SIGCHLD (Berkeley) auswerten

  ◆ großer Unterschied zwischen UNIX SystemV und Berkeley-UNIX!

▲ Verfahren bei Berkeley-UNIX

  ◆ Prozeß setzt s*ignalhandler*-Funktion für Signal SIGCHLD

  ◆ in der *signalhandler*-Funktion wird so lange *wait3( )* mit Option *WNOHANG* aufgerufen, bis *wait3( )* meldet, daß keine weiteren Söhne in gestopptem oder terminiertem Zustand existieren

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 typische *wait( )*-Anwendung

▲ Verfahren bei SystemV

- Prozeß setzt s*ignalhandler*-Funktion für Signal SIGCLD
- in der *signalhandler*-Funktion wird **genau ein** *wait( )*-Aufruf ausgeführt, durch den der Prozeß nicht blockiert werden kann
- da die *signalhandler*-Funktion bei Eintreffen eines Signals zurückgesetzt wird, muß sie am Ende der Signalbearbeitung neu aufgesetzt werden
- beim Setzen einer *Handler*-Funktion für SIGCLD stellt der Systemkern ein neues SIGCLD zu, wenn noch Sohnprozesse im ZOMBIE-Zustand existieren

■ weitere Besonderheiten von SIGCLD in SystemV

- SIGCLD führt (im Gegensatz zu anderen Signalen) per default nicht zum Abbruch eines Prozesses - es wird einfach ignoriert
- wird für SIGCLD explizit vereinbart, daß es ignoriert werden soll, werden Sohnprozesse automatisch - ohne *wait( )*-Aufruf - vom Kern aus dem ZOMBIE-Zustand befreit

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.8 MACH — Tasks und Threads

1 Motivation

UNIX-Prozeßkonzept ist für viele heutige Anwendungen unzureichend

- in Multiprozessorsystemen werden häufig parallele Abläufe in einem virtuellen Adreßraum benötigt
- zur besseren Strukturierung von Problemlösungen sind oft mehrere Aktivitätsträger innerhalb eines Adreßraums nützlich
- typische UNIX-Server-Implementierungen benutzen die *fork*-Operation, um einen Server für jeden Client zu erzeugen
  - ➥ Verbrauch unnötig vieler System-Ressourcen (Datei-Deskriptoren, Page-Table, Speicher, ...)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Vergleich von Prozeß- und Thread-Konzepten

- mehrere **UNIX-Prozesse** mit gemeinsamen Speicherbereichen

Bewertung:

+ echte Parallelität möglich

– viele Betriebsmittel zur Verwaltung eines Prozesses notwendig; Prozeßumschaltungen aufwendig → teuer

– innerhalb einer solchen Prozeßfamilie wäre häufig ein anwendungsorientiertes Scheduling notwendig: schwierig realisierbar

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Vergleich von Prozeß- und Thread-Konzepten (2)

- ***User-Level-Threads*** (Koroutinen) — Realisierung von Threads auf Benutzerebene innerhalb eines Prozesses

Bewertung:

+ Erzeugung von Threads und Umschaltung extrem billig
+ Verwaltung und Scheduling anwendungsorientiert möglich
– Systemkern hat kein Wissen über diese Threads
  - ➥ Scheduling zwischen den Koroutinen schwierig (Verdrängung meist nicht möglich)
  - ➥ in Multiprozessorsystemen keine parallelen Abläufe möglich
  - ➥ wird eine Koroutine wegen eines *page faults* oder in einem Systemaufruf blockiert, ist der gesamte Prozeß blockiert

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Vergleich von Prozeß- und Thread-Konzepten (3)

- ***Kernel-Threads***: leichtgewichtige Prozesse *(lightweight processes)*

Bewertung:

\+ eine Gruppe leichtgewichtiger Prozesse nutzt gemeinsam eine Menge von Betriebsmitteln

\+ jeder leichtgewichtige Prozeß ist aber als eigener Aktivitätsträger dem Betriebssystemkern bekannt

- eigener Programmzähler
- eigener Registersatz
- eigener Stack

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Vergleich von Prozeß- und Thread-Konzepten (3)

... Bewertung *Kernel-Threads (lightweight processes)*

+ Umschalten zwischen zwei leichtgewichtigen Prozessen einer Gruppe ist erheblich billiger als eine normale Prozeßumschaltung
  - ➥ es müssen nur die Register und der Programmzähler gewechselt werden (entspricht dem Aufwand für einen Funktionsaufruf)
  - ➥ Adreßraum muß nicht gewechselt werden
  - ➥ alle Systemressourcen bleiben verfügbar

– Kosten für Erzeugung und Umschaltung zwar erheblich geringer als bei "schwergewichtigen" Prozessen, aber erheblich teurer als bei *user-level-Threads*

– Verwaltung und Scheduling meist durch Kern vorgegeben

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Vergleich von Prozeß- und Thread-Konzepten (4)

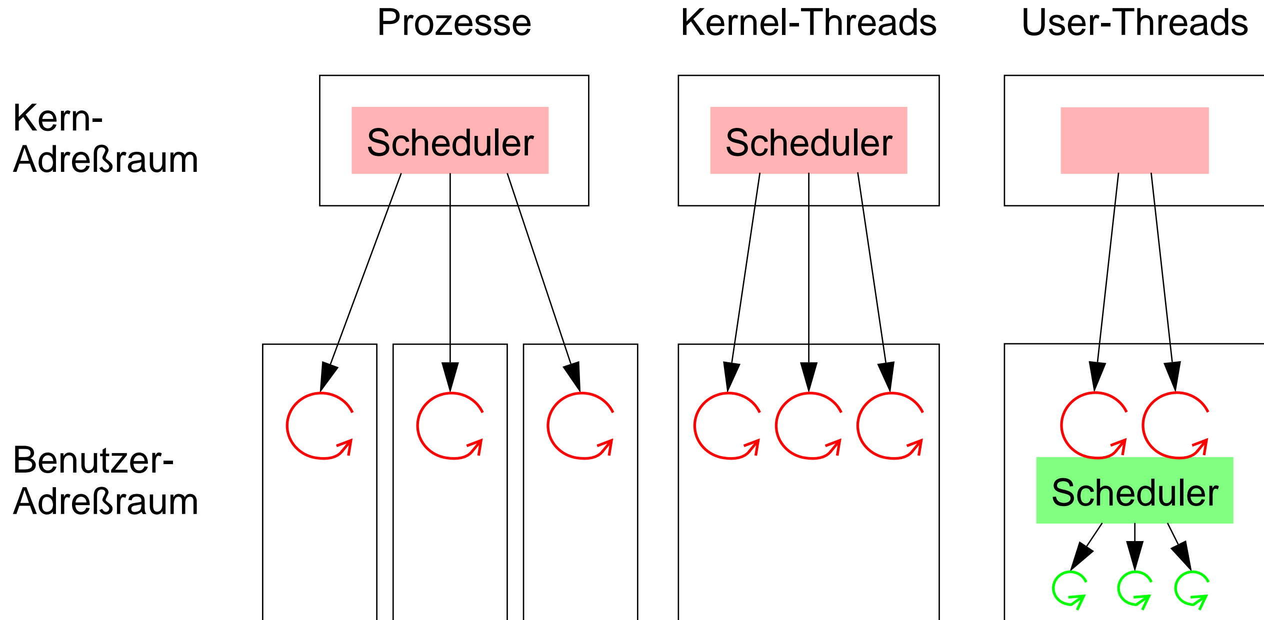
- Vergleich

	Prozesse	Kernel-Threads	User-Threads
Kosten	– teuer	O mittel	+ billig
Betriebssystemeingliederung	+ gut	+ gut	– schlecht
Interaktion untereinander	– schwierig	+ einfach	+ einfach
Benutzerkonfigurierbarkeit	– nein	– nein	+ ja
Gerechtigkeit	– nein	+ ja	± teils

– Gerechtigkeit bedeutet:
wie kommt das System damit klar, wenn eine Anwendung eine große Anzahl von Aktivitätsträgern erzeugt, eine andere dagegen eine geringe — werden Zeitscheiben an Anwendungen oder an Aktivitätsträger vergeben?

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Vergleich von Prozeß- und Thread-Konzepten (5)



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Abstraktionen in MACH: Tasks und Threads

▲ ***Task:*** Betriebsumgebung (System-Resourcen) für Aktivitätsträger

- ➤ virtueller Adreßraum
- ➤ Zugriffsrechte (= Portrechte)
- ➤ Betriebsmittel-Informationen
- ➤ kein Programm-Ablauf und keine Register

▲ ***Thread:*** Aktivitätsträger und seine Ablaufumgebung

- ➤ Registersatz
- ➤ Stack
- ➤ Programmzähler

◆ Innerhalb einer *Task* können beliebig viele *Threads* existieren

◆ In einer echten Multiprozessorumgebung können verschiedene *Threads* einer *Task* parallel auf mehreren Prozessoren ablaufen

◆ Traditioneller UNIX-Prozeß ≙ MACH-*Task* mit einem *Thread*

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.9 UNIX — Prozesse, LWPs & Threads

- Thread-Konzept zunehmend auch in UNIX-Systemen realisiert
  - Solaris
  - HP UX
  - Digital UNIX
  - Linux
  - …
- Programmierschnittstelle standardisiert: **Pthreads-Bibliothek**
  - ➥ IEEE POSIX Standard P1003.4a
- Pthreads-Implementierungen aber sehr unterschiedlich!
  - ➤ reine User-level-Threads (Linux, HP-UX)
  - ➤ reine Kernel-Threads (MACH, KSR-UNIX, Digital UNIX)
  - ➤ parametrierbare Mischung (Solaris)
- Daneben z. T. auch andere Thread-Bibliotheken (z. B. Solaris-Threads)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 pthread-Benutzerschnittstelle

Pthreads-Schnittstelle (Basisfunktionen):

pthread_create	Thread erzeugen & Startfunktion angeben
pthread_exit	Thread beendet sich selbst
pthread_cancel	Anderen Thread beenden
pthread_join	Auf Ende eines anderen Threads warten
pthread_self	Eigene Thread-Id abfragen
pthread_yield	Prozessor zugunsten eines anderen Threads aufgeben

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 pthread-Benutzerschnittstelle (2)

- Threaderzeugung

```
pthread_create(pthread_t *thread,
               const pthread_attr_t *attr,
               void *(*start_routine)(void *),
               void *arg)
```

thread Thread-Id

attr modifizieren von Attributen des erzeugten Threads (z. B. Stackgröße). **NULL** für Standardattribute.

Thread wird erzeugt und ruft Funktion **start_routine** mit Parameter **arg** auf

- Thread beenden (bei return aus **start_routine** oder):

```
void pthread_exit(void *retval)
```

- Auf Thread warten und exit-Status abfragen:

```
pthread_join(pthread_t thread, void **retvalp)
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

- Beispiel (Multiplikation Matrix mit Vektor)

```
double a[100][100], b[100], c[100];

main() {
    pthread_t tids[100];
...
    for (i = 0; i < 100; i++)
        pthread_create(tids + i, NULL, mult,
                                (void *)(c + i));
    for (i = 0; i < 100; i++)
        pthread_join(tids[i], NULL);
...
}

void *mult(void *cp) {
    int j, i = (double *)cp - c;
    double sum = 0;

    for (j = 0; j < 100; j++)
        sum += a[i][j] * b[j];
    c[i] = sum;
    return 0;
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

★ fork-Semantik

- Problem
  - alle Threads duplizieren?
  - nur den aufrufenden Thread duplizieren?
- Lösung im Pthreads-Standard
  - im neu erzeugten Prozeß existiert nur der Thread, der fork aufgerufen hat
- Alternative bei Solaris-Threads
  - ***Fork One Model***:
    nur der aufrufende Thread wird in den Sohnprozeß übernommen
  - ***Fork All Model:***
    Sohnprozeß enthält alle Threads

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Threads & UNIX-Semantik (2)

★ Signal-Semantik

- Grundlegendes Signal-Behandlungskonzept unverändert:
  - ➤ Signal ignorieren / Default-Reaktion / Signal abfangen
  - ➥ Einstellung gilt immer für alle Threads eines Prozesses
- Problem
  - ◆ welchem Thread wird ein eintreffendes Signal zur Bearbeitung zugestellt

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Signal-Semantik (2)

Signal-Zustellung an Threads

- Lösung unterscheidet nach der Art der Signal-Entstehung
  - Traps: synchron durch die Programmausführung erzeugt (Segmentation fault, Illegal instruction, …)
    - ➤ Signal wird an den verursachenden Thread zugestellt
  - Explizite Signalerzeugung durch das Programm (Funktion *pthread_kill*)
    - ➤ Signal wird an den angegebenen Ziel-Thread zugestellt
  - Interrupts: asynchron von "außen" erzeugt (Interrupt, Quit, Hangup, SIGIO, …)
    - ➤ Signal wird dem gesamten Prozess zugestellt

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Threads & UNIX-Semantik (3)

★ Signal-Semantik (3)

- Signal-Masken
  - ◆ Signale können in UNIX maskiert werden — die Zustellung zurückgehalten
- Signal-Masken werden von Threads an neu erzeugte Threads vererbt
- Threads können Thread-spezifische Signal-Masken setzen (Funktion *pthread_sigmask*)

▲ Bearbeitung von Signalen an den gesamten Prozess

- ◆ Zustellung erfolgt an einen der Threads, die das Signal nicht blockiert haben
- ◆ Auswahl des Threads erfolgt zufällig

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.10 Koordinierungsmechanismen

1 UNIX — Semaphore

- Atomare Operationen auf einem Semaphorvektor
  Realisierung folgender theoretischer Konzepte:
  - ◆ Vektoradditionssystem (VAS)
    - ➥ mehrere Semaphorvariablen können in einer atomaren Operation individuell dekrementiert (P-Operation) bzw. inkrementiert (V-Operation) werden
  - ◆ up/down-System - ähnliche Funktionalität
    (Lösung des 2. Leser/Schreiber-Problems)
    - ➥ in einer atomaren Operation können mehrere Variablen auf 0 getestet werden und im Erfolgsfall werden Variablen des Semaphorvektors modifiziert
- Include-Dateien: `<sys/types.h>` `<sys/ipc.h>` `<sys/sem.h>`

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 UNIX — Semaphore (2)

- Systemschnittstelle:

semget(2) Erzeugen eines neuen Semaphorverktors, bzw. Zugriff zu einem bestehenden Semaphor besorgen

semcntl(2) Statuskontrolle, Initialisierung und Löschen

semop(2) Semaphor-Operationen

- Semaphor-Operation

➥ **Aufruf:**

```
int semid;
int nsops = N;
struct sembuf sbuf[N];
...
semop(semid, sbuf, nsops);
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 UNIX — Semaphore (3)

- Eine Semaphor-Operation ist aus mehreren Einzeloperationen zusammengesetzt, jede Einzeloperation ist durch eine Struktur ***sembuf*** beschrieben:
  - ➥ Struktur:

```
struct sembuf {
    short sem_num;              /* Sem. Nummer
*/
    short sem_op;               /* Operation
*/
```

sem_num	Nummer der Semaphorkomponente auf die sich die Operation bezieht
sem_op < 0	P-Operation (VAS)
sem_op > 0	V-Operation (VAS)
sem_op = 0	Test auf 0 (2. Leser/Schreiber-Problem)

- Eine Semaphor-Operation blockiert immer dann als ganzes, wenn eine der Einzeloperationen blockiert!

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 UNIX — Semaphore (4)

★ UNIX-Semaphore als Vektoradditionssystem

- vgl. F. Hofmann: Betriebssysteme: Grundkonzepte und Modellvorstellungen; 2. Aufl., Kap. 2.4, S. 73

Vektoraddition: $e \in \mathbb{Z}^m$, Semaphorvektor $S \in \mathbb{Z}^m$,

Φ: $'S + e \geq 0$ Θ: $S = {'S} + e$

Realisierung:

Eine Operation e auf dem Semaphorvektor wird durch ein Feld von sembuf-Strukturen beschrieben:

$\forall\ e_i \in e \wedge e_i \neq 0$: $\exists$ sembuf (sembuf.sum_num = i $\wedge$ sembuf.sem_op = e_i)

$\forall\ e_i \in e \wedge e_i = 0$: $\nexists$ sembuf (sembuf.sum_num = i)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

… UNIX-Semaphore als Vektoradditionssystem

sem_op < 0 P-Teiloperation

sem_op wird vom Wert der Semaphorkomponente abgezogen, wenn diese dadurch nicht kleiner 0 wird.
Gesamtoperation blockiert oder liefert Fehler (je nach ***flag***) wenn nicht möglich

sem_op > 0 V-Teiloperation

sem_op wird zum Wert der Semaphorkomponente addiert

kein sembuf für Semaphorkomponenten, die unverändert bleiben sollen!

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 UNIX — Semaphore (6)

★ **2. Leser-Schreiber-Problem**

◆ vgl. F. Hofmann: Betriebssysteme: Grundkonzepte und Modellvorstellungen; 2. Aufl., Kap. 2.4, S. 76

■ UNIX-Sempahore realisieren kein echtes *up/down*-System, entscheidend für das 2. Leser-Schreiber-Problem ist aber:

- ➥ atomarer Test mehrerer Variablen, verbunden mit Modifikation anderer Variablen
- ➥ beliebig viele P-Operationen ermöglichen und so lange warten können, bis die entsprechenden V-Operationen komplett durchgeführt wurden
  - ➤ alle anderen Koordinierungssysteme können nur von einem endlichen Wert bis 0 belegen!

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

... **2. Leser-Schreiber-Problem**

- Lösung mit UNIX-Semaphoren
  - ➥ Semaphorvektor mit 3 Komponenten
    1. gegenseitiger Ausschluß zwischen Schreibern
    2. Zähler für Leser
    3. Anmeldung der Schreiber

    Vorbelegung: $\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$

  - ➥ **Leser**

    semop $\begin{pmatrix} - \\ 1 \\ 0 \end{pmatrix}$; *lesen*; semop $\begin{pmatrix} - \\ -1 \\ - \end{pmatrix}$;

  - ➥ **Schreiber**

    semop $\begin{pmatrix} - \\ - \\ 1 \end{pmatrix}$; semop $\begin{pmatrix} -1 \\ 0 \\ - \end{pmatrix}$; *schreiben*; semop $\begin{pmatrix} 1 \\ - \\ -1 \end{pmatrix}$;

    – bedeutet, daß bei semop für die entsprechende Vektorkomponente keine sembuf-Struktur übergeben wird!

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Pthreads-Koordinierung

- UNIX-Semaphore für Koordinierung von leichtgewichtigten Prozessen zu teuer
  - ◆ Implementierung durch den Systemkern
  - ◆ komplexe Datenstrukturen
- Bei Koordinierung von Threads reichen meist einfache ***mutex***-Semaphore
  - ◆ gewartet wird durch Blockieren des Threads oder durch *busy wait (Spinlock)*
- Komplexere Semaphore können alleine mit Mutexes nicht implementiert werden
  - ➥ Problem:
    - • Ein Mutex sperrt die Datenstruktur des komplexen Semaphors
    - • Der Zustand der Datenstruktur erlaubt die Operation nicht
    - • Blockieren an einem weiteren Mutex kann zu Verklemmungen führen
  - ➥ Lösung: mutex in Verbindung mit sleep/wakeup-Mechanismus
  - ➥ **Condition Variables**

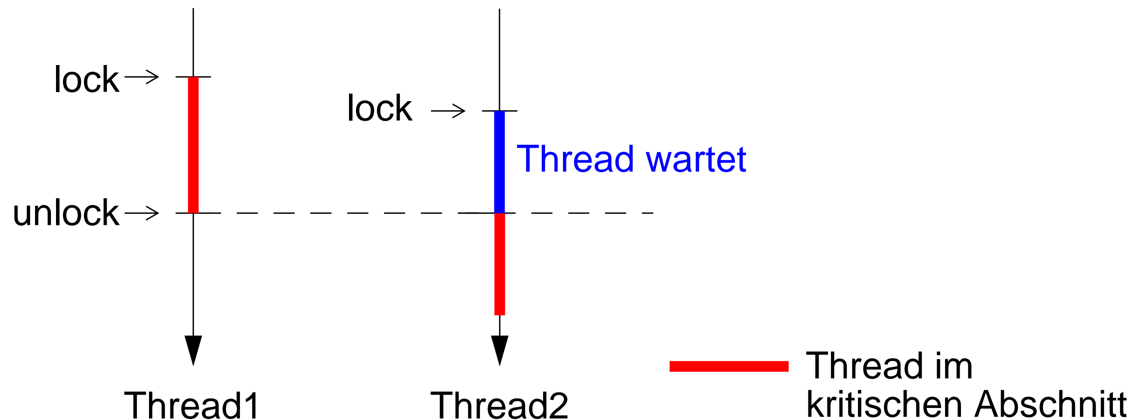
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Pthreads-Koordinierung (2)

★ Mutexes

- Koordinierung von kritischen Abschnitten



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

… Mutexes (2)

- Realisierung:
  - Implementierung weitgehend im Anwendungsmodus
  - Voraussetzung: atomarer Maschinenbefehl zum Testen und Belegen der Semaphorvariablen
    (z. B. Test-and-Set oder Compare-and-Swap)
  - Ist Semaphor frei kann, der Thread den kritischen Abschnitt ohne Verzögerung betreten
  - Ist Sempahor belegt kann alternativ
    - der Thread den Prozessor aufgeben (thread_yield)
      (normale Pthreads-Vorgehensweise)
    - der Thread aktiv auf die Freigabe warten (Spin)
      (Option z. B. bei der KSR Pthreads-Implementierung)
  - Bei der unlock-Operation werden blockierte Threads aufgeweckt (thread_resume)

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

… Mutexes (3)

Schnittstelle

- Mutex erzeugen

```
pthread_mutex_t m1
s = pthread_mutex_init(&m1, pthread_mutexattr_default);
```

- Lock & unlock

```
s = pthread_mutex_lock(&m1);
... kritischer Abschnitt
s = pthread_mutex_unlock(&m1);
```

- weitere Funktionen

pthread_mutex_trylock: wie lock, blockiert aber nicht wenn krit. Abschnitt belegt ist

pthread_mutex_getowner: liefert pthread-ID des aktuellen lock-Besitzers

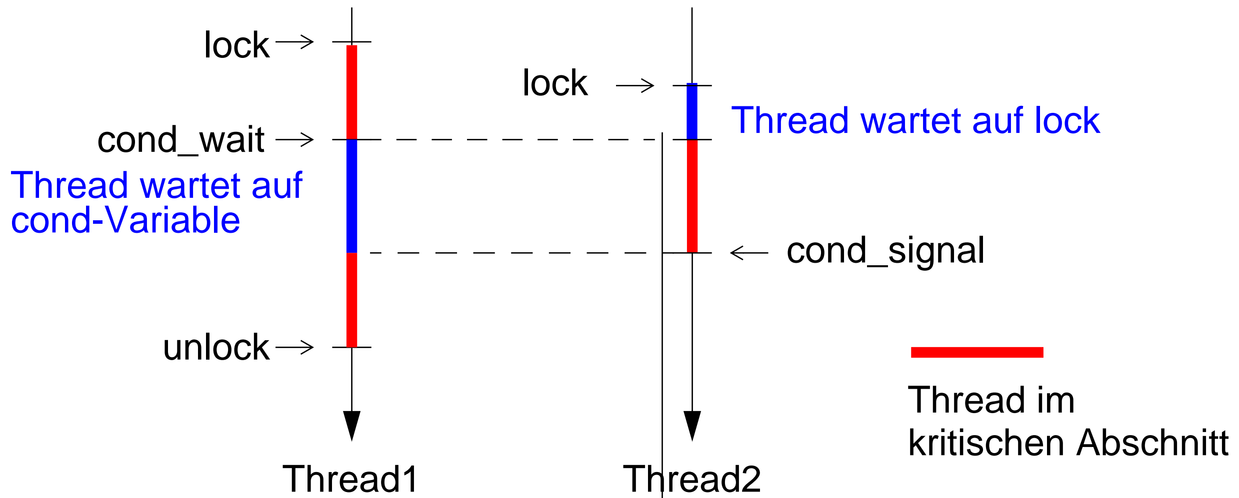
pthread_mutex_destroy: löscht Mutex

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Pthreads-Koordinierung (5)

★ Condition Variables

- Mechanismus zum Blockieren (mit gleichzeiter Freigabe des aktuellen kritischen Abschnitts) und Aufwecken (mit neuem Betreten des kritischen Abschnitts) von Threads



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Pthreads-Koordinierung (6)

... Condition Variables (2)

- Realisierung
  - Thread reiht sich in Warteschlange der Condition Variablen ein
  - Thread gibt Mutex frei
  - Thread gibt Prozessor auf
  - Ein Thread der die Condition Variable "frei" gibt weckt einen (oder alle) darauf wartenden Threads auf
  - Deblockierter Thread muß als erstes den kritischen Abschnitt neu betreten (lock)
  - Da möglicherweise mehrere Threads deblockiert wurden, muß die Bedingung nochmals überprüft werden
    (Analogie zu UNIX sleep/wakeup !)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Pthreads-Koordinierung (7)

... Condition Variables (3)

■ Schnittstelle

◆ Condition Variable erzeugen

```
pthread_cond_t c1;
s = pthread_cond_init(&c1, pthread_condattr_default);
```

◆ Betriebsmittel belegen

```
pthread_mutex_lock(&m1);
    while ( resource_busy )
      s = pthread_cond_wait
            (&c1, &m1);
    resource_busy = TRUE;
pthread_mutex_unlock(&m1);
... /* Betriebsmittel nutzen */
```

Betriebsmittel freigeben

```
pthread_mutex_lock(&m1);
    resource_busy = FALSE;
    pthread_cond_signal(&c1);
pthread_mutex_unlock(&m1);
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Pthreads-Koordinierung (8)

... Condition Variables (4)

- Bei **`pthread_cond_signal`** wird mindestens einer der wartenden Threads augeweckt — es ist allerdings nicht definiert welcher

- Mit **`pthread_cond_broadcast`** werden alle wartenden Threads aufgeweckt

- Ein aufwachender Thread wird als erstes den Mutex neu belegen — ist dieser gerade gesperrt bleibt der Thread solange blockiert!

- Lost Wake-Up Problem
  - Aufruf von **`pthread_cond_signal`** nach Freigabe des Mutex-LOcks würde eine Thread-Umschaltung sparen
  - Aber: wenn Aufruf von **`pthread_con_signal`** erfolgt, während der andere Thread zwischen **`resource_busy`**-Abfrage und **`pthread_cond_wait`** ist, geht das Signal verloren

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Thread-Cancellation

Probleme

- Thread hält Locks, Thread hat Ressourcen allokiert
- Thread bearbeitet gerade komplexe Datenstrukturen und diese sind momentan in inkonsistentem Zustand

Lösungen

- Cancellation in bestimmten Situationen verhindern,
  - ➥ cancelability state: PTHREAD_CANCEL_DISABLE
- Cancellation verzögern, wird er am nächsten *Cancellation Point* wirksam
  - ➥ cancelability type: PTHREAD_CANCEL_DEFERRED
- *Cleanup Stack*: Liste von Funktionen, die beim Termineren aufgerufen werden

Funktionen

- *pthread_cancel(3T), pthread_setcancelstate(3T), pthread_setcanceltype(3T), pthread_textcancel(3T), pthread_cleanup_push(3T), pthread_cleanup_pop(3T)*

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Koordinierung im UNIX-Kern (sleep/wakeup)

- der Ablauf eines Prozesses muß häufig mit dem anderer Prozesse oder mit äußeren Ereignissen *(external event)* koordiniert werden
  - Daten sind nicht immer unmittelbar verfügbar, wenn sie angefordert werden (z. B. Zeichen von Terminal, Block von Platte) *(ext. event)*
  - Semaphoroperationen blockieren
  - (quasi-)gleichzeitige Zugriffe auf globale Datenstrukturen des Systemkerns müssen geeignet koordiniert werden z. B. ***mutual exclusion***
- Betriebssystem-interner Koordinierungsmechanismus
  - Prozessor muß für andere, lauffähige Prozesse freigeben werden
  - sobald der Grund für die Blockierung nicht mehr besteht, muß der blockierte Prozeß in den Zustand **bereit** zurückgeführt werden
  - hat ein Prozeß, als er sich blockiert hat, bereits wichtige Betriebsmittel belegt (z. B. *inode* gesperrt, *inode* muß von Platte geladen werden), muß er möglichst schnell in den Zustand **laufend** zurückgelangen

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Koordinierung im UNIX-Kern (2) — sleep —

- Prozeß blockiert sich an einem Ereignis: ***wchan (wait channel)***
- in der Praxis ist *wchan* die Adresse einer Datenstruktur, die mit dem Ereignis korrespondiert - z. B. *inode*-Struktur-Adresse
- die *proc*-Strukturen blockierter Prozesse werden verkettet, die Anker dieser Queues werden durch Hashing aus dem *wchan* ermittelt
- Parameter gibt an, ob Prozeß durch Signal unterbrechbar ist
  - ◆ Abbruch wird verboten, wenn Prozeß in einem Vorgang blockiert wird, der in jedem Fall zu Ende gebracht werden muß
    (z. B. weil noch Datenstrukturen gesperrt sind)
    - ➥ Prozeß wird nur durch *wakeup* auf *wchan*, nicht aber durch eintreffende Signale aufgeweckt
  - ◆ sonst wird der Prozeß entweder durch *wakeup* auf *wchan* oder Zustellung eines Signals deblockiert

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Koordinierung im UNIX-Kern (3) — sleep —

- es wird die Priorität angegeben, mit der der Prozeß wieder ausgeweckt werden soll
  - ◆ sehr hohe Priorität bei Vorgängen, die schnell fortgeführt werden müssen (z. B. weil noch *locks* gehalten werden)
- es wird ein Prozeßwechsel durchgeführt

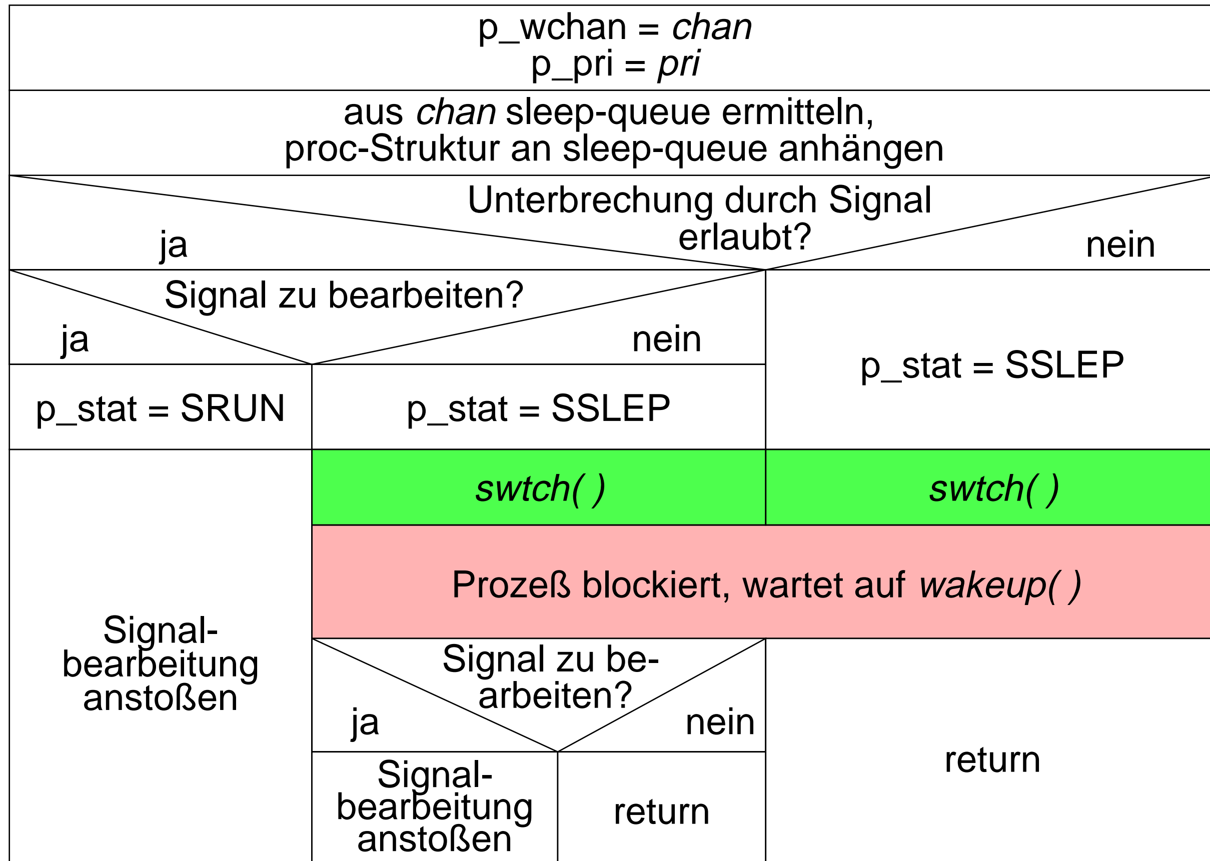
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Koordinierung im UNIX-Kern (4) — wakeup —

- für Ereignisse, auf die Prozesse warten können, wird ein ***wakeup( )*** ausgeführt
- es werden **alle** Prozesse in den Zustand **bereit** versetzt, die auf den angegebenen ***wchan*** gewartet haben
- jeder Prozeß muß überprüfen, ob der Blockierungsgrund wirklich nicht mehr besteht (z. B. mehrere Prozesse haben auf Freigabe eines *locks* gewartet und der erste davon hat das *lock* bereits wieder belegt)
- jeder Prozeß erhält die bei seinem ***sleep( )***-Aufruf angegebene Priorität
- die *wakeup*-Priorität wird bis zum Verlassen des Systemkerns (oder erneutem *sleep/wakeup*) beibehalten

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

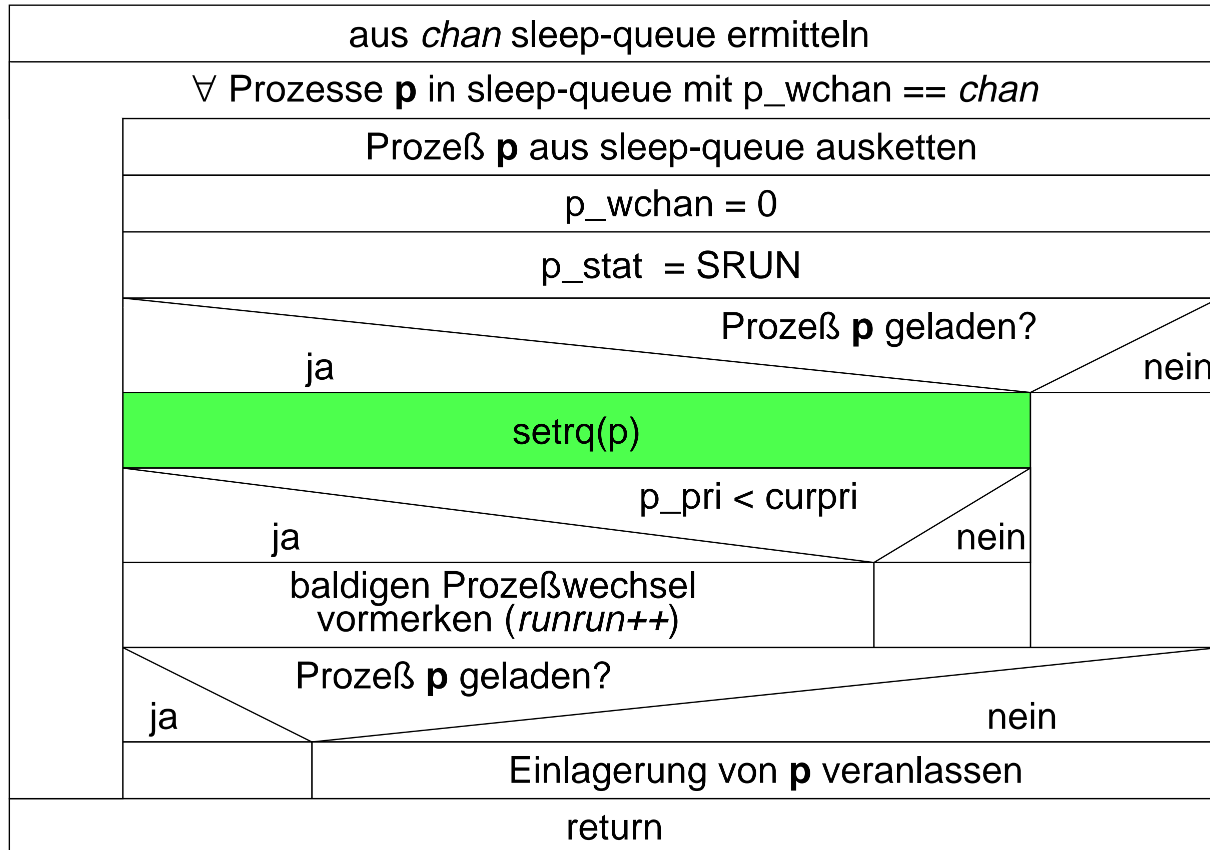
sleep(chan, pri)



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

wakeup(chan)



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.11 UNIX — Signale

- Signale können von einem Prozeß mit ***kill(2)*** oder ***sigsend(2)*** gezielt an andere Prozesse oder Prozeßgruppen zugestellt werden
- Kommunikationsablauf
  - ➥ der Signal-Empfänger hat für die Signale, die zur Kommunikation genutzt werden sollen, eine *signal-handler*-Funktion installiert
  - ➥ bei Eintreffen eines Signals wird die *signal-handler*-Funktion beim Empfänger automatisch angesprungen
  - ◆ mit einem Signal kann nur 1 Bit Information übertragen werden
- Systemaufrufe:

sigaction(2)	Signalbearbeitung steuern
sigprocmask(2)	Signalmaske installieren

- Prozeß schlafen legen:
  - ◆ ***pause(2)*** blockiert Prozeß bis Signal eintrifft
  - ◆ ***sigsuspend(2)*** installiert erst eine Signalmaske und blockiert Prozeß dann (zum Warten auf bestimmte Signale)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Ablauf der Signalverarbeitung im Systemkern

(1) Signal zustellen

- das Signal wird durch Setzen eines Bits in der Komponente `p_sig` der *proc*-Struktur des Empfängerprozesses zugestellt
- falls der Prozeß blockiert ist (→ *sleep*), wird er in den Zustand "bereit" überführt (*setrq( )*)
- die Signalbearbeitung wird dem Empfängerprozeß selbst überlassen
- nur die Stop-Signale bewirken einen direkten Eingriff in den Ablauf des Empfängerprozesses:
  - ➥ der Prozeß wird durch die Signalzustellung blockiert

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Ablauf der Signalverarbeitung im Systemkern (2)

(2) Signal verarbeiten

- jeder Prozeß prüft vor Verlassen des Systemkerns (am Ende eines Systemaufrufs) und vor sowie nach einem Blockieren (*sleep( )*), ob Signale für ihn vorliegen
- blockierte Signale werden nicht bearbeitet, bleiben aber in der Signal-Bitmaske `p_sig` vorgemerkt
- für alle anderen Signale wird der Reihe nach die Signalbearbeitung veranlaßt
  - ◆ *core-dump* erzeugen (wenn nötig) und *exit( )* aufrufen, oder
  - ◆ Signalbearbeitungsfunktion aufsetzen
  - ◆ während des Ablaufs einer Signalbearbeitungsfunktion können andere Signale eintreffen, die (sofern nicht blockiert) die Bearbeitung des ersten Signals sofort unterbrechen

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 SystemV ↔ BSD/POSIX-Semantik

SystemV-Probleme

- unsicher
  da die Signal-Reaktion vor Aufruf der Bearbeitungsfunktion zurückgesetzt wird, kann man zwei schnell aufeinanderfolgend eintreffende Signale nicht sicher bearbeiten
- man kann Signale nur ignorieren, nicht aber verzögern

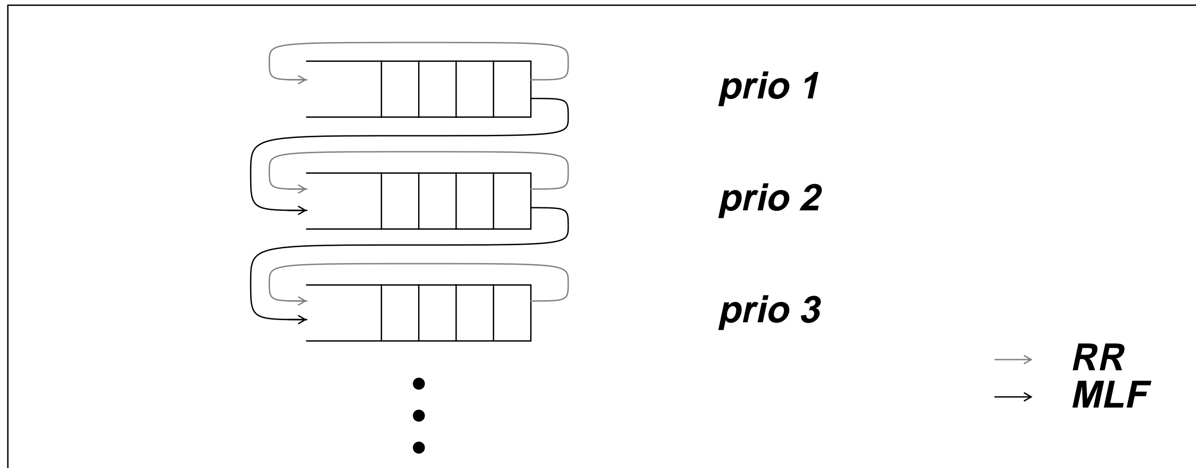
Vorteil der BSD/POSIX-Semantik

- entspricht vollständig der Bearbeitung von Hardware-Interrupts
  - während ein Signal bearbeitet wird, werden weitere blockiert
  - es kann maximal nur ein weiteres Signal gespeichert werden
  - zur Signalbearbeitung kann ein spezieller Stack verwendet werden (***sigaltstack(2)*** oder ***sigstack(2)***)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.12 UNIX — Scheduling

- der Scheduling-Algorithmus legt fest, welcher der lauffähigen Prozesse den Prozessor wie lange maximal zugeteilt bekommt
- Strategie: ***multilevel feedback with round robin***



➥ entscheidende Parameter

- ➤ Länge der Zeitscheibe für *Round Robin*
- ➤ Algorithmus zur Berechnung der Priorität

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.12 UNIX — Scheduling

- grundsätzliche Unterschiede zwischen UNIX SystemV und BSD UNIX

Wertung: Berkeley-Strategie besser für interaktives Arbeiten + bessere Einflußnahme auf einzelne Prozesse möglich

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Scheduling in UNIX System V

- **Zeitscheibe** für Prozesse, die Prozessor nicht vorher aufgeben: **1 s**
- Prioritätsberechnung

  Priorität ***p_pri*** (Element der proc-Struktur) wird ermittelt aus:

  p_cpu Maß für die CPU-Nutzung des Prozesses

  p_nice vom Benutzer einstellbarer Wert zwischen -20 und +20 (eigentlich zwischen 0 und 40, wovon dann NZERO=20 abgezogen wird)

$$\boldsymbol{p_pri} = PUSER + \frac{p_cpu}{2} + p_nice$$

Berechnung einmal pro Sekunde und nach jedem Systemaufruf bei Veränderungen an den ***runqueues*** wird der gerade aktive Prozeß in einen Pseudo-Systemaufruf gezwungen

$$\boldsymbol{p_cpu} = \frac{p_cpu}{2}$$ einmal pro Sekunde ∀ Prozesse

$$\boldsymbol{p_cpu} = p_cpu + 1$$ alle 10 ms für den aktiven Prozeß

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Scheduling in UNIX System V (2)

- generelle Auswirkung des Algorithmus:
  - rechenintensive Prozesse neigen zu niedriger Priorität (großes *p_pri*), stark interaktive Prozesse (die häufig blockiert werden) zu höherer Priorität (kleines *p_pri*)
- Probleme bei hoher Systemlast
  - es existieren viele Prozesse, wobei jeder Prozeß nur relativ selten den Prozessor zugeteilt bekommt
    - → *p_cpu* der rechenintensiven Prozesse ist im Verhältnis zum wenig belasteten System niedrig
    - → geringer Prioritätsunterschied zwischen stark interaktiven und rechenintensiven Prozessen
    - → Antwortverhalten schlecht, da die rechenintensiven Prozesse ziemlich gleichberechtigt werden

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Scheduling in BSD UNIX

- **Zeitscheibe** für Prozesse, die Prozessor nicht vorher aufgeben: **100 ms**
- Prioritätsberechnung

  Priorität ***p_usrpri*** (Element der ***proc*-Struktur**) wird ermittelt aus:

  p_cpu Maß für die CPU-Nutzung des Prozesses

  p_nice vom Benutzer einstellbarer Wert zwischen -20 und +20

  load mittlere Anzahl lauffähiger Prozesse in der letzten Minute

$$\mathbf{p_usrpri} = PUSER + \frac{p_cpu}{4} + 2 \cdot p_nice$$

wird alle 40 ms für den gerade aktiven Prozeß ermittelt

$$\mathbf{p_cpu} = \frac{2 \cdot load}{2 \cdot load + 1} \cdot p_cpu + p_nice$$

einmal pro Sekunde ∀ Prozesse

$\mathbf{p_cpu} = p_cpu + 1$ alle 10 ms für den aktiven Prozeß

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
F-Prozesse.doc 1999-02-24 18.31
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Scheduling in BSD UNIX (2)

- Unterschiede zu SystemV
  - durch kürzere Zeitscheibe wird die CPU hochprioren Prozessen gleichmäßiger zugeteilt
  - Priorität wird regelmäßig und nicht abhängig von Systemaufrufen neu berechnet
  - *p_nice* hat stärkeren Einfluß auf *p_usrpri*
  - *p_nice* beeinflußt Veränderung von *p_cpu*
  - *load* beeinflußt Veränderung von *p_cpu*
    $\rightarrow$ bei hoher Last erholt sich die Priorität langsamer
- besseres interaktives Verhalten, weil
  - kürzere Zeitscheibe
  - rechenintensive Prozesse erhalten bei hoher Last und interaktiven Anforderungen kaum die aktuell beste Priorität

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Scheduling in BSD UNIX (3)

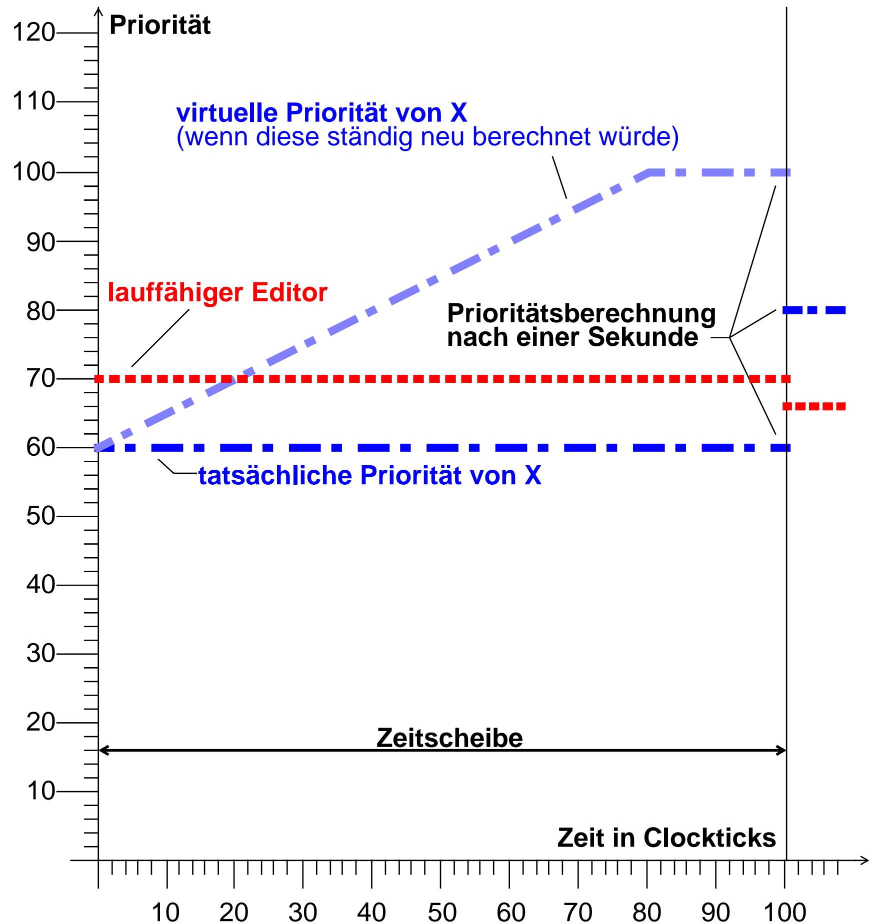
- Prioritätsverhalten mit ***nice(2)*** leichter zu beeinflussen
  - ◆ *p_nice* wirkt sich erheblich stärker aus als in SystemV
- Verwaltungaufwand erheblich größer (z. B. häufigere Prozeßwechsel)
- Rechner, deren CPU durch den interaktiven Betrieb bereits voll ausgelastet ist, bearbeiten andere Aufgaben praktisch nicht mehr

  → Berkeley-Scheduling nur für leistungsfähige Anlagen geeignet

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Vergleich: SystemV ↔ BSD

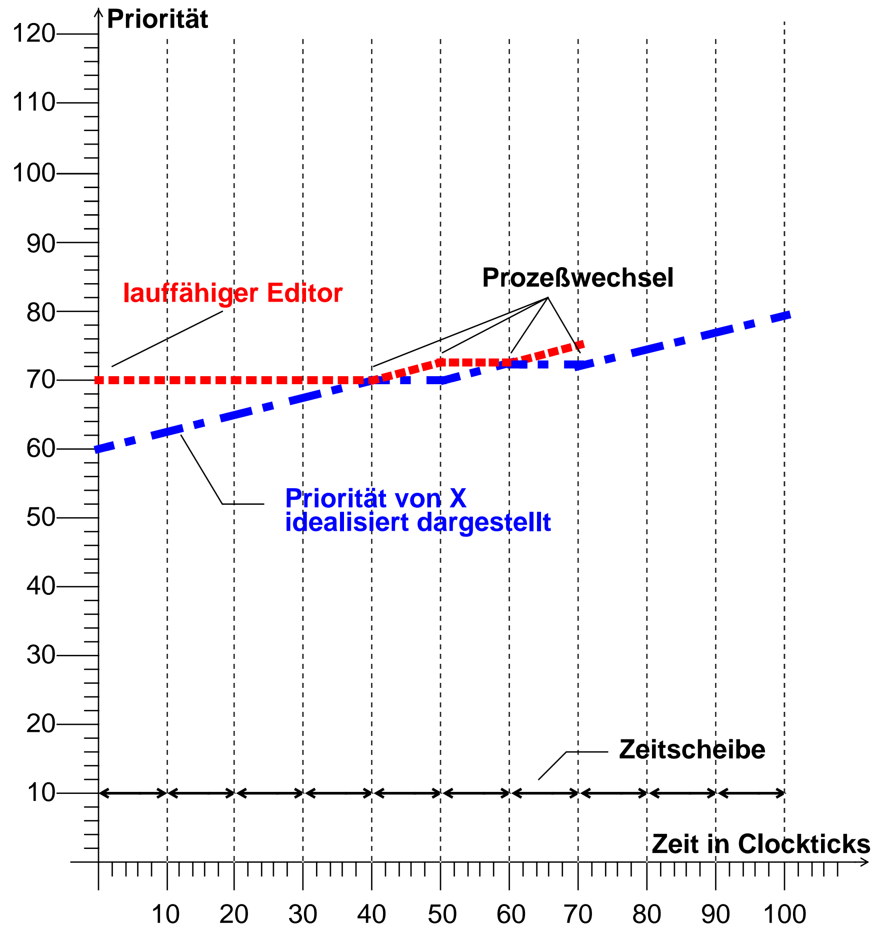
- Prioritätsverhalten und Prozeßwechsel in SystemV
  - Rechenprozeß X, interaktiver Prozeß (z. B. Editor),
  - anfangs $p_cpu = 0$
  - ohne Aufruf von Systemdiensten (dadurch wird Priorität nicht neu berechnet), keine Interrupts



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Vergleich: SystemV ↔ BSD (2)

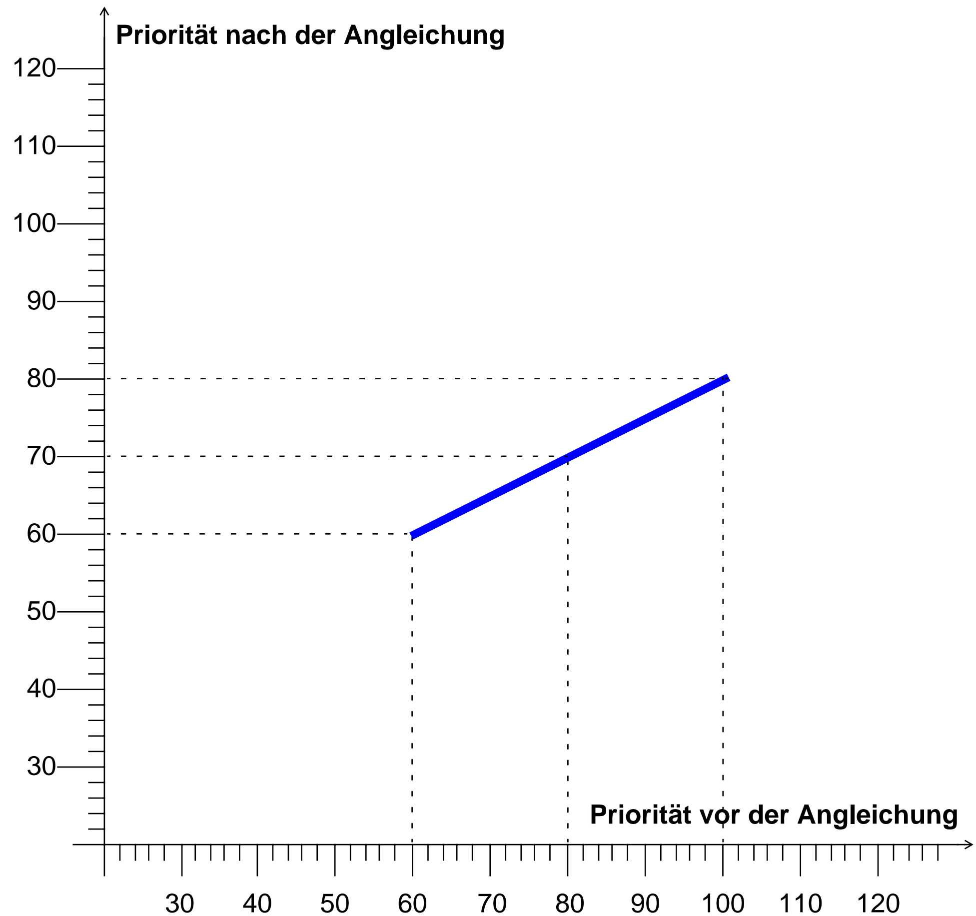
- Prioritätsverhalten und Prozeßwechsel in Berkeley-UNIX
  - Rechenprozeß X, interaktiver Prozeß (z. B. Editor),
  - gleiche Bedingungen wie Beispiel für SystemV



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Vergleich: SystemV $\leftrightarrow$ BSD (3)

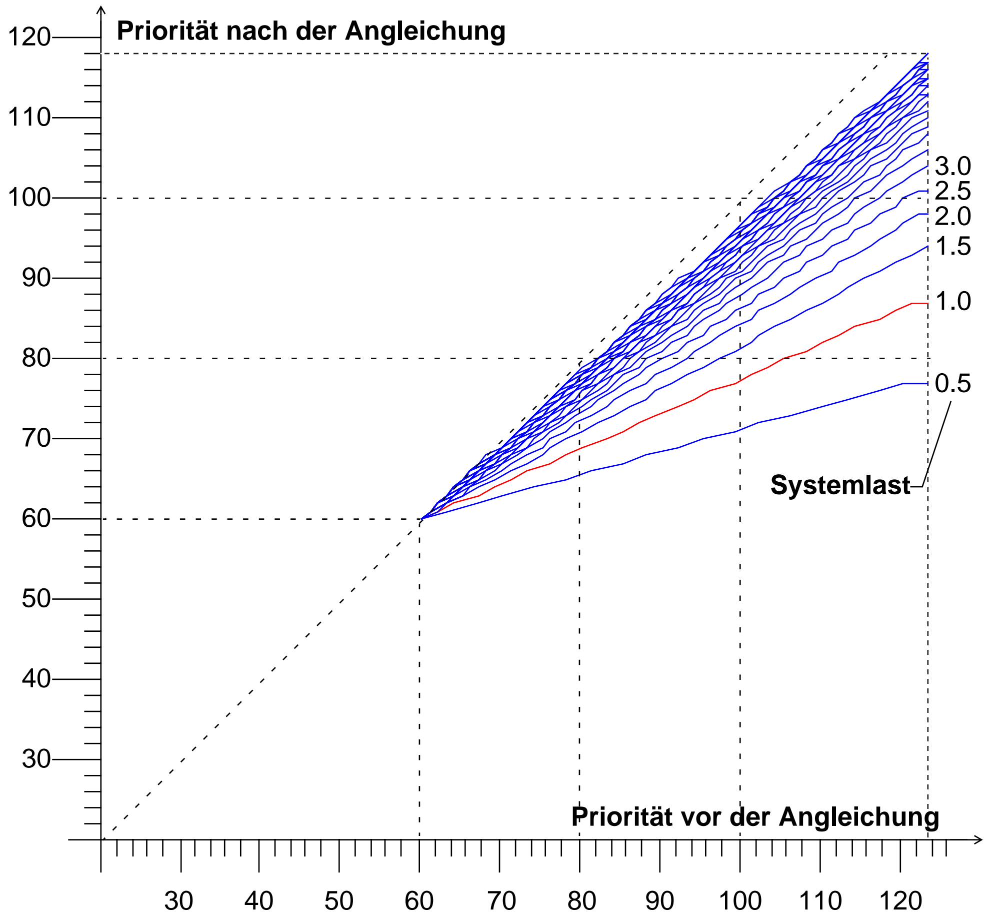
- Verbesserung der Prioritäten durch Zerfallsfunktion für *p_cpu*
  - SystemV



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Vergleich: SystemV ↔ BSD (4)

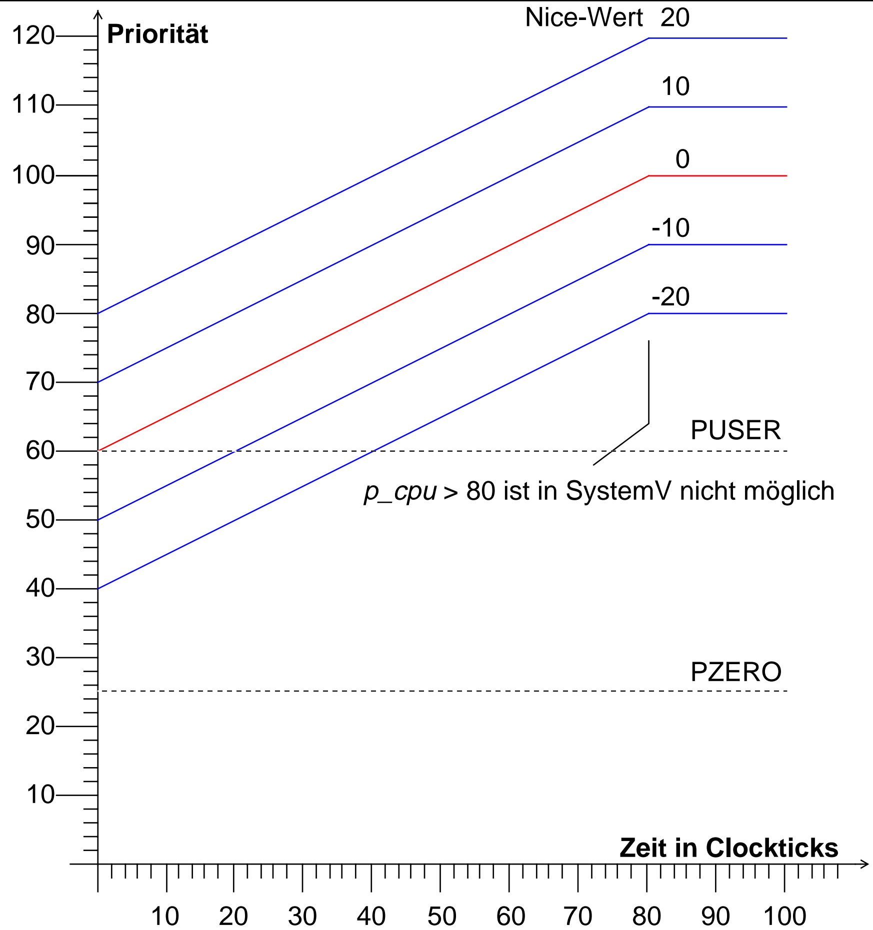
- Verbesserung der Prioritäten durch Zerfallsfunktion für *p_cpu*
  - Berkeley-UNIX - abhängig von der aktuellen Systemlast



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Vergleich: SystemV ↔ BSD (5)

- Einfluß des *nice*-Wertes auf die Priorität eines Prozesses
  - SystemV



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

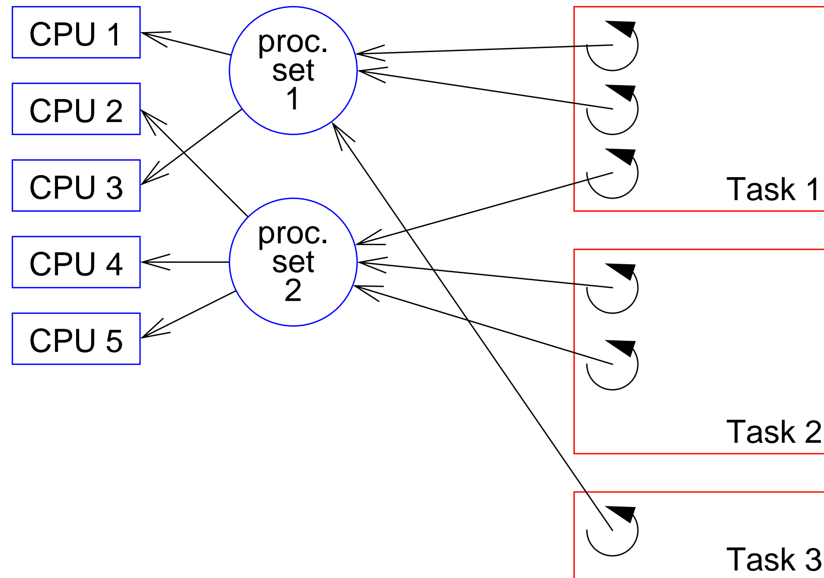
3 Vergleich: SystemV ↔ BSD (6)

- Einfluß des *nice*-Wertes auf die Priorität eines Prozesses
  - ◆ Berkeley-UNIX

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Architektur

- Scheduling auf Thread-Basis — unabhängig von Tasks
- Prozessoren sind zu ***processor sets*** zusammengefaßt
  - jeder Prozessor ist genau einem *processor set* zugeordnet
  - Threads können dynamisch verschiedenen *processor sets* zugeordnet werden
  - eigenes Scheduling für jedes *processor set*



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Schedulingstrategie & Prioritäten

- Multi-level feedback Strategie mit 32 globalen Warteschlangen pro *processor set* für bereite Prozesse
- Pro Prozessor eine lokale Warteschlange für prozessor-gebundene Threads
  - hat immer Vorrang vor den globalen Warteschlangen
  - z. B. für Threads, die E/A-Funktionen für dem Prozessor zugeordnete Geräte abwickeln

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Schedulingstrategie & Prioritäten (2)

- Dynamische Prioritätsberechnung ähnlich wie bei UNIX
  - ◆ Priorität aktiver Threads sinkt
  - ◆ Priorität passiver Threads wird abhängig von der Wartezeit verbessert

current priority	aktuelle Priorität, wird vom Kern auf der Basis der CPU-Nutzung des Threads, der aktuellen Systemlast und der *base priortity* berechnet
min. base priority	untere Grenze für *base priority*
base priority	vom Thread selbst festlegbare Basis für die Prioritätsberechnung

- Länge von Zeitscheiben abhängig von der Systemlast
  – kurze Zeitscheiben bei hoher Last (→ bessere Antwortzeiten)
  – lange Zeitscheiben bei geringer Last (→ weniger Overhead)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Scheduling-Unterstützung für nebenläufige Anwendungen

- MACH bietet Mechanismen um im Rahmen einer Anwendung das Scheduling von Threads der Anwendung zu beeinflussen

★ **Discouragement**

➥ Hinweis an das System, daß der aktuelle Thread nicht laufen soll (weil er z. B. aufgrund des Anwendungszustands ohnehin keinen Fortschritt machen kann)

- drei Stufen:

mild	Prozessor wenn möglich an anderen Thread abgeben
strong	Prozessor aufgeben und vorübergehend Priorität senken
absolute	Thread für bestimmten Zeitraum blockieren

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
F-Prozesse.doc 1999-02-24 18.31
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Scheduling-Unterstützung für nebenläufige Anwendungen (2)

★ Handoff-Scheduling

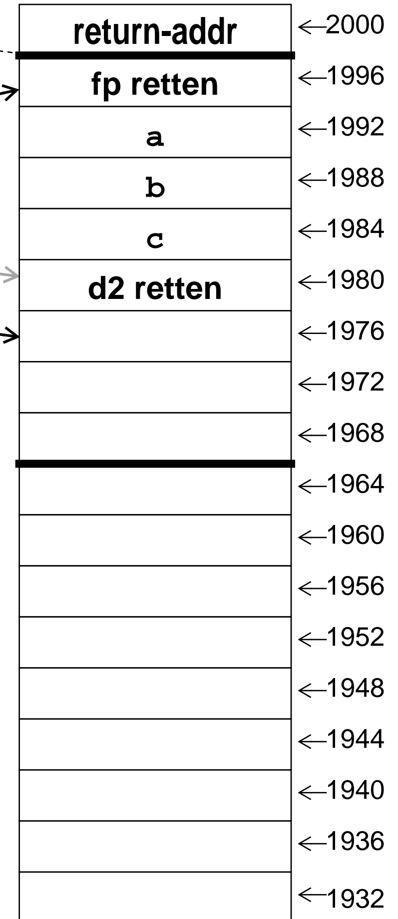
- ➥ Ein Thread gibt den Prozessor auf und bestimmt dabei seinen Nachfolger
  - ➤ Scheduling-Mechanismen des Kerns werden dabei umgangen
  - ➤ Anwendung z. B. wenn ein Thread auf einen leeren Puffer trifft und den Thread, der Daten nachfüllen kann kennt
  - ➤ Lösung des *priority inversion* Problems:
    ein hochpriorer Thread blockiert sich an einer Sperre, die von einem Thread mit niedriger Priorität gehalten wird

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 ■ Stack mehrerer Funktionsaufrufe

```
main() {
        int a, b, c;
        register r;

        a = 10;
        b = 20;
        r = 40;

        f1(a, b);

        return(a);
}
```

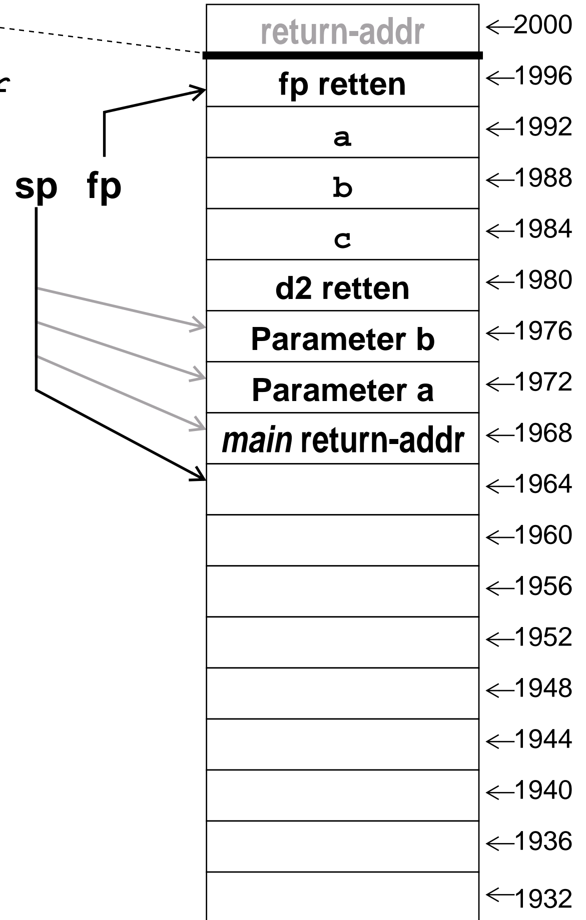
Reg. d2: r



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Stack mehrerer Funktionsaufrufe

```
main() {
        int a, b, c;
        register r;

        a = 10;
        b = 20;
        r = 40;

        f1(a, b);

        return(a);
}
```

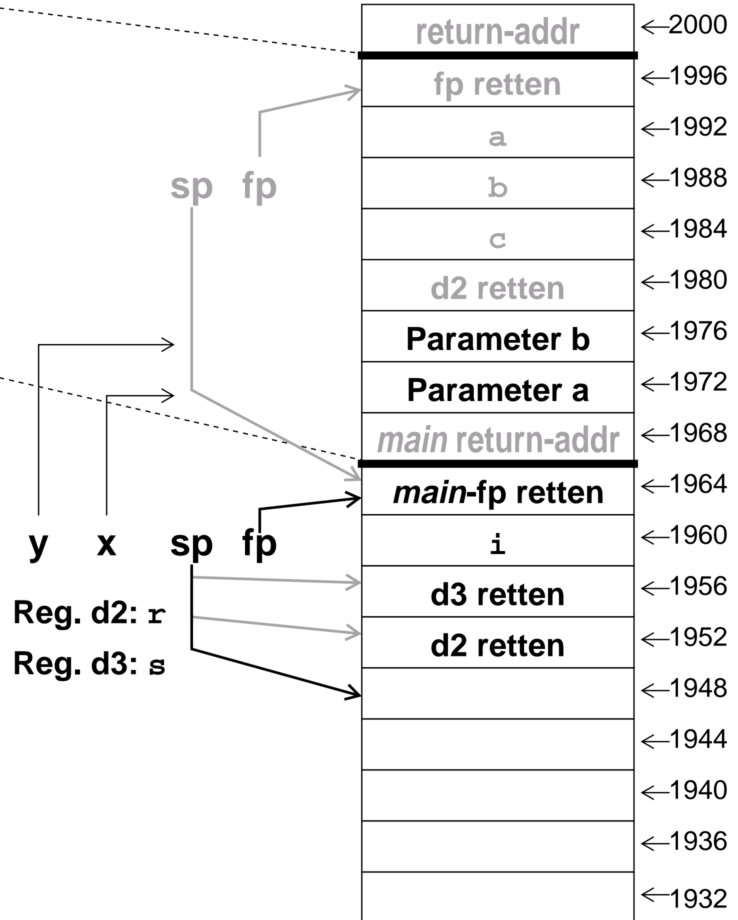


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Stack mehrerer Funktionsaufrufe

```
main() {
        int a, b, c;
        register r;

        a = 10;
        b = 20;
        r = 40;

        f1(a, b);

        return(a);
}
f1(x,y) {
        int i;
        register r;
        register s;

        x++;

        i = f2(x);

        return(i);
}
```

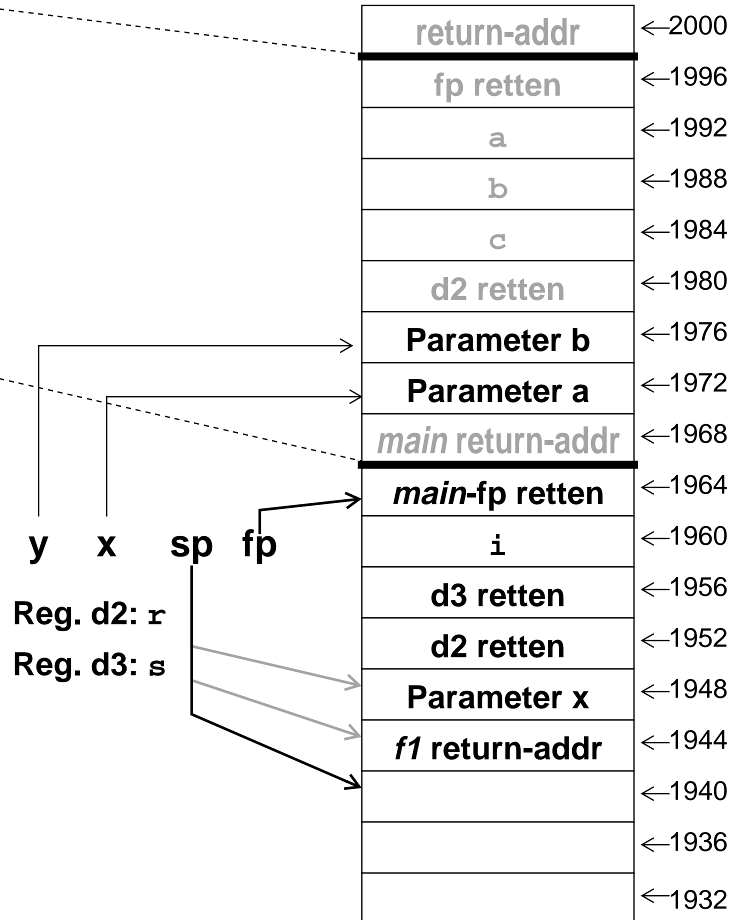


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Stack mehrerer Funktionsaufrufe

```
main() {
        int a, b, c;
        register r;

        a = 10;
        b = 20;
        r = 40;

        f1(a, b);

        return(a);
}
```

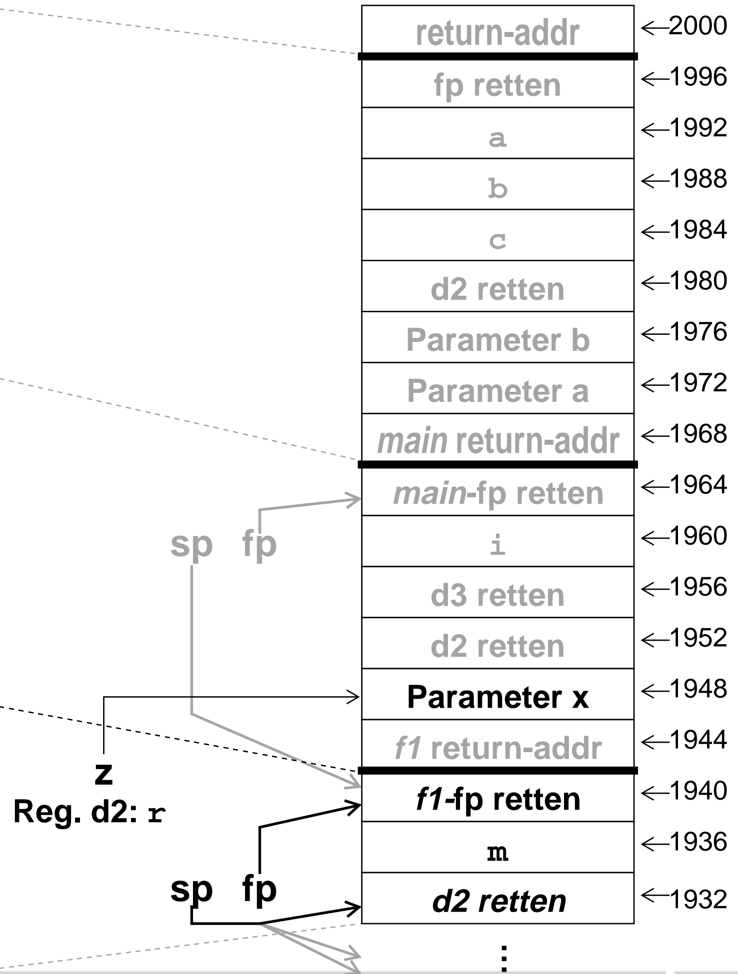
```
f1(x,y) {
        int i;
        register r;
        register s;

        x++;

        i = f2(x);

        return(i);
}
```



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Stack mehrerer Funktionsaufrufe

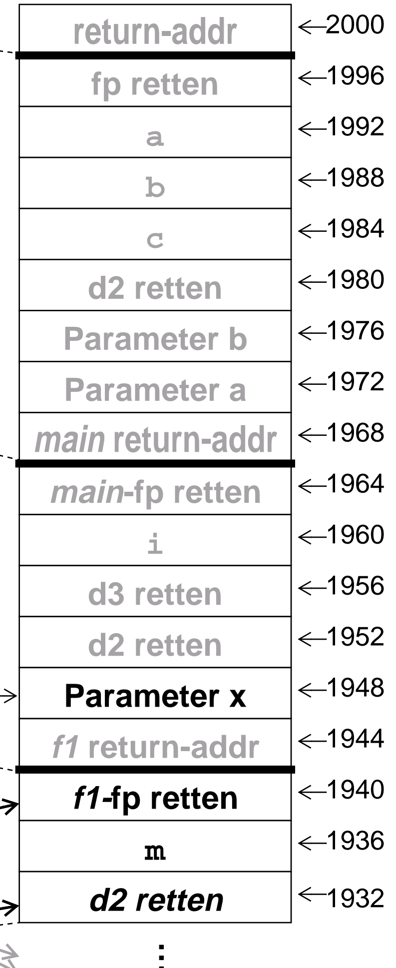
```
main() {
        int a, b, c;
        register r;

        a = 10;
        b = 20;
        r = 40;

        f1(a, b);

        return(a);
}
f1(x,y) {
        int i;
        register r;
        register s;

        x++;

        i = f2(x);

        return(i);
}
f2(z) {
        int m;
        register r;

        r = z;
        m = 100;

        return(z+1);
}
```



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Stack mehrerer Funktionsaufrufe

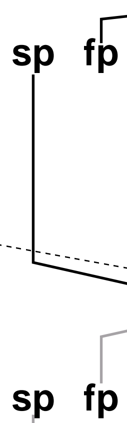
```
main() {
        int a, b, c;
        register r;

        a = 10;
        b = 20;
        r = 40;

        f1(a, b);

        return(a);
}
f1(x,y) {
        int i;
        register r;
        register s;

        x++;

        i = f2(x);

        return(i);
}
f2(z) {
        int m;
        register r;

        r = z;
        m = 100;

        return(z+1);
}
```

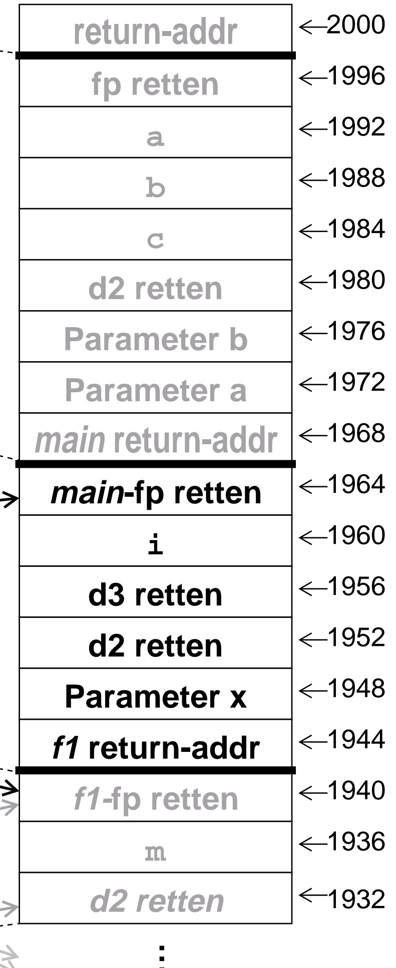


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Stack mehrerer Funktionsaufrufe

```
main() {
        int a, b, c;
        register r;

        a = 10;
        b = 20;
        r = 40;

        f1(a, b);

        return(a);
}
f1(x,y) {
        int i;
        register r;
        register s;

        x++;

        i = f2(x);

        return(i);
}
f2(z) {
        int m;
        register r;

        r = z;
        m = 100;

        return(z+1);
}
```

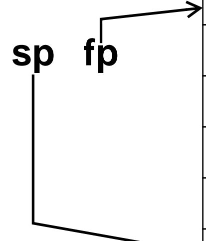


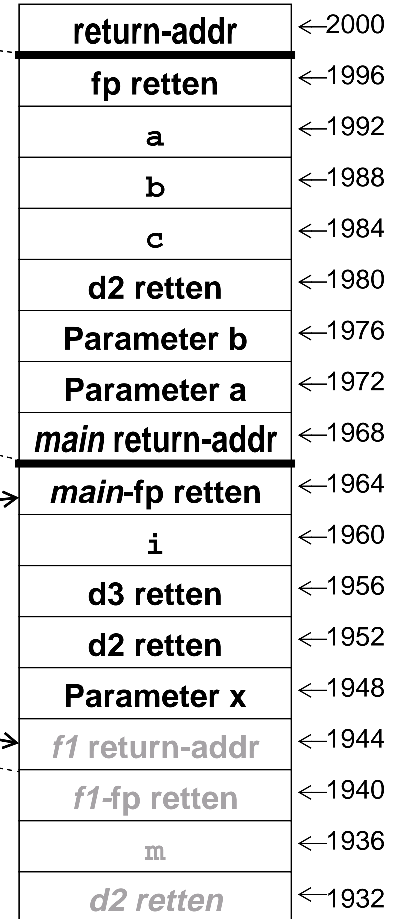


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Stack mehrerer Funktionsaufrufe

```
main() {
        int a, b, c;
        register r;

        a = 10;
        b = 20;
        r = 40;

        f1(a, b);

        return(a);
}
f1(x,y) {
        int i;
        register r;
        register s;

        x++;

        i = f2(x);

        return(i);
}
```



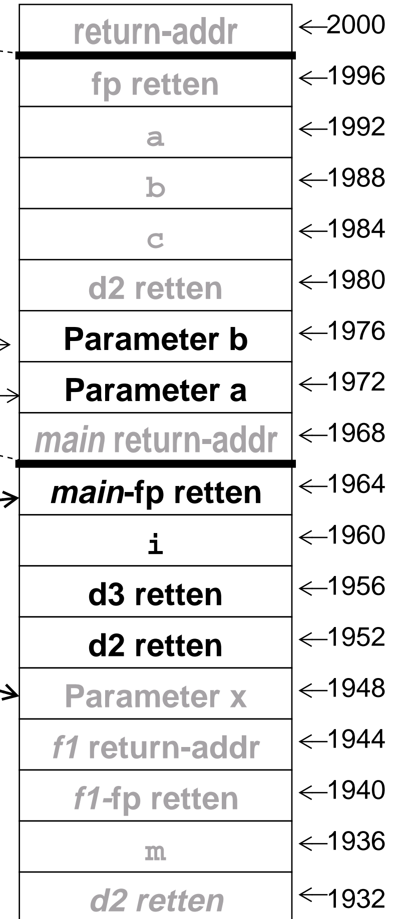


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Stack mehrerer Funktionsaufrufe

```
main() {
        int a, b, c;
        register r;

        a = 10;
        b = 20;
        r = 40;

        f1(a, b);

        return(a);
}
f1(x,y) {
        int i;
        register r;
        register s;

        x++;

        i = f2(x);

        return(i);
}
```

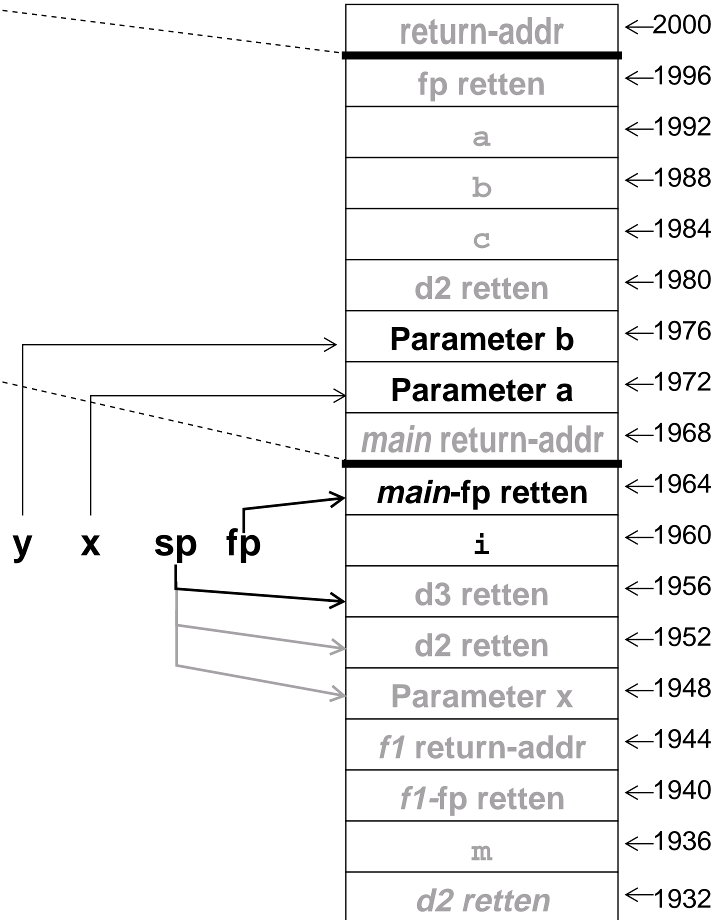


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Stack mehrerer Funktionsaufrufe

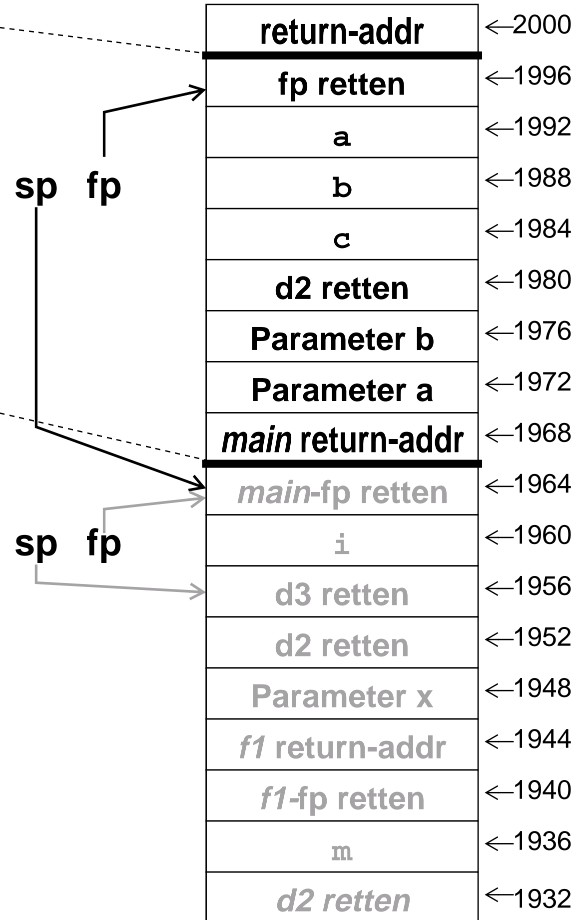
```
main() {
        int a, b, c;
        register r;

        a = 10;
        b = 20;
        r = 40;

        f1(a, b);

        return(a);
}
f1(x,y) {
        int i;
        register r;
        register s;

        x++;

        i = f2(x);

        return(i);
}
```



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Stack mehrerer Funktionsaufrufe

```
main() {
        int a, b, c;
        register r;

        a = 10;
        b = 20;
        r = 40;

        f1(a, b);

        return(a);
}
f1(x,y) {
        int i;
        register r;
        register s;

        x++;

        i = f2(x);

        return(i);
}
```

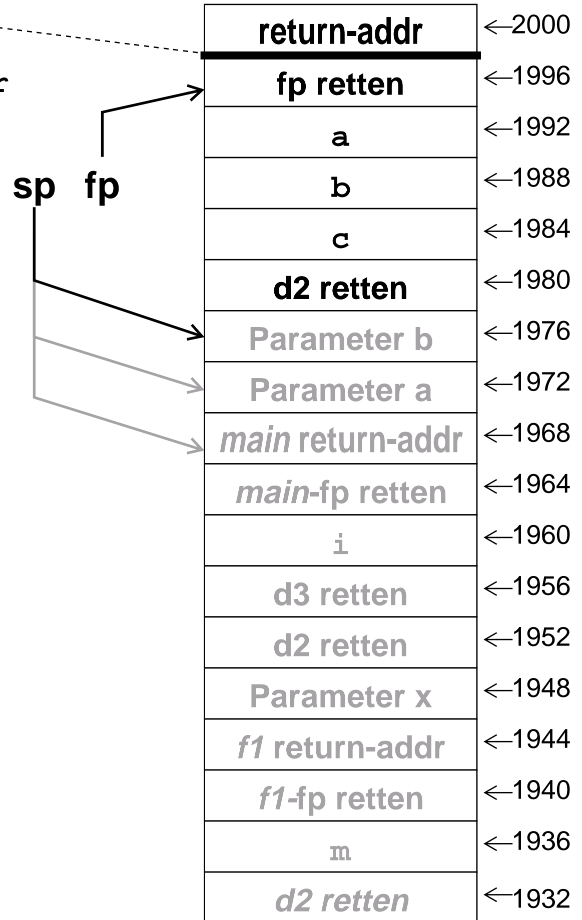


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Stack mehrerer Funktionsaufrufe

```
main() {
        int a, b, c;
        register r;

        a = 10;
        b = 20;
        r = 40;

        f1(a, b);

        return(a);
}
```

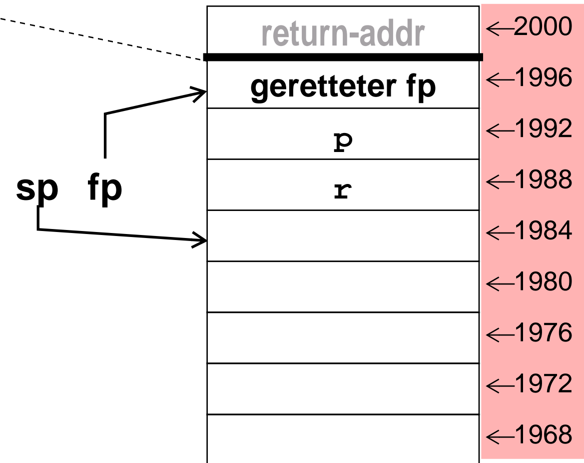
Reg. d2: r

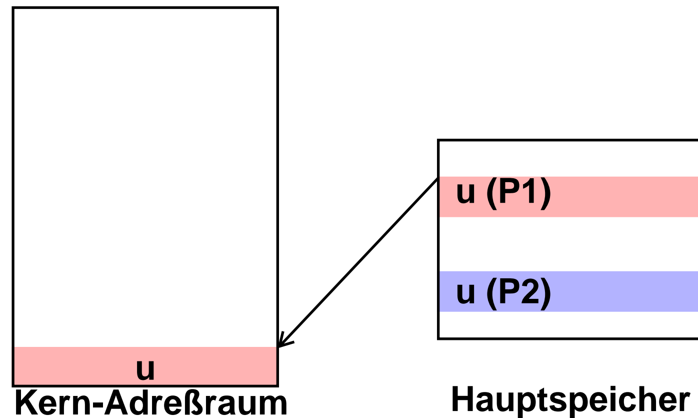


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Ablauf (3)

```
swtch() {
     struct proc *p;
     int r;
     p = getrq();

     r = save(u.u_ssave);

     if (r == 0) {
        resume(p);
     } else {
        return(1);
     }
}
```

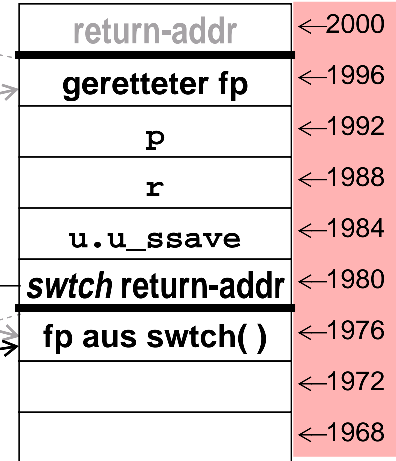


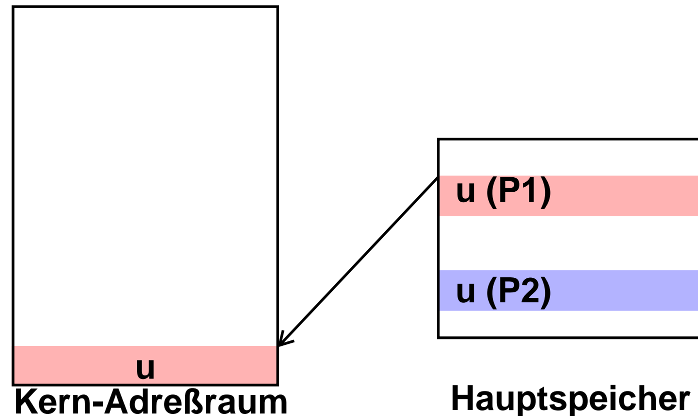


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Ablauf (3)

```
swtch() {
     struct proc *p;
     int r;
     p = getrq();

     r = save(u.u_ssave);

     if (r == 0) {
        resume(p);
     } else {
        return(1);
     }
}
save(savearea *s) {
     /* Register,
        sp, fp,
        fp aus swtch und
        swtch return-addr
        -> s sichern */

     return(0);
}
```

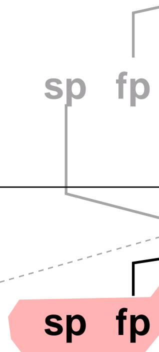


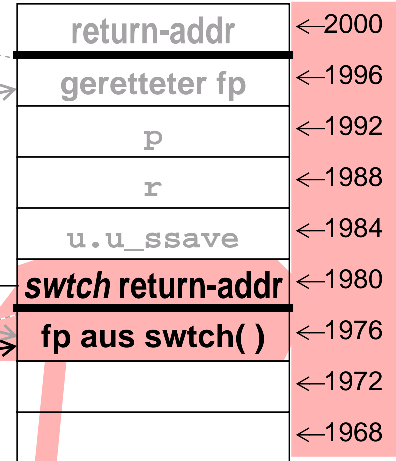


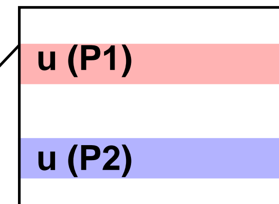
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Ablauf (3)

```
swtch() {
     struct proc *p;
     int r;
     p = getrq();

     r = save(u.u_ssave);

     if (r == 0) {
        resume(p);
     } else {
        return(1);
     }
}
save(savearea *s) {
     /* Register,
        sp, fp,
        fp aus swtch und
        swtch return-addr
        -> s sichern */

     return(0);
}
```







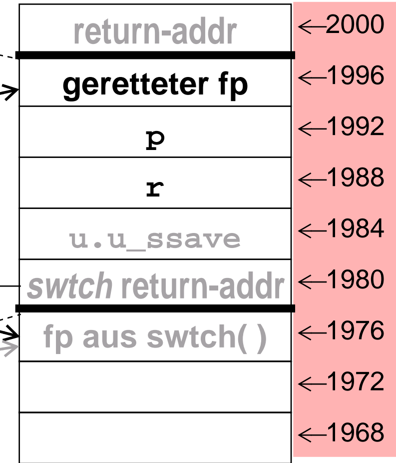
Hauptspeicher

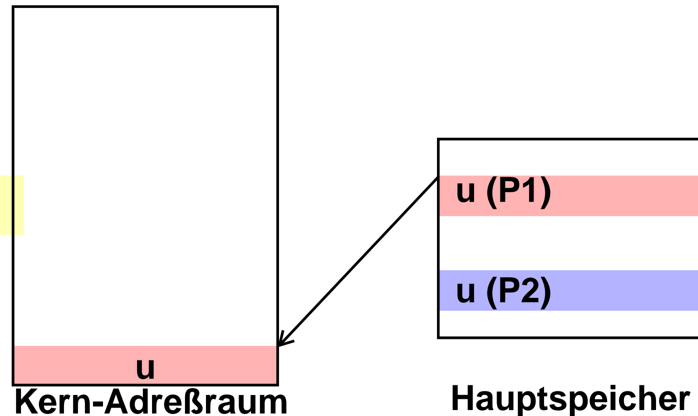
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Ablauf (3)

```
swtch() {
     struct proc *p;
     int r;
     p = getrq();

     r = save(u.u_ssave);

     if (r == 0) {
         resume(p);
     } else {
         return(1);
     }
}
save(savearea *s) {
     /* Register,
         sp, fp,
         fp aus swtch und
         swtch return-addr
         -> s sichern */

     return(0);
}
```

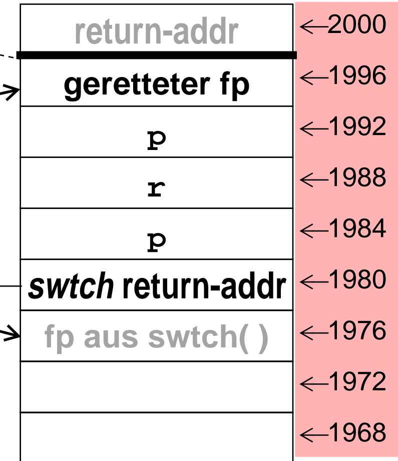


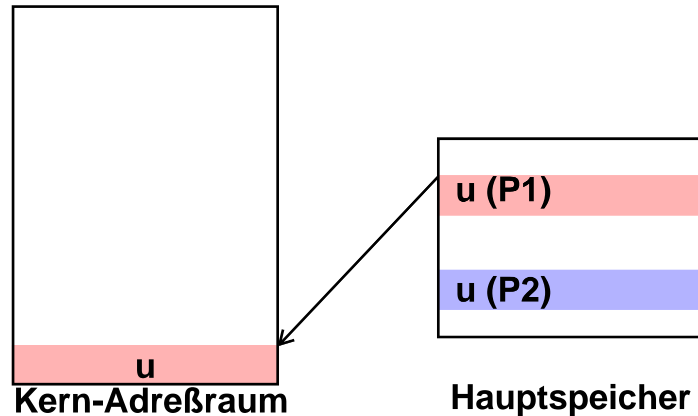


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Ablauf (3)

```
swtch() {
     struct proc *p;
     int r;
     p = getrq();

     r = save(u.u_ssave);

     if (r == 0) {
         resume(p);
     } else {
         return(1);
     }
}
```

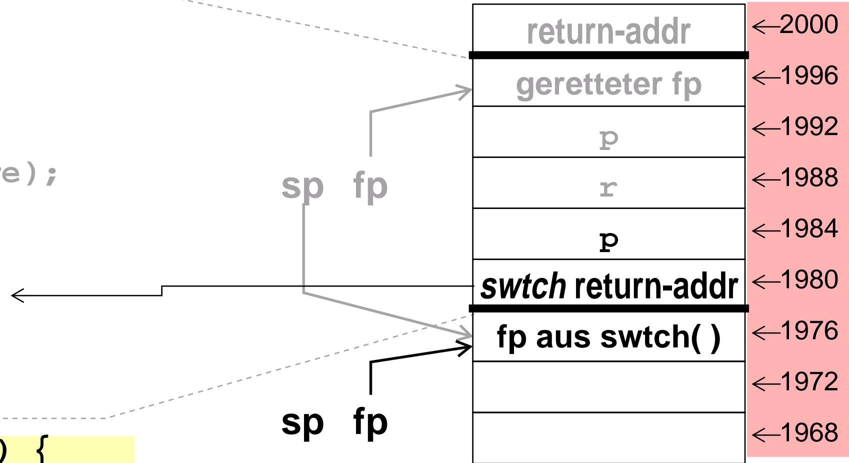


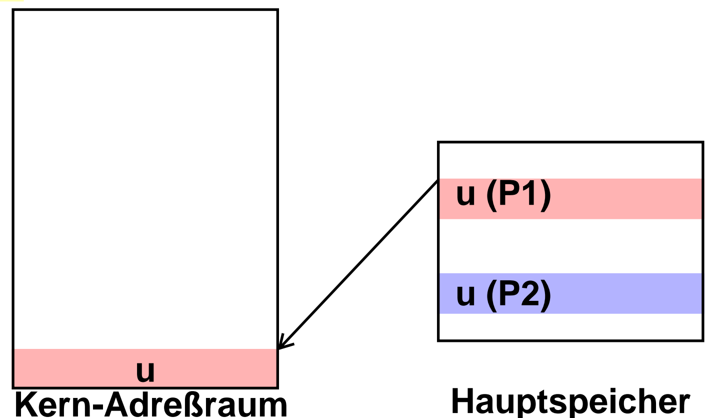


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Ablauf (4)

```
swtch() {
    struct proc *p;
    int r;
    p = getrq();

    r = save(u.u_ssave);

    if (r == 0) {
        resume(p);
    } else {
        return(1);
    }
}
resume(struct proc *p) {
    /* u-area und
       kernel-stack
       von p mappen */

    /* gesicherte Register,
       fp und return-addr
       restaurieren */

    return(1);
}
```



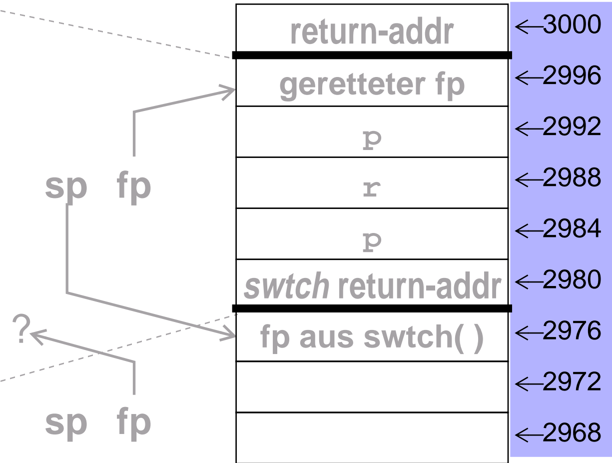


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

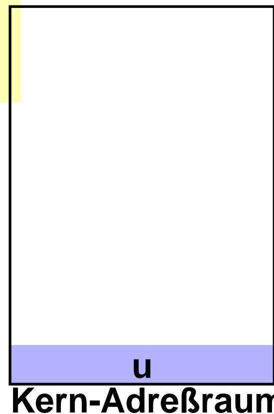
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

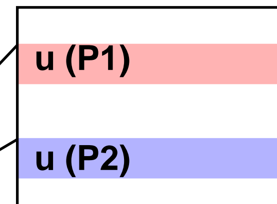
Ablauf (4)

```
swtch() {
     struct proc *p;
     int r;
     p = getrq();

     r = save(u.u_ssave);

     if (r == 0) {
        resume(p);
     } else {
        return(1);
     }
}
```




```
resume(struct proc *p) {
     /* u-area und
❶       kernel-stack
        von p mappen */

     /* gesicherte Register,
        fp und return-addr
        restaurieren */

     return(1);
}
```

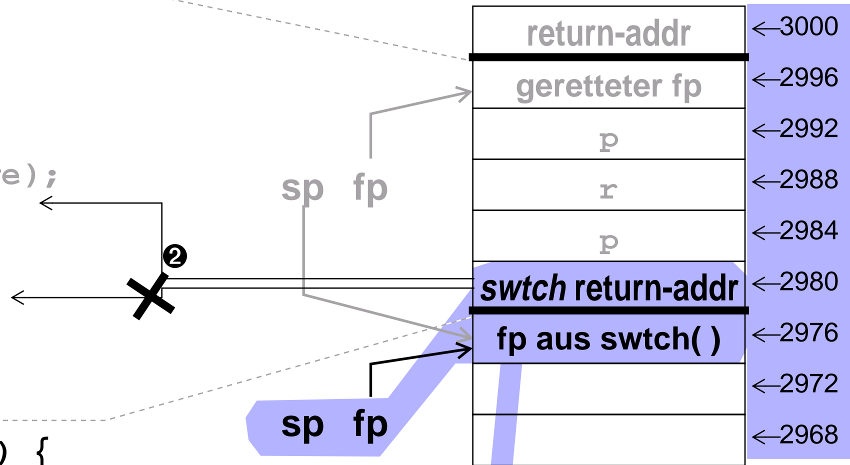


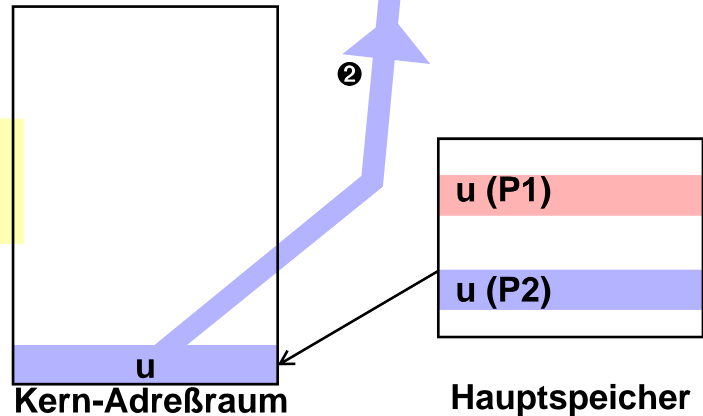


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Ablauf (4)

```
swtch() {
     struct proc *p;
     int r;
     p = getrq();

     r = save(u.u_ssave);

     if (r == 0) {
        resume(p);
     } else {
        return(1);
     }
}
resume(struct proc *p) {
     /* u-area und
        kernel-stack
        von p mappen */

     /* gesicherte Register,
❷       fp und return-addr
        restaurieren */

     return(1);
}
```

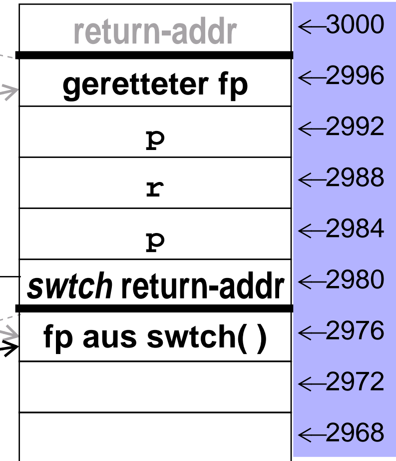


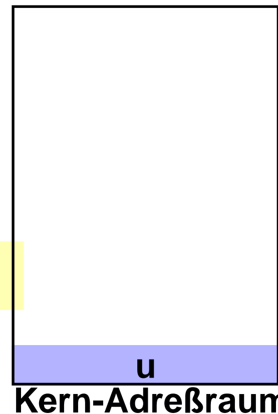


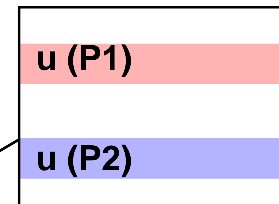
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Ablauf (4)

```
swtch() {
     struct proc *p;
     int r;
     p = getrq();

     r = save(u.u_ssave);

     if (r == 0) {
        resume(p);
     } else {
        return(1);
     }
}
resume(struct proc *p) {
     /* u-area und
        kernel-stack
        von p mappen */

     /* gesicherte Register,
        fp und return-addr
        restaurieren */

     return(1);
}
```





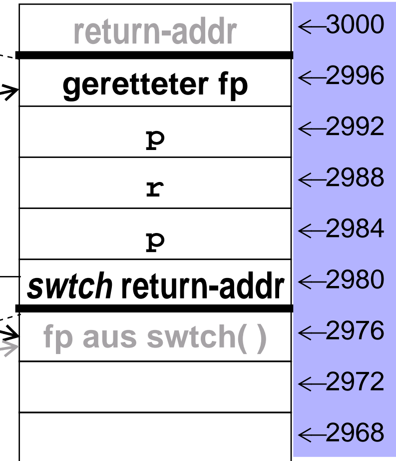


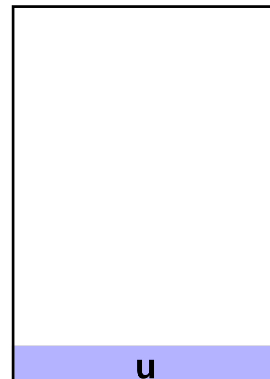
Hauptspeicher

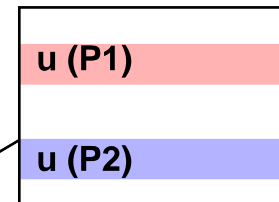
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Ablauf (4)

```
swtch() {
     struct proc *p;
     int r;
     p = getrq();

     r = save(u.u_ssave);

     if (r == 0) {
        resume(p);
     } else {
        return(1);
     }
}
resume(struct proc *p) {
     /* u-area und
        kernel-stack
        von p mappen */

     /* gesicherte Register,
        fp und return-addr
        restaurieren */

     return(1);
}
```





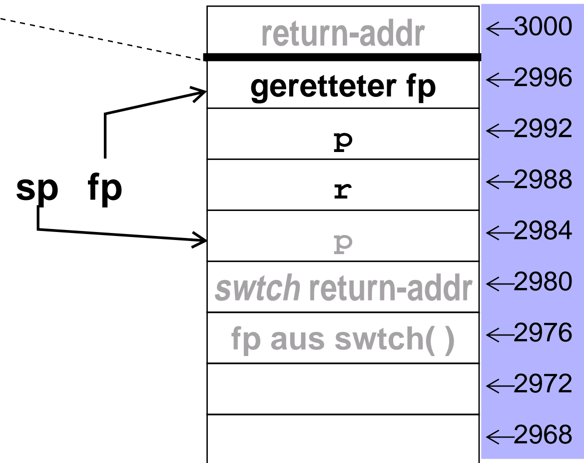


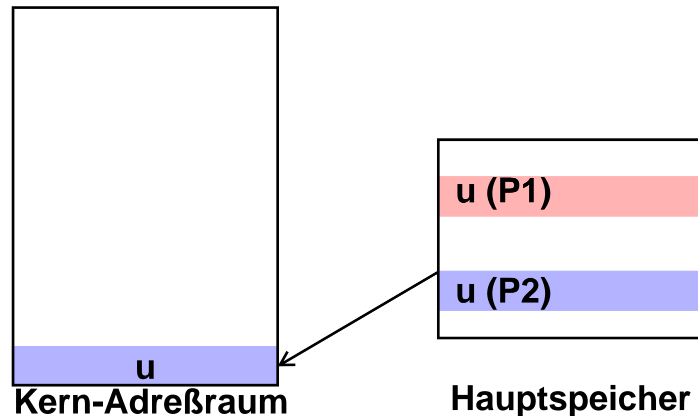
Hauptspeicher

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Ablauf (4)

```
swtch() {
     struct proc *p;
     int r;
     p = getrq();

     r = save(u.u_ssave);

     if (r == 0) {
        resume(p);
     } else {
        return(1);
     }
}
```





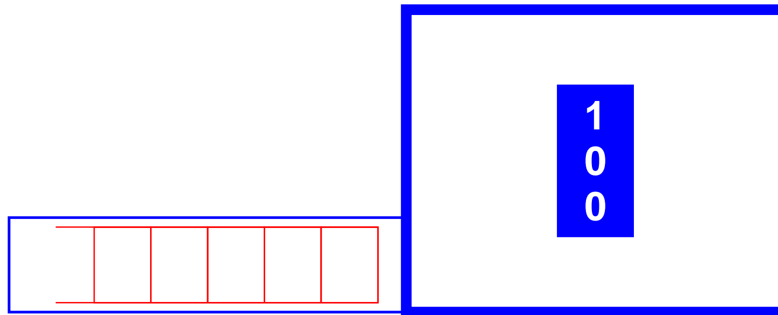
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

UNIX — Semaphore (8)

…

2. Leser-Schreiber-Problem

Leser



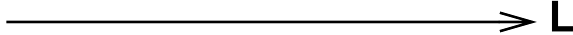
Schreiber

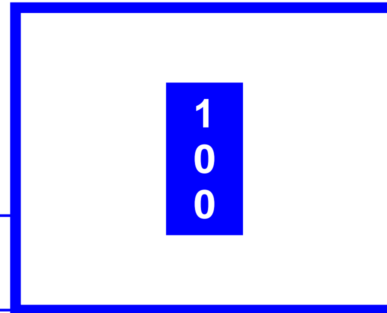
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

...

2. Leser-Schreiber-Problem

Leser



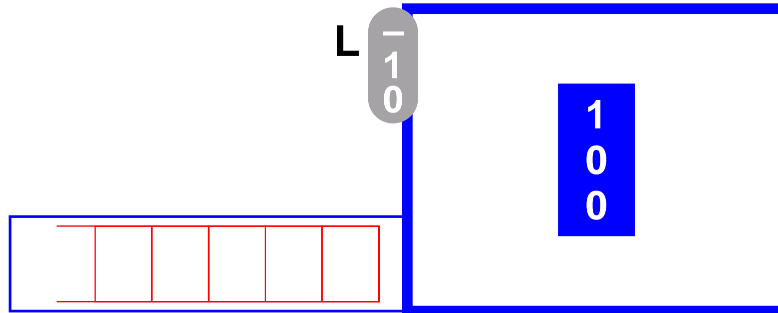


Schreiber

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2. Leser-Schreiber-Problem

Leser



Schreiber

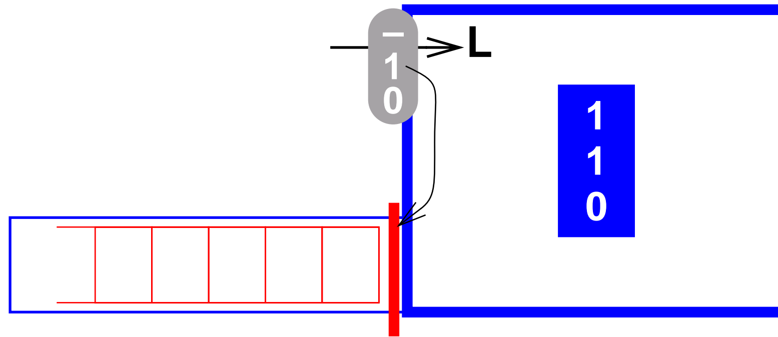
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

… 2. Leser-Schreiber-Problem

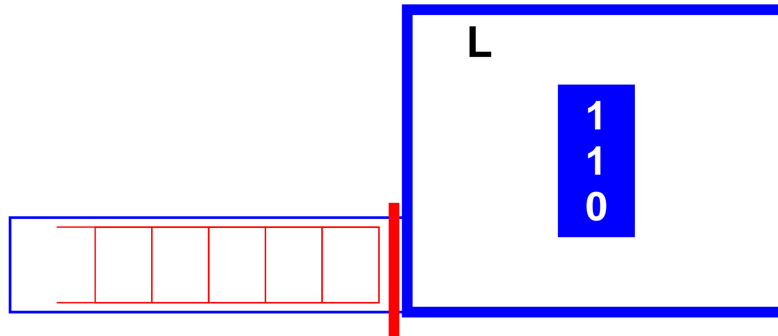
Leser

Schreiber



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

… 2. Leser-Schreiber-Problem



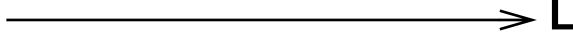
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

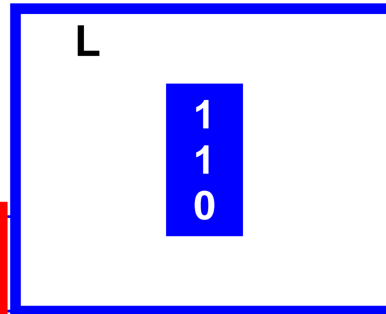
UNIX — Semaphore (8)

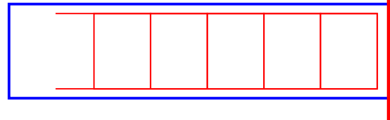
...

2. Leser-Schreiber-Problem

Leser







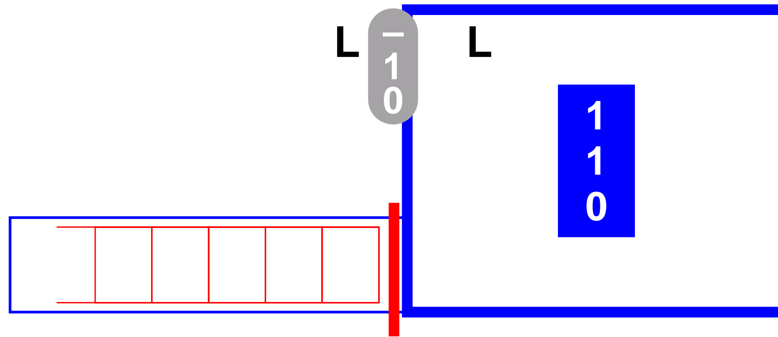
Schreiber

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

... 2. Leser-Schreiber-Problem

Leser

Schreiber

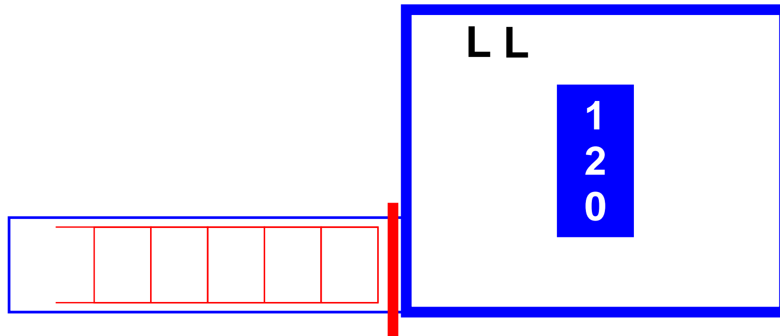


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

... 2. Leser-Schreiber-Problem

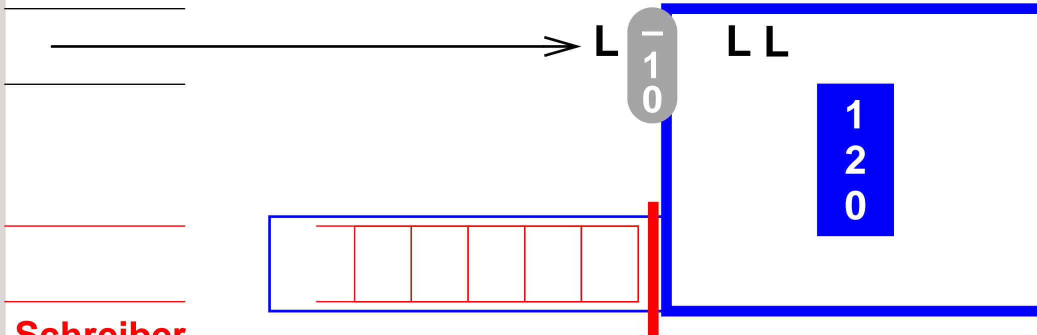
Leser

Schreiber



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2. Leser-Schreiber-Problem



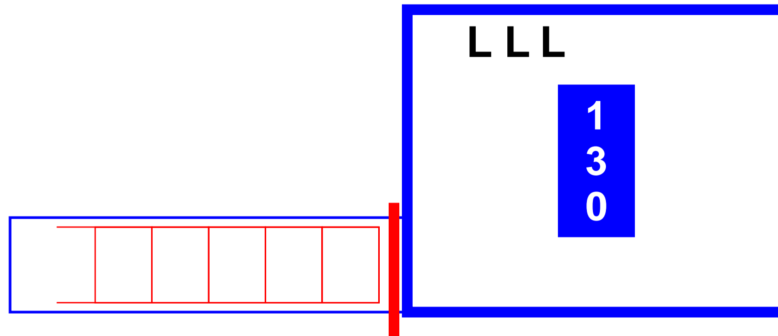
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

... 2. Leser-Schreiber-Problem

Leser

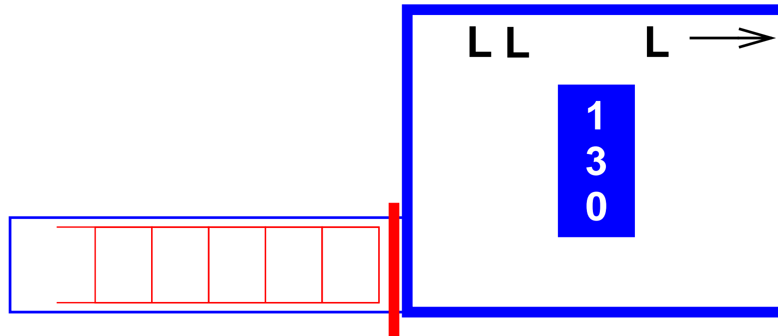
Schreiber



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

UNIX — Semaphore (8)

… **2. Leser-Schreiber-Problem**



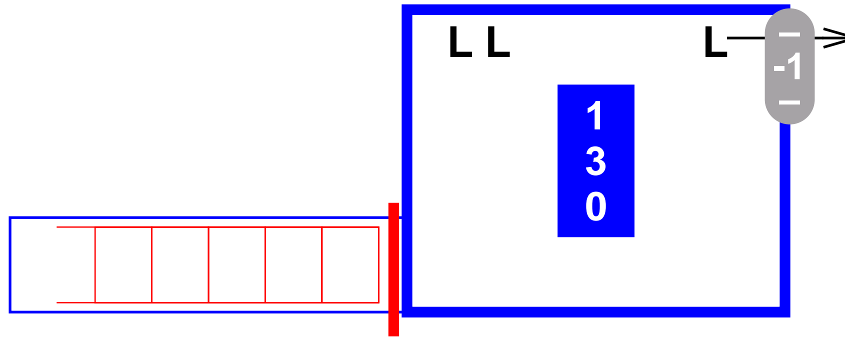
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

... 2. Leser-Schreiber-Problem

Leser

Schreiber

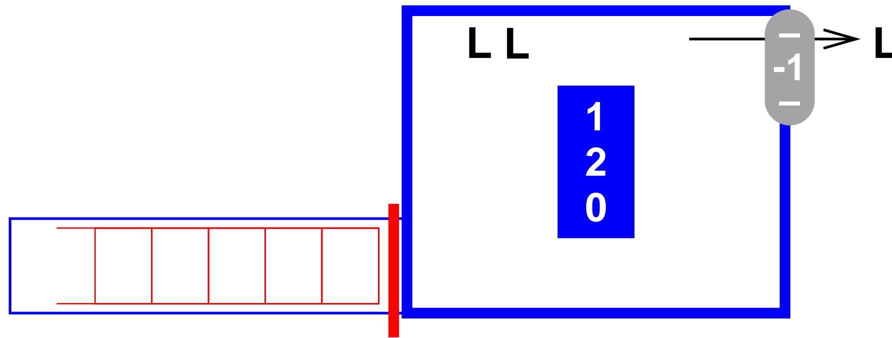


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

... 2. Leser-Schreiber-Problem

Leser

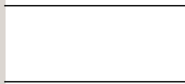
Schreiber

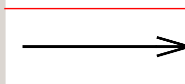


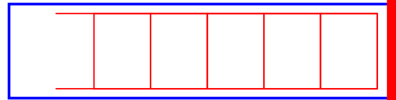
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

... 2. Leser-Schreiber-Problem

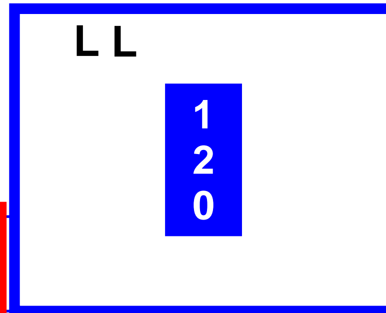
Leser







Schreiber



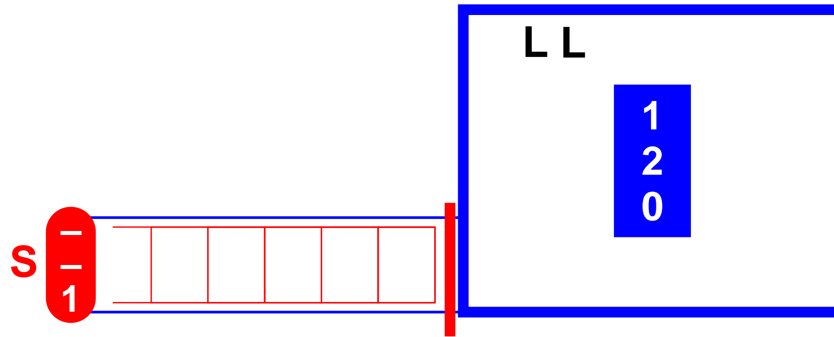
L

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

UNIX — Semaphore (8)

2. Leser-Schreiber-Problem

Leser

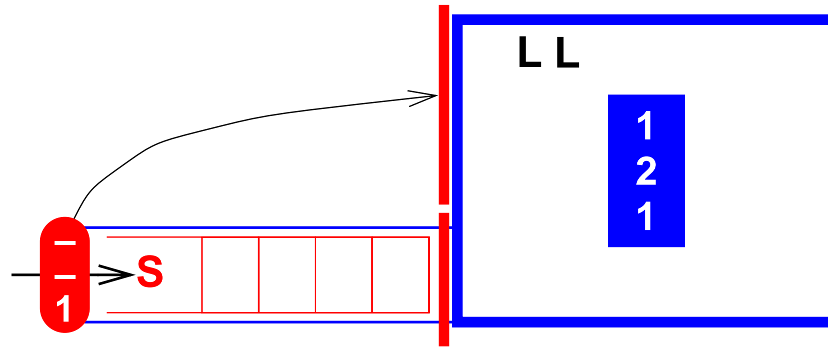


Schreiber

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

… 2. Leser-Schreiber-Problem

Leser

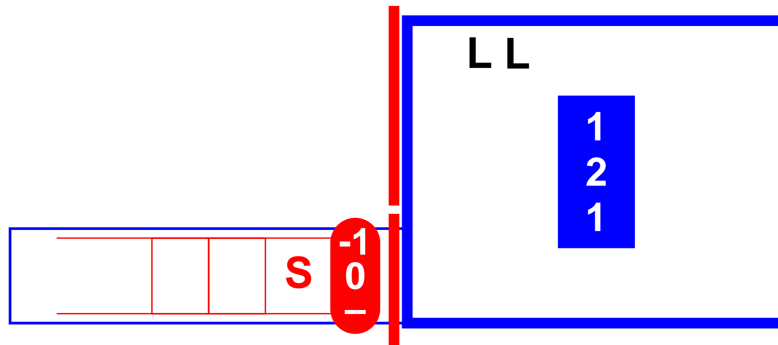


Schreiber

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2. Leser-Schreiber-Problem

Leser

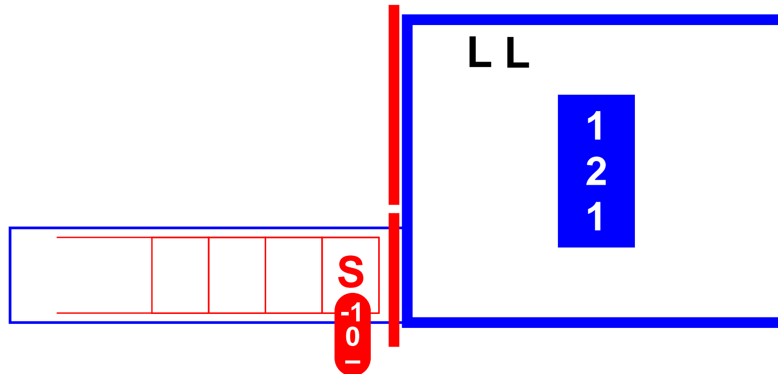


Schreiber

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

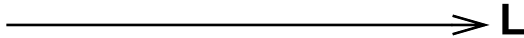
... 2. Leser-Schreiber-Problem

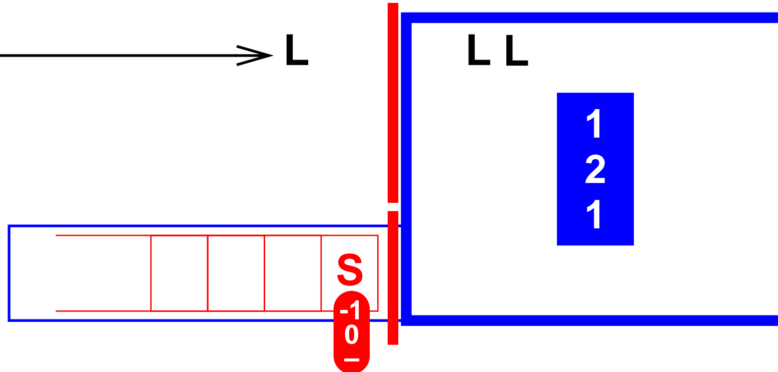


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

... 2. Leser-Schreiber-Problem



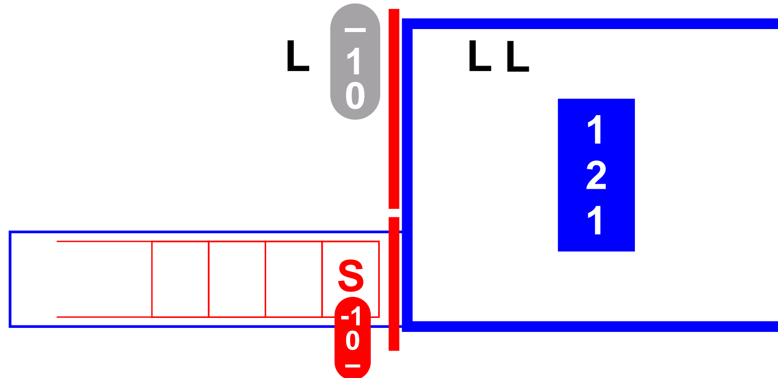


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

… 2. Leser-Schreiber-Problem

Leser

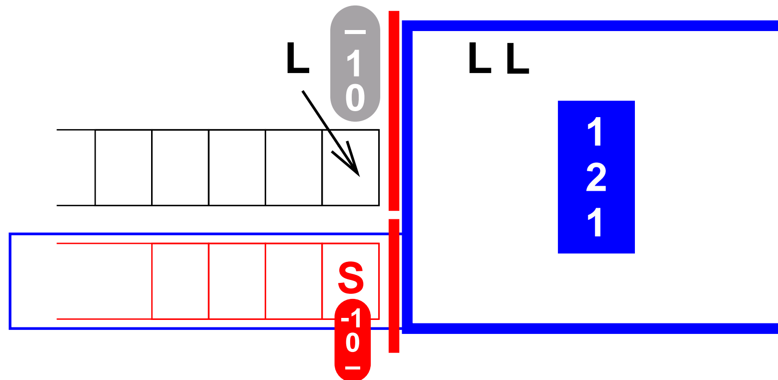
Schreiber



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

... 2. Leser-Schreiber-Problem

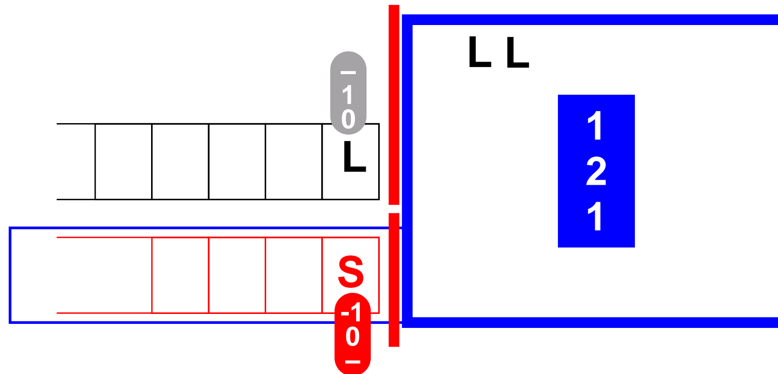
Leser



Schreiber

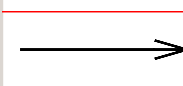
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

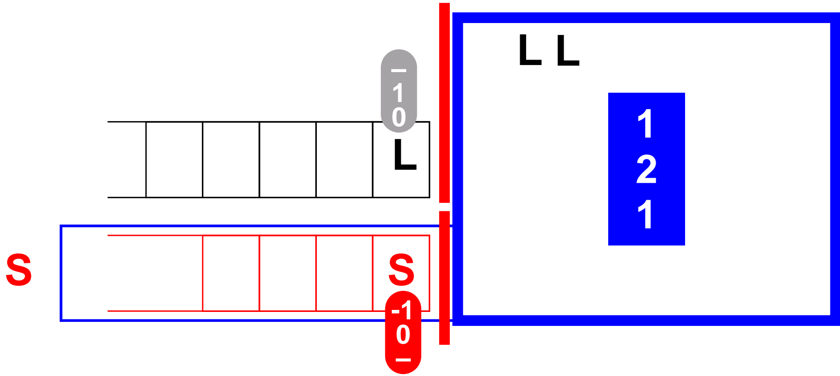
... 2. Leser-Schreiber-Problem



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

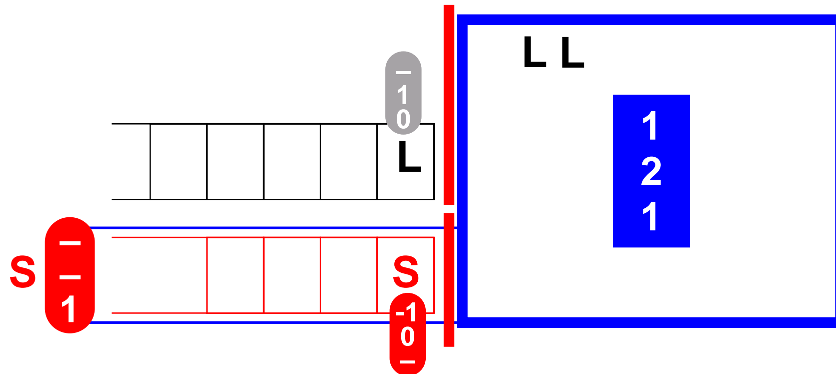
... 2. Leser-Schreiber-Problem





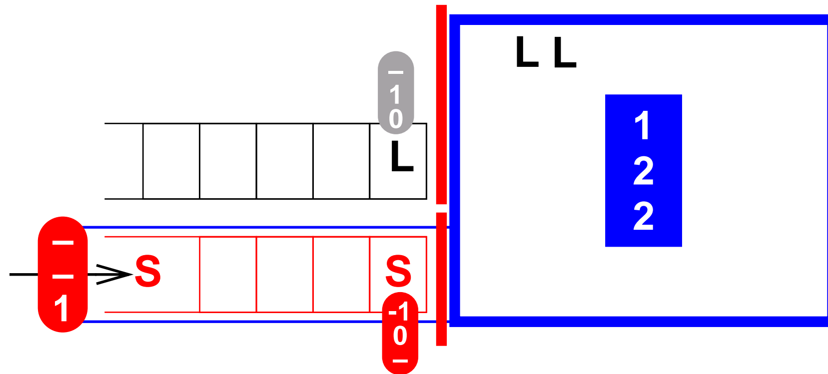
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

... **2. Leser-Schreiber-Problem**



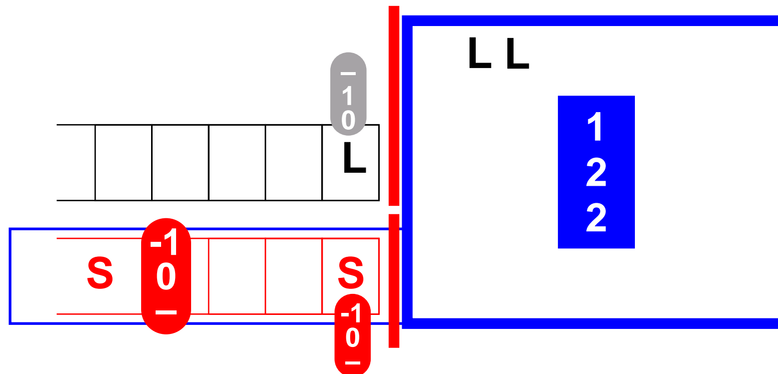
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

... 2. Leser-Schreiber-Problem



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

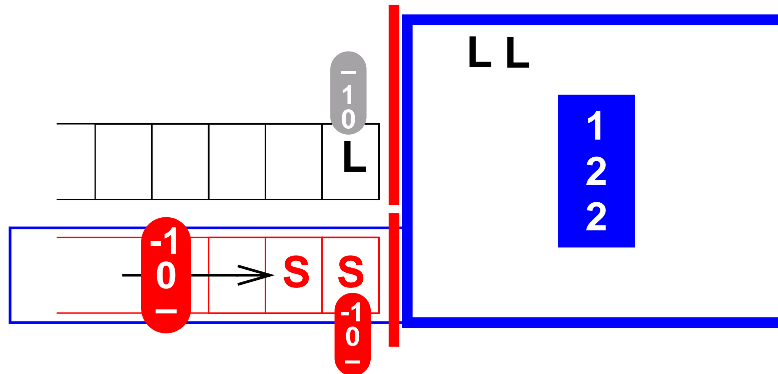
… 2. Leser-Schreiber-Problem



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

… 2. Leser-Schreiber-Problem

Leser



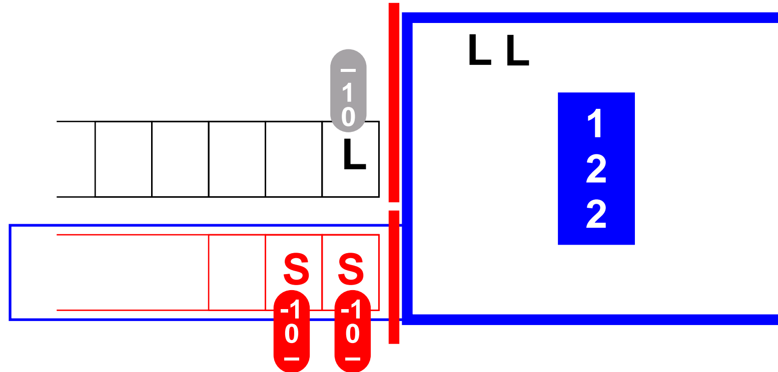
Schreiber

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

...

2. Leser-Schreiber-Problem

Leser



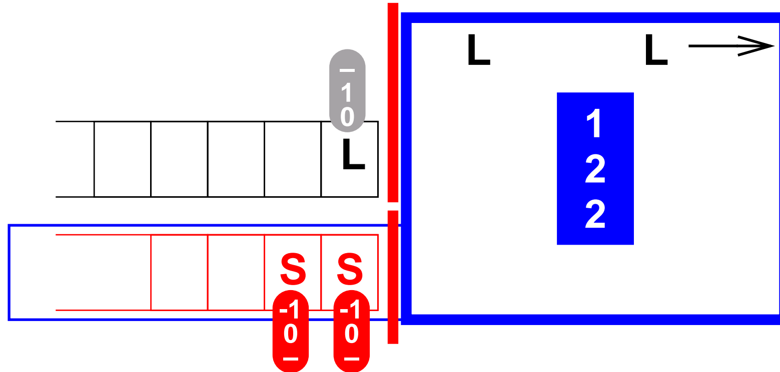
Schreiber

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

... 2. Leser-Schreiber-Problem

Leser

Schreiber

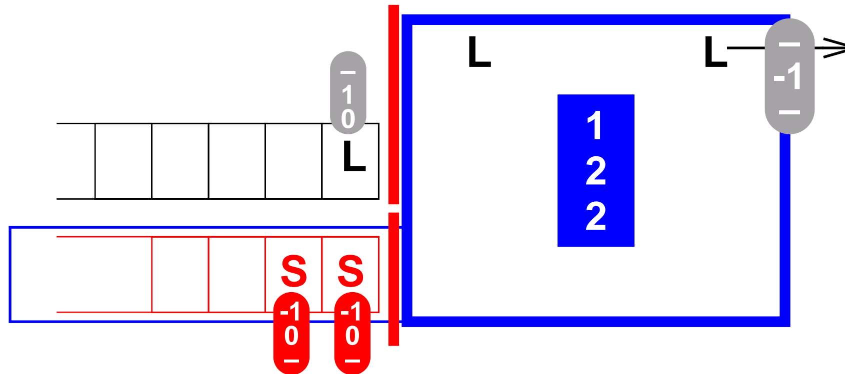


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

... 2. Leser-Schreiber-Problem

Leser

Schreiber

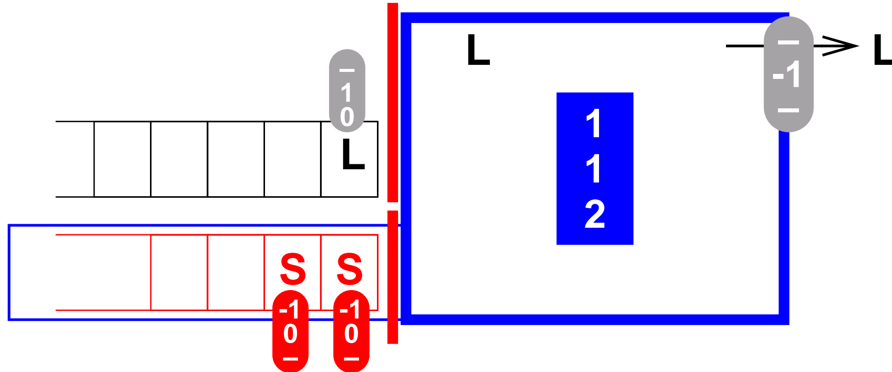


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

... 2. Leser-Schreiber-Problem

Leser

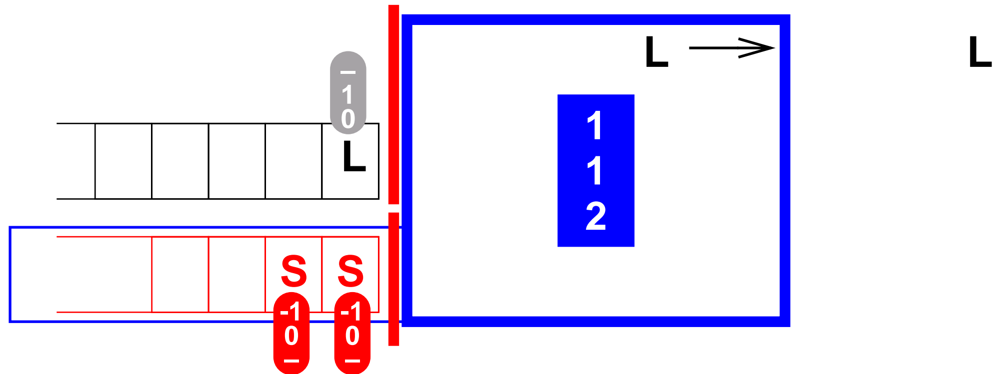
Schreiber



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

... 2. Leser-Schreiber-Problem

Leser



Schreiber

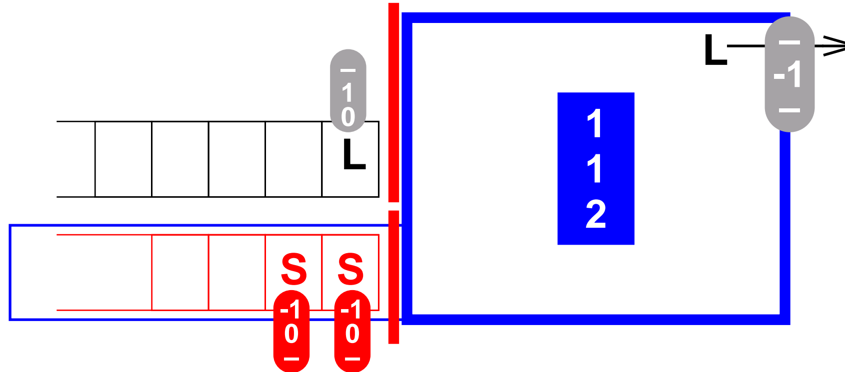
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

UNIX — Semaphore (8)

2. Leser-Schreiber-Problem

Leser

Schreiber

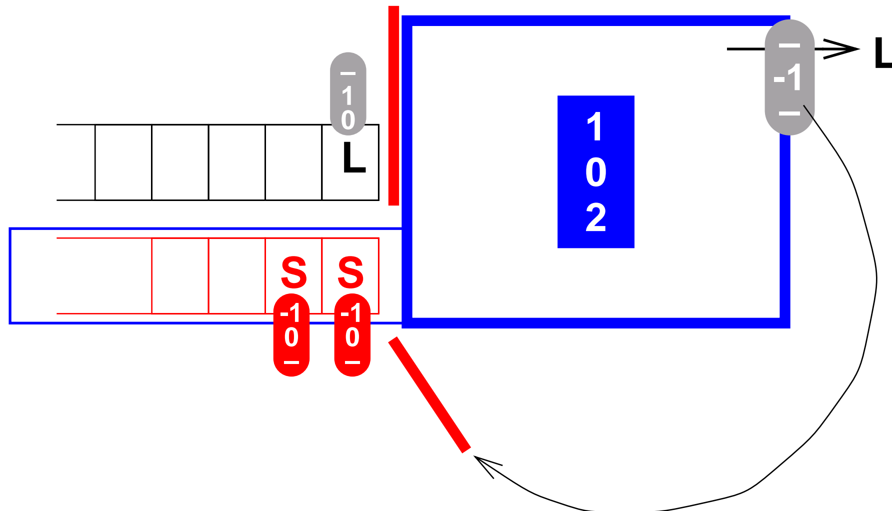


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

... **2. Leser-Schreiber-Problem**

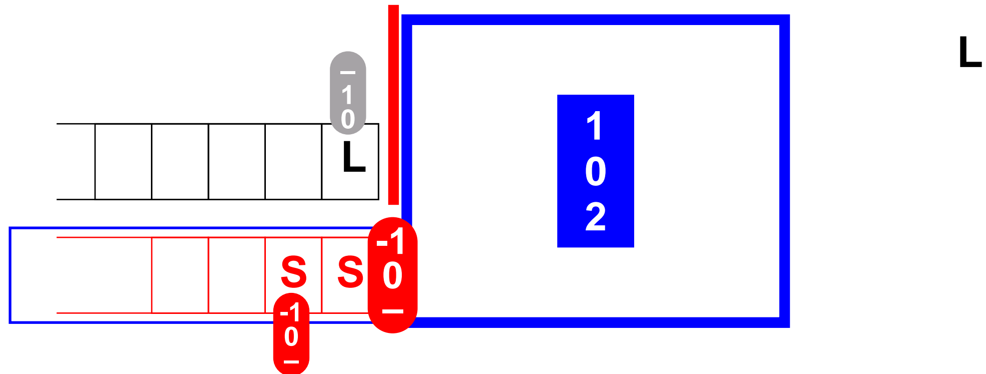
Leser

Schreiber



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

… 2. Leser-Schreiber-Problem

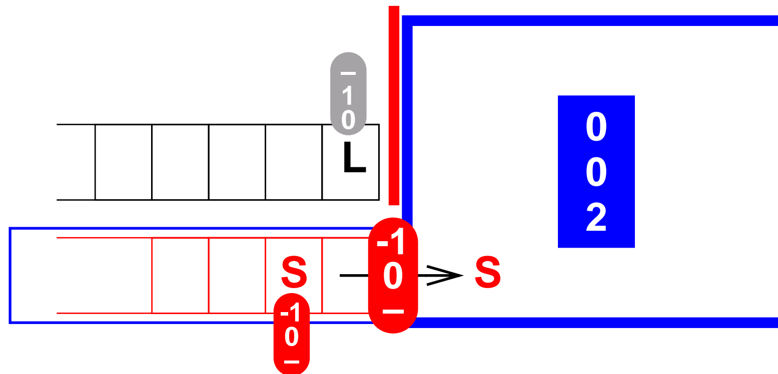


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 UNIX — Semaphore (8)

... **2. Leser-Schreiber-Problem**

Leser



Schreiber

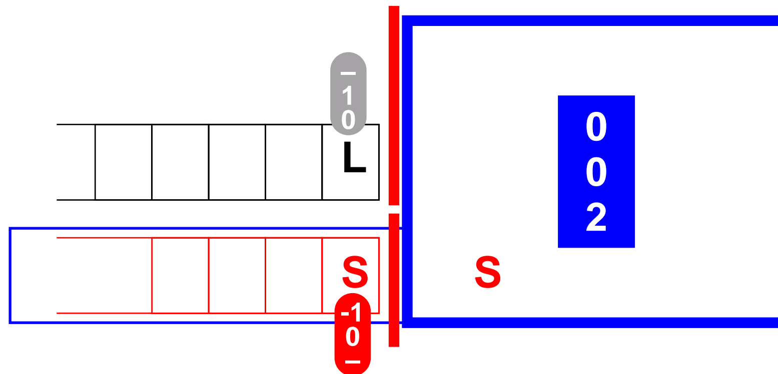
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

… 2. Leser-Schreiber-Problem

Leser

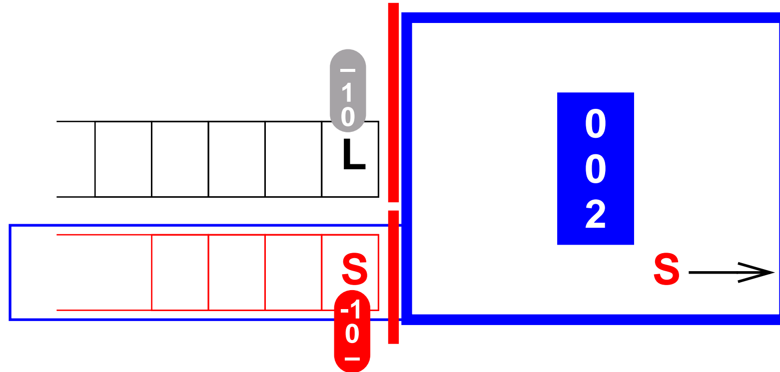
Schreiber



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

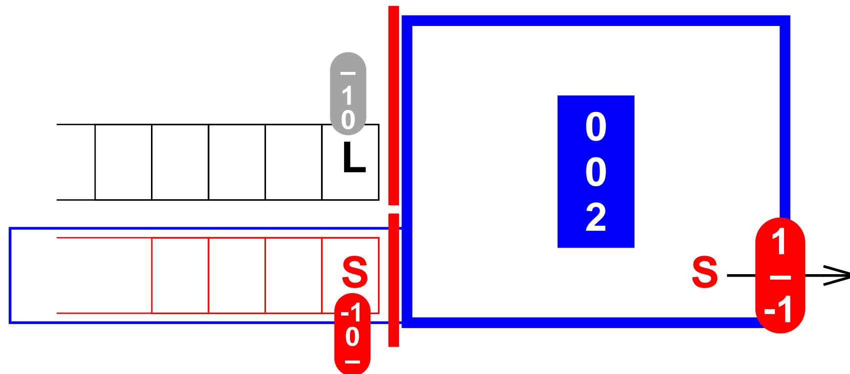
… 2. Leser-Schreiber-Problem



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

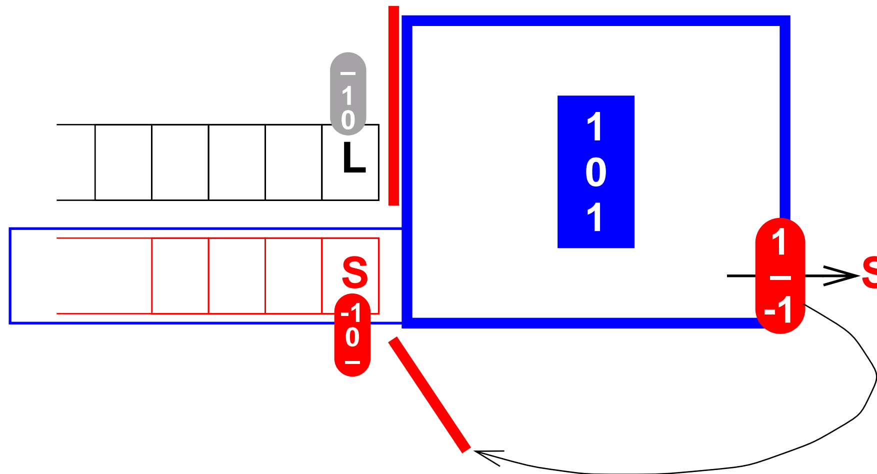
1 UNIX — Semaphore (8)

... **2. Leser-Schreiber-Problem**



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

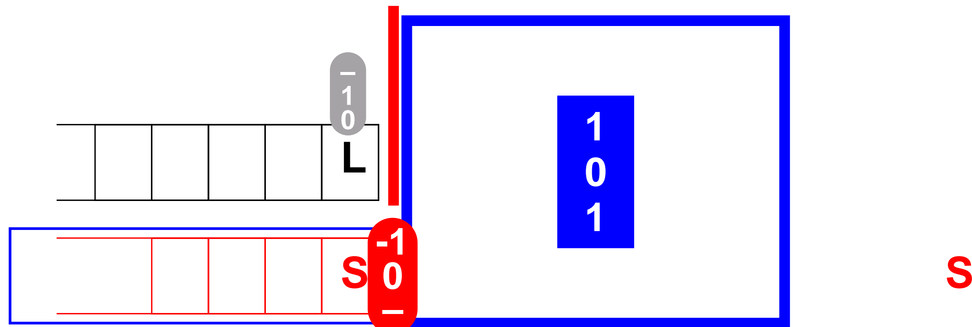
... 2. Leser-Schreiber-Problem



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

... 2. Leser-Schreiber-Problem

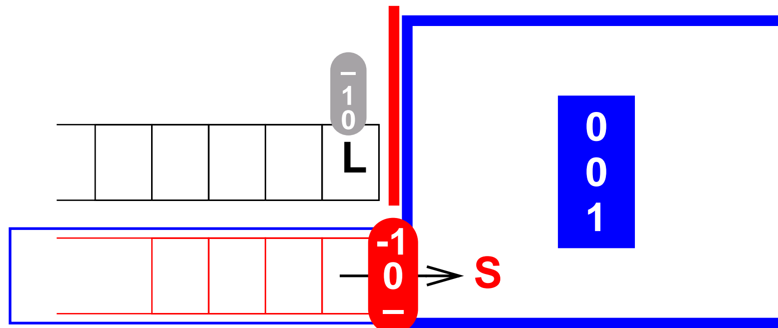
Leser



Schreiber

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

... **2. Leser-Schreiber-Problem**

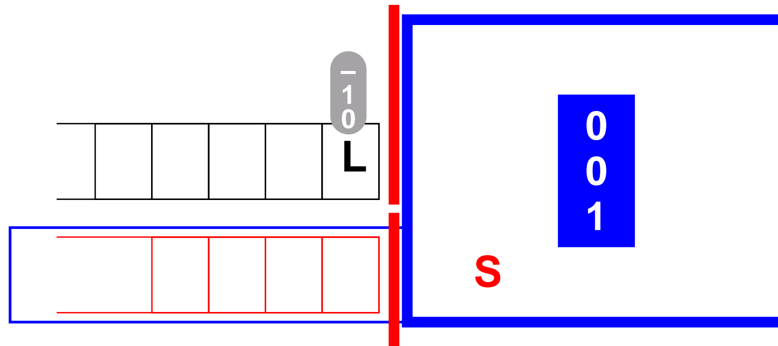


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

... 2. Leser-Schreiber-Problem

Leser

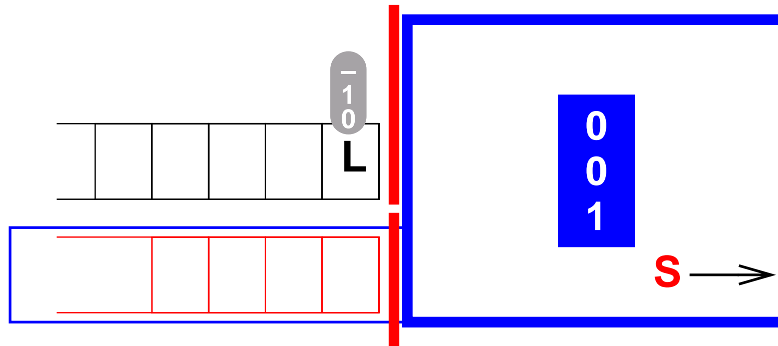


Schreiber

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

… 2. Leser-Schreiber-Problem

Leser

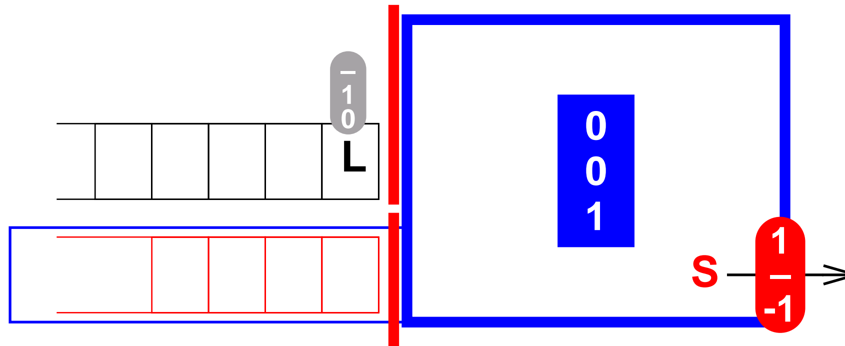


Schreiber

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2. Leser-Schreiber-Problem

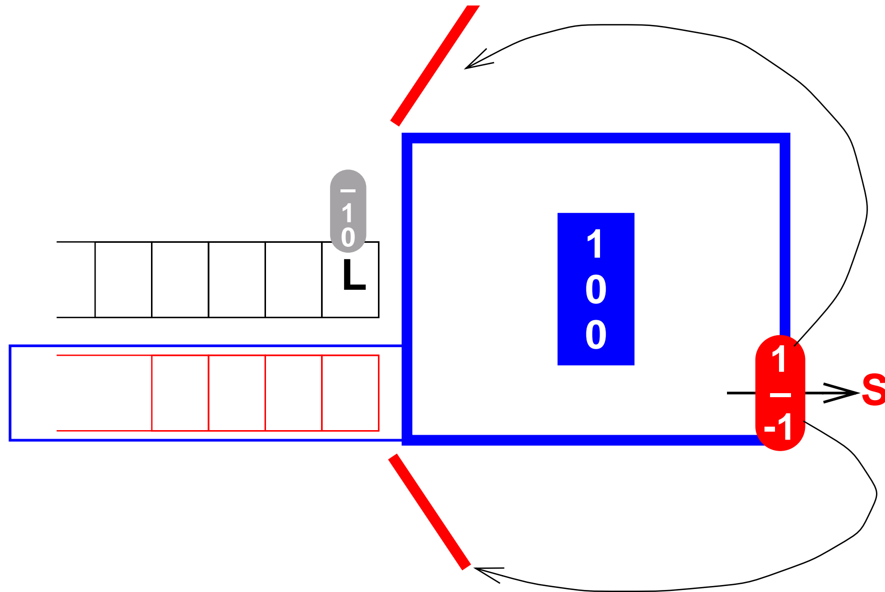
Leser



Schreiber

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

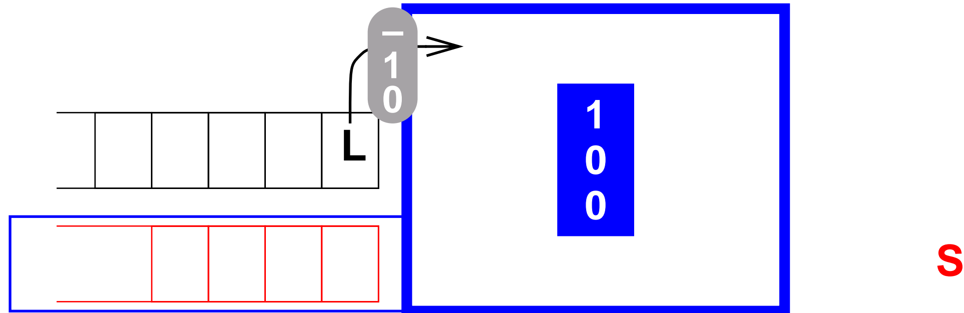
2. Leser-Schreiber-Problem



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

... 2. Leser-Schreiber-Problem

Leser

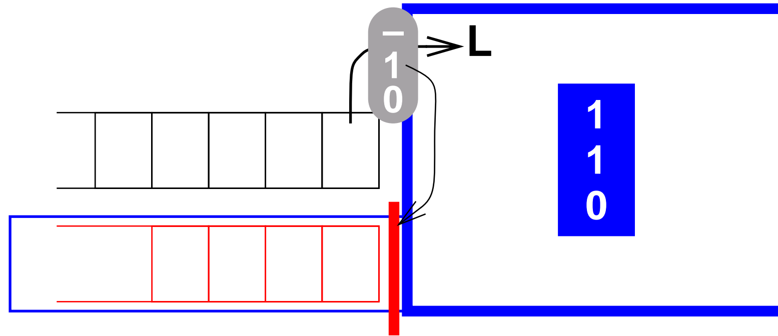


Schreiber

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

... **2. Leser-Schreiber-Problem**

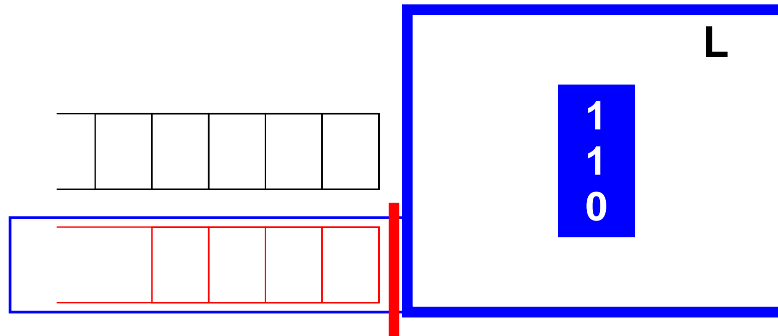
Leser



Schreiber

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2. Leser-Schreiber-Problem

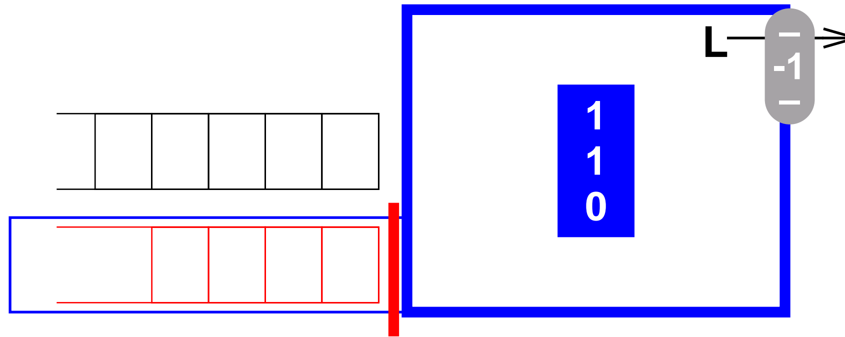


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 UNIX — Semaphore (8)

... 2. Leser-Schreiber-Problem

Leser



Schreiber

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

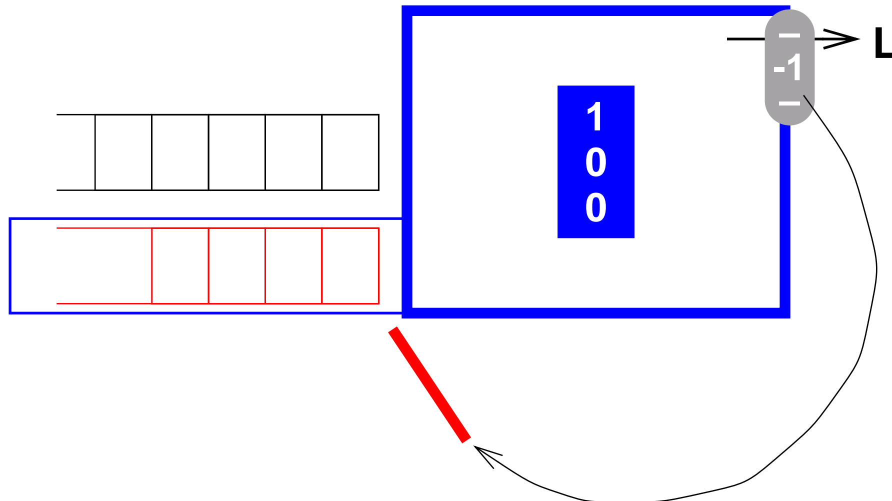
UNIX — Semaphore (8)

...

2. Leser-Schreiber-Problem

Leser

Schreiber

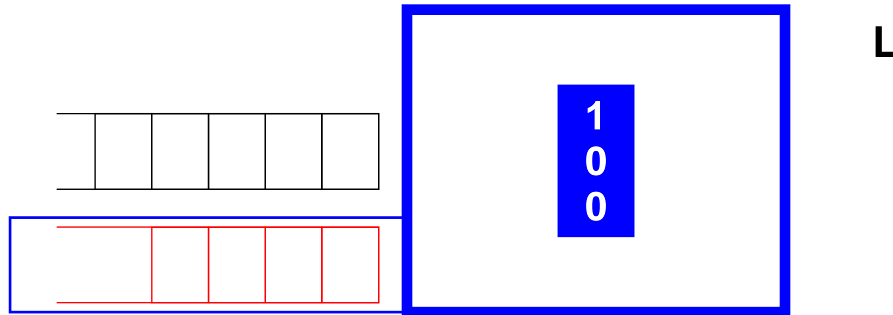


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

… **2. Leser-Schreiber-Problem**

Leser



Schreiber

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

G Interprozeßkommunikation

G.1 Überblick

- ursprüngliche UNIX IPC-Mechanismen (UNIX V7)
  - Signale
  - wait / exit-Wert
  - Pipes
  - Dateien
- UNIX System V - Erweiterungen
  - Named Pipes
  - Shared Memory
  - Messages
  - Semaphore
- Berkeley-UNIX Erweiterungen
  - Sockets
- MACH-IPC
  - Ports & Messages

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

G.2 Sockets

- Endpunkte einer Kommunikationsverbindung
- Arbeitsweise: FIFO, bidirektional
- Attribute:
  - ➤ **Name** (durch *Binding*)
  - ➤ **Communication Domain**
  - ➤ **Typ**
  - ➤ **Protokoll**

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Communication Domain und Protokoll

- **Communication Domain** legt die **Protokoll-Familie**, in der die Kommunikation stattfindet, fest
- durch die Protokoll-Familie wird gleichzeitig auch die Adressierungsstruktur **(Adreß-Familie)** festgelegt
(war unabhängig geplant, wurde aber nie getrennt)
- das **Protokoll**-Attribut wählt das Protokoll innerhalb der Familie aus
- ursprünglich (bis BSD 4.3) existierten nur zwei Communication Domains
  - ➤ UNIX-Domain (AF_UNIX)
  - ➤ Internet-Domain (AF_INET)
- nur AF_INET ist generell vorhanden
daneben derzeit ca. 25 AF definiert
(ISO-Protokolle, DECnet, SNA, Aplletalk, …)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Communication Domain und Protokoll (2)

UNIX Domain

- für lokale Prozeßkommunikation
- bidirektionale Pipes
- Namen aus dem UNIX Dateisystem
  - Eintrag im Dateisystem (Dateityp Socket)
  - verwendbar wie Named Pipes
  - Anlegen und Öffnen nicht mit *mknod( )* und *open( )*, sondern über die normale Socket-Schnittstelle
  - Schließen und Löschen normal über *close( )* und *unlink( )*

Internet Domain

- für Kommunikation über Rechnernetz
- Protokolle: **TCP/IP** oder **UDP/IP**
- Namen: **IP-Adressen** und **Port-Nummern**

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Socket Typen

Stream-Sockets

- unterstützen bidirektionalen, zuverlässigen Datenfluß
- gesicherte Kommunikation (gegen Verlust und Duplizierung von Daten)
- die Ordnung der gesendeten Daten bleibt erhalten
- Vergleichbar mit einer *pipe* - allerdings bidirektional
  (UNIX-Domain- und Internet-Domain-Sockets mit TCP/IP)

Datagramm-Sockets

- unterstützen bidirektionalen Datenfluß
- Datentransfer unsicher (Verlust und Duplizierung möglich)
- die Reihenfolge der ankommenden Datenpakete stimmt nicht sicher mit der der abgehenden Datenpakete überein
- Grenzen von Datenpaketen bleiben im Gegensatz zu **Stream-Socket** -Verbindungen erhalten
  (Internet-Domain Sockets mit UDP/IP)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Socket Typen (2)

■ Raw-Sockets

◆ ermöglichen Zugriff auf das darunterliegende Kommunikationsprotokoll (z. B. IP); sind nicht für den allgemeinen Gebrauch gedacht.

■ Kern-Sockets

◆ *Bi-direktionale, asynchrone* Kommunikation mit Kern-Subsystemem (z.B. Routing-Tabellen-Verwaltung)

◆ *Broadcast* von Kern an User-Level-Prozesse möglich

■ SEQPACKET-Sockets

◆ gesicherter Paket-Strom
(Paketgrenzen bleiben erhalten, aber sichere Übertragung,
wie bei Stream-Sockets)

★ nur Stream- und Datagramm-Sockets sind allgemein implementiert!

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Client-Server Modell

★ Ein ***Server*** ist ein Programm, das einen Dienst ***(Service)*** anbietet, der über einen Kommunikationsmechanismus erreichbar ist

Server

- **akzeptieren Anforderungen**, die von der Kommunikationsschnittstelle kommen
- **führen** ihren angebotenen **Dienst aus**
- **schicken** das **Ergebnis zurück** zum Sender der Anforderung
- *Server* sind normalerweise als normale Benutzerprozesse realisiert

Client

- ein Programm wird ein ***Client***, sobald es
  - eine **Anforderung an einen Server** schickt und
  - auf eine Antwort wartet

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Generieren eines Sockets

- Sockets werden mit dem Systemaufruf *socket(2)* angelegt

```
s = socket(Domain, Typ, Protokoll);
```

5 Namensgebung

- Sockets werden ohne Namen generiert
- durch den Systemaufruf *bind(2)* wird einem Socket ein Name zugeordnet

```
bind (s, Name, Namenslänge);
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

6 Verbindungsanforderungen annehmen

- ein Socket wird durch ***listen(2)*** auf Verbindungsanforderungen vorbereitet, die in einer Warteschlange zwischengespeichert werden

- die angeforderten Verbindungen werden einzeln aufgebaut/ zurückgewiesen

```
listen (s, queuelength);
```

7 Verbindungsaufbau und Kommunikation (Stream-Sockets)

- Über einen Socket ist eine Kommunikation zwischen Prozessen möglich:
  - **UNIX-Domain**-Sockets: Kommunikation lokal auf einem Rechner
  - **Internet**-Sockets: Kommunikation über Netzwerk(e)

- Der Kommunikationsaufbau ist asymmetrisch -
ein Prozeß agiert als ***Client***, der andere als ***Server***.

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

7 Verbindungsaufbau und Kommunikation (Stream-Sockets) (2)

- Ablauf:

Client

```
...
connect(s, &server,
          sizeof(server))

...
write(s, buf, 100);
...
read(s, buf1, 20);
```

Server

```
...
listen(s, 5);
s_new = accept(s, from,
           sizeof(from));

...
read(s_new, buf, 100);
...
write(s_new, bufx, 20);
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

8 Stream-Sockets — Systemschnittstelle

- Verbindungsaufbau

listen(2) teilt dem System mit, daß man bereit ist, Verbindungen auf dem Socket entgegenzunehmen

accept(2) nimmt eine Verbindung an und generiert einen neuen Socket (***s_new***), der für die Kommunikation verwendet werden kann - steht kein Verbindungwunsch an, blockiert *accept*

connect(2) baut die Verbindung vom ***client*** zum ***server*** auf

- Kommunikation

read(2), write(2) verhalten sich wie gewohnt

send(2), recv(2) wie *write( )* bzw. *read(2)* + zusätzliche Funktionalität (Daten ansehen ohne zu lesen*, out-of-band-data, ...)*.

- Verbindungsabbau

close(2) Socket schließen

shutdown(2) Elimieren von bereitliegenden Daten vor einem *close( )*

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 Verbindungslose Sockets

- Für Kommunikation über Datagramm-Sockets kein Verbindungsaufbau notwendig
- Systemaufrufe

  sendto(2) Datagramm senden
  recvfrom(2) Datagramm empfangen
- **Besonderheit: *Broadcasts*** über Datagramm-Sockets (Internet Domain)

10 weitere Systemaufrufe der Socketschnittstelle

getpeername(2)
Namen der mit dem Socket verbundenen Gegenstelle abfragen

getsockname(2)
Namen eines Sockets abfragen

getsockopt(2), setsockopt(2)
Parametrierung eines Sockets abfragen / setzen

- weitere Manual-Seiten (Solaris): ***socket(5), in(5), tcp(7p), ip(7p), udp(7p)***

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

H Rechnerkommunikation (TCP/IP)

H.1 Überblick

- Ende der 70er Jahre in den USA Entwicklung neuer Netzwerkprotokolle, gefördert durch ***DARPA*** *(Defense Advanced Research Project Agency*)
- daraus entstandene Protokollfamilie: **TCP/IP**
  (nach den Namen der zentralen Protokolle -
  T*ransmission* ***C****ontrol* ***P****rotocol* und ***I****nternet* ***P****rotocol*)
- ab 1983 Standardprotokoll im ARPANet
- in Berkeley Implementierung der Protokolle für UNIX (4.1c bsd)
- ab 1985/86 Einsatz von TCP/IP in den Netzen der **NSF** *(National Science Foundation)*, **NASA** und anderer Institutionen
- in den 90er Jahren Übergang zu kommerziellen Netzprovidern (ISPs)
  (z. B. USA: MCI, Sprint, AT&T, D: EUnet, XLink, MAZ)

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
H-Inet.doc 1999-02-24 18.31
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

H.1 Überblick (2)

- IP ermöglicht den Verbund von Netzen,
  das entstandene Gesamtnetz wird als **Internet** bezeichnet

- Größe: 1987: ca. 20.000 Rechner, mehrere hundert Netze
  1994: über 3 Mio. Rechner in 61 Ländern
  Ende 98: >1,6 Mio. in D, > 8 Mio. in EU, ? weltweit
  (sichtbare Rechner), exponentieller Anstieg

- seit Ende der 80er Jahre starke Ausbreitung des **Internet** in Europa (NORDUnet, XLINK, SURFnet, ...)

- in Deutschland zuerst regionale Cluster (Bayern und Baden Württemberg), seit Ende 1989 bundesweites Internet mit dem WIN (Wissenschaftsnetz) als Transportnetz

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

H.1 Überblick (2)

- Implementierungen der Protokolle sind heute für alle in für Netzbetrieb relevanten Betriebssysteme verfügbar
- die TCP/IP-Protokolle sind unabhängig von der verwendeten Netzwerktechnologie - verwendbar sind z. B.
  - ◆ Ethernet, Token Ring, FDDI, ATM
  - ◆ serielle Leitungen, Modemverbindungen (PPP), ISDN
- dieTCP/IP-Protokolle sind keine Standards im Sinn internationaler Standardisierungsgremien (ISO, CCITT), aber ***de facto Standards***
- Arbeitsberichte, Protokollvorschläge und -standards werden in einer Serie von technischen Berichten veröffentlicht, den ***Requests for Comments - RFC***s

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

H.2 Einordnung in das ISO/OSI Referenzmodell

7	Anwendungsschicht	*ftp, telnet, rlogin, rsh, ...*
6	Darstellungsschicht	-
5	Sitzungsschicht	
4	Transportschicht	*TCP, UDP*
3	Netzwerkschicht	*IP*
2	Verbindungsschicht	*Ethernet*
1	physikalische Schicht	

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

H.3 Internet Protocol - IP

- Netzwerkprotokoll zur Bildung eines virtuellen Netzwerkes auf der Basis mehrerer physischer Netze
- definiert Format der Datengrundeinheit - IP-Datagramm
- enthält Regeln, wie Pakete verarbeitet und Fehler behandelt werden (Fragmentierung, maximale Lebenszeit, ...)
- unzuverlässige Datenübertragung
- Routing-Konzepte

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Adressierung

Grundprinzipien

- einheitliche Adressierung, unabhängig von Netzwerk-Technologie (Ethernet, Token-Ring)
  - ➥ virtuelles Netz: ***Internet***
- Netzwerkeinheitlich (über Router) - transparent für Benutzer
- Aufbau einer IP-Adresse:
  - ◆ 4 Byte: **a.b.c.d**, unterteilt in Netzwerk- und Rechneradresse

a < 128:	**a** = Netzwerk, **b.c.d** = Rechner *(Class A Net)* große Organisationen
127 < **a** < 192:	**a.b** = Netzwerk, **c.d** = Rechner *(Class B Net)* Organisationen mittlerer Größe
a > 191:	**a.b.c** = Netzwerk, **d** = Rechner *(Class C Net)* kleinere Organisationen

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
H-Inet.doc 1999-02-24 18.31
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Probleme des Adressierungsschemas

- Mehrere physische Netze in einem Class A oder B Netz
  - ***Subnetting***: Der Rechneranteil der Adresse kann durch sog. ***Subnetmask*** in einen Unternetz- und den eigentlichen Rechneranteil unterteilt werden (Bit-Maske)
  - Nach "außen" wird nur das Class A/B-Netz bekanntgegeben, intern hat man mehrere Netze
- Nur 127 Class A- und ca. 17.000 Class B-Netze,
  aber über 2 Mio Class-C-Netze → *ROADS*-Problem
  - Mehrere Class-C-Netze werden zusammengefaßt (***Supernetting***)
    — statt einem Class-B-Netz z. B. 256 Class-C-Netze
  - Beim Routing wird der Netzblock gemeinsam behandelt

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Adressierung (2)

IP-Adressen an der Uni Erlangen:

- **REVUE-Netz**
  Class B Netz 131.188
- Subnetmask: 0xFFFFFF00
  (teilweise auch 0xFFFFFFC0)
- Beispiele:
  Informatik Hauptnetz: 131.188.2
  Inf.-CIP Subnetz: 131.188.30
  Inf. 4 Subnetz: 131.188.34

  faui01: 131.188.2.1,
  131.188.30.1
  faui04a: 131.188.30.240

LRS-Netz 1:	131.188.137.0
LRS-Netz 2:	131.188.137.64
LRS-Netz 3:	131.188.137.128
LRS-Netz 4:	131.188.137.192
hal.e-technik:	131.188.137.6, 131.188.137.65

- zusätzlich Medizin-internes Netz
  UNI_ERL_MED
  Class B Netz 141.69

➥ Rechner mit mehreren IP-Adressen unerwünscht — besser spezielle Geräte (Router) für die Verbindung zwischen

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Routing

Motivation

- **Routing** bezeichnet die Aufgabe, einen Weg zu finden, über den IP-Pakete geschickt werden
- Prinzipiell gibt es zwei Arten des Routings:
  - **Direktes Routing:** Ein Paket wird direkt von einem Rechner zu einem anderen geschickt (**auf einem physischen Netz**, Grundlage für alle Internet-Kommunikation)
  - **Indirektes Routing:** Der Zielrechner ist **nicht auf demselben physischen Netz**, so daß das Paket über (einen) Router geschickt werden muß

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Routing (2)

Direktes Routing

- Der Absender verpackt das IP-Datagramm in ein Paket des physischen Netzes (Encapsulation)
- Die Ziel-IP-Adresse wird in eine Adresse des physischen Netzes abgebildet
- Das Paket wird mittels der Netzwerk-Hardware übertragen
- häufigstes Verfahren: **ARP** *(address resulution protocol)* (auf Broadcast-Basis)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Routing (3)

Indirektes Routing

- Beim indirekten Routing muß der Absender das verpackte Datagramm **an einen Router schicken**, der selbst wiederum das Datagramm zum Ziel-Netzwerk weiterleiten muß
- **Router** im Internet bilden eine **kooperative, verbundene Struktur**
- **Datagramme werden von Router zu Router weitergereicht**, bis sie das Ziel-Netzwerk erreicht haben.
- Da sowohl der Absender-Rechner als auch ein Router wissen müssen, welchen Router sie für ein bestimmtes Ziel-Netzwerk benutzen müssen, brauchen sie **Routing-Informationen.**
- Routing-Informationen werden in Rouing-Tabellen gehalten

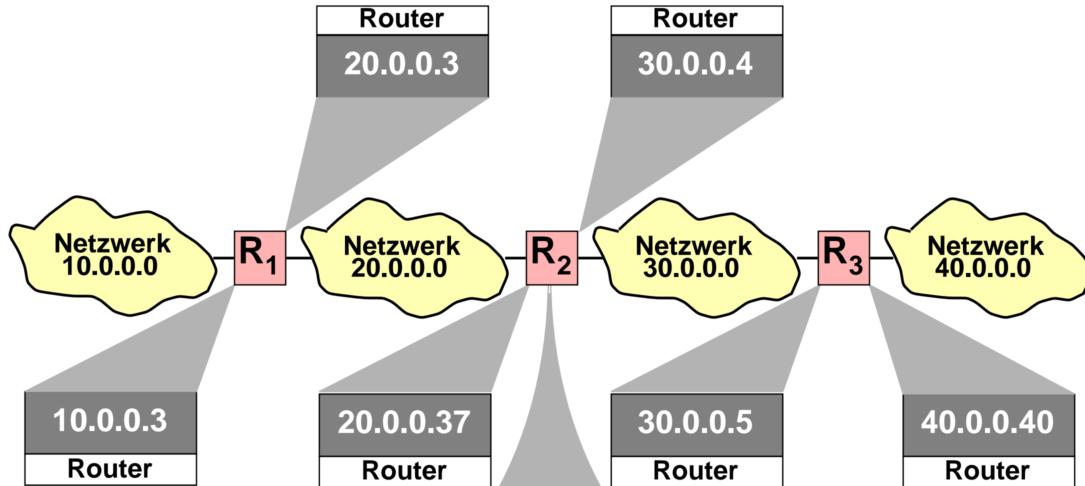
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Routing (4)

Routing-Informationen

- Um die **Routing-Tabellen so kompakt** wie möglich zu halten, beziehen die Routing-Informationen sich (im allgemeinen) **nicht auf Ziel-Adressen, sondern auf Zielnetzwerke**
- IP-Software schickt das jeweilige Datagramm **entsprechend dem Netzwerk-Teil der IP-Adresse** an einen bestimmten Router
- Routing-Tabelleneinträge sind normalerweise ein **Tupel (Netw,Router)**
- Um den Unfang der Routing-Tabellen zu begrenzen, kann ein ***Default Router*** angegeben werden, der für alle nicht explizit aufgeführten Netze benutzt wird.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



Router und Routing-Tabellen

Routing-Tabelle Router R_2	
Ziel-Netzwerk	**Route**
20.0.0.0	Deliver Direct
30.0.0.0	Deliver Direct
40.0.0.0	30.0.0.5
10.0.0.0	20.0.0.3

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Routing (6)

Routing-Protokolle

- **Autonomes System** (AS): Menge von Netzen und Routern einer Administrationseinheit
(z. B. alle Netze im deutschen Wissenschaftsnetz WiN)

- Autonome Systeme tauschen Informationen über Netze und zugehörige AS-Pfade untereinander aus (*Exterior Gateway Protocol — EGP,* speziell *Border Gateway Protocol* — **BGP**)

- Innerhalb eines AS werden Routing-Informationen mit *Interior Gateway Protocols* ausgetauscht (vor allem **RIP** und **OSPF**)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Routing-Protokolle

- **RIP** — Vektordistanz-Algorithmus, Distanz-Metrik = *hop-count*
  - jeder aktive RIP-Rechner überträgt seine Routing-Tabelle regelmäßig per Broadcast
  - jeder RIP-Rechner übernimmt Routing-Information, falls er nicht schon mindestens eine gleichwertige Route kennt
  - Routing-Information von RIP wird nach einer *timeout*-Phase ungültig, wenn sie nicht erneuert wurde
- **OSPF** — Shortest Path First-Algorithmus
  - es werden Informationen über die Netzanschlüsse jedes Knotens ausgetauscht, jeder OSPF-Rechner erstellt für sich daraus den Verbindungsgraphen des Netzes und routet danach
  - viele administrative Einflußmöglichkeiten (*load balancing*, etc.)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

H.4 User Datagram Protocol - UDP

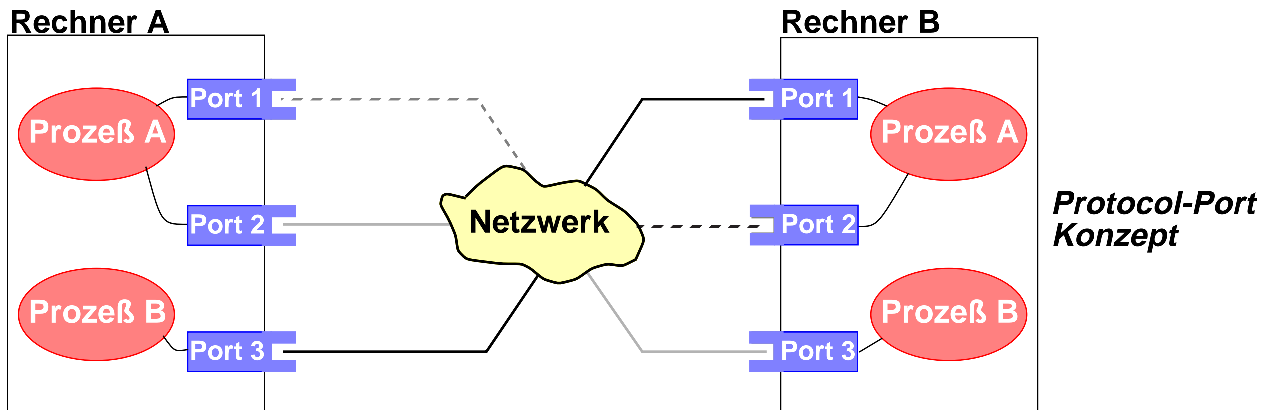
1 Motivation

- bei IP definiert eine Adresse einen Rechner
- keine Möglichkeit, einen bestimmten Benutzer oder Prozeß (Dienst) anzusprechen
- die intuitive Lösung, als Ziel einen Prozeß zu nehmen hat Nachteile
  - ➤ Prozesse werden dynamisch erzeugt und vernichtet
  - ➤ Prozesse können ersetzt werden - die *PID* ändert sich dadurch
  - ➤ Ziele sollten aufgrund ihrer Funktion (Dienst) ansprechbar sein
  - ➤ Prozesse könnten mehrere Dienste anbieten (vgl. *inetd*)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 *Protocol-Port* Konzept

- *Protocol-Ports* sind abstrakte Kommunikations-Endpunkte
- Pakete, die einem Port zugestellt werden, werden in einer Warteschlange gehalten, bis sie abgeholt werden
- Empfangsanforderungen an einen Port werden blockiert, bis ein Paket eintrifft
- zur Kommunikation mit einem anderen Port müssen **Port-Nummer** und IP-Adresse bekannt sein



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 … *Protocol-Port* Konzept (2)

- für eine Reihe von Standard-Diensten (route-Protokoll, rwho, ...) sind Portnummern festgelegt (UNIX → Datei **`/etc/services`**)
- UDP agiert als Multiplexer/Demultiplexer auf IP
- UDP bietet sonst keine zusätzliche Funktionalität gegenüber IP - UDP-Pakete können
  - verloren gehen
  - in falscher Reihenfolge beim Empfänger ankommen

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

H.5 Transmission Control Protocol - TCP

- IP stellt nur unzuverlässige Datenübertragung zur Verfügung
- TCP sichert die Übertragung gegen Fehler (Datenverlust, Paketverdopplung, falsche Reihenfolge)
- TCP versucht die zur Verfügung stehende Netzkapazität möglichst "*freundlich" und optimal* auszunutzen
- das Port-Konzept von UDP ist auch in TCP realisiert
- TCP-Kommunikation hat folgende Eigenschaften
  - ◆ ***Stream*-Orientierung** - die Daten werden als Byte-Strom entgegengenommen / abgeliefert
    - – unstrukturiert (keine Paketgrenzen!)
    - – gepuffert (IP-Pakete werden - wenn möglich - gefüllt)
  - ◆ **virtuelle Verbindung** - vor Beginn der Datenübertragung muß eine Verbindung zum Kommunikationspartner aufgebaut werden
  - ◆ voll-duplex Verbindung

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

H.6 Systemschnittstelle

Internet-Domain Sockets

1 Adressierung

- Rechner-Adresse (32-bit, ***IP-Address***) und Portnummer
- Include-Datei: `<netinet/in.h>`,
  Struktur *sockaddr_in* :

```
struct sockaddr_in {
    short  sin_family;                 /* =AF_INET */
    short  sin_port;                   /* Port */
    struct in_addr sin_addr;           /* IP-Addr */
    char   sin_zero[8];
};
```

sin_port: Portnummer (1 bis 1023 reserviert!)
ab 1024 frei wählbar
0 = Port wird vom System gewählt

sin_addr: IP-Adresse, mit ***gethostbyname(3)*** zu finden

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Rechnernamen / IP-Adressen

- IP-Adressen sind für den Benutzer unhandlich
  → Vergabe von Rechnernamen

- Form der Rechnernamen bei größerer Rechnerzahl wichtig
  - Abbildung Rechnername ↔ Adresse muß möglich sein
  - es dürfen keine Namenskonflikte auftreten

- flacher Namensraum
  - Namen sind Sequenz von Zeichen, ohne Strukturierung
  - zentrale Vergabe notwendig
  - bei großem Namensraum unmöglich zu verwalten

- hierarchischer Namensraum
  - vergleichbar mit UNIX-Dateinamen
  - Namensvergabe dezentral möglich
  - Abbildung Name ↔ Adresse ebenfalls dezentral möglich

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Rechnernamen / IP-Adressen (2)

- dezentrale Abbildung über ***Nameserver***
  - ◆ Beispiel: **`faui04.informatik.uni-erlangen.de`**
- Abbildung Rechnernamen ↔ IP-Adressen (Schnittstelle)
- Funktionen: ***gethostent(3)***
  gethostbyname(3)
  gethostbyaddr(3)
- Include-Datei: `<netdb.h>`
  Struktur *hostent* :

```
struct hostent {
    char *h_name;                /* official name of host
*/
    char  **h_aliases;           /* alias list */
    int   h_addrtype;            /* host address type
(AF_INET) */
    int   h_length;              /* length of address */
    char  **h_addr_list          /* list of addresses from
                                    name server */
```


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 weitere Verwaltungsdaten

★ analog zu den Rechnerdaten gibt es Zuordnungsmechnismen für:

- Netzwerkadressen
  Datei: */etc/networks*
  Funktionen: ***getnetent(3), getnetbyname(3), getnetbyaddr(3)***

- Protokollnamen (TCP, UDP, ...)
  Datei: */etc/protocols*
  Funktionen: ***getprotoent(3), getprotobyname(3), getprotobynumber(3)***

- Service ⇔ Portnummer
  Datei: */etc/services*
  Funktionen: ***getservent(3), getservbyname(3), getservbyport(3)***

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
H-Inet.doc 1999-02-24 18.31
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

H.7 spezielle Funktionalität

- Broadcast-Pakete
  - nur bei Datagramm-Sockets möglich
  - kann von allen Rechnern auf dem Netz empfangen werden
  - netzwerkunabhängige Programmierung kompliziert, da zuerst die angeschlossenen Netzwerke erfragt werden müssen

- Out of band data
  - dringende Daten, die bevorzugt empfangen werden sollen
  - nur bei Stream-Sockets
  - senden mit ***send(2)***, empfangen mit ***recv(2)***
  - bewirkt beim Empfänger das Signal **SIGURG**

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

I Sicherheit in heutigen Betriebssystemen

I.1 Überblick

- UNIX-Sicherheitsmechanismen
- Angriffsmöglichkeiten
- Netzdienste unter UNIX
- Verschlüsselungsverfahren
- Firewalls

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

I.2 UNIX-Sicherheitsmechanismen

- Authentisierung durch private Paßwörter
- Paßwörter werden verschlüsselt gespeichert:
  `/etc/passwd`, NIS (Network Information Service)
- Verschlüsselungsalgorithmus DES:
  - ◆ Digital Encryption Standard von IBM,
    1977 standardisiert, 56 Bit Schlüssellänge
  - ◆ gilt als dekodierbar
- Problem:
  - ◆ Verschlüsselte Paßwörter sind öffentlich zugänglich
  - ◆ Wörterbuchattacken möglich (crack)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

I.2 UNIX-Sicherheitsmechanismen (2)

- Lösungsansatz:
  - ◆ Verschlüsselte Paßwörter werden in Shadow-Dateien gespeichert (`/etc/shadow`), die nur mit Systemverwalterprivilegien lesbar sind
  - ◆ Problem: alle Programme, die Paßwörter überprüfen (z.B. `xlock`), müssen Systemverwalterprivilegien erhalten

- Weitergehende Lösungen:
  - ◆ Verwendung verbesserter Authentisierungsverfahren:
    - Kerberos (am MIT entwickeltes Sicherheitssystem)
    - DCE (Distributed Computing Environment)
  - ◆ Integration dieser Authentisierungsverfahren in bestehende Betriebssystemdienste z.B. durch PAM (Pluggable Authentication Modules)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Angriffspunkte

- Programme, die mit erweiterten Rechten ausgeführt werden müssen (**`passwd`**, **`su`**, ...) verwenden den S-Bit Mechanismus

- Programmierfehler in diesen Programmen
  - unkontrollierte Dateizugriffe
    z.B. Öffnen einer Datei **`/tmp/<PID>`** um zu schreiben
  - buffer overflow

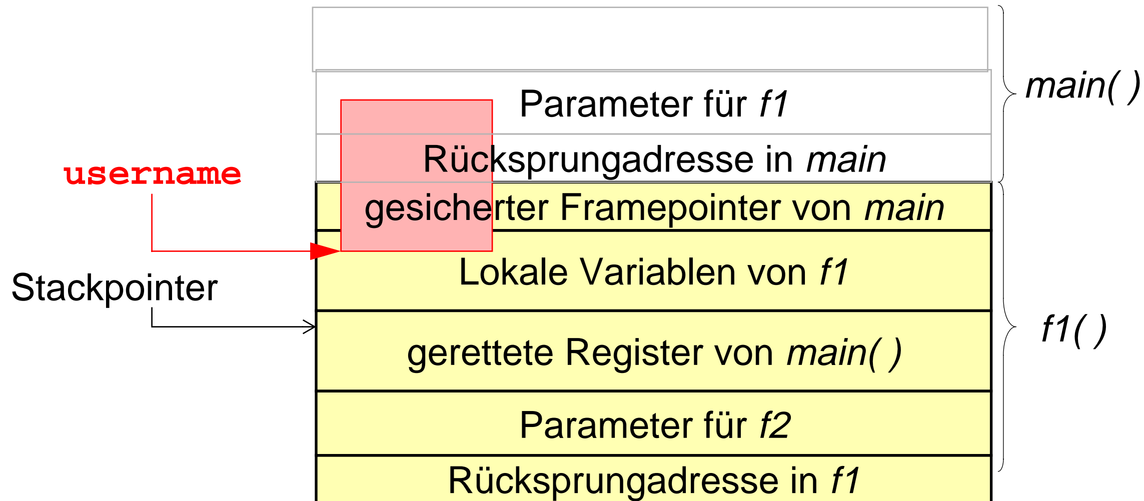
```
...
char username[9];
char buffer[256];
...
strcpy(username, argv[1]);
...
gets(buffer);
...
sprintf(buffer, "%s", argv[1]);
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Vorgehensweise

- Zugriff auf das System über beliebigen Account
  (z.B. Paßwort mit crack geraten)

- Auslösen eines buffer overflow



- Erlangung von Systemverwalterprivilegien

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Gegenmaßnahmen

- Kapselung der sicherheitskritischen Abschnitte (TCB: Trusted Computing Base)
- Verwendung der zusätzlichen Privilegien nur in den TCBs (hilft nicht gegen buffer overflow Probleme)

```
...
seteuid(0);
fd = open(“/etc/passwd”, O_RDWR);
seteuid(getuid());
...
```

- Sorgfältige Programmierung

```
...
strncpy(username, argv[1], 8); username[8] = 0;
...
fgets(buf1, sizeof(buf1), stdin);
...
snprintf(buf2, sizeof(buf2), "%s", argv[1]);
...
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Risikoabschätzung

- Bekannte Sicherheitslücken verbreiten sich sehr schnell!!!
  - bugtraq-Mailingliste
  - `http://www.rootshell.com/`
  - `http://www.dhp.com/~fyodor/`
  - IRC (Internet Relay Chat)

- Alle schlecht gewarteten Systeme im Internet sind angreifbar (Auch Heimrechner, die über PPP angebunden sind!!!)
  - Beispiel Linux:
    - In Standard-Distributionen sind oft 20-30 Dienste eingeschaltet.
    - Bereits wenige Wochen nach Erscheinen werden Sicherheitslücken bekannt.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

I.4 Netzdienste unter UNIX

- Datentransfer- / Kontrolltransferdienste rlogin, rsh, telnet, ftp
  - ◆ Authentisierung mit Klartextpaßwort
  - ◆ Angriffsmöglichkeiten:
    - Paßwörter sammeln durch Abhören des Netzverkehrs
    - Übernahme von Verbindungen

- Datentransfer- / Mitteildienste E-Mail, WWW
  - ◆ Keine Authentisierung
  - ◆ Absender kann beliebig angegeben werden, Mails können modifiziert werden.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

I.5 Verschlüsselungsverfahren

- Verschlüsselung über Netzwerk allgemein:

- Symmetrische Verfahren (DES, IDEA, ...):
  - ◆ Verschlüsselung und Entschlüsselung mit dem gleichen Schlüssel

- Asymmetrische Verfahren (RSA, El Gamal, Diffie/Hellman):
  - ◆ Verschlüsselung und Entschlüsselung mit unterschiedlichem Schlüssel
  - ◆ ein Schlüssel läßt sich nicht aus dem jeweils anderen berechnen
  - ◆ Idee:
    - Verbreitung eines Schlüssels ("öffentlicher Schlüssel")
    - Geheimhaltung des anderen Schlüssels ("privater Schlüssel")

  → Alle können Daten verschlüsseln, die man nur selbst entschlüssel kann

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Anwendung von Verschlüsselung

- Mail, WWW:
  - Signieren und Verschlüsseln von Mail und WWW-Seiten
  - Dazu nötig: öffentlicher Schlüssel des Kommunikationspartners
  - Beispiele: Mail: PGP, S/MIME (Netscape)
    WWW: TLS (Transport Level Security, vormals SSL)

- Datentransfer- / Kontrolltransferdienste ssh, scp, slogin
  - Verschlüsseln bei Verbindungsaufnahme und Verschlüsselung der Daten
  - Jeder Rechner besitzt einen öffentlichen Schlüssel
  - Problem: Die öffentlichen Schlüssel müssen verteilt werden
  - Bei **slogin/ssh/scp** bin ich sicher, daß keiner mithört und daß ich auch genau mit dem Rechner verbunden bin, den ich angebe

```
usa> slogin faui01a.informatik.uni-erlangen.de
Password: xxx                    # nur die faui01a bekommt das
Paßwort
faui01a>                         #
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

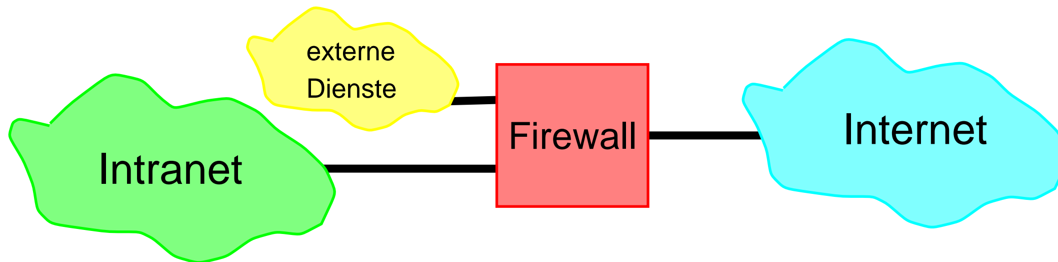
2 IP Security

- Eigenständiges Sicherheitskonzept für IP
- sichert Integrität, Authentizität und / oder Vertraulichkeit zu
- Protokolldefinition unabhängig von den verwendeten Verschlüsselungsverfahren (z.B. MD5, 3DES)
- Kernprotokoll:
  - AH: Authentication Header
    Integrität, Authentizität, Sicherheit vor Wiederholungen (replay)
  - ESP: Encapsulating Security Payload Header
    zusätzlich zu AH: Vertraulichkeit
- Header können getrennt oder gemeinsam verwendet werden
- Zusatzprotokolle für Schlüsselaustausch (IKE),
  Definition von öffentlichen Schlüsseln und Zertifikaten (SPKI) und Namensauflösung (DNSsec)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

I.6 Firewalls

- Mit einer Firewall werden vertrauenswürdige und nicht vertrauenswürdige Netzwerksegmente getrennt.



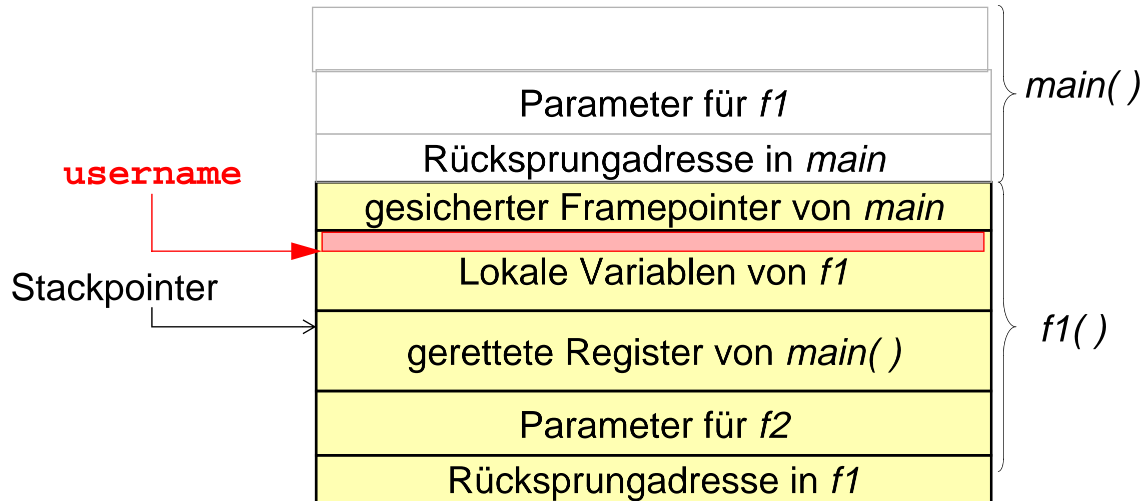
- Firewall-Funktionalität:
  - ◆ Intelligenter Router (erlaubt nur gewisse Dienste)
  - ◆ Paket-Filter: filtert "defekte" Pakete / SYN-Attacken
  - ◆ Gateway:
    - WWW-Inhalt filtern
    - Adressen umsetzen
  - ◆ Sicheres Tunneln: z.B. mit "virtual private network"

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Vorgehensweise

- Zugriff auf das System über beliebigen Account (z.B. Paßwort mit crack geraten)

- Auslösen eines buffer overflow



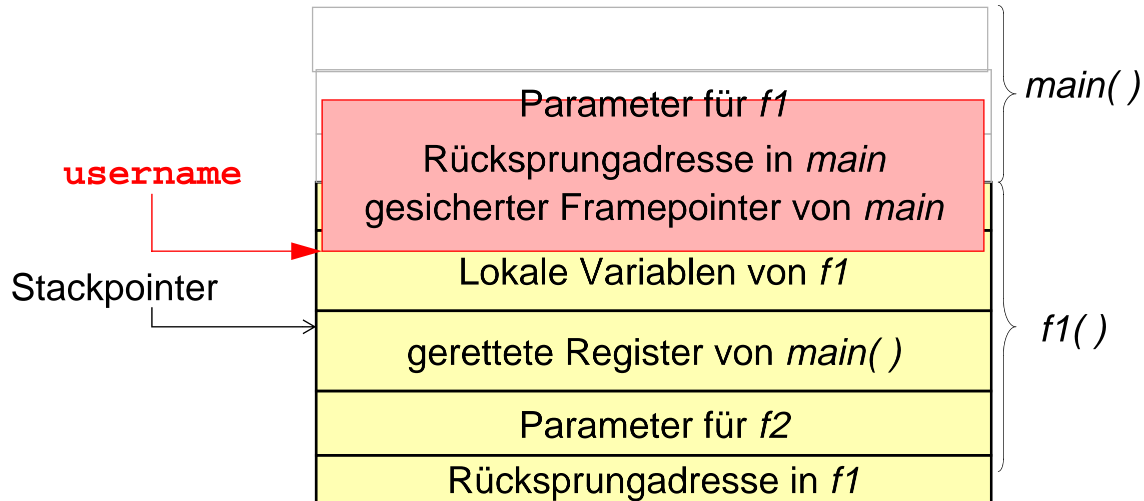
- Erlangung von Systemverwalterprivilegien

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Vorgehensweise

- Zugriff auf das System über beliebigen Account
(z.B. Paßwort mit crack geraten)

- Auslösen eines buffer overflow



- Erlangung von Systemverwalterprivilegien

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

J UNIX-Dateisystem

J.1 Funktionalität

- Abstraktionen für Benutzersicht:
  1. Pfade, Dateinamen
  2. Dateibaum verdeckt mehrere Platten (bzw. Partitionen)
  3. Dateien = unstrukturierte Byteströme
  4. *special files* fast wie reguläre Dateien zugreifbar
- diese Abstraktionen müssen vom Dateisystem auf konkrete Objekte abgebildet werden
- den obigen Abstraktionen entsprechen etwa folgende Objekte:

  1. / 2. Plattenpositionen, Adressen von Gerätecontrollern, etc.

  3. Platten verwalten Sektoren, Spuren und Zylinder

  4. unterschiedliche Gerätetreiber werden für E/A-Operationen auf verschiedenen Peripheriegeräten benötigt

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
J-Filesystem.doc 1999-02-24 18.02
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

J.2 Directories (Kataloge)

- wichtigster Bestandteil zum Aufbau eines hierarchischen Dateisystems
- Funktion: Strukturierung des Namensraums
- *Directories* enthalten Verweise auf Inodes
  (von Dateien oder weiteren Directories)

J.3 Inodes

- ein Inode enthält alle Informationen, die zur Verwaltung einer Datei und zum Auffinden ihres Inhalts benötigt werden
- beim Öffnen einer Datei werden die Inode-Daten von der Platte ***(disk inode)*** in die Inode-Tabelle des Systemkerns **(*in-core inode)*** gelesen

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Umwandlung: Pfad : Inode

- Root = Inode 2
- Inode-Nummer ist nur in dem Directory zu finden, in dem die Datei eingetragen ist
  - ➥ die Pfadkomponenten müssen einzeln bearbeitet werden
  - ➥ um den Inode der jeweils nächsten Pfadkomponente zu bestimmen, muß der Inhalt des voranstehenden Directories eingelesen werden
- Systemkern-Funktion: ***lookup( )***
- Directory Lookup Cache zur Beschleunigung
  - ◆ LRU Cache mit Einträgen
    Directory-Inode-Nummer / Datei(name) in Directory / Inode der Datei

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Umwandlung: Pfad : Inode (2)

- Berücksichtigt werden hierbei:
  - ◆ Symbolic Links
    - – werden beim Absteigen im Baum verfolgt
  - ◆ Mount-Punkte
    - – beim Absteigen im Baum am Mount-Punkt Wechsel zur *root* des eingehängten Dateisystems
    - – beim Aufsteigen im Baum (über "..") an der Root eines Dateisystems Wechsel auf den Mount-Punkt
  - ◆ Current Root
    - – Überschreiten beim Aufsteigen im Baum (über "..") wird verhindert

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Datei öffnen

- Aufruf:

```
fd = open(path, flags, modes);
```

fd	Filedescriptor (kleiner Integer-Wert)
path	Pfadname der Datei
flags	Angabe über Zugriffsform (lesen, schreiben, lesen+schreiben), ob Datei neu erzeugt / vorher gelöscht wird
modes	Zugriffsrechte, falls die Datei neu erzeugt wird

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Datei öffnen (2)

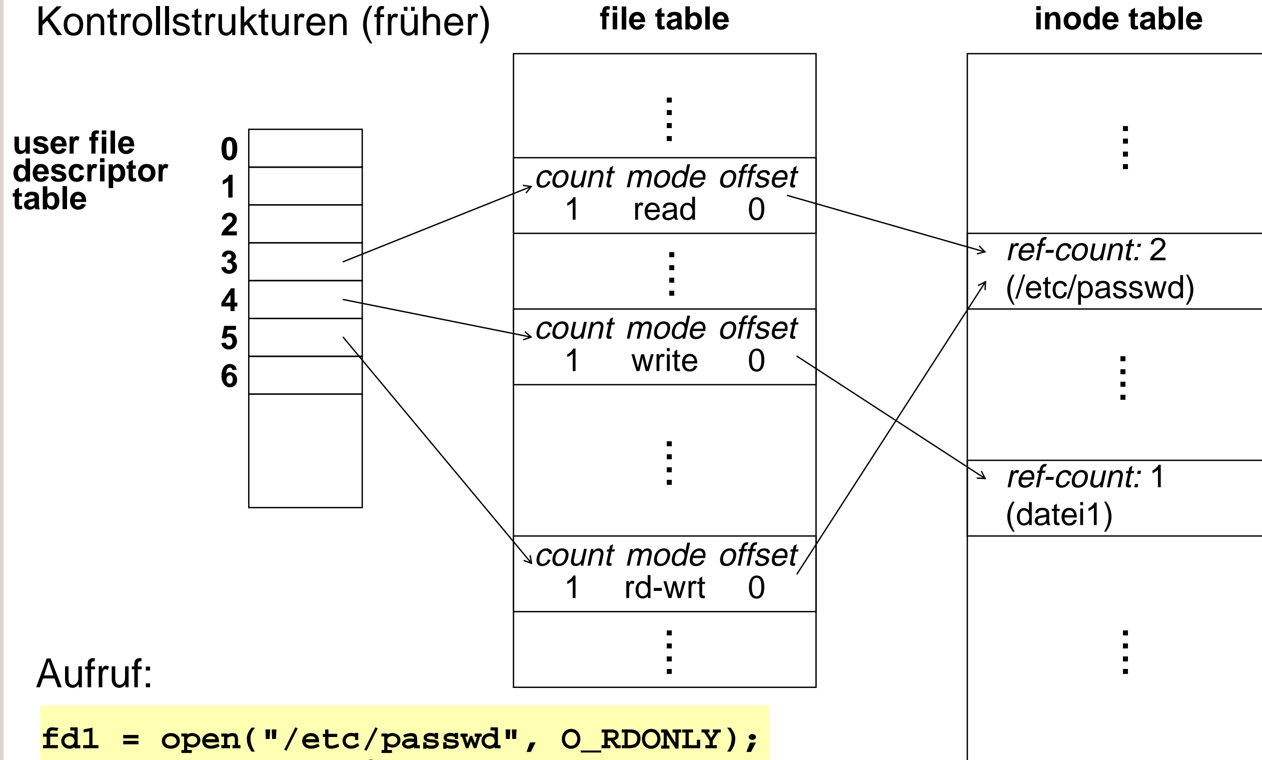
- open-Algorithmus

in-core inode zu Pfadnamen erzeugen ***(namei, lookup)*** (Inode suchen und einlesen)	
Datei existiert nicht oder keine Zugriffsrechte	
ja	nein
in-core Inode freigeben	***file table*** entry für Inode belegen ***count*** und ***offset*** initialisieren
return(Fehler)	
	user file descriptor belegen, Zeiger auf ***file table*** *entry* setzen
	falls ***truncate***, alte Blöcke freig.
	return(***user file descriptor)***

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Datei öffnen (3)

- Kontrollstrukturen (früher)



- Aufruf:

```
fd1 = open("/etc/passwd", O_RDONLY);
fd2 = open("datei1", O_WRONLY);
fd3 = open("/etc/passwd, O_RDWR);

-> fd1 == 3, fd2 == 4, fd3 == 5
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Datei öffnen (4)

user file descriptor table

- Feld von Zeigern auf Einträge in der ***file table***
- Teil der ***user-area*** des Prozesses

  ```
  struct file *u_ofile[NOFILE]
  ```

  siehe *<sys/user.h>*
- Filedescriptor (aus ***open(2)***) ist ein Index in diese Tabelle

  ➥ Entkopplung von Systemdatenstrukturen und Benutzerdaten
  Filedeskriptor = Capability
  (Sicherheit!)

file table

- Feld von ***file***-Strukturen (siehe *<sys/file.h>*), enthält
  - Zeiger auf Eintrag in der ***inode table***
  - Eintrag für Zugriffsmodus
  - Schreib/Lesezeiger (offset)
  - Referenzzähler

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
J-Filesystem.doc 1999-02-24 18.02

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

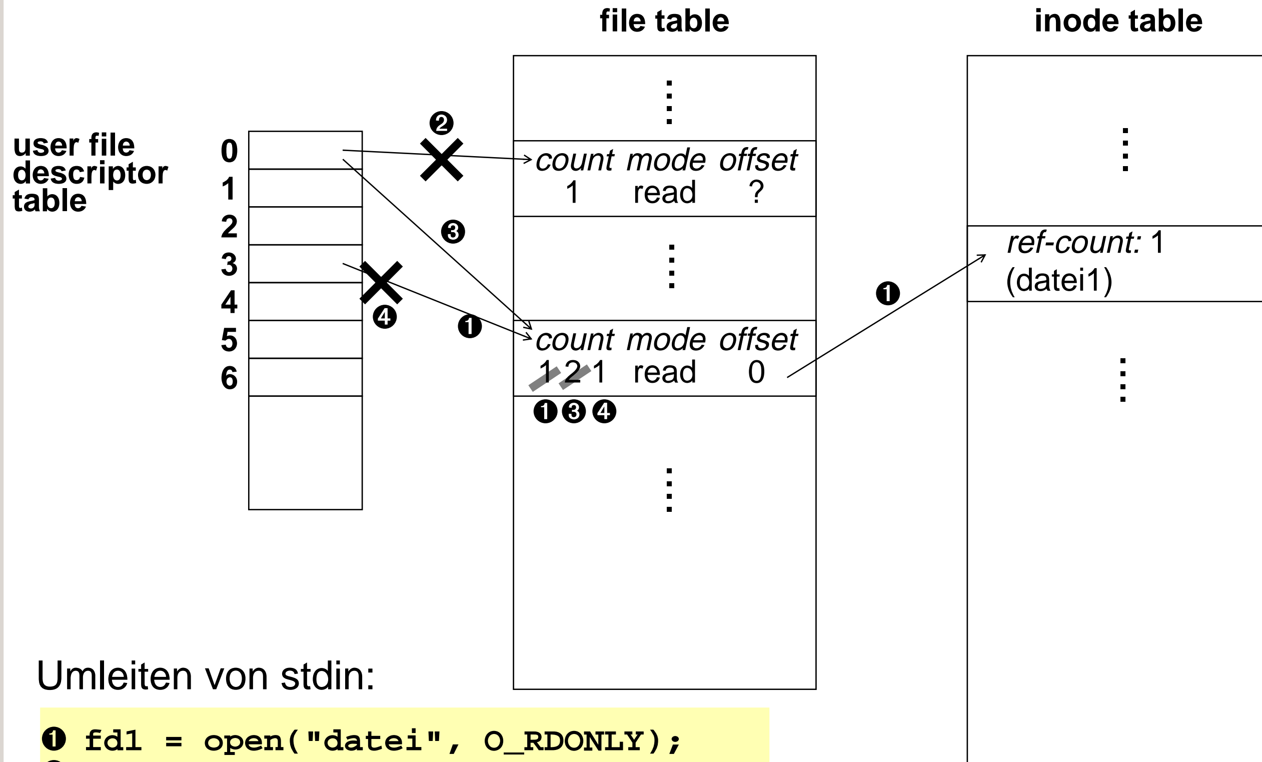
2 Datei öffnen (5)

inode table

- Feld von ***in-core inodes*** (siehe *<ufs/inode.h>*), enthält
  - Referenzzähler
  - Info über physikalisches Gerät (Platte, Partition)
  - *disk inode* Struktur
  - Informationen für ***lock***-Funktionen (***flock(2), lockf(2)***)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Kontrollstrukturen und Systemaufruf *dup( )*

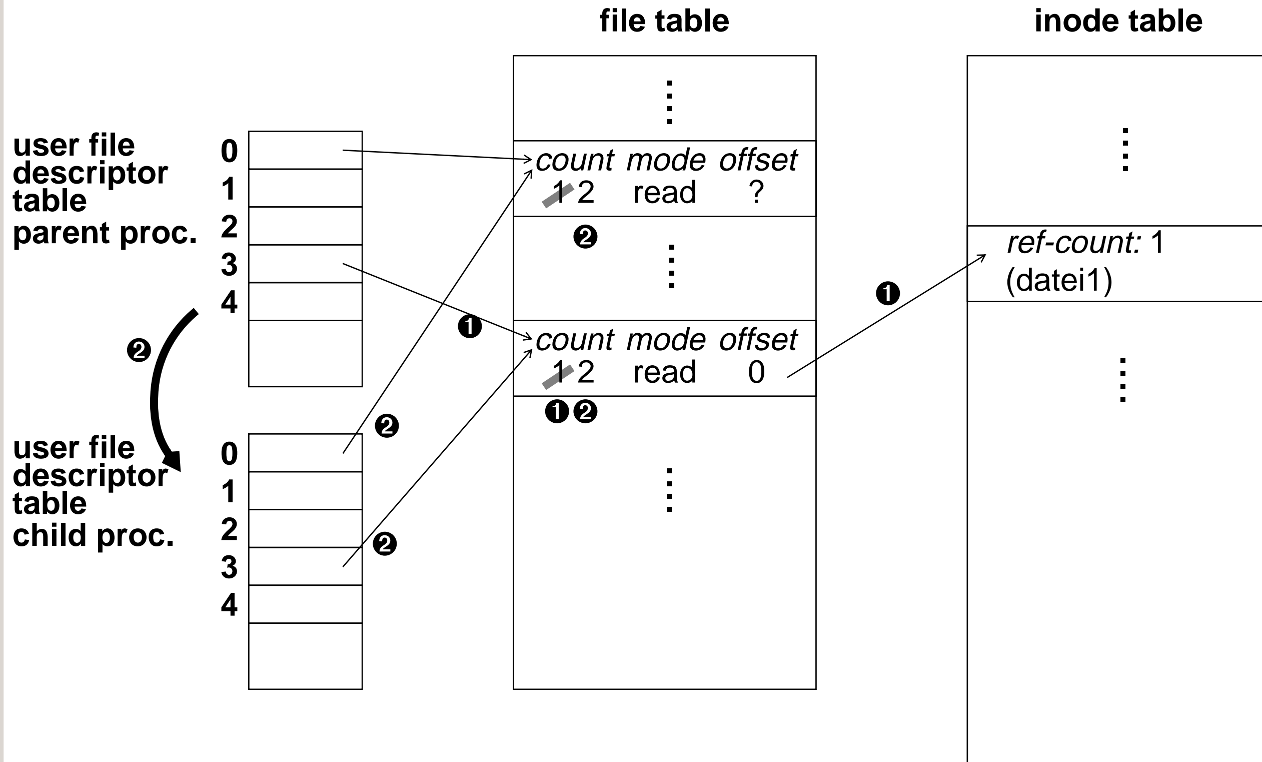


- Umleiten von stdin:

```
❶ fd1 = open("datei", O_RDONLY);
❷ close(0);
❸ dup(fd1);
❹ close(fd1);
```


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.




```
❶ fd1 = open("datei", O_RDONLY);
❷ fork( );
```


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
J-Filesystem.doc 1999-02-24 18.02
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

J.4 Vnodes

1 Motivation

- Inode = konkrete Implementierung eines Dateiverwaltungsblocks eines speziellen Dateisystems

- Problem: unterschiedliche Dateisysteme in einem UNIX-System
  - S5 – SystemV-File-System
  - UFS - UNIX-File-System
  - NFS - Network File-System
  - specFS - special File-System (Gerätedateien, FIFO-Dateien)

➥ unterschiedliche Implementierung (Inode, *lookup*-Funktion, …)

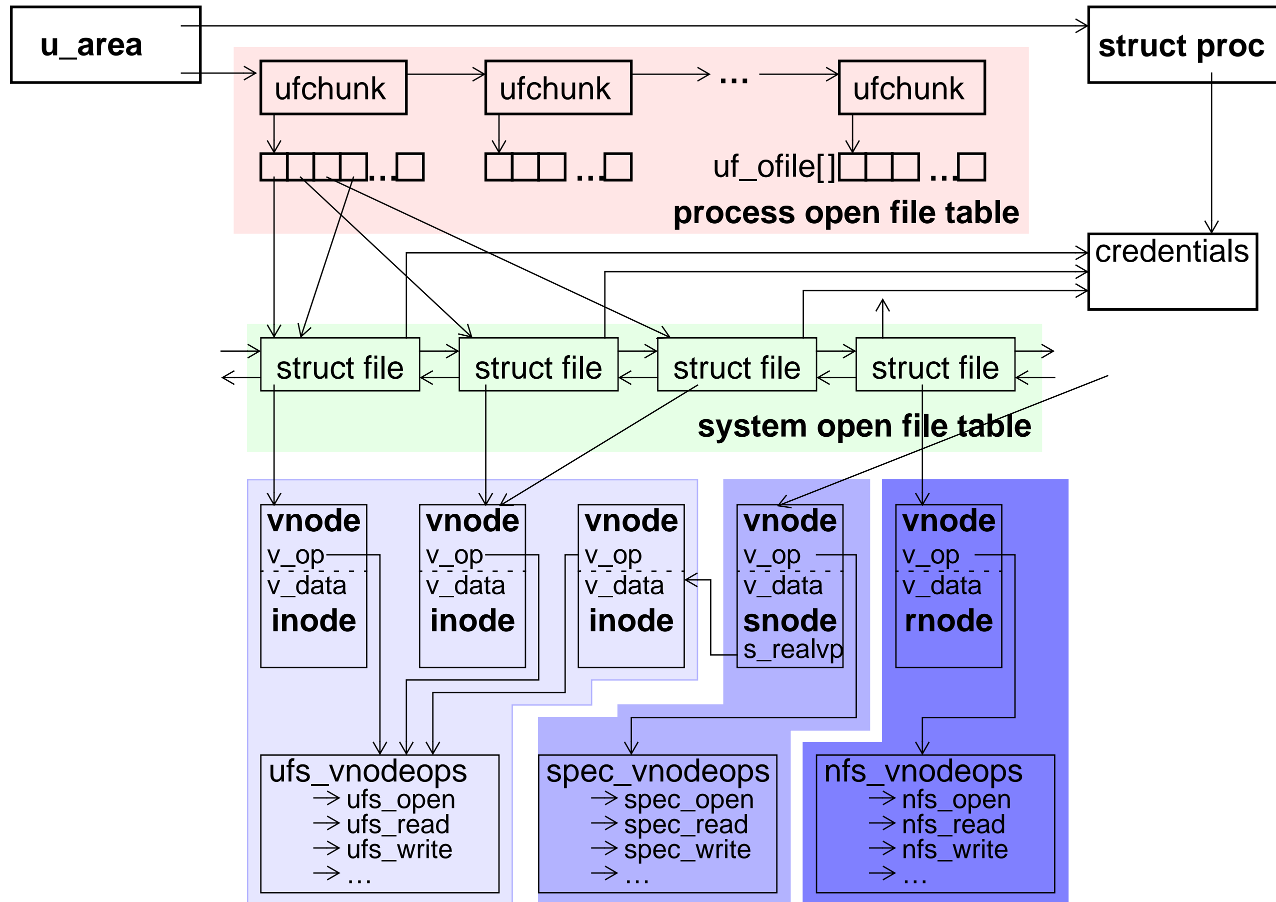
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Motivation (2)

- Lösung: **Vnode** = abstrakte Darstellung einer Datei
  - ➥ definiert Operationen auf Dateien
    - ➤ *open, close, read/write, lookup*, …
  - ➥ + allgemeine Zustandsinformationen
    - ➤ Dateityp (*regular file/directory*)
    - ➤ Referenzzähler
    - ➤ Verwaltungsinformationen für *locking* etc.
    - ➤ Zeiger auf konkrete Operationen und konkreten Zustand
- konkrete Darstellung einer Datei
  - ➥ Implementierung der Operationen
    - ➤ *ufs_open*, *ufs_close*, *ufs_read, ufs_lookup*, …
  - ➥ + konkreter Zustand
    - ➤ *in-core Inode*

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Kontrollstrukturen (SystemV.4)

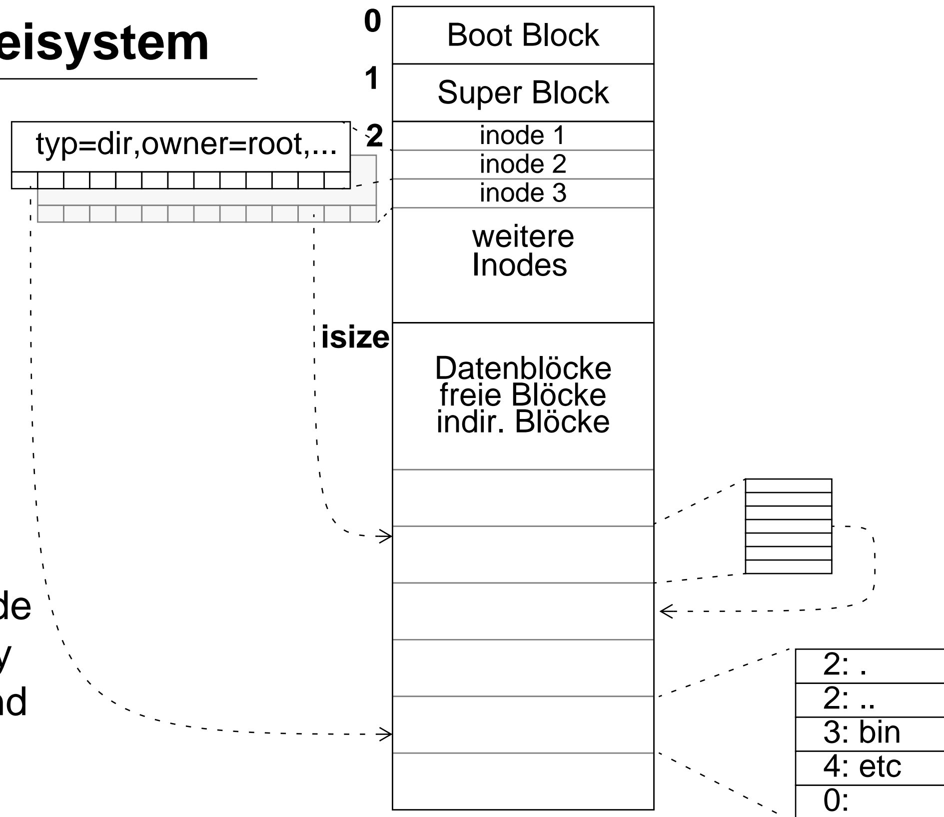


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

J.5 Dateisysteme

Aufbau eines Dateisystems auf einer Disk

1 SystemV-Dateisystem



- Blockgröße 512 oder 1024 Byte

- große Unterschiede zwischen Berkeley fast-file-system und UNIX SystemV-Dateisystemen!

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 SystemV-Dateisystem (2)

Super-Block

- Der *Super-Block* enthält die Verwaltungsinformationen für ein Dateisystem
  - Größe
  - Anzahl der freien Blöcke
  - Liste freier Blöcke *(free block list)*
  - Zeiger auf nächsten freien Block in der *free block list*
  - Größe der Inode-Liste
  - Anzahl freier Inodes
  - Liste freier Inodes *(free inode list)*
  - Zeiger auf ersten freien Inode
  - Lock-Felder für *free block* und *free inode list*
  - *modified flag* für Super-Block

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Berkeley Fast File System (UNIX File System — UFS)

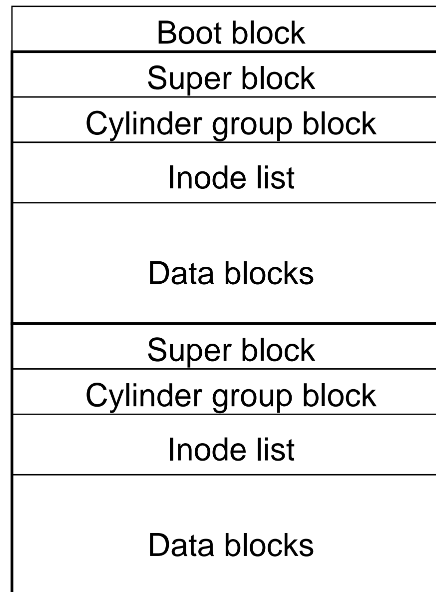
- für UNIX 4.2bsd entwickelt - effizienter als SystemV-Dateisysteme
- Blockgröße minimal 4k, meist 8k
  - ⇒ über zweifach indirekte Blöcke können bis zu 64 GB adressiert werden
- Directories anders aufgebaut - Dateinamen bis zu 255 Zeichen lang
- Dateisystem in ***cylinder groups*** aufgeteilt (*cylinder group* = Gruppe von aufeinanderfolgenden Zylindern auf Platte, häufig 16)
- pro *cylinder group* werden folgende Daten gehalten:
  - ◆ Backup-Kopie des Superblocks
  - ◆ Bitmap für freie Blöcke (statt *free list*)
  - ◆ Inodes
  - ◆ Datenblöcke

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Berkeley Fast File System (2)

- Um Verschnitt gering zu halten, können Blöcke in 2, 4 oder 8 ***Fragmente*** aufgeteilt werden, die einzeln adressierbar sind

- Eine Datei wird - wenn möglich - in einer *cylinder group* gehalten



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

J.6 Virtuelles Dateisystem

1 Überblick

- Vnodes bilden Schnittstelle zu dateisystemunabhängigen Operationen auf Dateien
- das *Virtual File System* (**VFS**) stellt eine vergleichbare Schnittstelle für Operationen auf Dateisystemen zur Verfügung — z. B.
  - ***vfs_mount***, ***vfs_unmount***
  - ***vfs_statvfs*** Statistikdaten ermitteln
  - ***vfs_root*** root-vnode ermitteln
  - ***vfs_sync*** gepufferte Daten schreiben
- Implementierung durch VFS-switch-Tabelle
  - Dateisystemname, Initialisierungsfunktion,
    vfsops — Feld mit Zeigern auf die FS-Funktionen

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
J-Filesystem.doc 1999-02-24 18.02
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Mount-Mechanismus

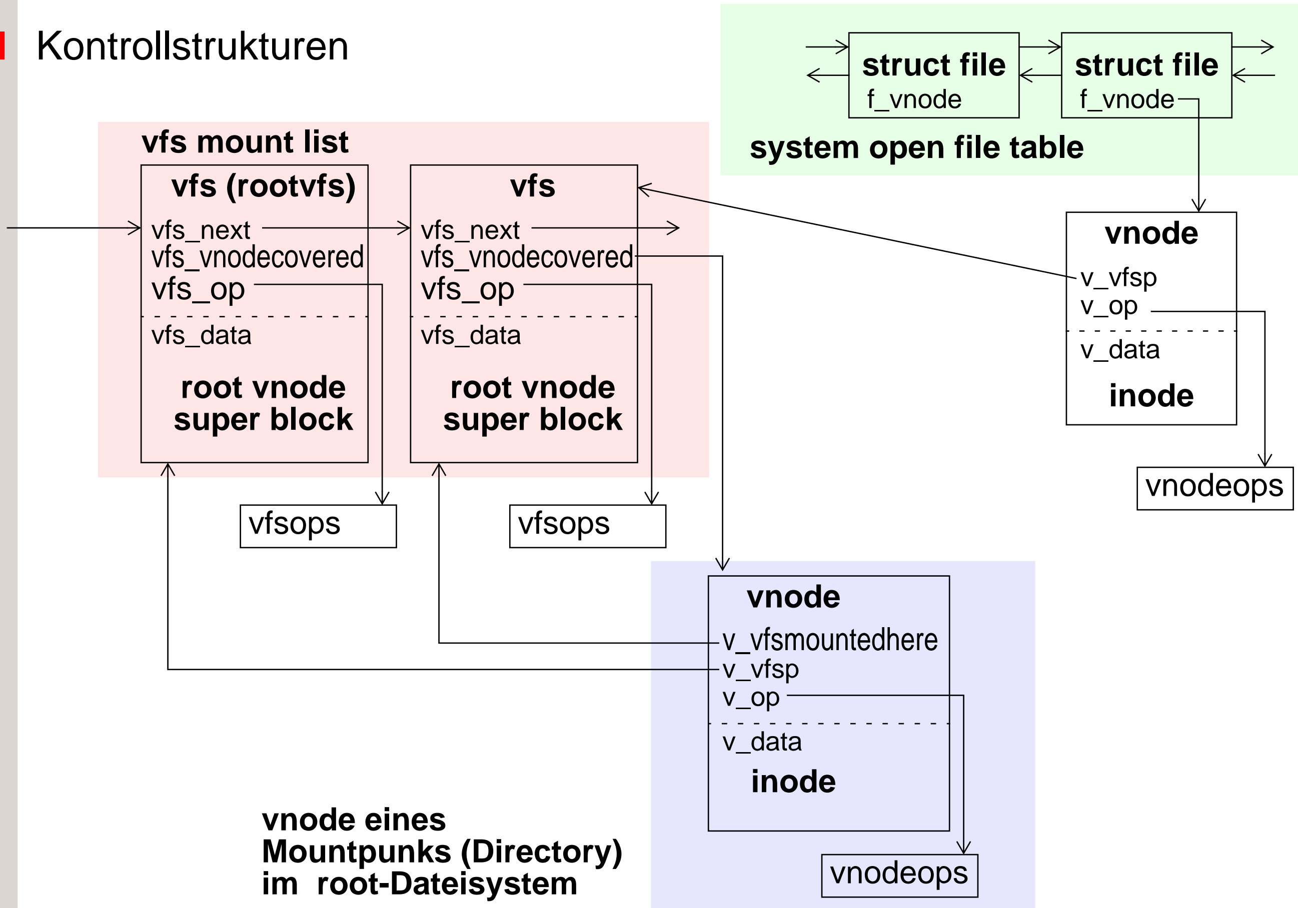
- über die mount-Funktion wird der root-Vnode eines Dateisystems auf einen Directory-Vnode des bestehenden Dateibaums montiert
  - Systemkern verwaltet die montierten Dateisysteme in verketteten *vfs*-Strukturen
  - über die vfs-Struktur kann ermittelt werden:
    - auf welchen Vnode das Dateisystem montiert ist
    - der root-Vnode des Dateisystems
  - Vnode des Mount-Punkts (Directory) enthält Eintrag mit Zeiger auf vfs-Datenstruktur des montierten Dateisystems

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Mount-Mechanismus (2)

- Kontrollstrukturen



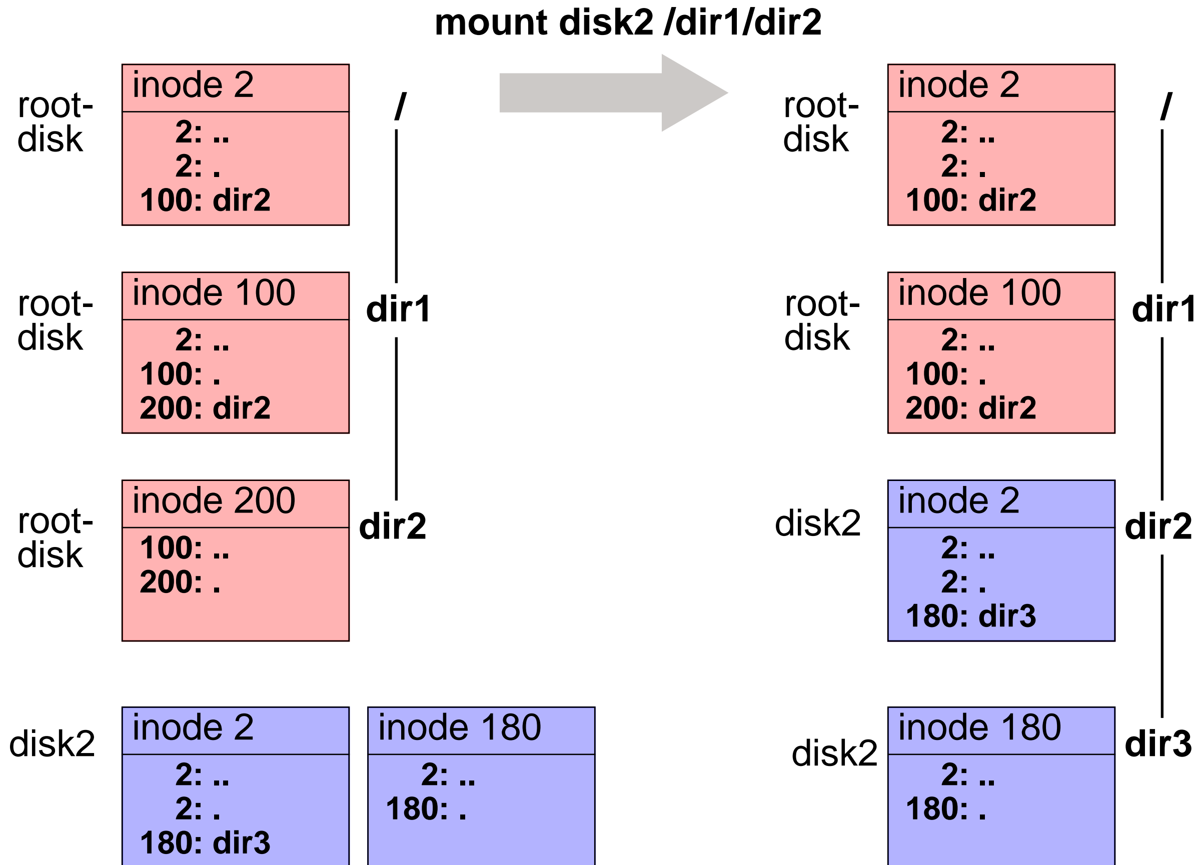
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Mount-Mechanismus (3)

- *lookup*-Funktion des UNIX-Kerns berücksichtigt bei Pfad-Vnode-Umwandlung Mount-Punkte
  - ◆ ist in einem vnode *vfs_mountedhere* gesetzt, wird der Vnode beim Abstieg im Pfad durch den *root-vnode* des *VFS* ersetzt
  - ◆ ein *root-vnode* eines *VFS* wird bei Verfolgung von ".." durch *vfs_vnodecovered* des *VFS* ersetzt
  - ◆ an Directory-Einträgen ändert sich nichts

  ➥ an Mount-Punkten Diskrepanz zwischen Directory-Einträgen und Ergebnis von ***stat(2)***

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

J-Filesystem.doc 1999-02-24 18.02

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

J.7 Network File System (NFS)

1 Überblick

- 1984 von Sun Microsystems Inc. entwickelt
- NFS ermöglicht, Teilbäume von Dateisystemen anderer Rechner in das lokale Dateisystem einzuhängen *(**mounten**)*
- ein Teilbaum eines *Remote*-Dateisystems verhält sich dadurch genau so, wie eine Partition einer lokalen Platte, die mit einem ***mount***-Befehl in den Dateibaum eingehängt wurde
- die Dateien des *Remote*-Dateisystems können mit den gleichen Kommandos und Systemaufrufen wie lokale Dateien bearbeitet werden

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Überblick (2)

- Abweichungen von der UNIX-Dateisemantik:
  - ➤ ***special files*** (Geräte-Dateien) sind nicht erreichbar
  - ➤ es werden nicht alle Systemdienste unterstützt (z. B. *File-locks*)
  - ➤ ***append***-Modus beim Öffnen einer Datei nicht möglich
  - ➤ wenn ein Dateisystem voll ist, wird dies nicht unmittelbar mitgeteilt

- auf der *Client*-Seite sind die NFS-Funktionen in den UNIX-Systemkern integriert **(virtuelles Dateisystem - *VFS*)**

- NFS-*Server* können, müssen aber nicht im Betriebssystem-Kern ablaufen

- Kommunikation zwischen *NFS-Client* und *-Server* erfolgt über ***SUN-RPC*** *(Remote Procedure Calls),* aufbauend auf ***UDP/IP*** oder (Solaris 2.5, 4.4bsd) ***TCP/IP***

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Überblick (3)

- *NFS-Server* sind zustandslos, d. h. sie halten keine Informationen über frühere Aufträge eines *Clients* (z. B. welche Dateien geöffnet, welche Position in einer Datei)
  → problemloser Neustart nach Systemabstürzen

- standardisierte Datendarstellung für Datenaustausch zwischen unterschiedlichen Rechnerarchitekturen:
  XDR *(e**X**ternal **D**ata **R**epresentation)*

- die Spezifikationen für NFS, XDR und RPC ist *public domain*, d. h. frei erhältlich - nicht jedoch die Implementierungen

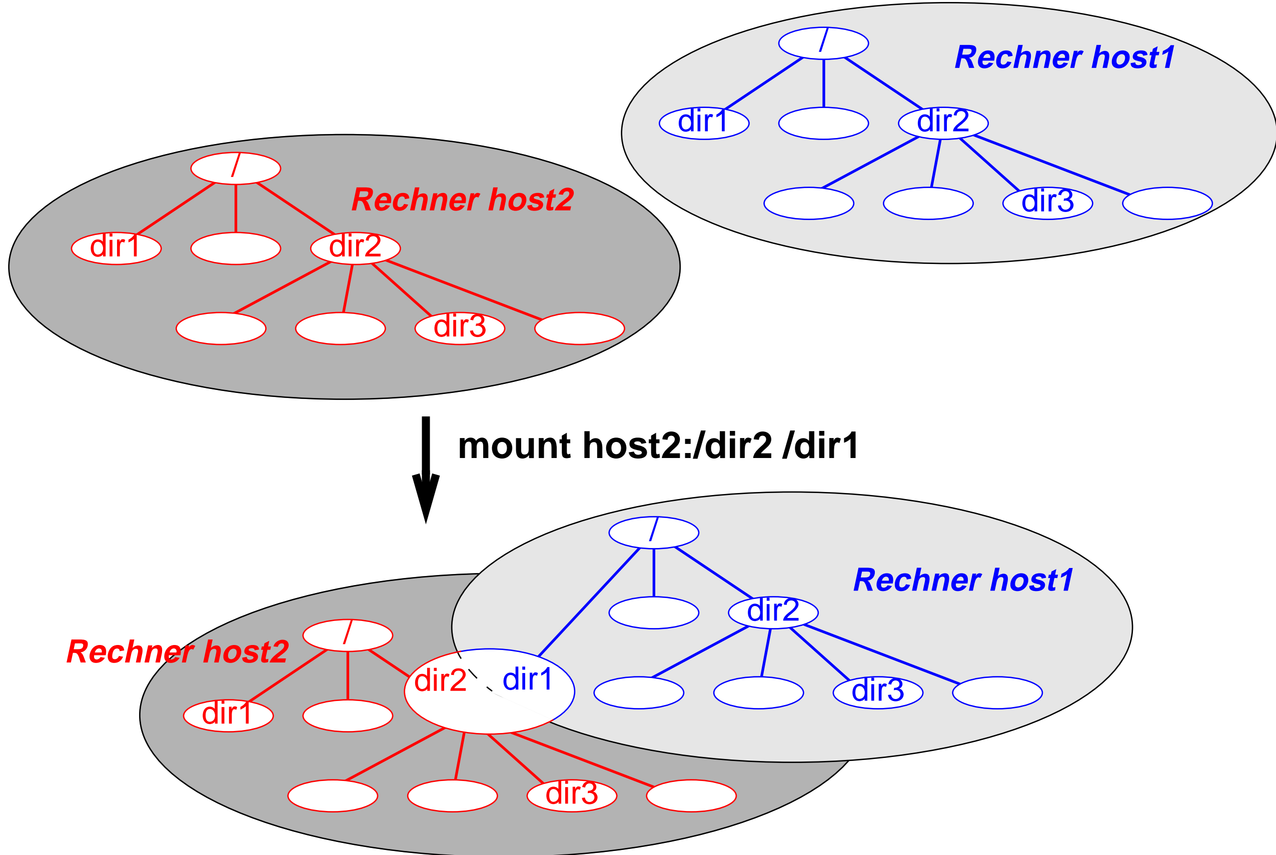
- Programme werden immer auf dem *aufrufenden* Rechner ausgeführt
  ➤ Betrieb von *diskless workstations* möglich

- Probleme bei unterschiedlichen Rechnertypen - unterschiedlich übersetzte Programme *(Binaries)* notwendig

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Beispiel



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Sicherheitsaspekte

- Dateisysteme müssen explizit exportiert werden um von anderen Rechnern *gemountet* werden zu können
  - ➤ der Systemverwalter des exportierenden Rechners kann festlegen, welche Rechner welches Dateisystem von ihm importieren dürfen
  - ➤ bei ***secure NFS*** kann darüber hinaus auch spezifiziert werden, welche Benutzer des anderen Rechners zugreifen dürfen
- die Benutzerabbildung erfolgt in UNIX über die *User-Id*
  - ➤ Beispiel: der Benutzer mit der *User-Id* 129 auf Rechner *host1* hat in einem Dateisystem, das von Rechner *host2* importiert ist, genau die Rechte, die der Benutzer mit *User-Id* 129 auf dem Rechner *host2* hat
- Der *Super-User* (*User-Id* 0) wird auf den Benutzer *nobody* abgebildet (die *Super-User*-Abbildung kann pro exportiertes Dateisystem/ Zielrechner auch explizit festgelegt werden)

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
J-Filesystem.doc 1999-02-24 18.02
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Implementierung

- NFS-Server erzeugen beim ***nfs_open()*** ein **file-handle** (Kombination aus Filesystem/Device-Id und Inode) und übergeben dies dem NFS-Client
  - es werden auf Server-Seite keine Informationen gehalten!
  - bei jedem Lese-/Schreib-Zugriff wird die Datei auf Basis der file-handle-Informationen neu geöffnet
- NFS-Client benutzt file-handle für alle weiteren Zugriffe auf die geöffnete Datei (Block lesen/schreiben)
  - file-handle wird im **rnode**-Teil des vnode gespeichert
  - Offset-Informationen sind in der file-Struktur vorhanden
  - vfs-Struktur enthält die Adresse des NFS-Servers

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Implementierung (2)

- Besonderheiten:
  - wird eine Datei auf Client-Seite gelöscht, obwohl noch Prozesse gültige Dateideskriptoren darauf halten:
    - wird sie auf der Serverseite nur umbenannt (nfs_...)
    - wird sie erst endgültig gelöscht, wenn die letzte Referenz auf den vnode auf der Client-Seite geschlossen wird

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Verwaltungsstrategien

- Pufferverwaltungssystem mit Verwaltungsalgorithmen, ähnlich denen in einem Paging System (nicht demand paging!) mit folgenden Eigenschaften:
  - **read ahead**
    wird ein Block gelesen, so werden auch der Transfer des folgenden Blocks angestoßen (wenn sequentiell gelesen wird)
  - **write after**
    wird ein Block geschrieben, bedeutet dies nicht den sofortigen physikalischen Transfer auf die Platte
  - **geschrieben** wird u. a. bei folgenden Ereignissen:
    - Datei wird geschlossen
    - keine freien Puffer mehr vorhanden
    - vom System *(clock)* wird ein ***update*** angestoßen
    - Systemaufruf ***sync(2)*** oder ***fsync(2)*** (BSD)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

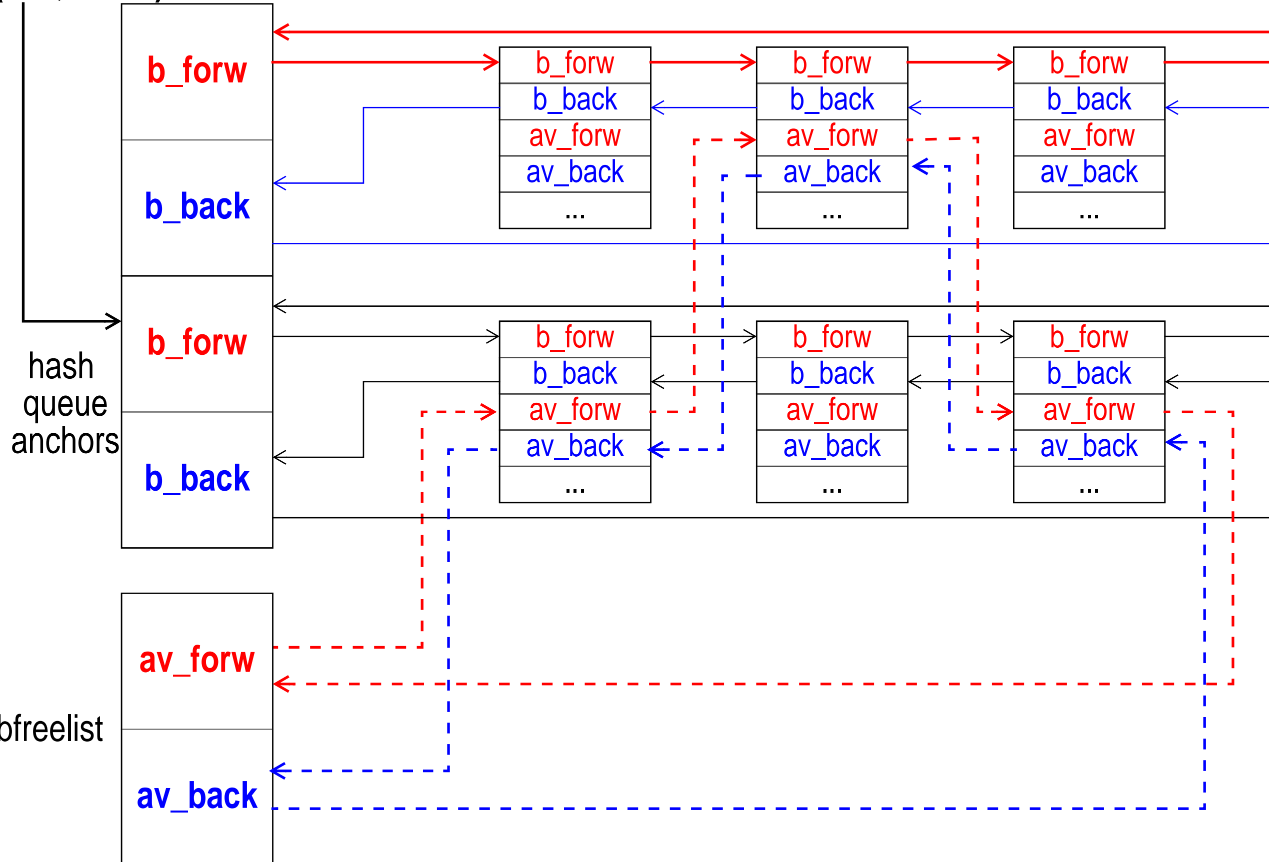
1 Verwaltungsstrategien (2)

- werden ausgelagerte Blöcke, deren Puffer noch nicht anderweitig benutzt wurde, erneut angesprochen, so werden sie reaktiviert ***(reclaim)***
- wenn zur Gewinnung von freien Blöcken eine Auslagerung angestoßen wird, werden die Kandidaten für die Freiliste nach dem LRU-Algorithmus ausgewählt

2 Integration von Buffer Cache und Speicherverwaltung

- Ab SystemV R4 werden die Puffer des Dateisystems mit dem normalen Paging-System verwaltet
  - Dateiblöcke werden in virt. Adreßraum des Kerns abgebildet
  - Daten werden dann einfach aus Kern in User-level Adreßraum kopiert
    - ➤ Page-fault bewirkt Einlagerung der Seite aus der Datei (über vnode-Operationen des jeweiligen Dateisystems)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



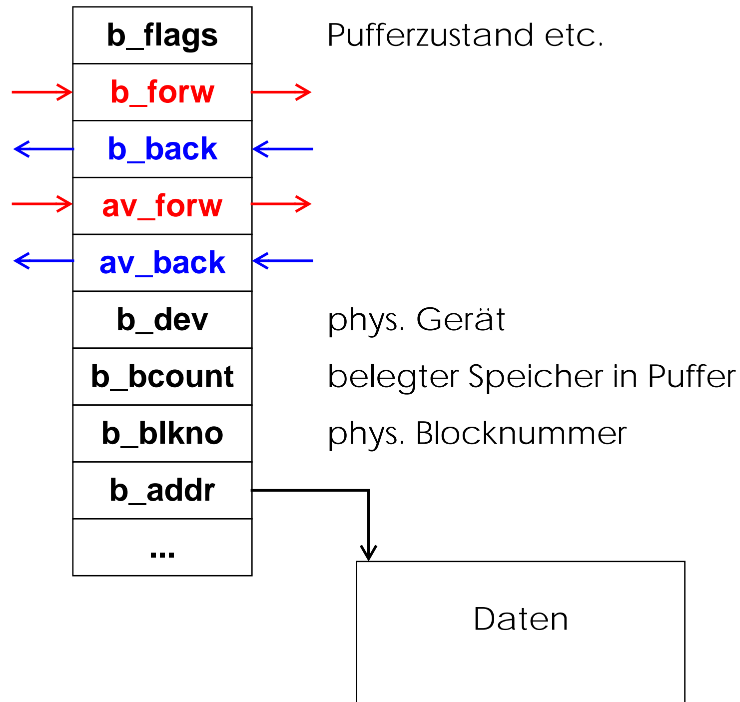
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

J-Filesystem.doc 1999-02-24 18.02

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Kontrollstrukturen

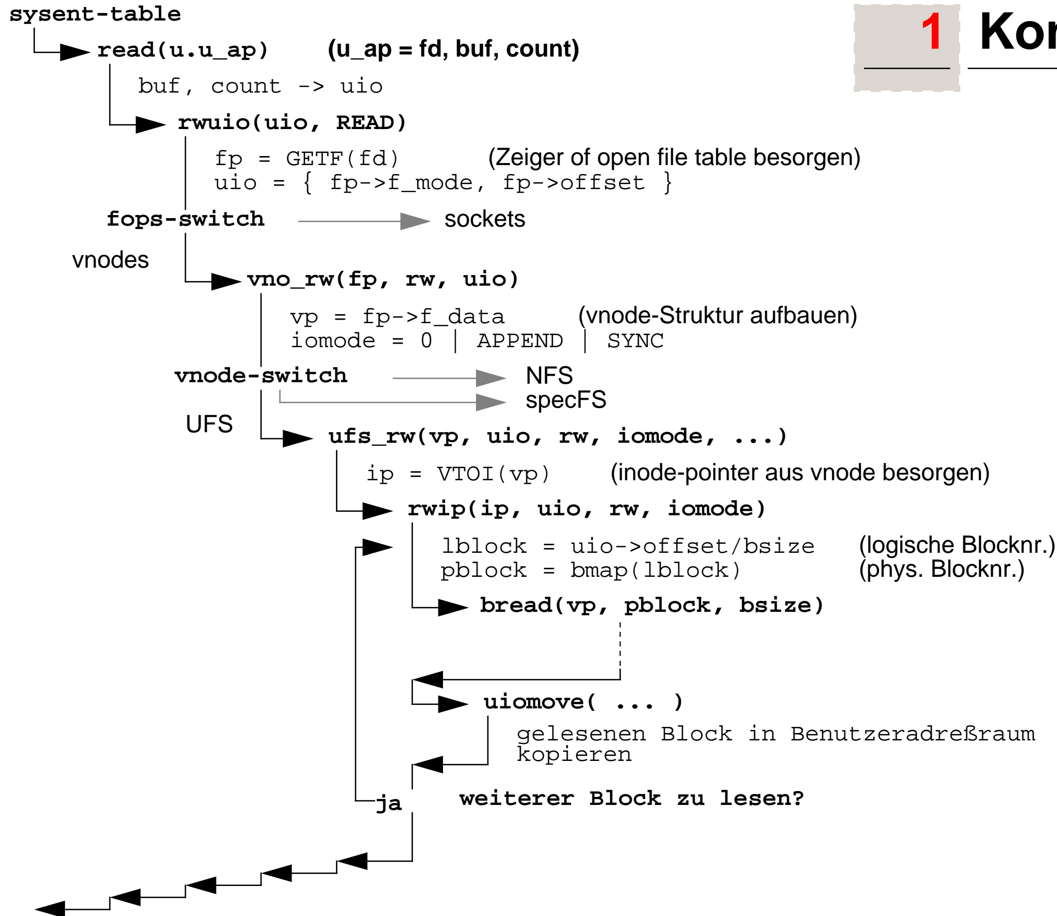
- in BSD4.4 sind die Puffer zusätzlich an den *vnodes* verankert
- Buffer-Header (struct buf, siehe *<sys/buf.h>)*



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

J.9 Dateisystem - Buffer Cache

1 Kontrollfluß



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

J.10 Treiber

1 Überblick

- Jede Gerätedatei entspricht einem physischen oder virtuellen Gerät
- bei den physischen Geräten werden zwei Ein-Ausgabesysteme unterschieden:
  - zeichenorientierte E/A (Terminals, Drucker)
  - blockorientierte E/A (vor allem Platten)
- für jedes Ein-Ausgabesystem existiert ein:
  - Puffersystem
  - Treiberfunktionsvektor, *cdevsw* und *bdevsw*

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Überblick (2)

- Für jeden Gerätetyp ist in mindestens einem dieser Vektoren eine Zeile mit den gerätespezifischen Operationen reserviert

- Jedes Gerät ist mit dem Tripel (E/A-Typ, *major number*, *minor number*) eindeutig beschrieben
  - E/A-Typ wählt Funktionsvektor aus
  - *major number* wählt die zugehörige Zeile im Funktionsvektor aus
  - *minor number* dient im gerätespezifischen Kontrollblock des Treibers als Index zur Identifikation des Geräts

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Treiber-Funktionsvektoren

- *block-device-switch*:

```
struct bdevsw {
        int         (*d_open)();
        int         (*d_close)();
        int         (*d_strategy)();
        int         (*d_dump)();
        int         (*d_psize)();
        int         d_flags;
};
```

- ◆ *read*- und *write*-Aufrufe werden über den Block-Puffer abgewickelt
- ◆ die *strategy*-Funktion regelt die E/A-Operation zwischen Block-Puffer und Gerät
- ◆ *major*- und *minor number* werden im Inode gespeichert

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Treiber-Funktionsvektoren (2)

- *character -device-switch*:

```
struct cdevsw {
        int      (*d_open)();
        int      (*d_close)();
        int      (*d_read)();
        int      (*d_write)();
        int      (*d_ioctl)();
        int      (*d_reset)();
        int      (*d_select)();
        int      (*d_mmap)();
        struct   streamtab *d_str;
};
```

◆ im Falle von *Streams*-Treibern sind alle Komponenten leer - nur die *streamtab*-Komponente enthält einen Verweis auf die Struktur der *Streams*-Verwaltung

- Block-orientierte Geräte (z. B. Platten) haben in der Regel einen zweiten Eintrag als zeichen-orientiertes Gerät um direkte Zugriffe unter Umgehung der Block-Puffer zu ermöglichen

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
J-Filesystem.doc 1999-02-24 18.02
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Treiber-Funktionsvektoren (3)

- der Systemaufruf *mknod* legt im Dateisystem Knoten für die Gerätedateien an

- Beispiel aus /dev

```
% ls -l /dev
crw--w--w-  1 root       0,    0 Feb  5 12:38 console
crw-r-----  1 root       3,    1 Dec  2 11:48 kmem
crw-rw-rw-  1 root      13,    0 Sep 15  1988 mouse
crwx-w----  1 eckert    20,    5 Feb  5 12:07 ttyp5
crw-rw-rw-  1 root      21,    5 Feb  5 12:06 ptyp5
brw-r-----  1 root       7,    1 Sep 15  1988 sd0b
crw-r-----  1 root      17,    1 Sep 15  1988 rsd0b
brw-r-----  1 root       3,    0 Dec  2  1988 xy0a
crw-r-----  1 root       9,    0 Jan 10 10:34 rxy0a
crw--w--w-  1 root      12,    0 Feb  5 12:11 ttya
crw--w--w-  1 root      12,    1 Jan 11 09:58 ttyb
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Treiber-Funktionsvektoren (4)

- Auszug aus der Konfigurationsdatei des Systemkerns

```
struct bdevsw   bdevsw[] =
{
  { nodev,    nodev,    nodev,       nodev,      /*0*/
    0,        0},                             /* was ip */
  { tmopen,   tmclose,  tmstrategy, tmdump,      /*1*/
    0,        B_TAPE },
  { nodev,    nodev,    nodev,       nodev,      /*2*/
    0,        B_TAPE},                        /* was ar */
  { xyopen,   nulldev,  xystrategy, xydump,      /*3*/
    xysize,   0 },
  ...
  { sdopen,   nulldev,  sdstrategy, sddump,      /*7*/
    sdsize,   0 },
  ...
}
```

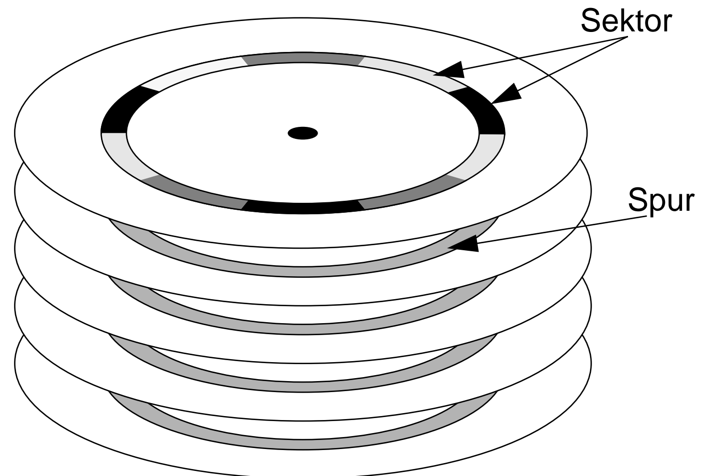

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Organisation einer Disk - Laufwerk

- typische Winchester-Festplatten bestehen aus mehreren, übereinander angeordneten Magnetplatten und einem Schreib-/Lese-Kopf pro Plattenoberfläche

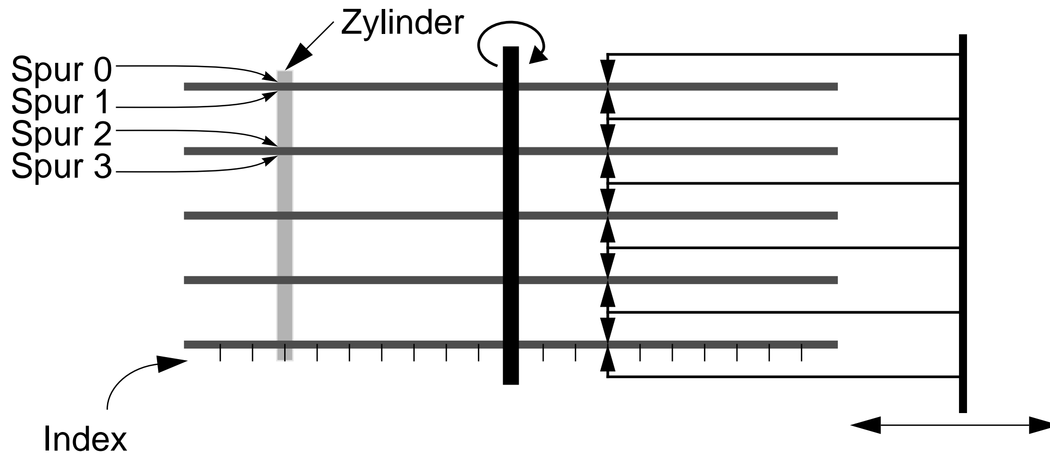
- Begriffe:

Spur *(Track)*	Umlaufbahn auf einer Diskoberfläche
Sektor	Teil einer Spur
Zylinder	Menge der Spuren mit gleichem Radius über alle Oberflächen



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Organisation einer Disk - Laufwerk (2)



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Organisation einer Disk - Laufwerk (3)

- Beispiele realer Disks:

	Fujitsu			*Seagate*	
	M2344K	*M2652*	*Elite9*	*Elite47*	*Cheetah18*
	1987	1990	1994	1998	1998
	9"	9"	5,25"	5,25"	3,5"
Bussystem:	SMD	IPI	SCSI-2	Ultra SCSI	Fibre
Kapazität:	690 MB	2,0GB	10,8GB	47 GB	18GB
Zylinder:	624	1893	4925	9996	6962
Spuren/Zyl.:	27	20	27	28	24
Bytes/Spur:	40960	52864	variabel		
Sektoren/Spur:	1 - 128 (konfigurierbar)				
Positionierzeit:					
(Spur-Spur)	4 ms	2 ms		1,1 ms	0,6 ms
(mittel)	16 ms	11 ms	11 ms	13 ms	5,7 ms
(max)	33 ms	22 ms		28 ms	12 ms
Transfer MB/s:	2,458	4,75	6 - 7	40	200
Umdr./Min.:	3600	5400	5400	5357	10025

➥ was bedeutet das konkret (z.B. bei Elite9)?
- – eine Umdrehung = 11 ms
- – bei DEC Alpha (300 MHz, 2 Instr./Takt = 600 Mio/Sek.)
  ca. 6,6 Mio. Befehle ausführbar

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Organisation einer Disk (SCSI)

- Platte = Sequenz von Datenblöcken (perfect media)
- lokaler Pufferspeicher (z. B. 1-4 MB) für Optimierungsstrategien (da Bus schneller als Medium-Transfer)

3 Organisation einer Disk — Betriebssystem

- UNIX kann eine Disk in **Partitionen** unterteilen
  - ➥ auf jeder Partition kann ein UNIX-Dateisystem angelegt werden
- jedes UNIX-Dateisystem enthält eigene, von anderen Partitionen unabhängige Inodes
- aus diesen Inodes wird auf jedem UNIX-Dateisystem über Directories ein eigener, hierarchisch aufgebauter Dateibaum erzeugt
- mehrere solche Bäume können durch ***mounten*** zu einem Dateibaum zusammengefaßt werden

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Organisation einer Disk — BS (2)

- die Position der einzelnen Partitionen wird festgelegt:
  – in älteren UNIX-Systemen statisch im Kern
  – in neueren UNIX-Systemen in Tabelle (*VTOC - Volume Table of Contents*) am Anfang der Platte (z. B. Zyl. 0, Spur 0, Sekt. 0)
- Partitionierung am Beispiel Solaris

```
Current partition table (original):
Total disk cylinders available: 2733 + 2 (reserved cylinders)

Part       Tag  Flag     Cylinders      Size            Blocks
 0        root   wm       0 -  101     75.70MB   (102/0/0)   155040
 1        swap   wu     102 -  446    256.05MB   (345/0/0)   524400
 2      backup   wm       0 - 2732      1.98GB   (2733/0/0) 4154160
 3  unassigned   wm       0             0        (0/0/0)          0
 4  unassigned   wm       0             0        (0/0/0)          0
 5  unassigned   wm       0             0        (0/0/0)          0
 6         usr   wm     447 - 2732      1.66GB   (2286/0/0) 3474720
 7  unassigned   wm       0             0        (0/0/0)          0
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Organisation einer Disk — BS (3)

- den einzelnen Partitionen sind jeweils *block special files* **(/dev/dsk)** und *character special files* **(/dev/rdsk)** zugeordnet
  - ➥ über diese Dateien sind *major-number* (=Disk-Typ) und *minor-number* (=Partition) zur Interaktion mit dem Treiber feststellbar
  - ➥ die *block special files* sind in *mount*-Kommandos verwendbar
  - ➥ über die *character special files* ist ungepufferter Zugriff auf eine Partition möglich (z. B. für file system check)
  - ➥ Beispiel Solaris:

```
% ls -lL /dev/dsk
brw-r-----   1 root       32,    0 Oct 23 17:21 c0t0d0s0
brw-r-----   1 root       32,    1 Oct 23 17:21 c0t0d0s1
brw-r-----   1 root       32,    2 Oct 23 17:21 c0t0d0s2
...
brw-r-----   1 root       32,    6 Oct 23 17:21 c0t0d0s6
brw-r-----   1 root       32,    7 Oct 23 17:21 c0t0d0s7

% fsck /dev/rdsk/c0t0d0s6
% mount /dev/dsk/c0t0d0s6 /usr
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Disk-Zugriffsstrategien

- die Verwaltungsstrategien des *Buffer Cache* haben u. a. zwei Auswirkungen auf den Platten-Treiber
  - weniger Plattenzugriffe bei häufigen Operationen auf einer Datei
  - Anforderungen an den Plattentreiber kommen nicht einzeln, sondern immer in Gruppen (z. B. wenn Auslagerung angestoßen wurde)
- der Plattendurchsatz kann gesteigert werden, wenn die Anforderungen geeignet sortiert werden
- Problem bei modernen Disks:
  - ➤ Disk hat internen Pufferspeicher und fährt eine eigene Strategie für Zugriffe auf das Medium

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Disk-Zugriffsstrategien (2)

- mögliche Strategien:
  - **FIFO**
    - + Aufträge werden fair behandelt
    - – Plattenzugriffe nicht optimal, da viele Positionierungen erforderlich
  - **SATF** *(shortest access time first)*
    - + optimiert die Positionierungen des Plattenarms
    - – bei vielen Aufträgen für einen begrenzten Bereich können aber Aufträge für entferntere Positionen ausgehungert werden
  - **SCAN**
    - + Aufträge werden nach Zylinderpositionen sortiert, so daß der Plattenarm der Reihe nach über alle Positionen bewegt wird
    - – viele Aufträge für einen kleinen Bereich benachteiligen entfernte Aufträge
  - gemischte Strategien

    z. B. SATF über die ersten 8 Aufträge → Mischung aus SATF und FIFO

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
J-Filesystem.doc 1999-02-24 18.02
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

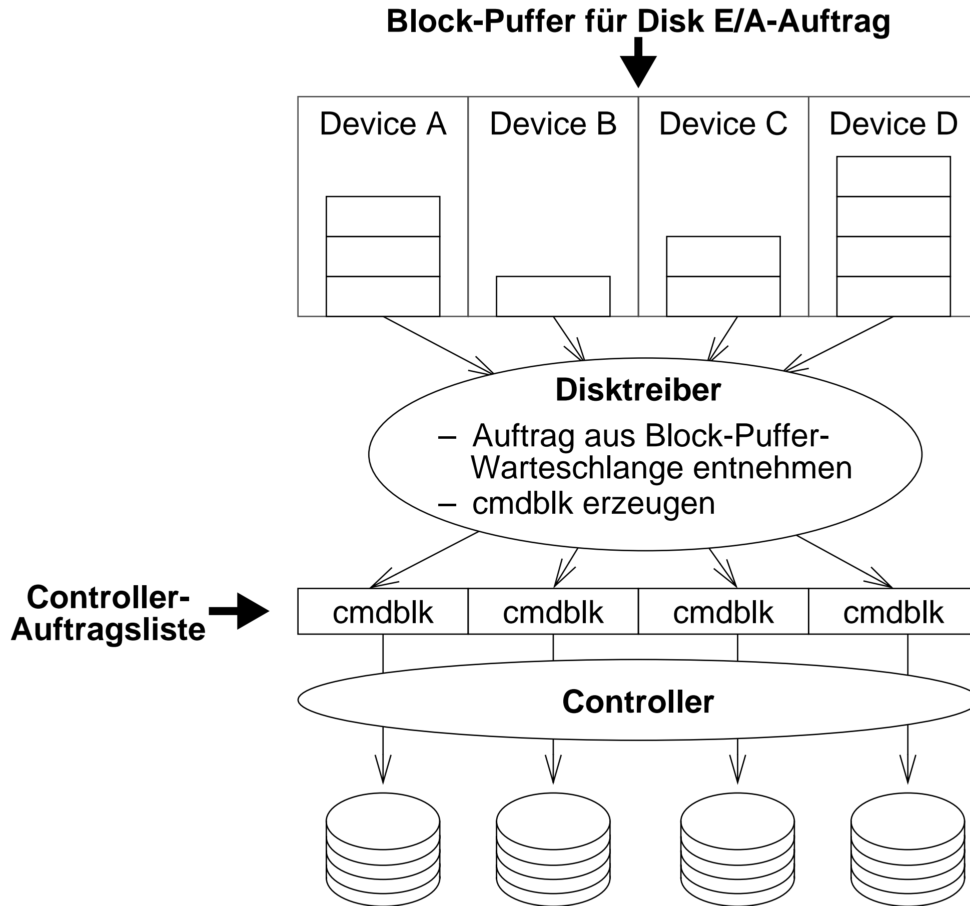
J.12 Disk-Treiber: Datenstrukturen und Kontrollfluß

1 Überblick

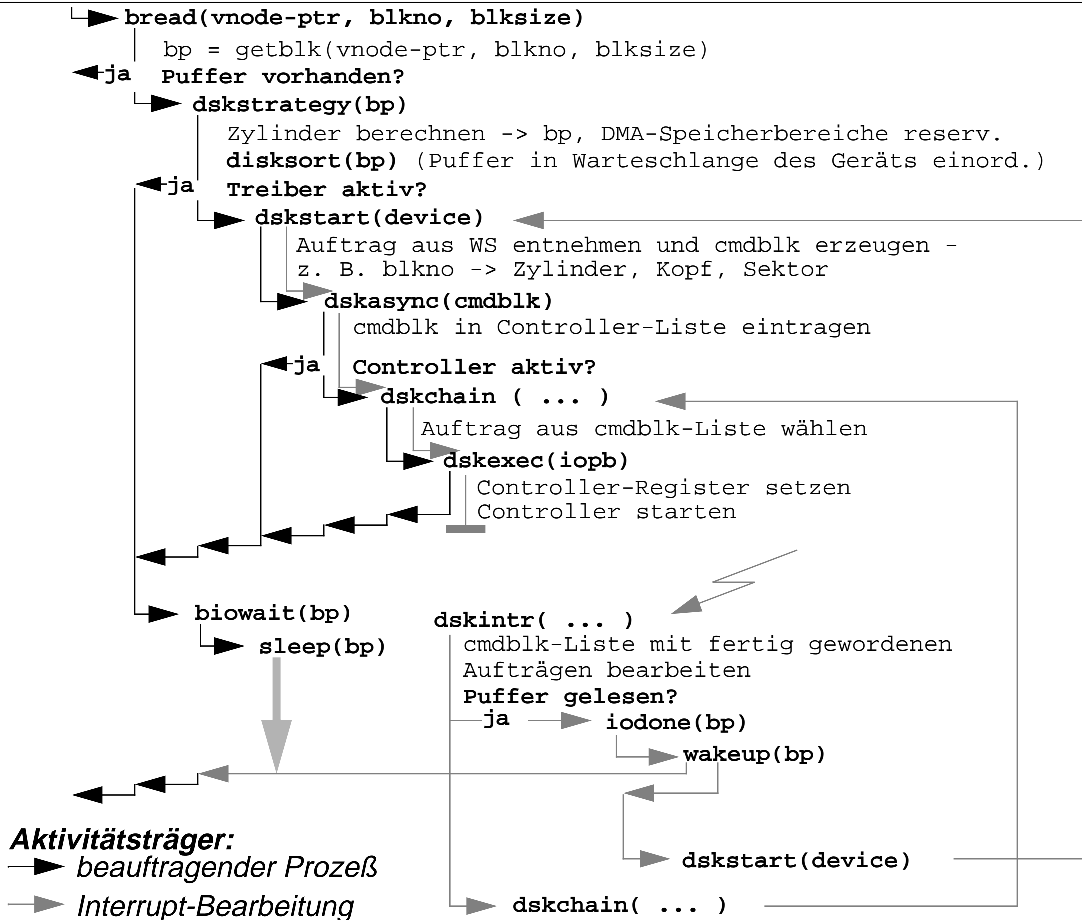
- Jedem Gerät (Platte) ist eine Warteschlange für Puffer des Buffer-Caches zugeordnet
- Puffer für Aufträge an den Disktreiber werden in die Warteschlange des zugehörigen Geräts einsortiert (*disksort( )*)
- Disktreiber entnimmt für freie Geräte Aufträge und startet den Controller
- bearbeitete Aufträge werden durch Interrupt gemeldet
  - ➥ Interrupt-Bearbeitung
    - – eigener Aktivitätsträger
    - – unabhängig von den beauftragenden Prozessen!
      - weckt wartende Prozesse
      - startet nächsten Auftrag aus Warteschlange

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
J-Filesystem.doc 1999-02-24 18.02
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Datenstrukturen



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

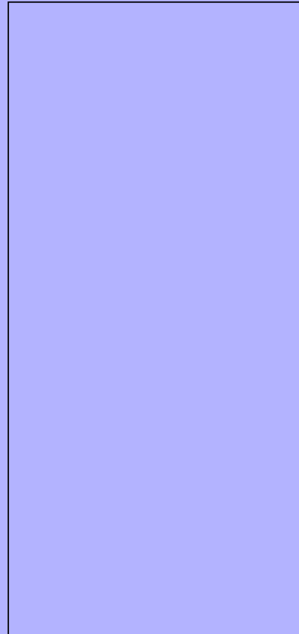
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Datei öffnen (3)

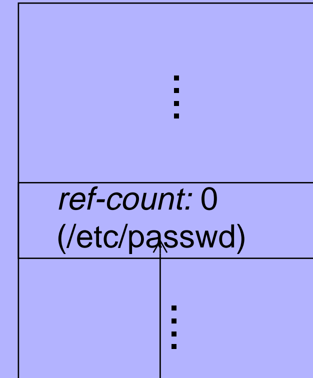
fd1
fd2
fd3

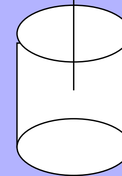
user-level Daten Proc. A

file table



inode table





```
fd1 = open("/etc/passwd", O_RDONLY);
```

globale Kern-Daten

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Datei öffnen (3)

file table

count mode offset
0 read 0

inode table

ref-count: 1
(/etc/passwd)

fd1
fd2
fd3

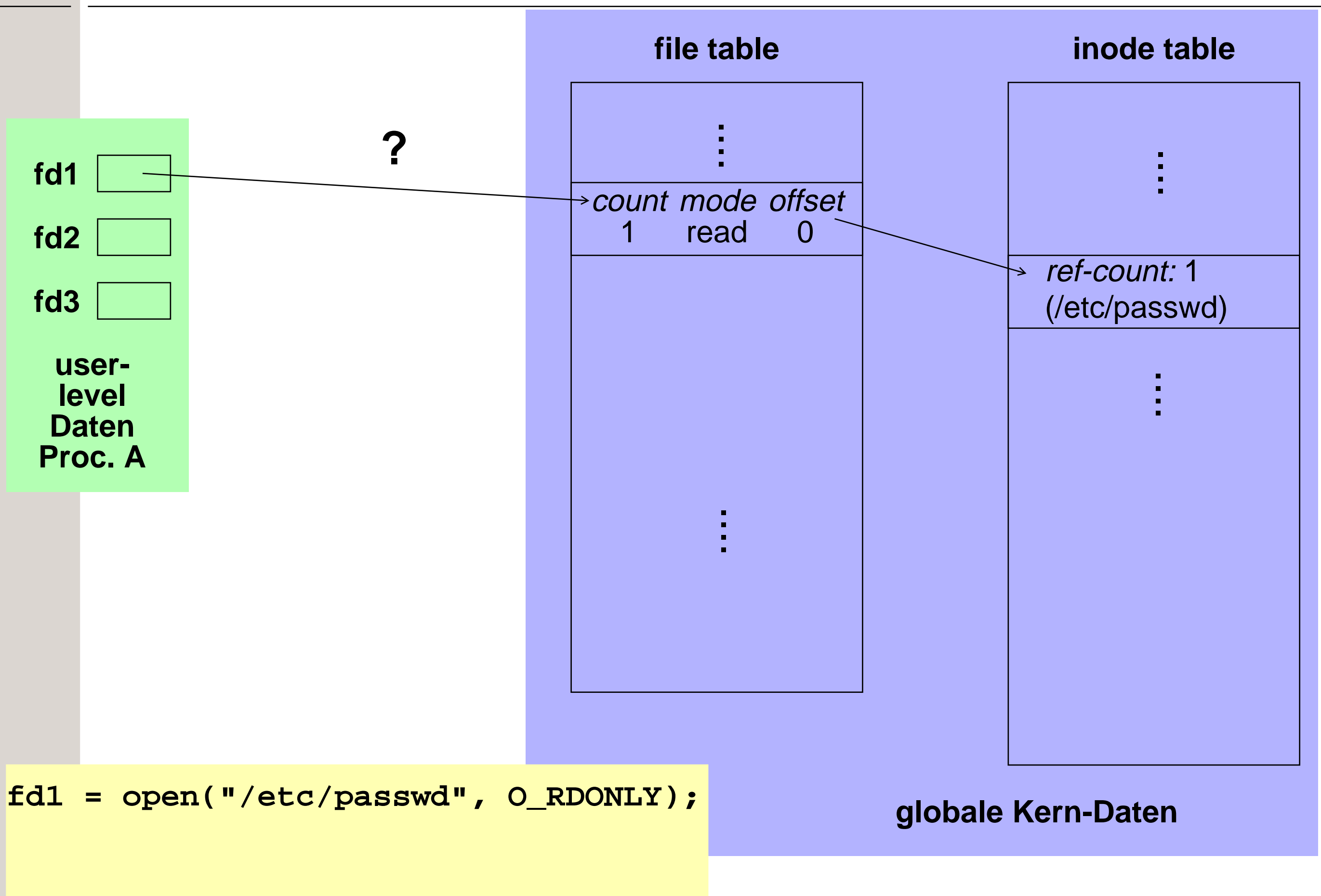
user-level Daten Proc. A

```
fd1 = open("/etc/passwd", O_RDONLY);
```

globale Kern-Daten

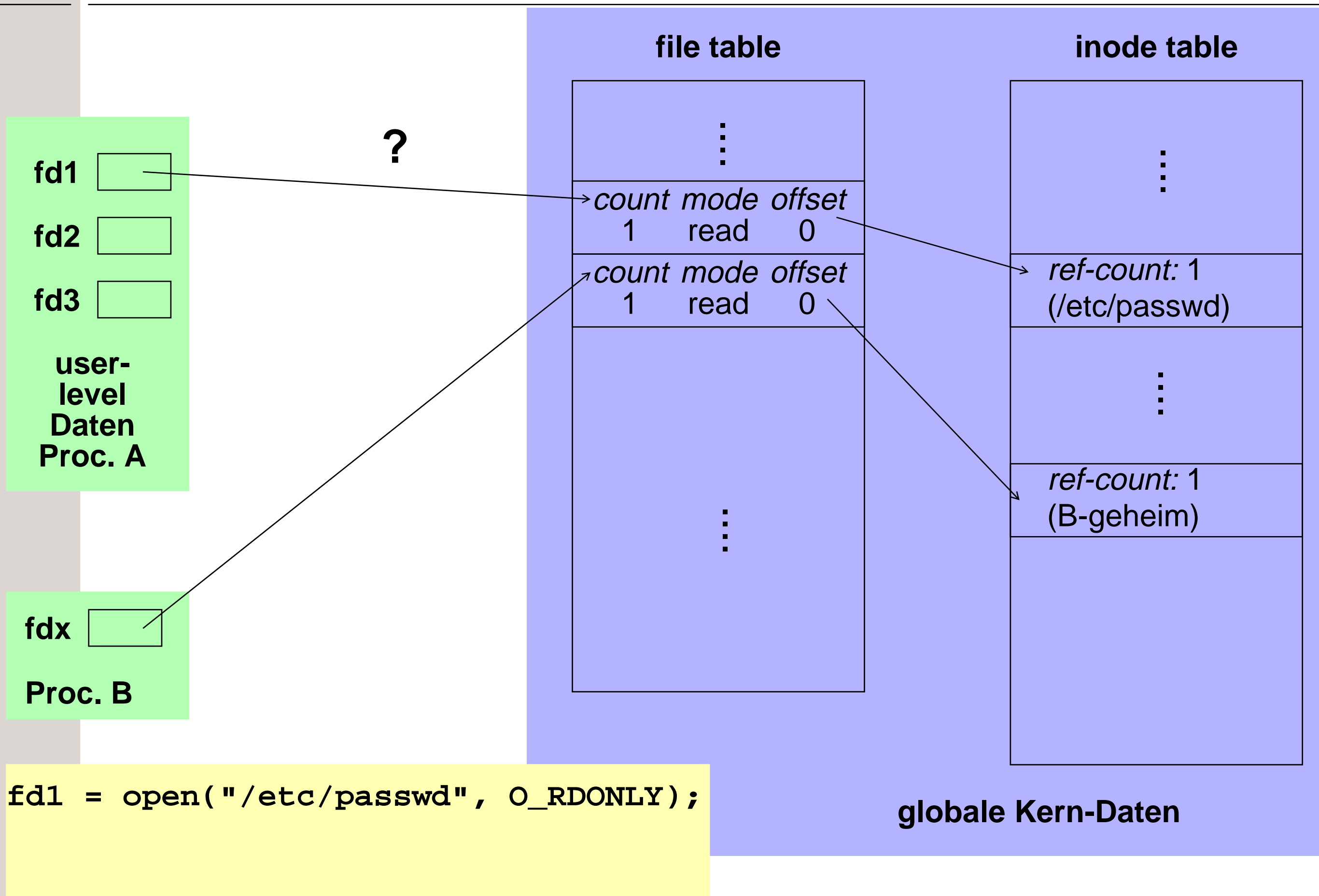
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Datei öffnen (3)



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

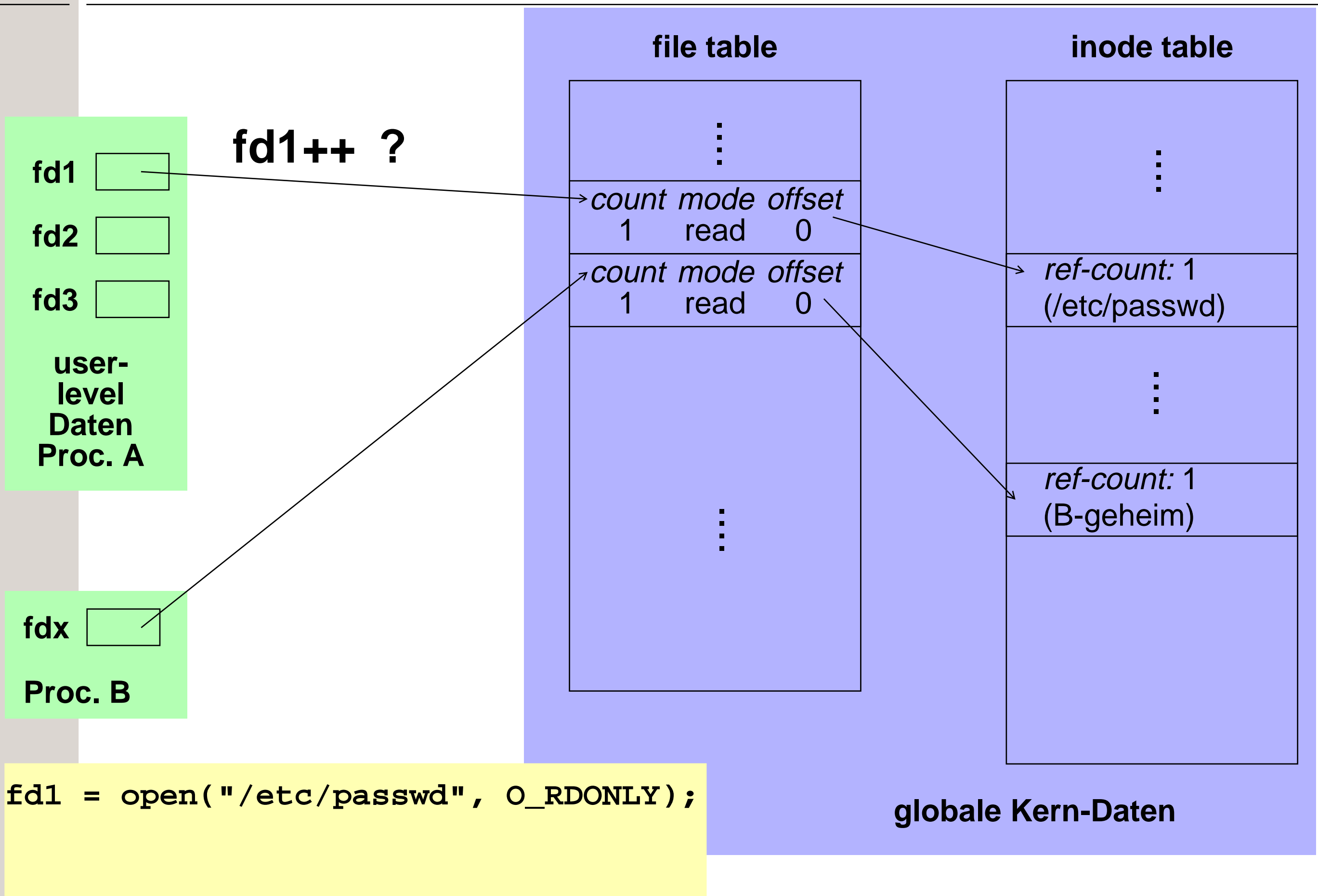
2 Datei öffnen (3)




```
fd1 = open("/etc/passwd", O_RDONLY);
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

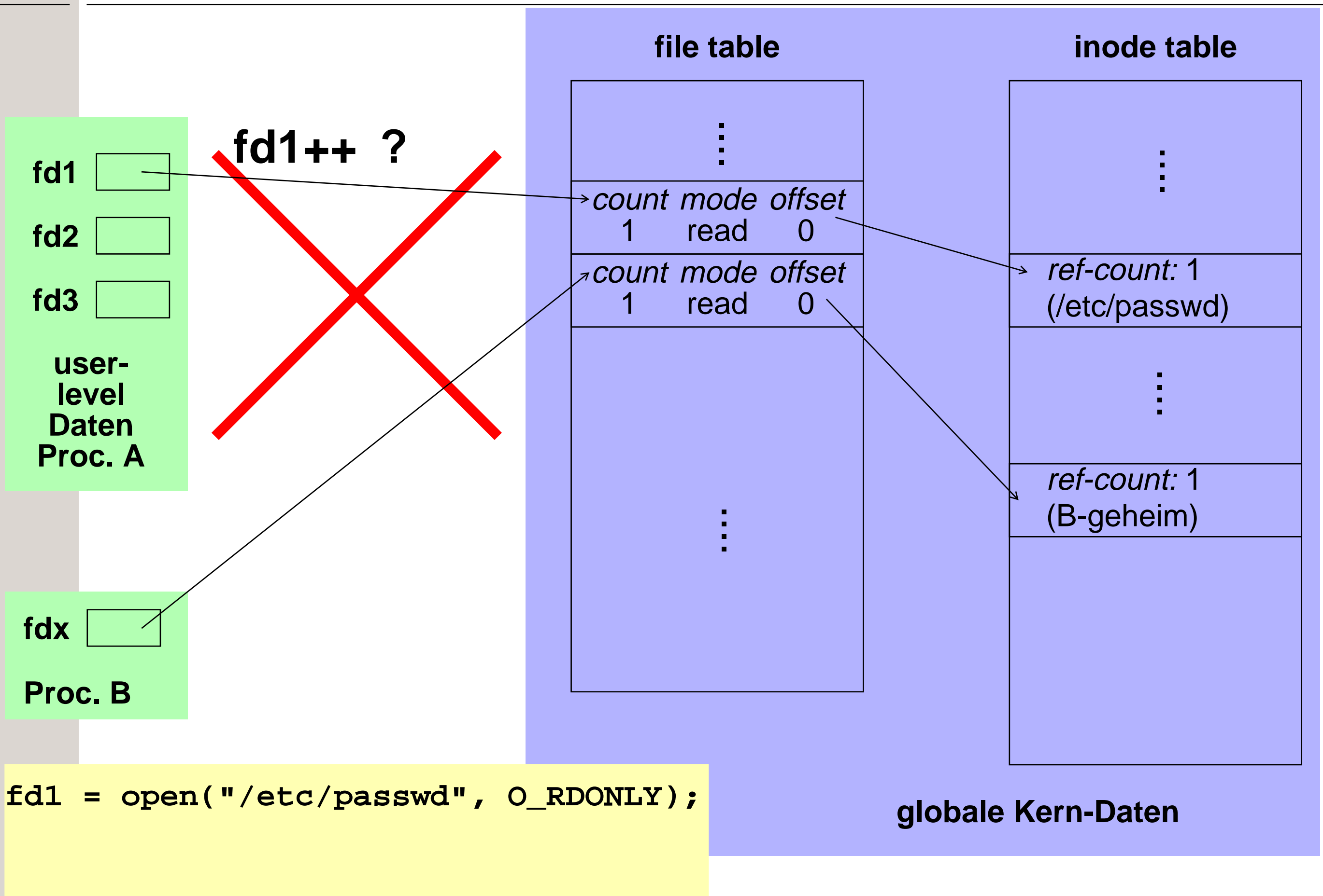
2 Datei öffnen (3)




```
fd1 = open("/etc/passwd", O_RDONLY);
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

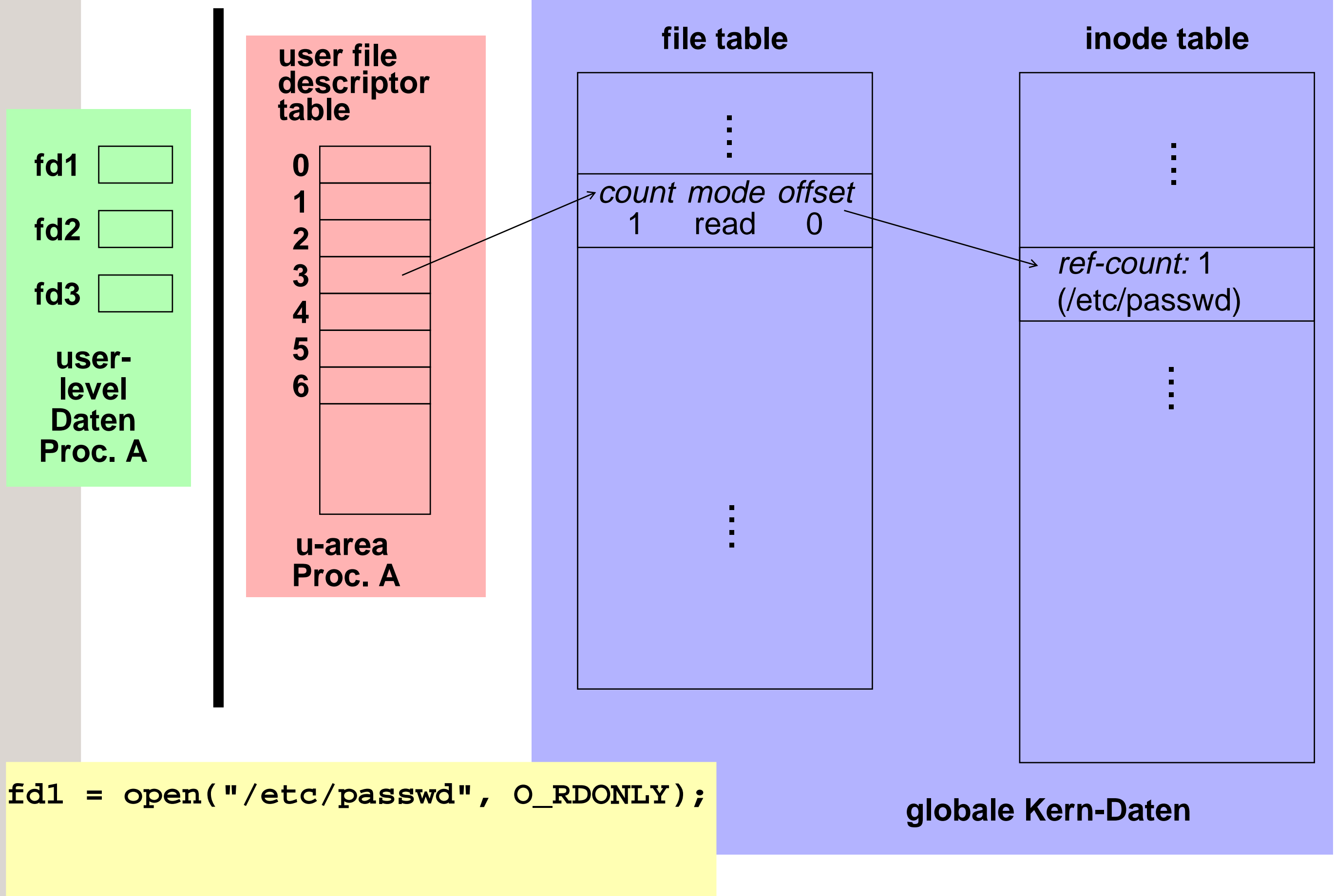
2 Datei öffnen (3)




```
fd1 = open("/etc/passwd", O_RDONLY);
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

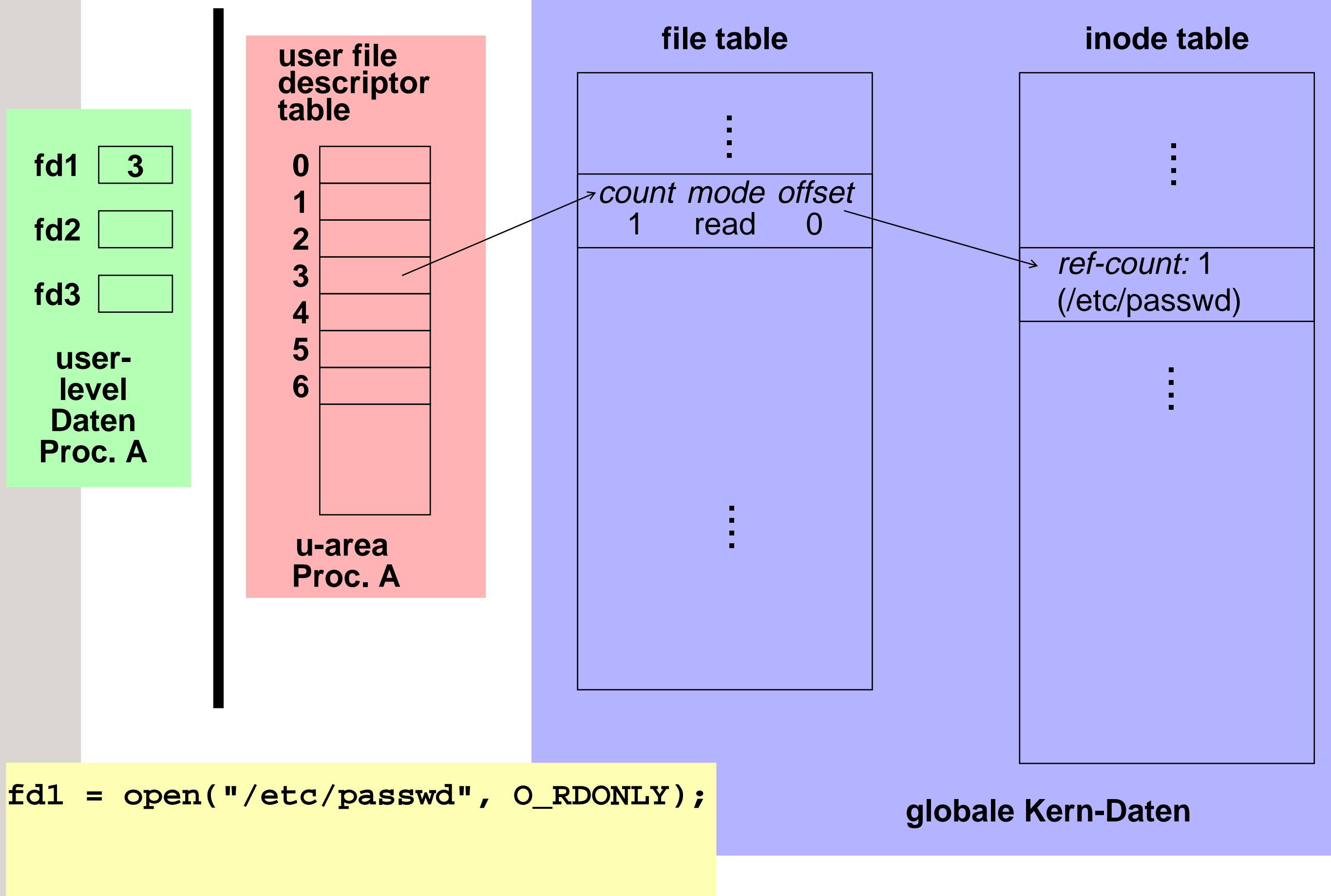
Datei öffnen (3)




```
fd1 = open("/etc/passwd", O_RDONLY);
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

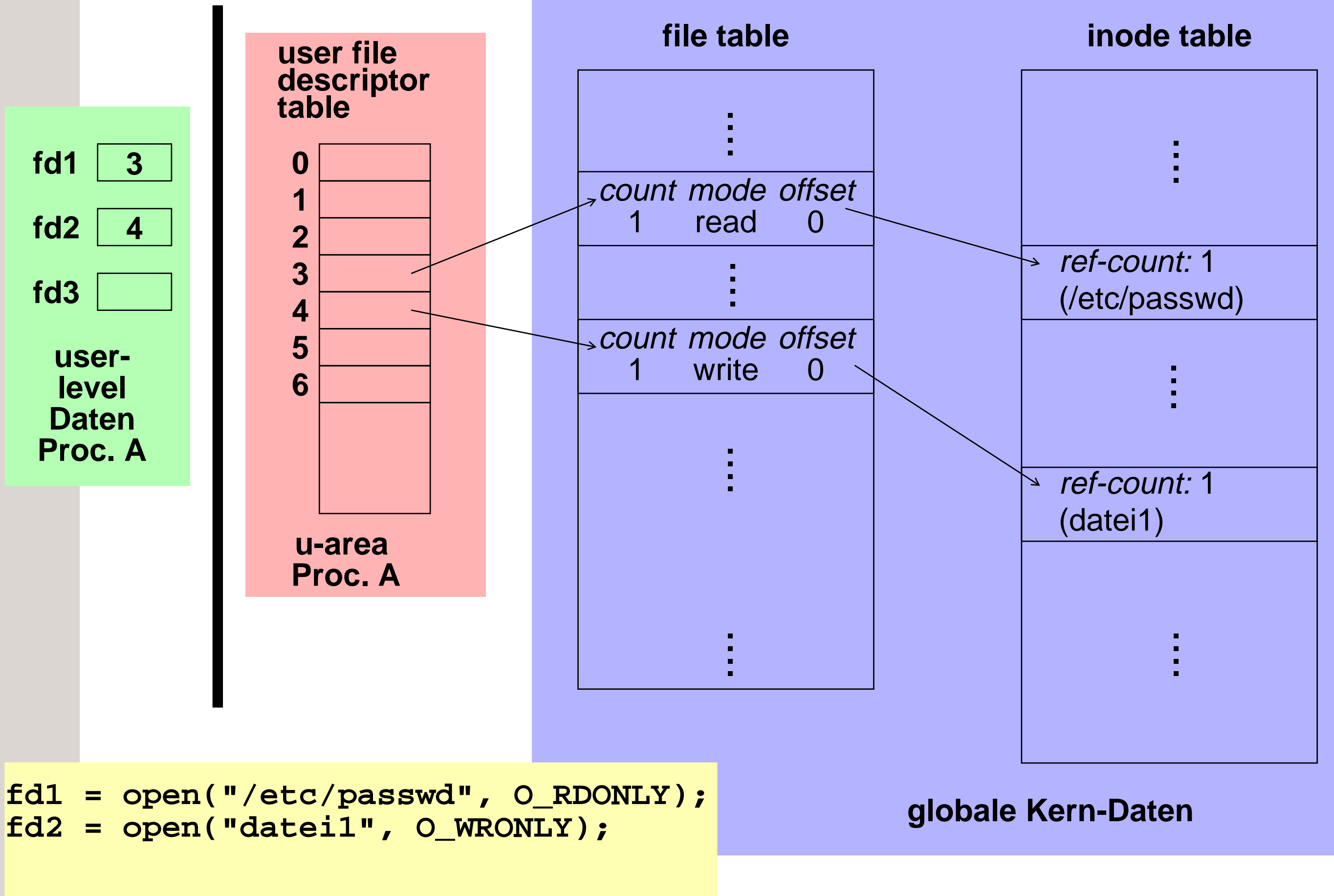
2 Datei öffnen (3)




```
fd1 = open("/etc/passwd", O_RDONLY);
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Datei öffnen (3)

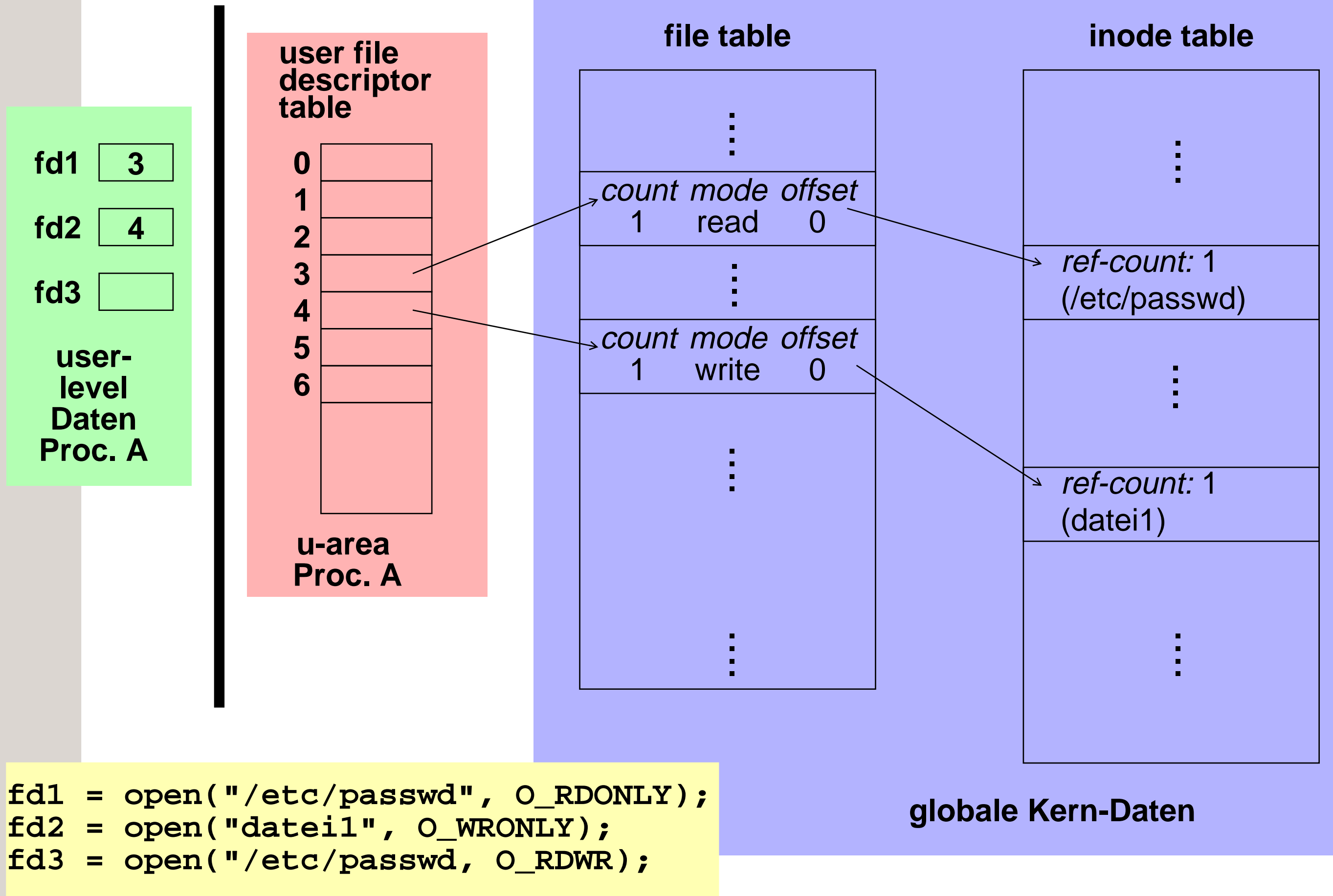



```
fd1 = open("/etc/passwd", O_RDONLY);
fd2 = open("datei1", O_WRONLY);
```


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Datei öffnen (3)

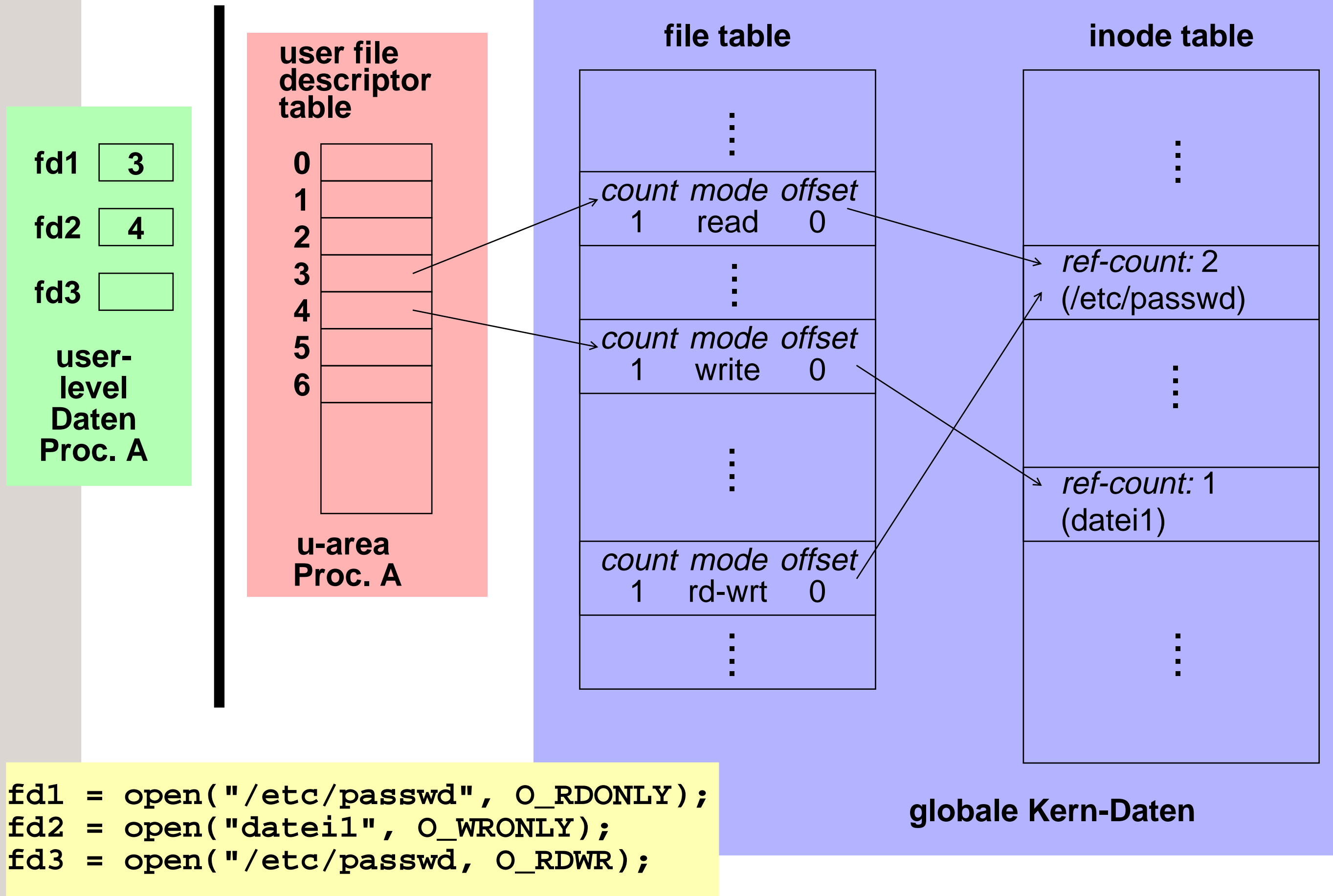



```
fd1 = open("/etc/passwd", O_RDONLY);
fd2 = open("datei1", O_WRONLY);
fd3 = open("/etc/passwd, O_RDWR);
```


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

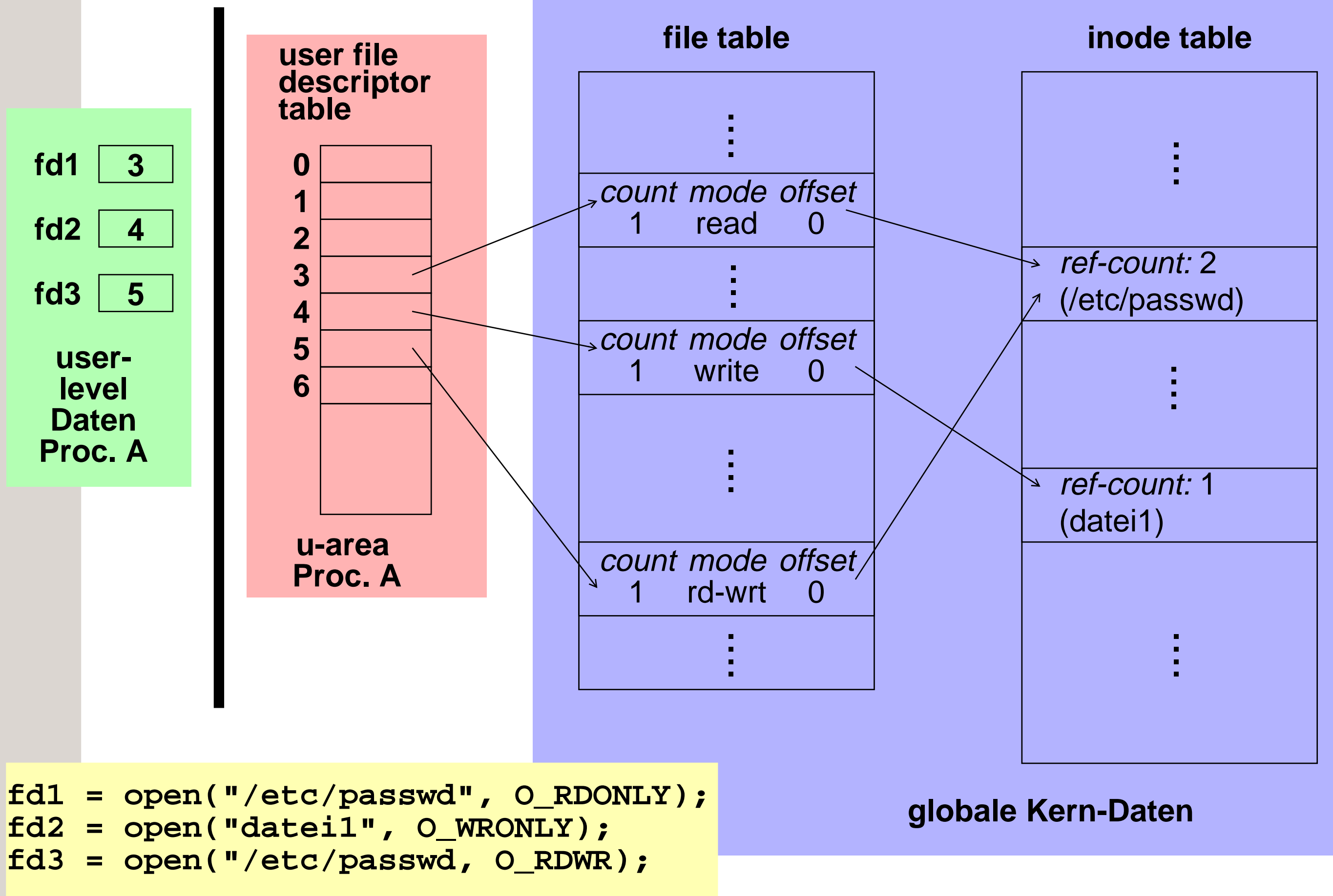
Datei öffnen (3)




```
fd1 = open("/etc/passwd", O_RDONLY);
fd2 = open("datei1", O_WRONLY);
fd3 = open("/etc/passwd, O_RDWR);
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

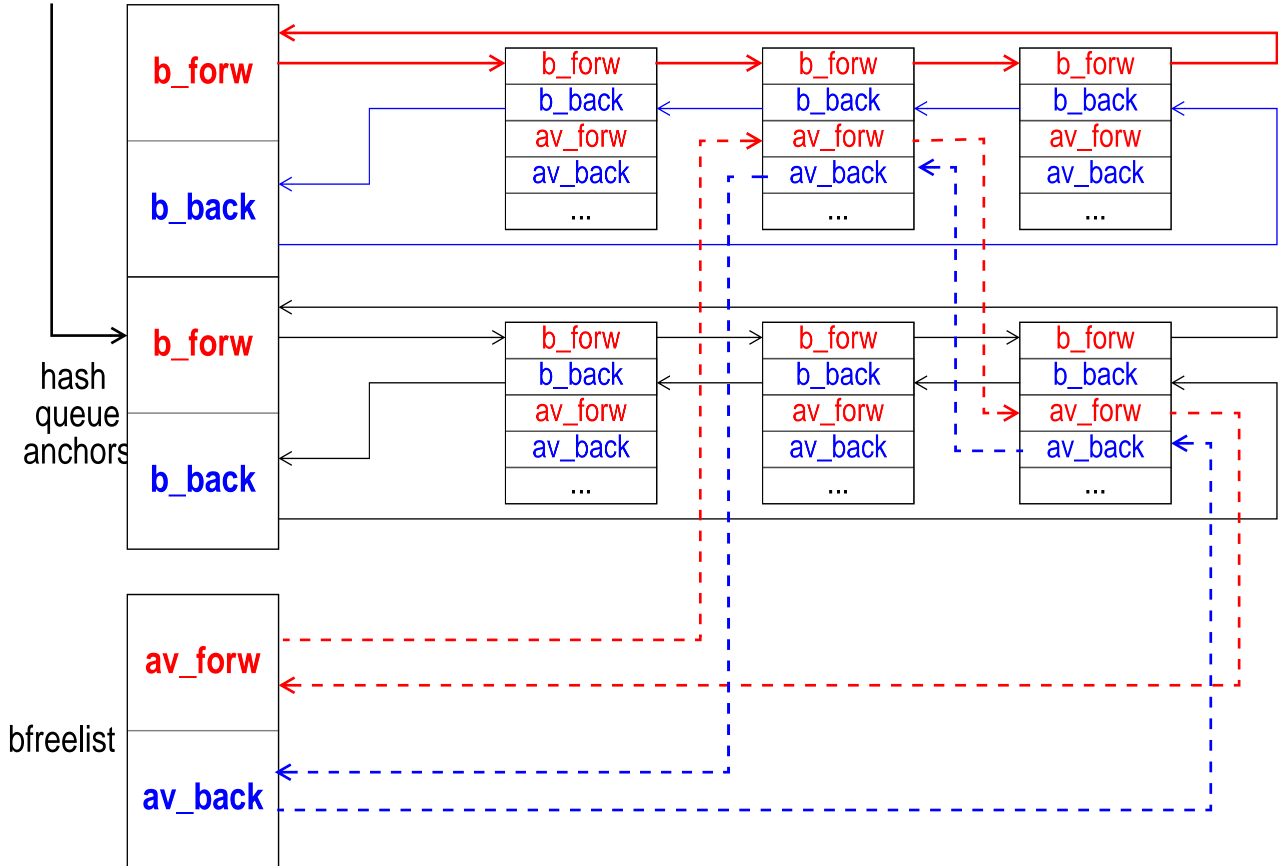
2 Datei öffnen (3)



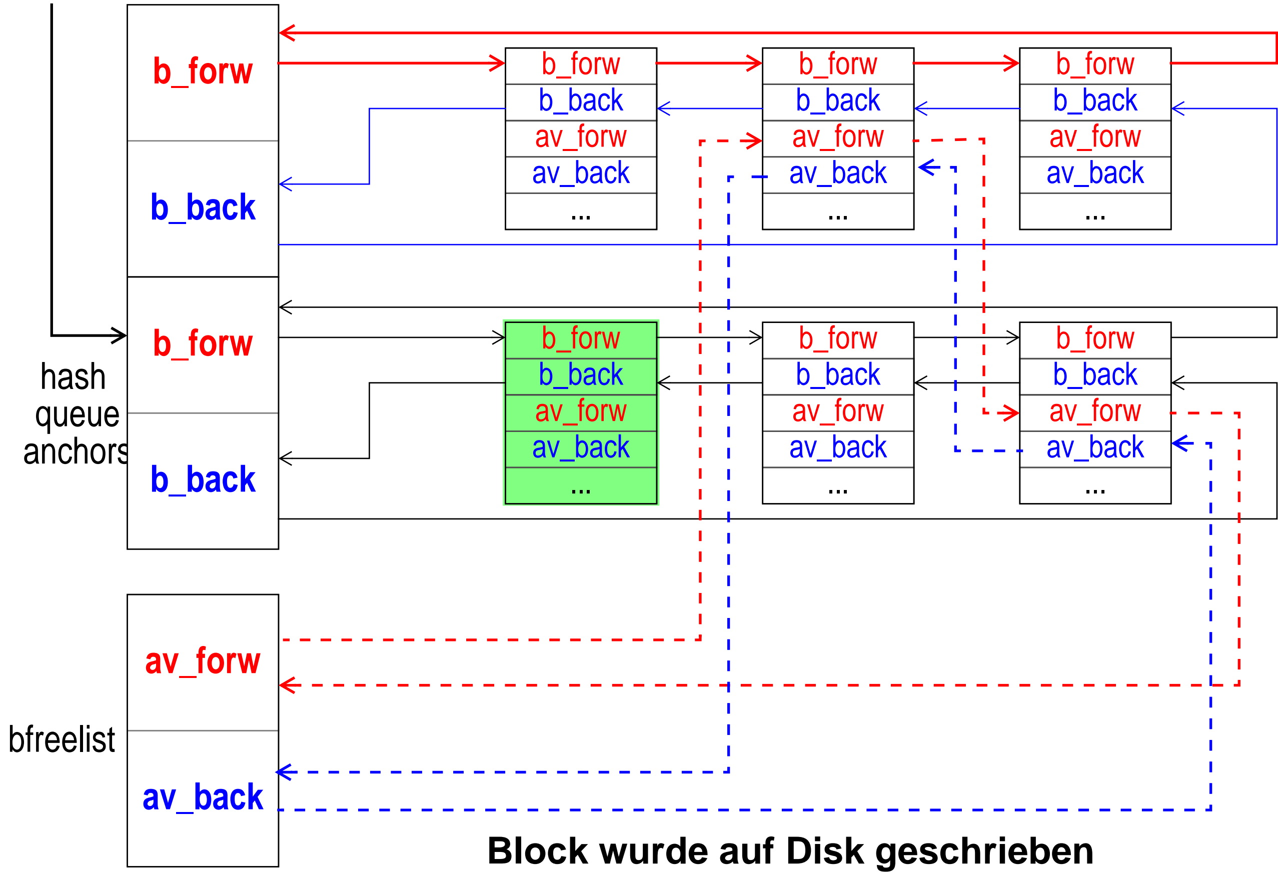

```
fd1 = open("/etc/passwd", O_RDONLY);
fd2 = open("datei1", O_WRONLY);
fd3 = open("/etc/passwd, O_RDWR);
```


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

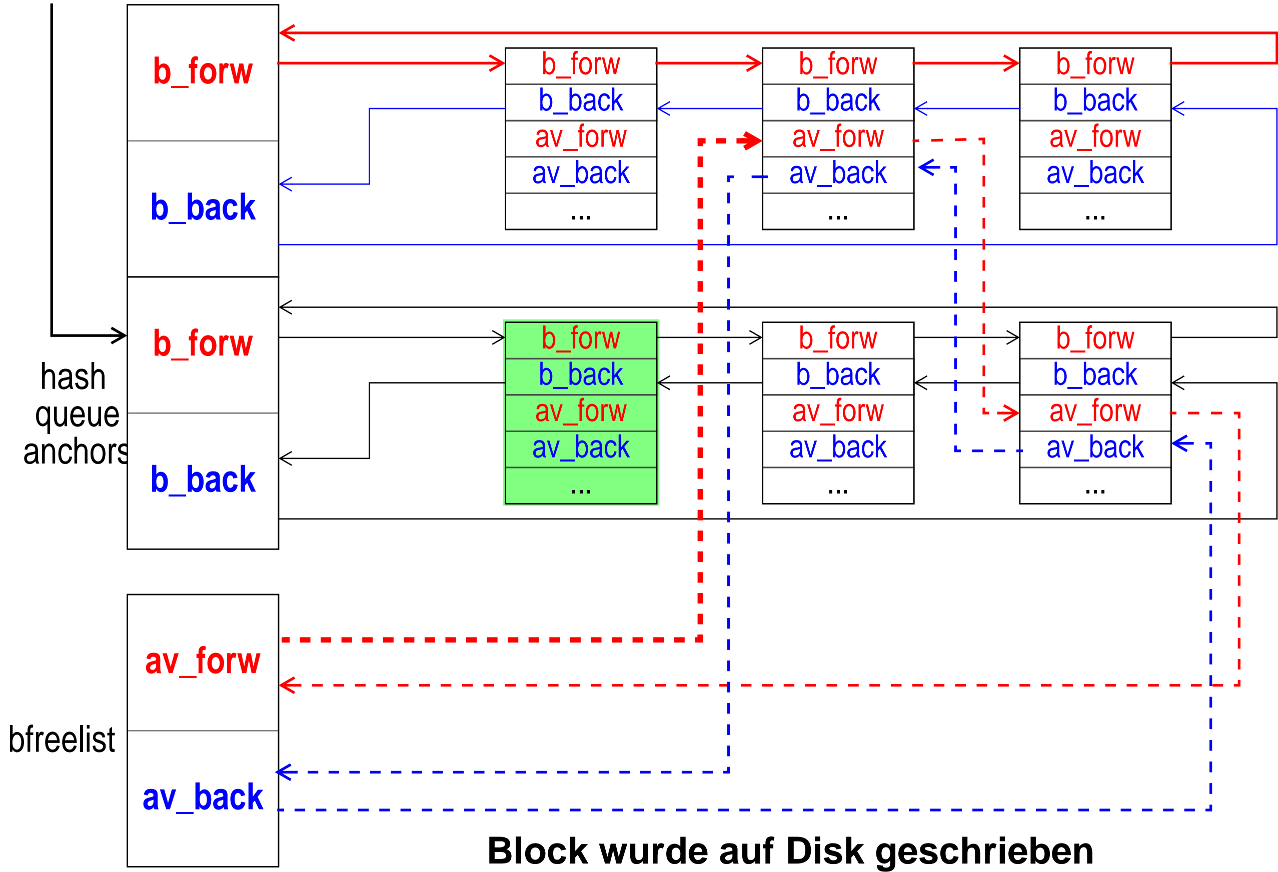


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



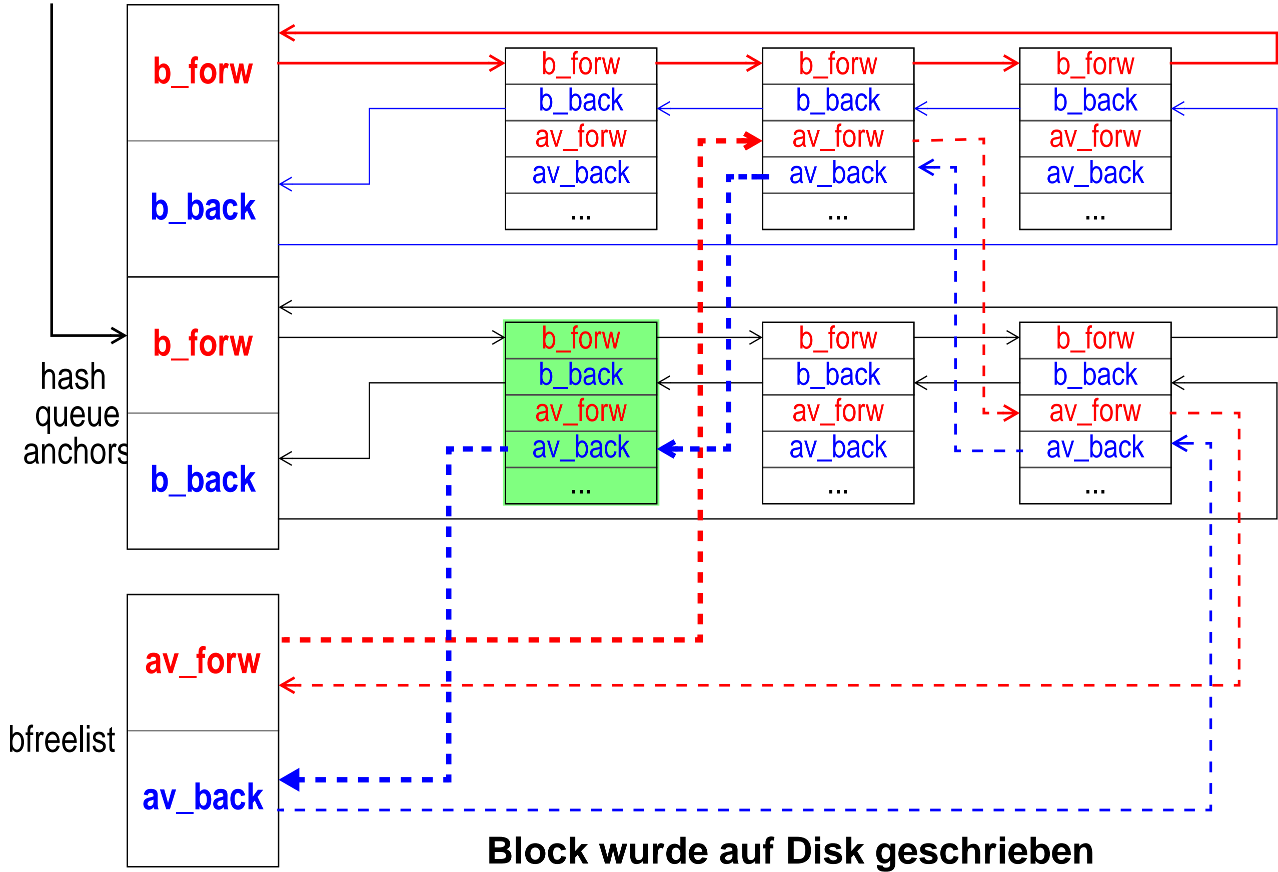
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

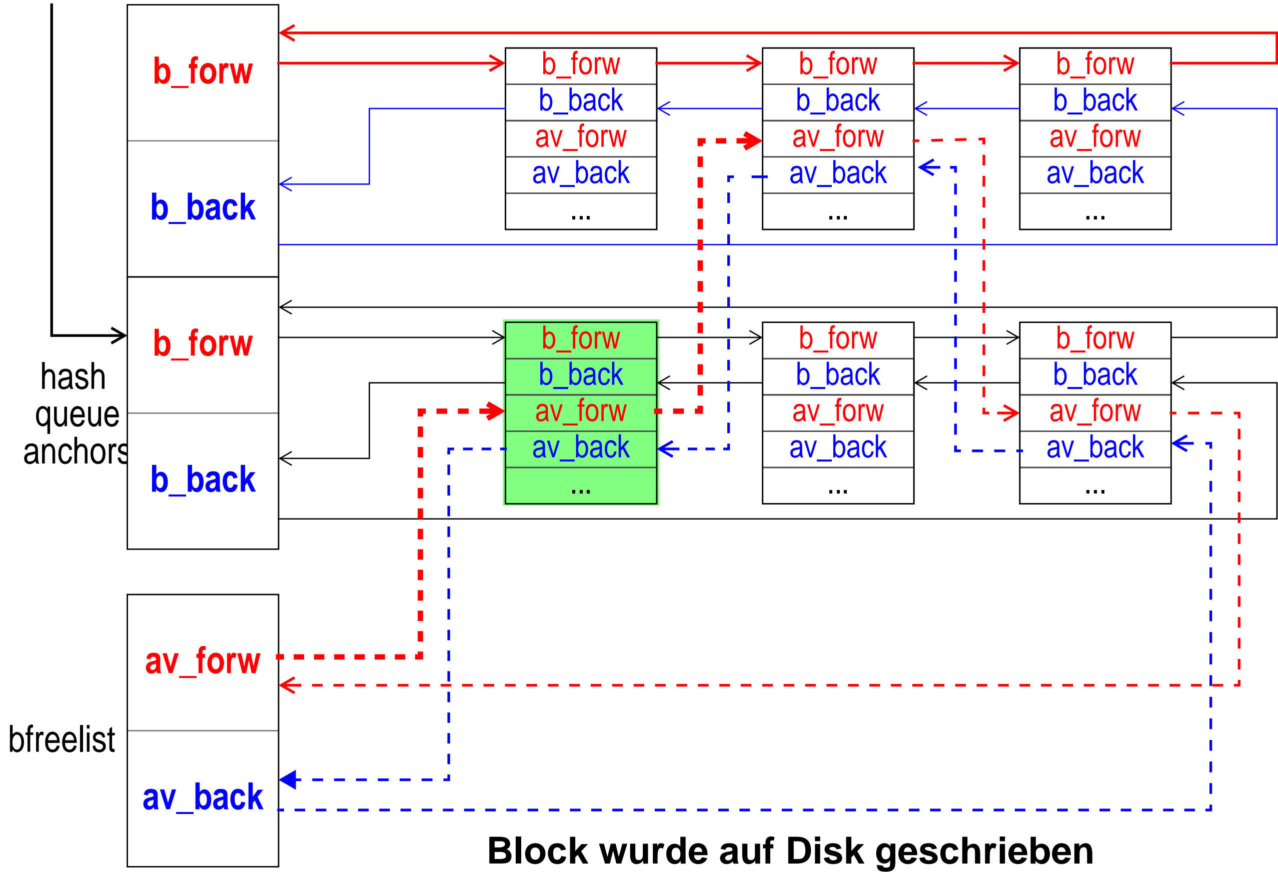


Block wurde auf Disk geschrieben

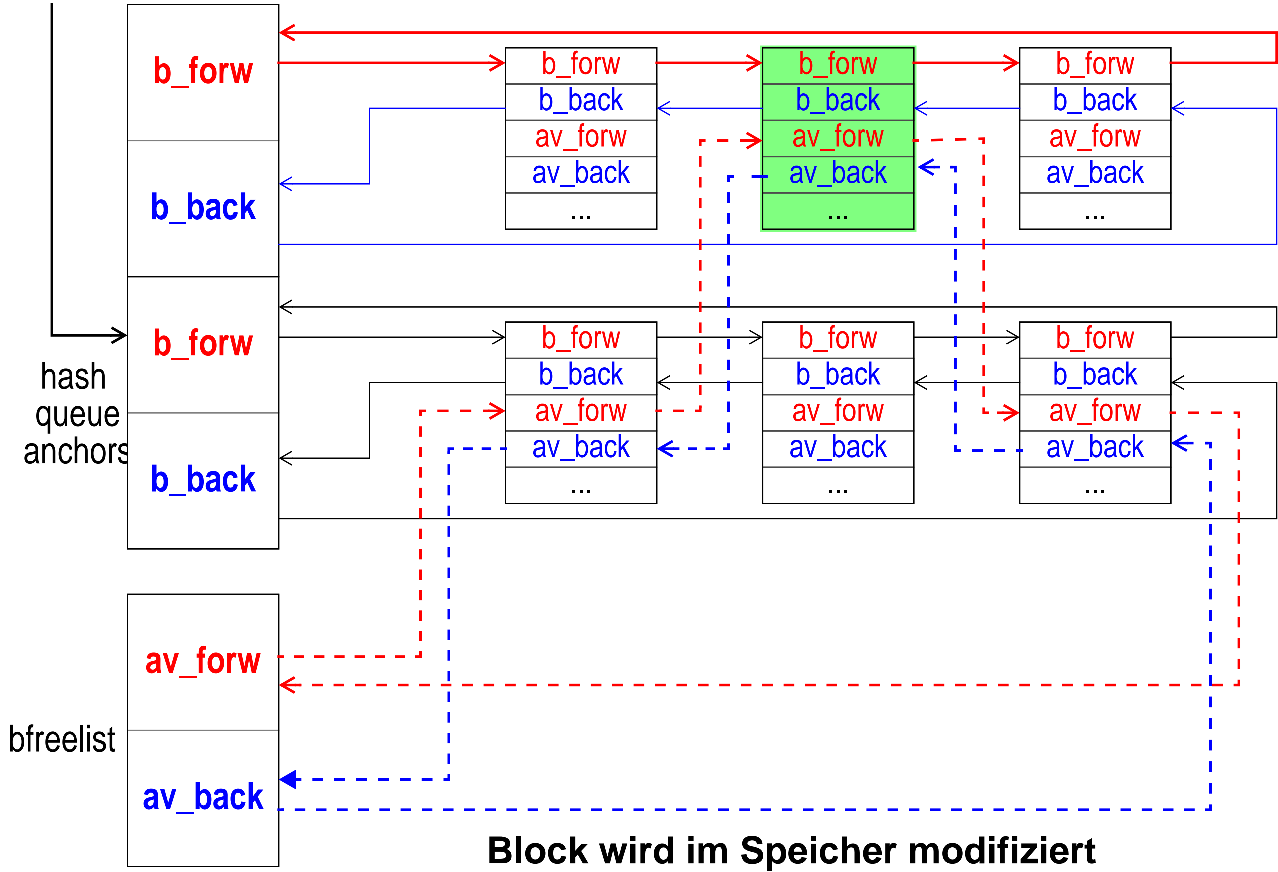
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



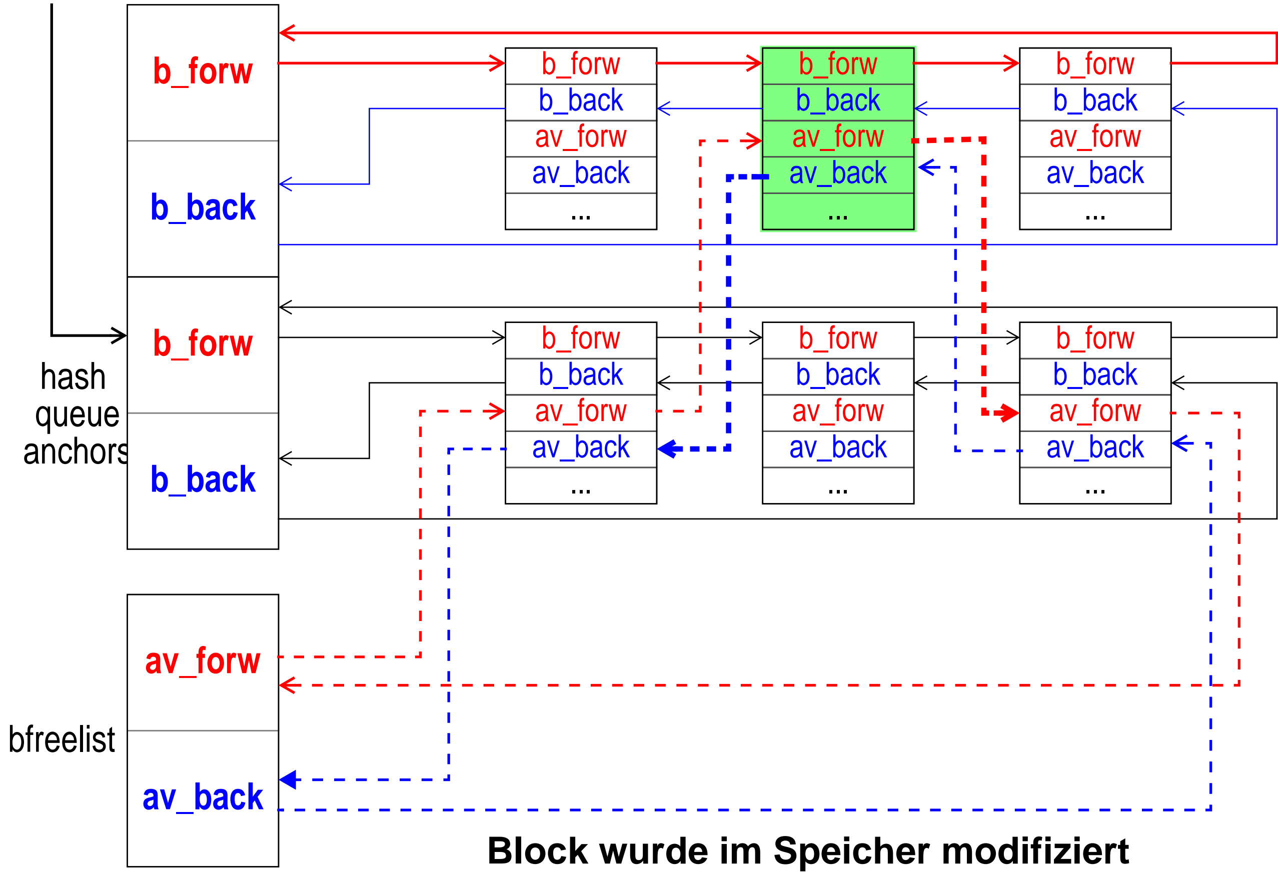
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

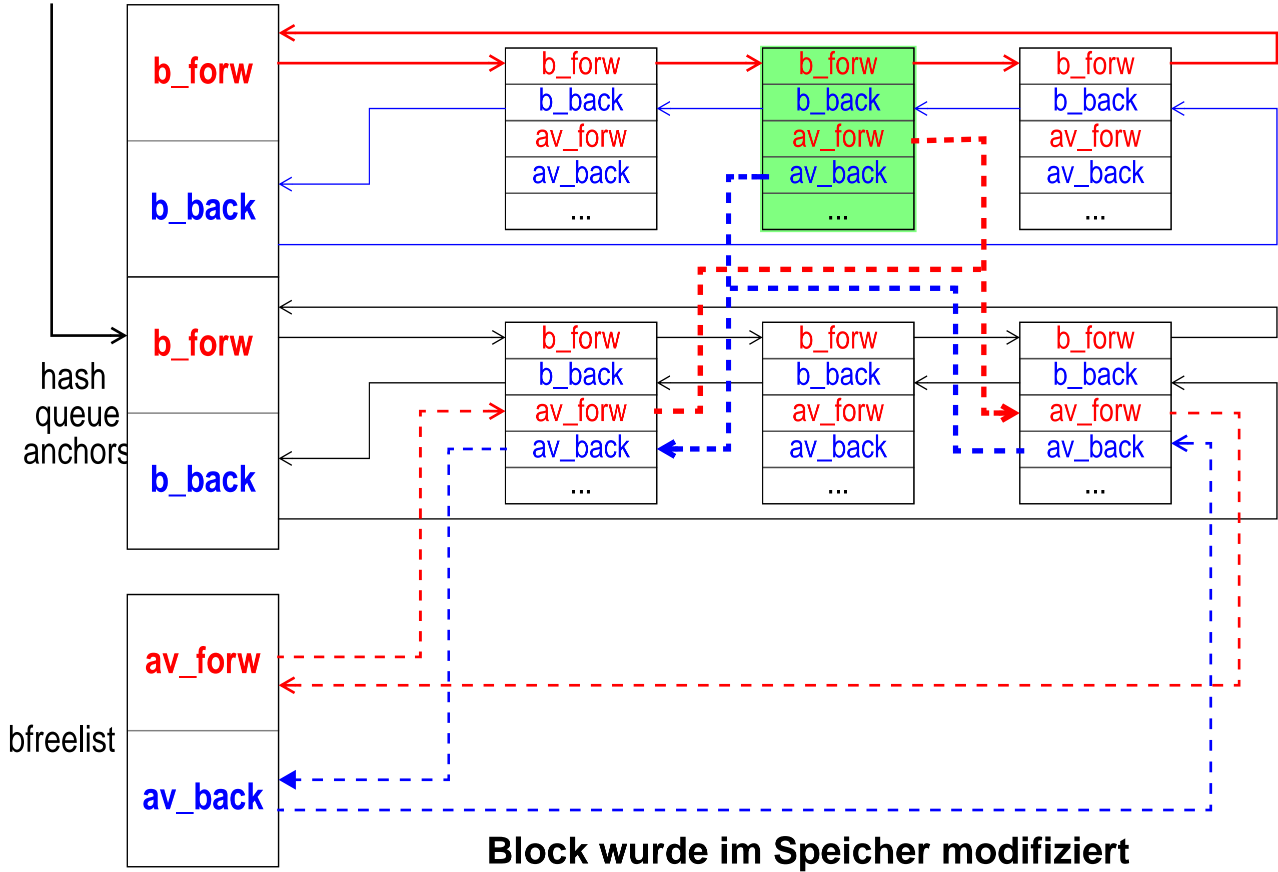


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

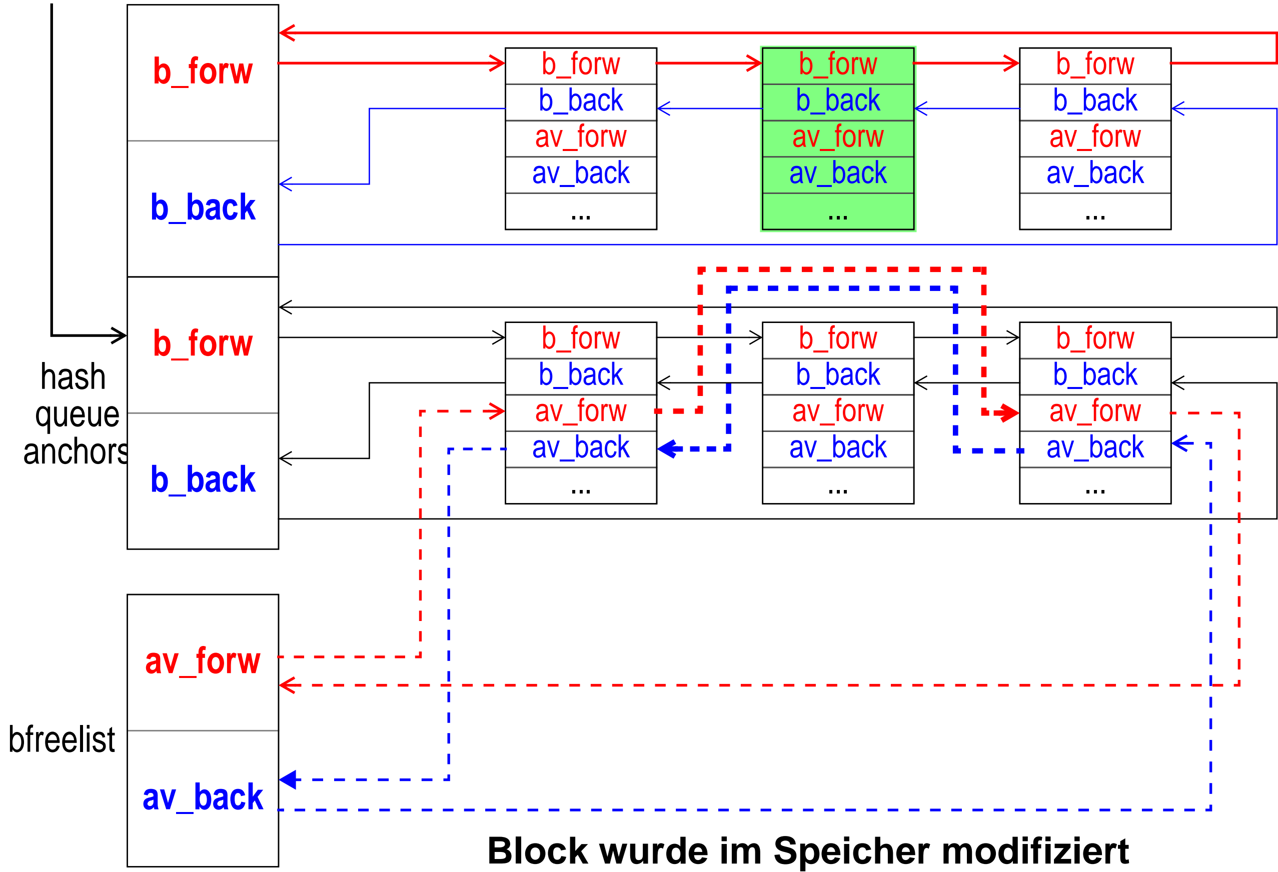


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Kontrollstrukturen

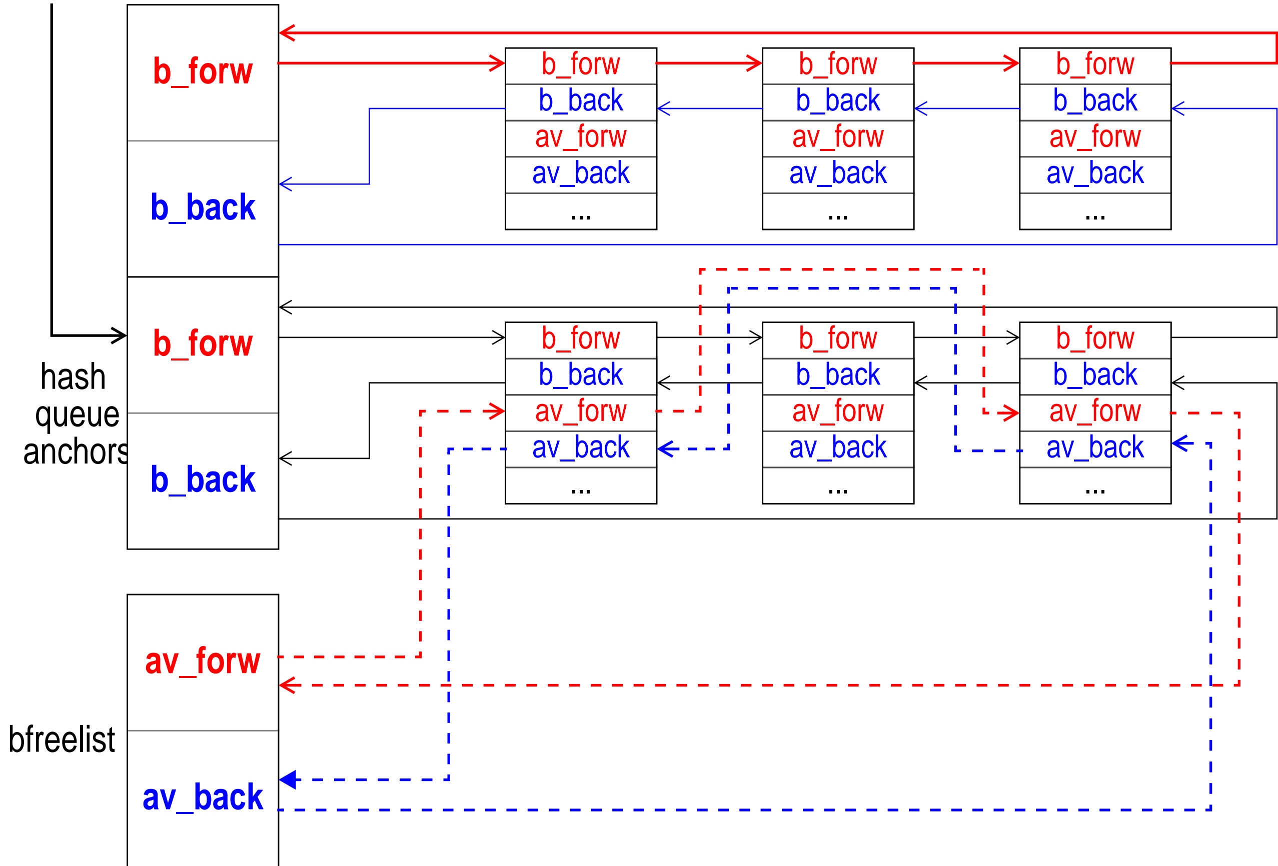


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

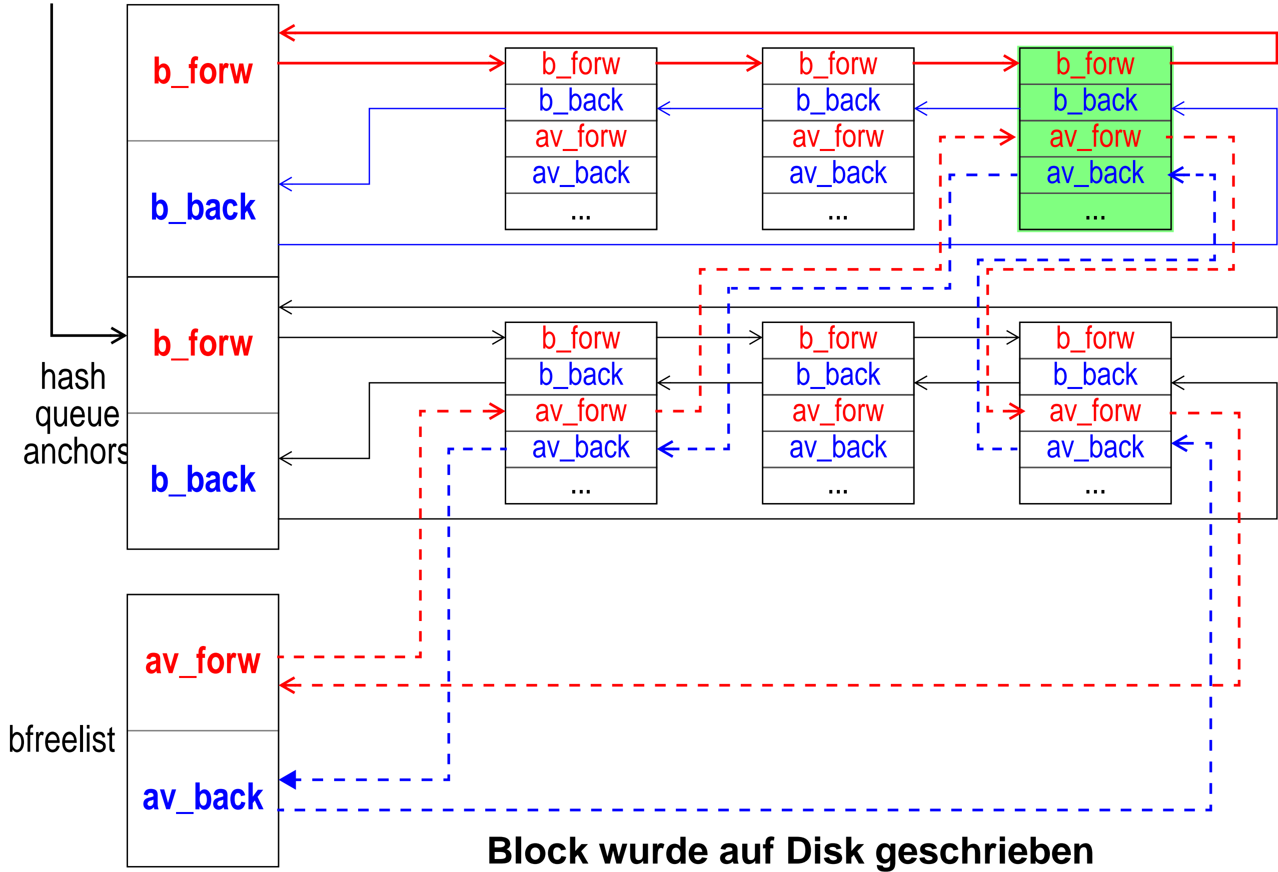


Block wurde im Speicher modifiziert

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



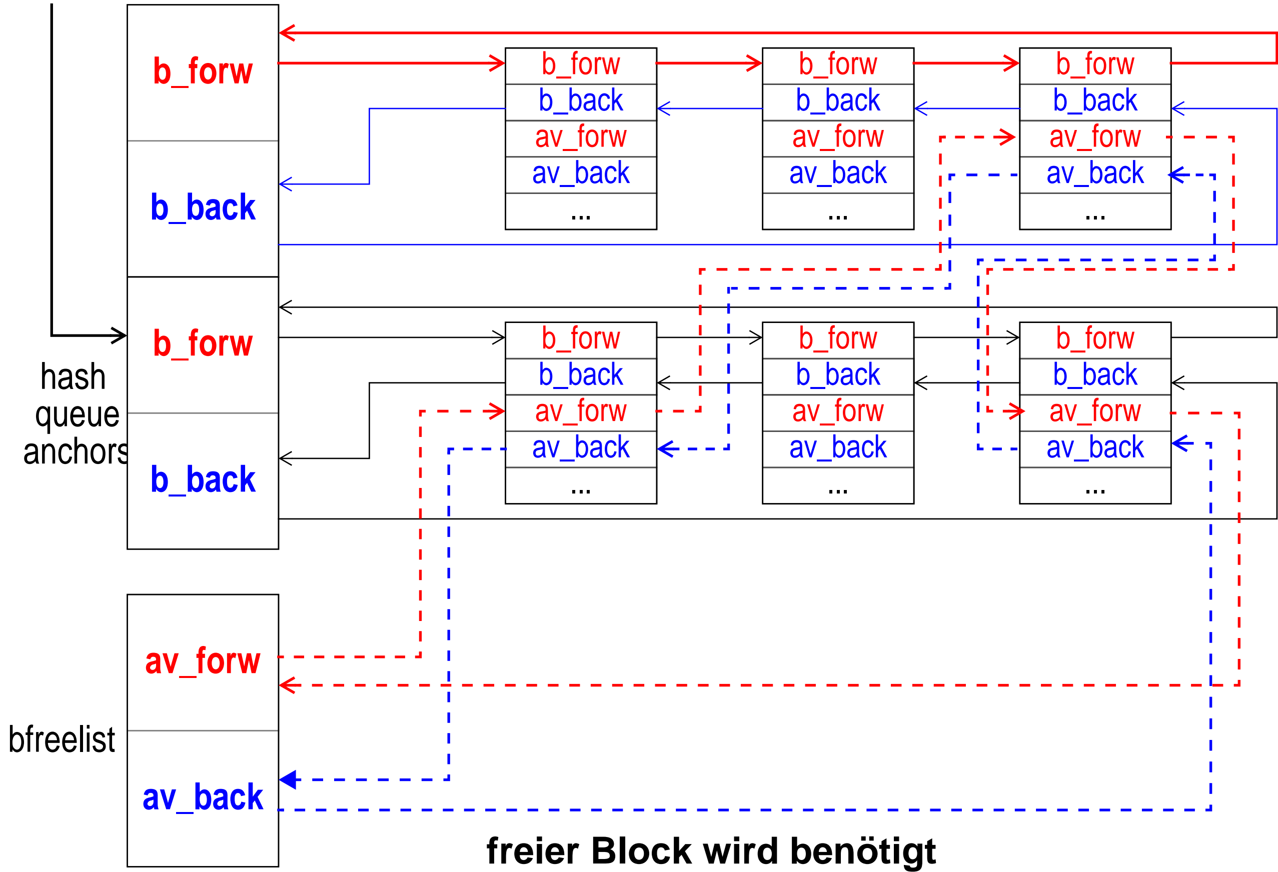
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



Block wurde auf Disk geschrieben

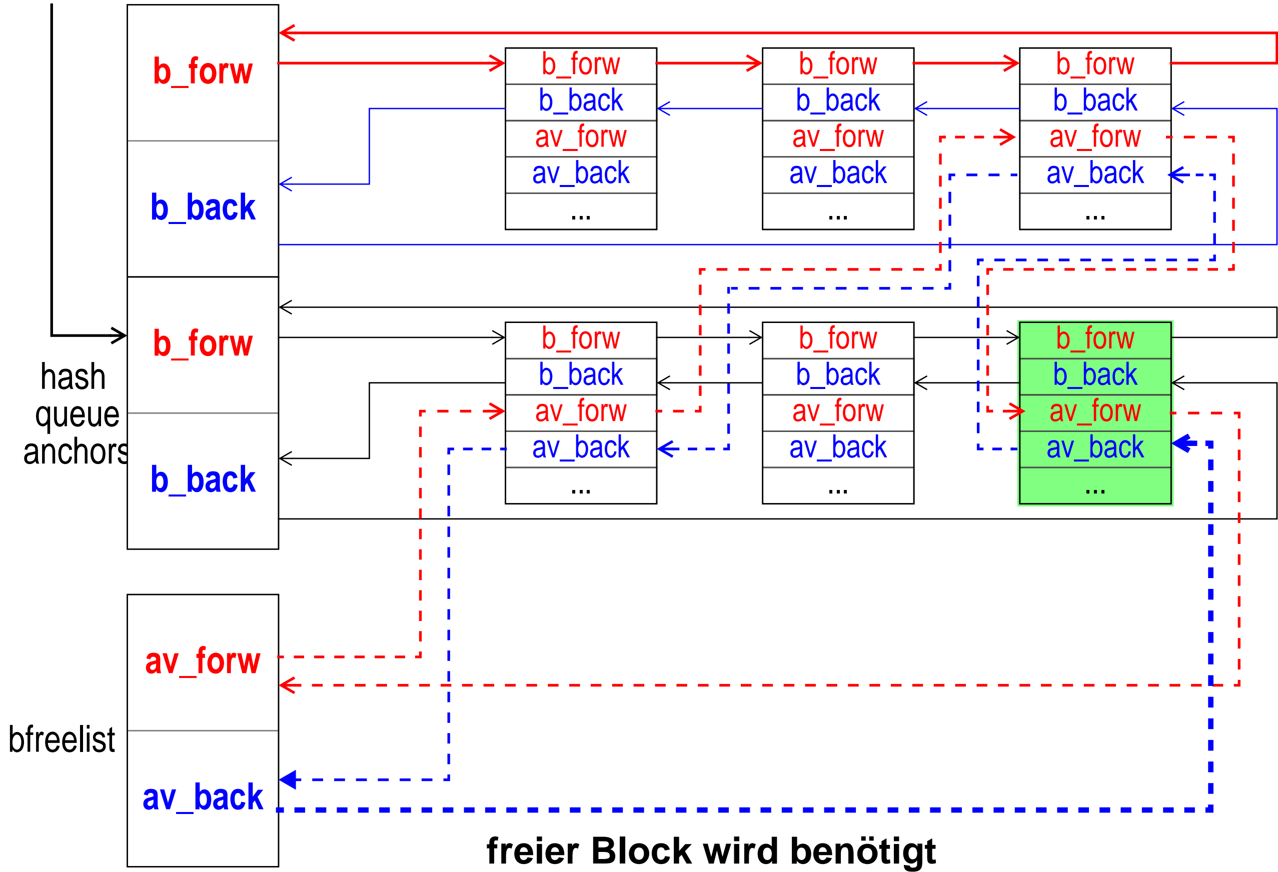
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



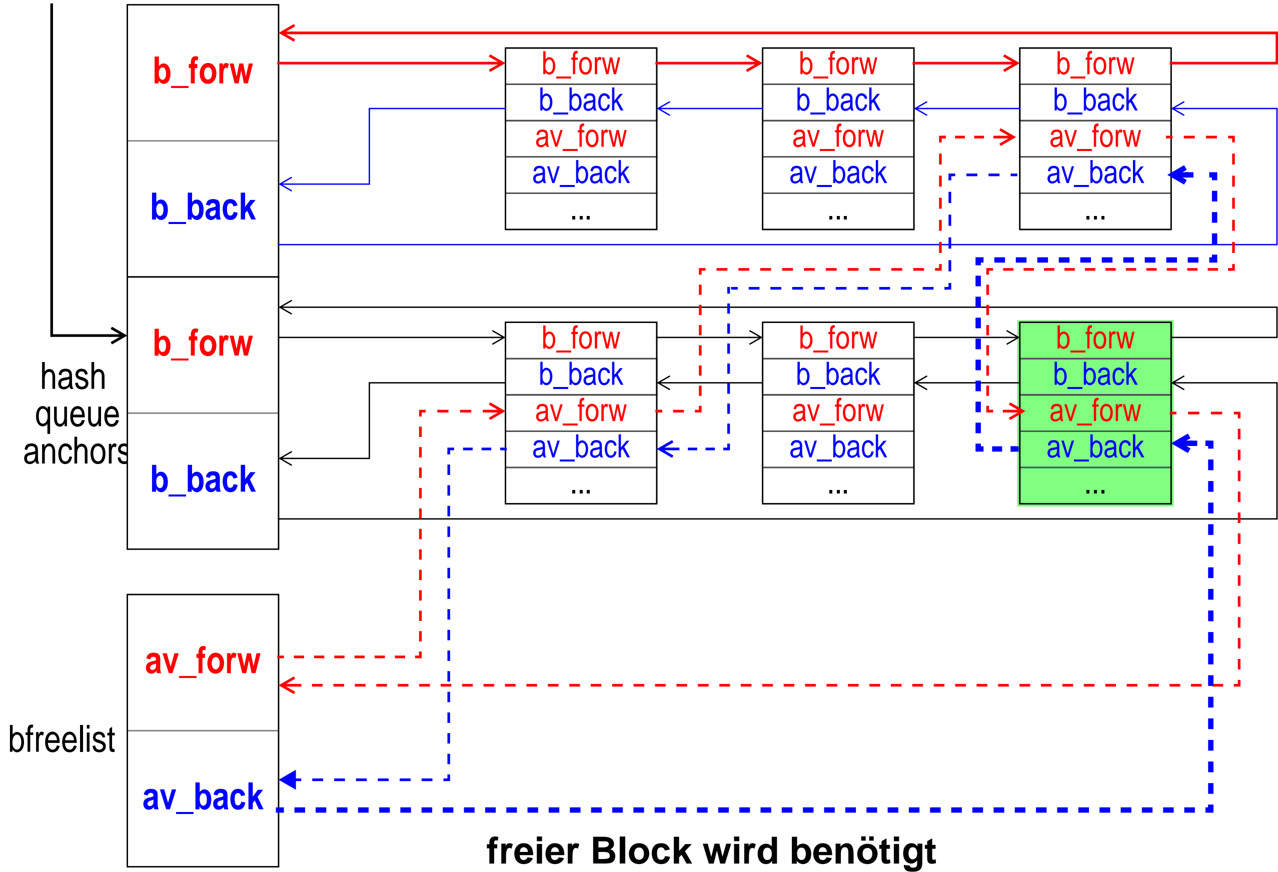
freier Block wird benötigt

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

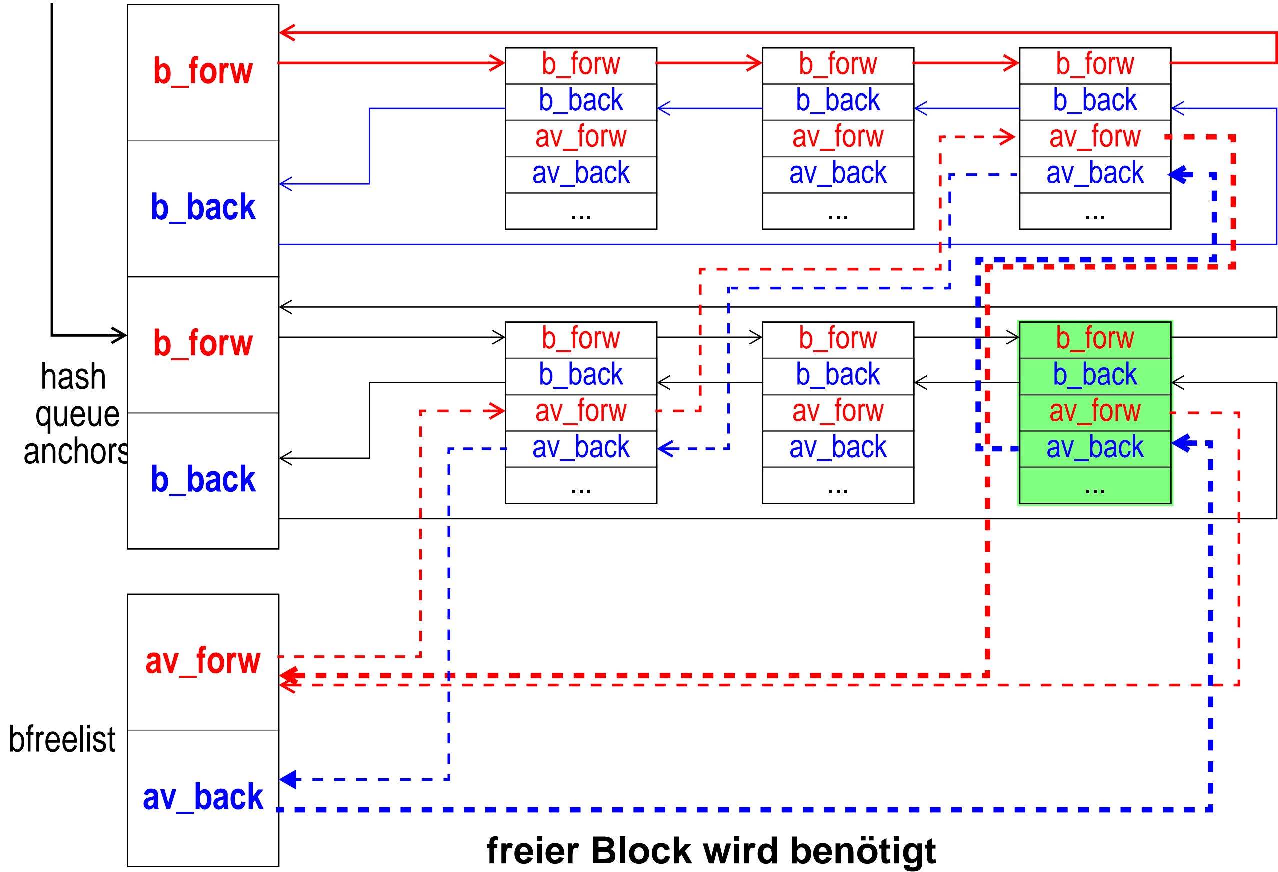
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Kontrollstrukturen

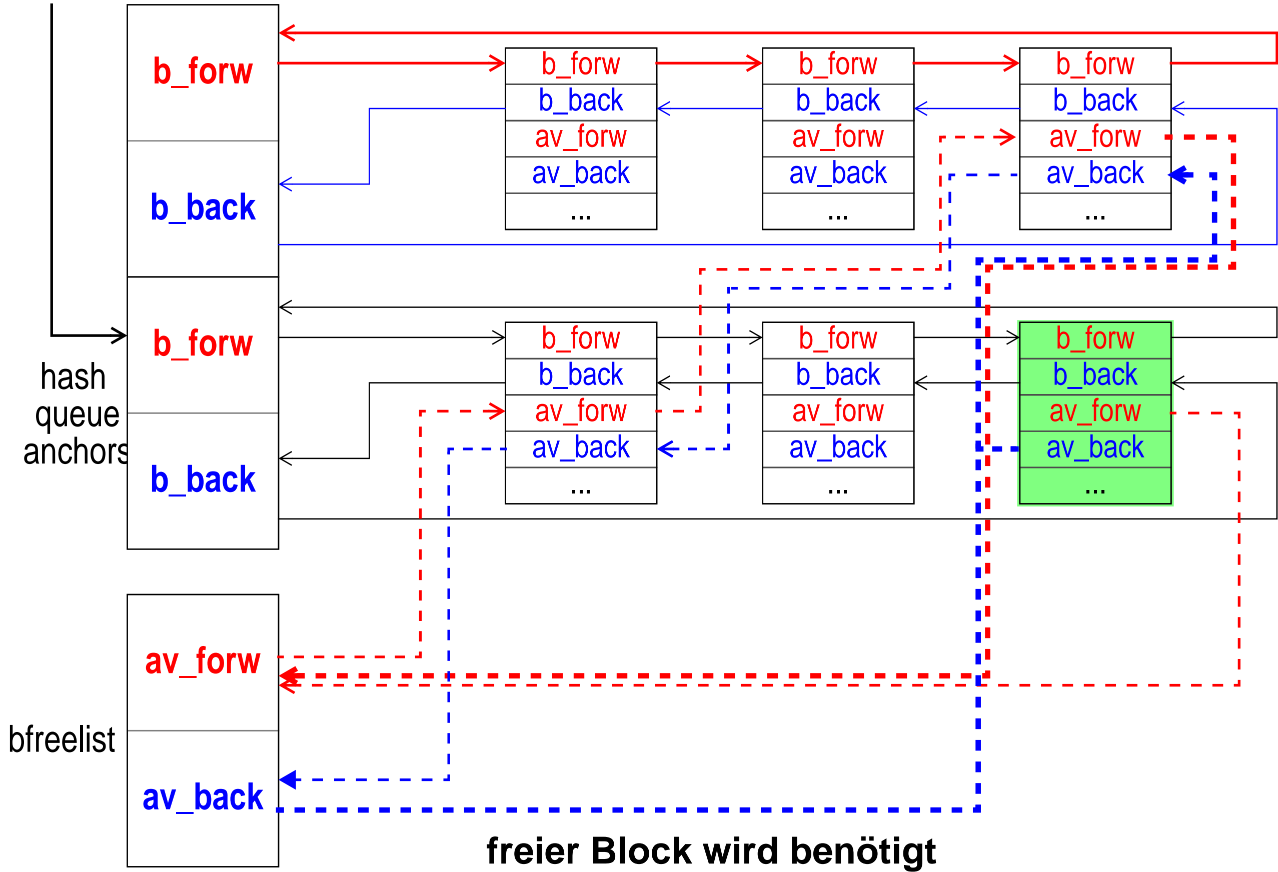


freier Block wird benötigt

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

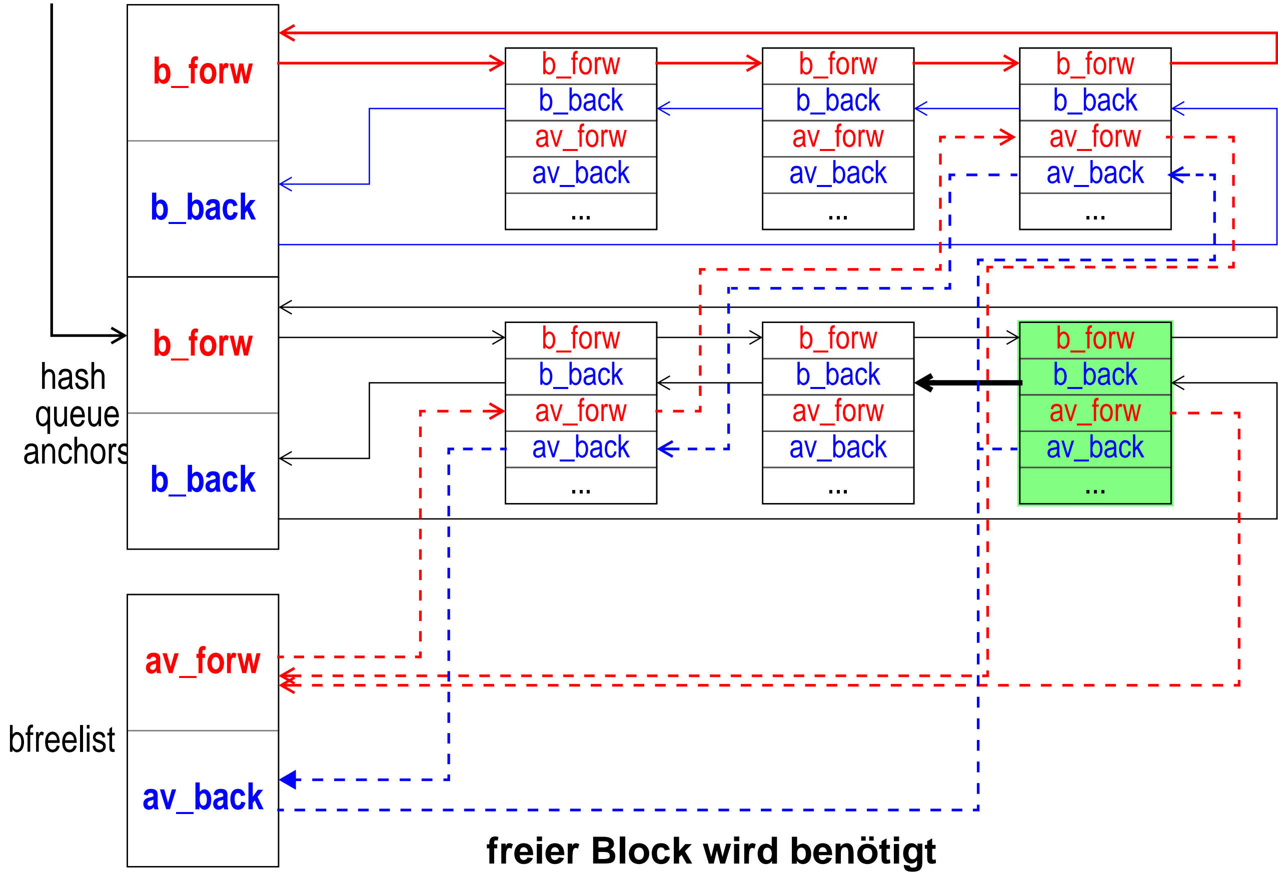
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Kontrollstrukturen



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

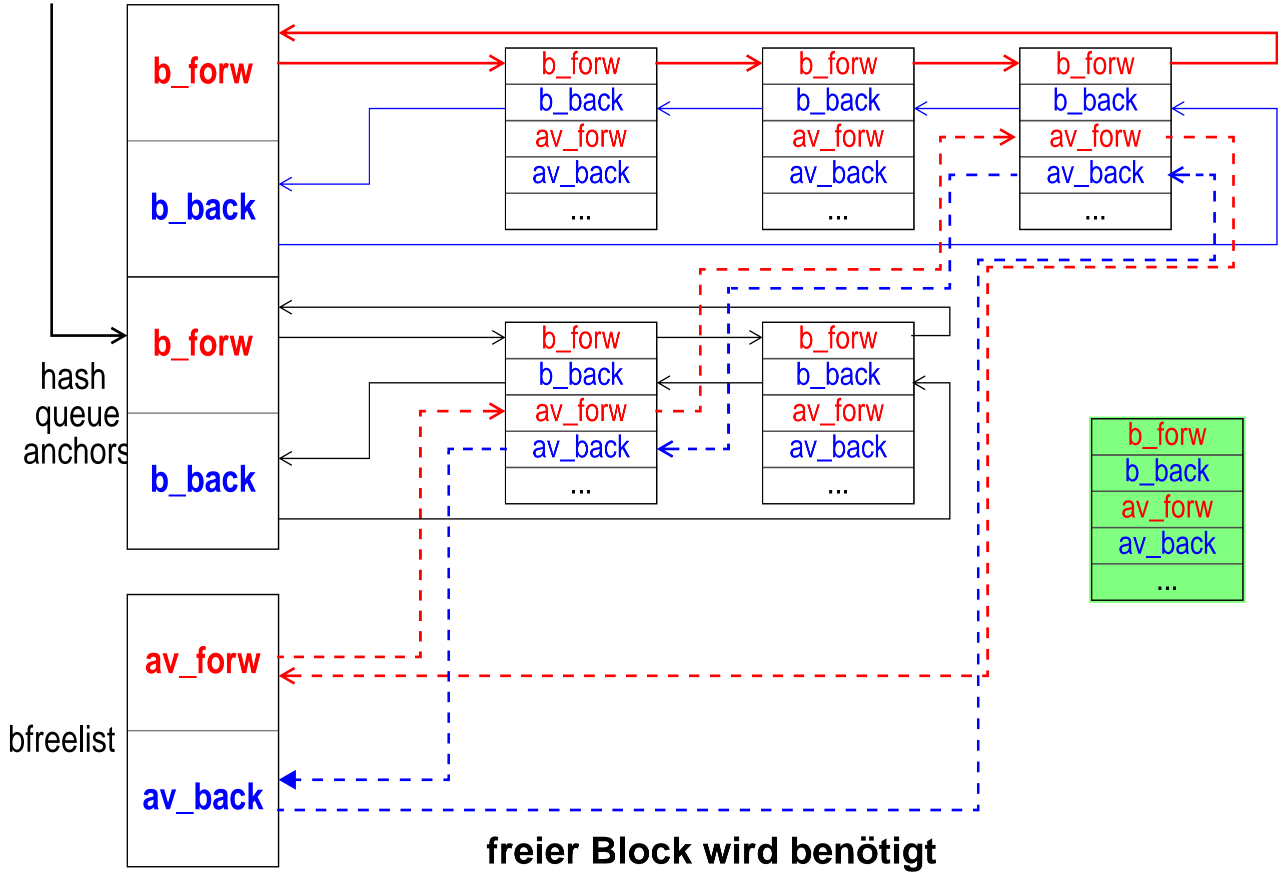
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

bhash(dev, blkno)

freier Block wird benötigt

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Kontrollstrukturen



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Kontrollfluß

```
sysent-table
  └─► read(u.u_ap)          (u_ap = fd, buf, count)
        buf, count -> uio
      └─► rwuio(uio, READ)
            fp = GETF(fd)           (Zeiger of open file table besorgen)
            uio = { fp->f_mode, fp->offset }
fops-switch ──────► sockets
vnodes
          └─► vno_rw(fp, rw, uio)
                vp = fp->f_data         (vnode-Struktur aufbauen)
                iomode = 0 | APPEND | SYNC
vnode-switch ──────► NFS
              └────► specFS
UFS
              └─► ufs_rw(vp, uio, rw, iomode, ...)
                    ip = VTOI(vp)       (inode-pointer aus vnode besorgen)
                  └─► rwip(ip, uio, rw, iomode)
                  ┌─►   lblock = uio->offset/bsize     (logische Blocknr.)
                  │     pblock = bmap(lblock)          (phys. Blocknr.)
                  │   └─► bread(vp, pblock, bsize)
                  │                 ┆
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Kontrollfluß

```
sysent-table
  └──► read(u.u_ap)          (u_ap = fd, buf, count)
         buf, count -> uio
         └──► rwuio(uio, READ)
                 fp = GETF(fd)          (Zeiger of open file table besorgen)
                 uio = { fp->f_mode, fp->offset }
       fops-switch ──────► sockets
       vnodes
                 └──► vno_rw(fp, rw, uio)
                         vp = fp->f_data          (vnode-Struktur aufbauen)
                         iomode = 0 | APPEND | SYNC
           vnode-switch ──────► NFS
                        └─────► specFS
                 UFS
                         └──► ufs_rw(vp, uio, rw, iomode, ...)
                                 ip = VTOI(vp)          (inode-pointer aus vnode besorgen)
                                 └──► rwip(ip, uio, rw, iomode)
                                   ┌──►  lblock = uio->offset/bsize      (logische Blocknr.)
                                   │     pblock = bmap(lblock)           (phys. Blocknr.)
                                   │     └──► bread(vp, pblock, bsize)
                                   │                    ┆
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Kontrollfluß

```
sysent-table
 └─► read(u.u_ap)          (u_ap = fd, buf, count)
      buf, count -> uio
      └─► rwuio(uio, READ)
              fp = GETF(fd)          (Zeiger of open file table besorgen)
              uio = { fp->f_mode, fp->offset }
     fops-switch ───────► sockets
     vnodes
          └─► vno_rw(fp, rw, uio)
                  vp = fp->f_data          (vnode-Struktur aufbauen)
                  iomode = 0 | APPEND | SYNC
          vnode-switch ───────► NFS
                 └────────────► specFS
           UFS
                 └─► ufs_rw(vp, uio, rw, iomode, ...)
                         ip = VTOI(vp)          (inode-pointer aus vnode besorgen)
                         └─► rwip(ip, uio, rw, iomode)
                           ┌─►    lblock = uio->offset/bsize       (logische Blocknr.)
                           │      pblock = bmap(lblock)            (phys. Blocknr.)
                           │      └─► bread(vp, pblock, bsize)
                           │                    ┆
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Kontrollfluß

```
sysent-table
 └─► read(u.u_ap)          (u_ap = fd, buf, count)
       buf, count -> uio
     └─► rwuio(uio, READ)
           fp = GETF(fd)          (Zeiger of open file table besorgen)
           uio = { fp->f_mode, fp->offset }
 fops-switch ──────► sockets
 vnodes
         └─► vno_rw(fp, rw, uio)
               vp = fp->f_data          (vnode-Struktur aufbauen)
               iomode = 0 | APPEND | SYNC
     vnode-switch ──────► NFS
                   └────► specFS
        UFS
             └─► ufs_rw(vp, uio, rw, iomode, ...)
                   ip = VTOI(vp)          (inode-pointer aus vnode besorgen)
                 └─► rwip(ip, uio, rw, iomode)
                 ┌─►    lblock = uio->offset/bsize       (logische Blocknr.)
                 │      pblock = bmap(lblock)            (phys. Blocknr.)
                 │    └─► bread(vp, pblock, bsize)
                 │              ¦
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Kontrollfluß

```
sysent-table
 └──► read(u.u_ap)        (u_ap = fd, buf, count)
       │  buf, count -> uio
       └──► rwuio(uio, READ)
             │  fp = GETF(fd)          (Zeiger of open file table besorgen)
             │  uio = { fp->f_mode, fp->offset }
fops-switch ─────────► sockets
vnodes
             └──► vno_rw(fp, rw, uio)
                   │  vp = fp->f_data       (vnode-Struktur aufbauen)
                   │  iomode = 0 | APPEND | SYNC
       vnode-switch ─────────► NFS
                   │ └────────► specFS
         UFS       └──► ufs_rw(vp, uio, rw, iomode, ...)
                         │  ip = VTOI(vp)      (inode-pointer aus vnode besorgen)
                         └──► rwip(ip, uio, rw, iomode)
                          ┌──►  │  lblock = uio->offset/bsize     (logische Blocknr.)
                          │     │  pblock = bmap(lblock)          (phys. Blocknr.)
                          │     └──► bread(vp, pblock, bsize)
                          │                  ¦
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Kontrollfluß

```
sysent-table
 └─► read(u.u_ap)        (u_ap = fd, buf, count)
       buf, count -> uio
     └─► rwuio(uio, READ)
           fp = GETF(fd)           (Zeiger of open file table besorgen)
           uio = { fp->f_mode, fp->offset }
  fops-switch ──────► sockets
 vnodes
         └─► vno_rw(fp, rw, uio)
               vp = fp->f_data         (vnode-Struktur aufbauen)
               iomode = 0 | APPEND | SYNC
    vnode-switch ──────► NFS
                 └─────► specFS
     UFS   └─► ufs_rw(vp, uio, rw, iomode, ...)
                 ip = VTOI(vp)        (inode-pointer aus vnode besorgen)
               └─► rwip(ip, uio, rw, iomode)
               ┌─►   lblock = uio->offset/bsize      (logische Blocknr.)
               │     pblock = bmap(lblock)           (phys. Blocknr.)
               │   └─► bread(vp, pblock, bsize)
```

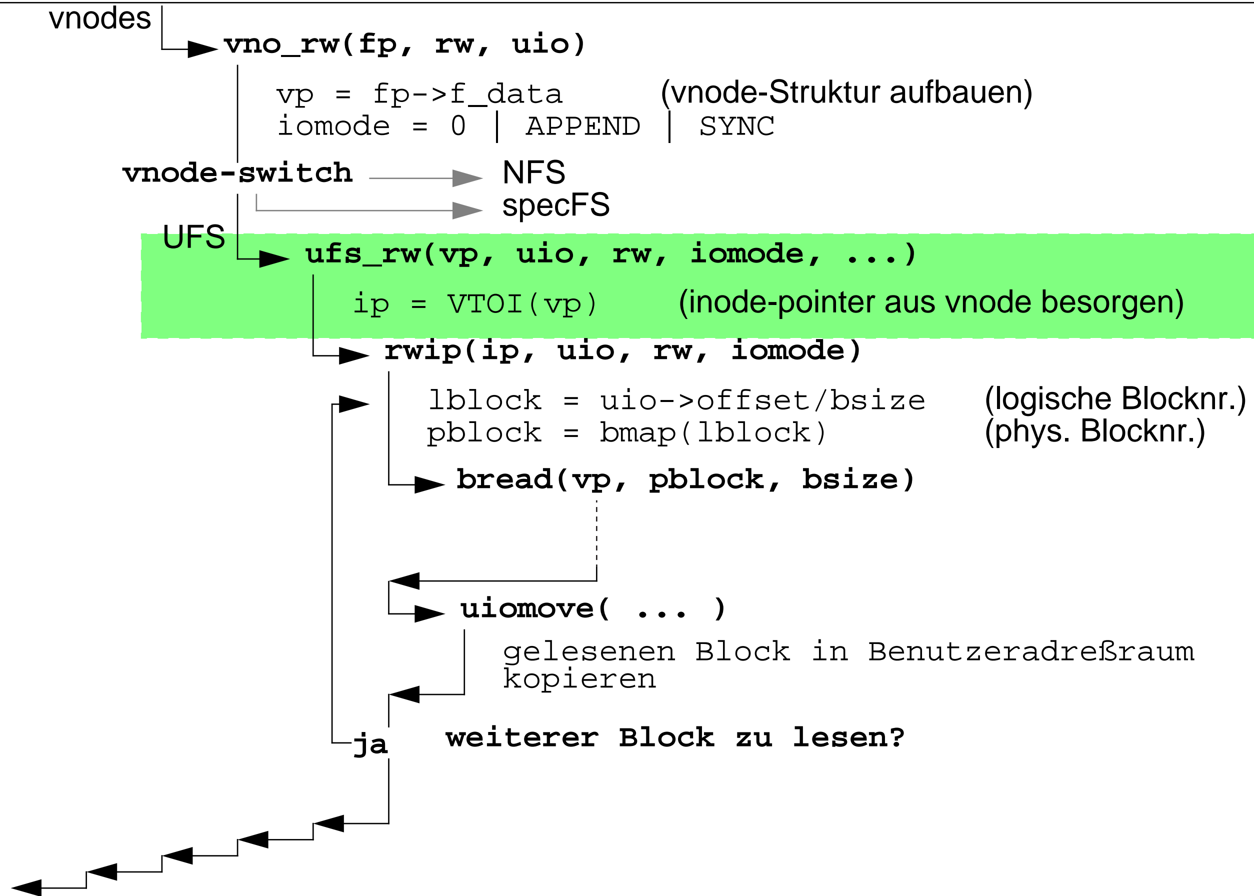

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Kontrollfluß

```
sysent-table
  └─► read(u.u_ap)        (u_ap = fd, buf, count)
        buf, count -> uio
      └─► rwuio(uio, READ)
            fp = GETF(fd)          (Zeiger of open file table besorgen)
            uio = { fp->f_mode, fp->offset }
    fops-switch ──────► sockets
   vnodes
          └─► vno_rw(fp, rw, uio)
                vp = fp->f_data        (vnode-Struktur aufbauen)
                iomode = 0 | APPEND | SYNC
          vnode-switch ──────► NFS
                       └─────► specFS
            UFS
                └─► ufs_rw(vp, uio, rw, iomode, ...)
                      ip = VTOI(vp)       (inode-pointer aus vnode besorgen)
                    └─► rwip(ip, uio, rw, iomode)
                   ┌─►    lblock = uio->offset/bsize       (logische Blocknr.)
                   │      pblock = bmap(lblock)            (phys. Blocknr.)
                   │    └─► bread(vp, pblock, bsize)
                   │                 ¦
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

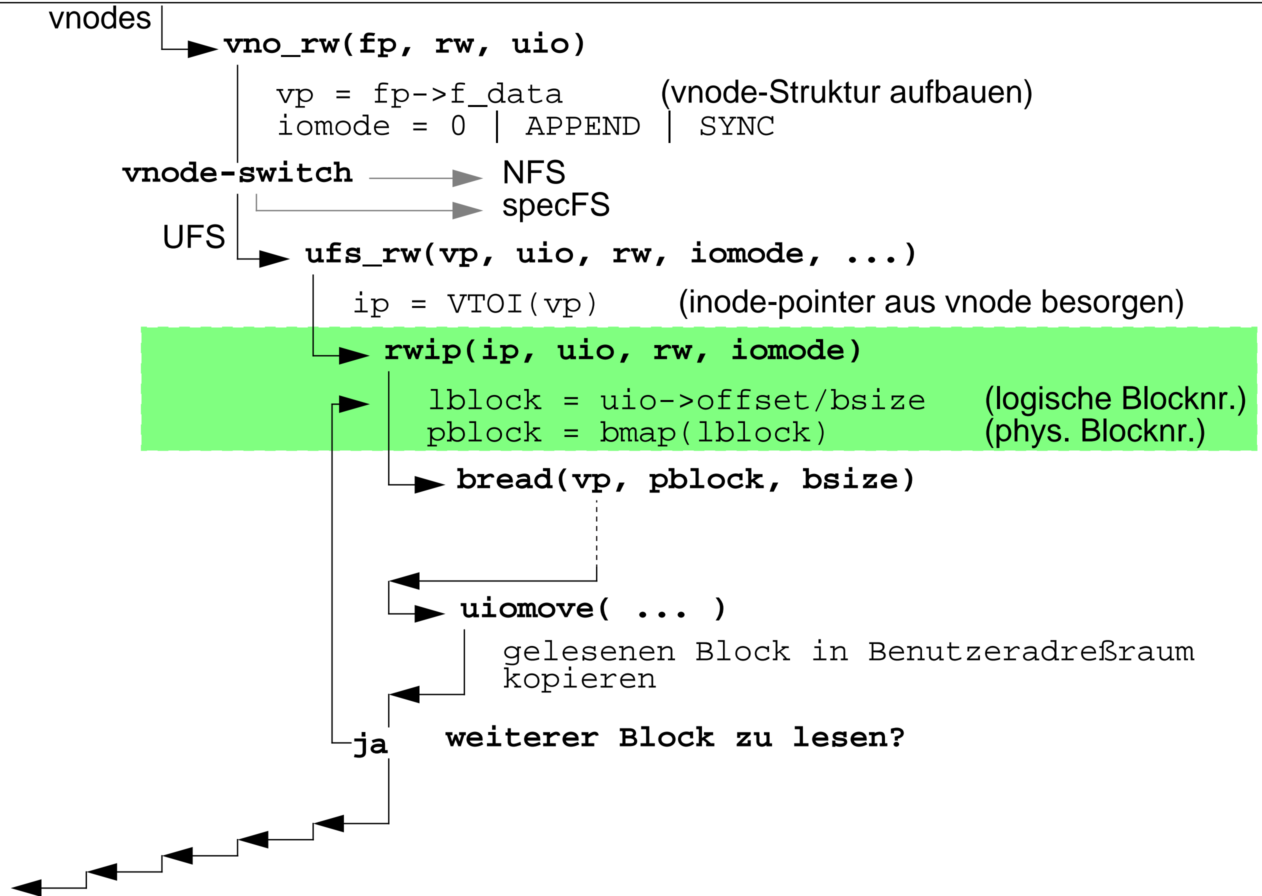
I.9 Dateisystem - Buffer Cache



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

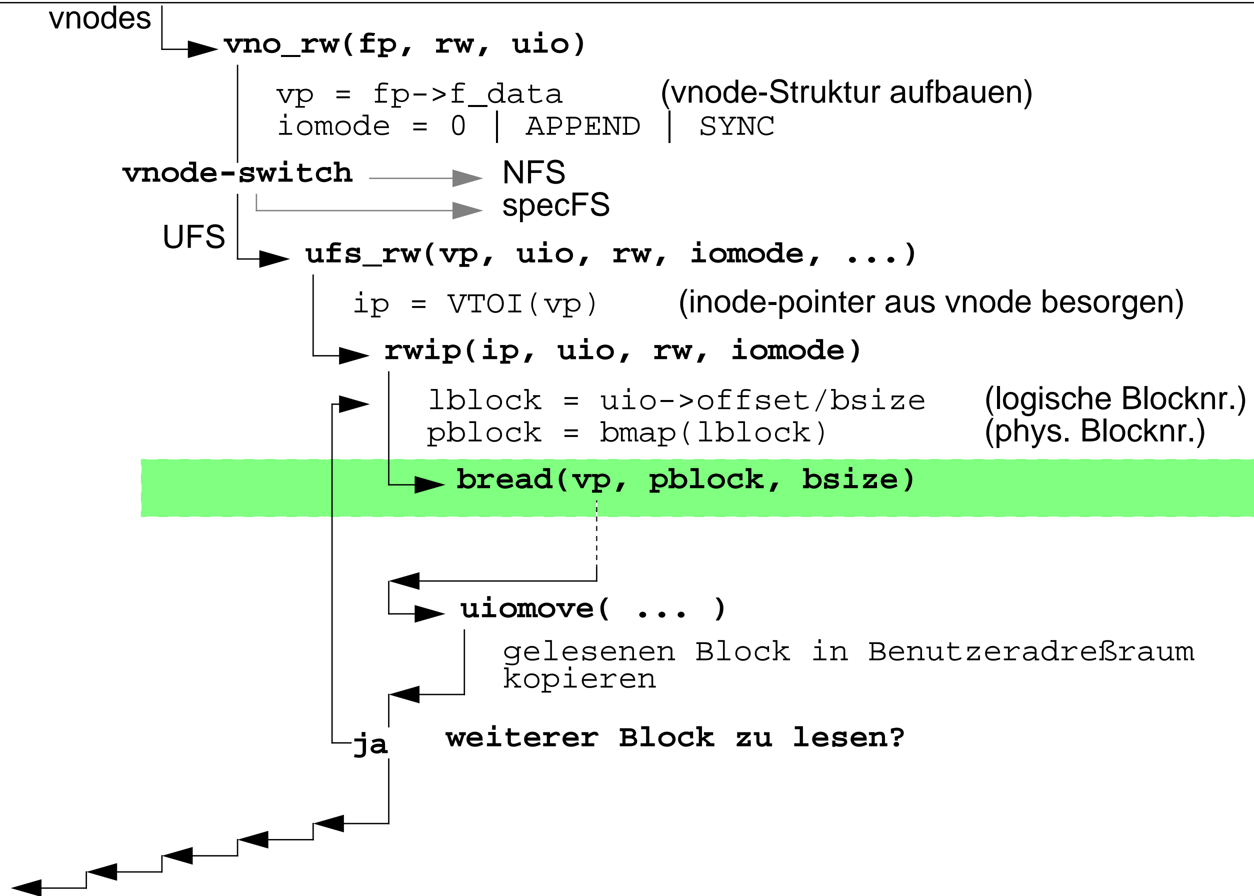
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

I.9 Dateisystem - Buffer Cache



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

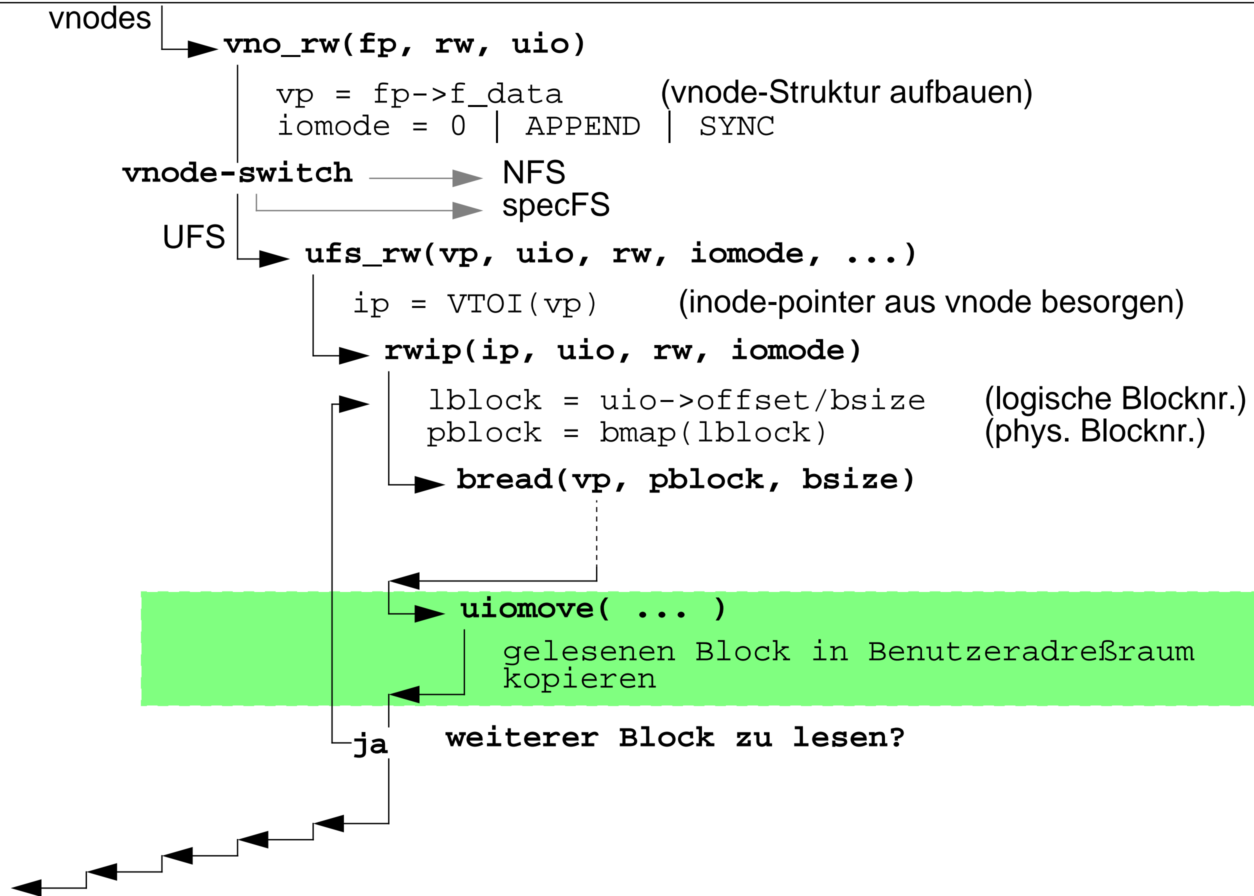
I.9 Dateisystem - Buffer Cache



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

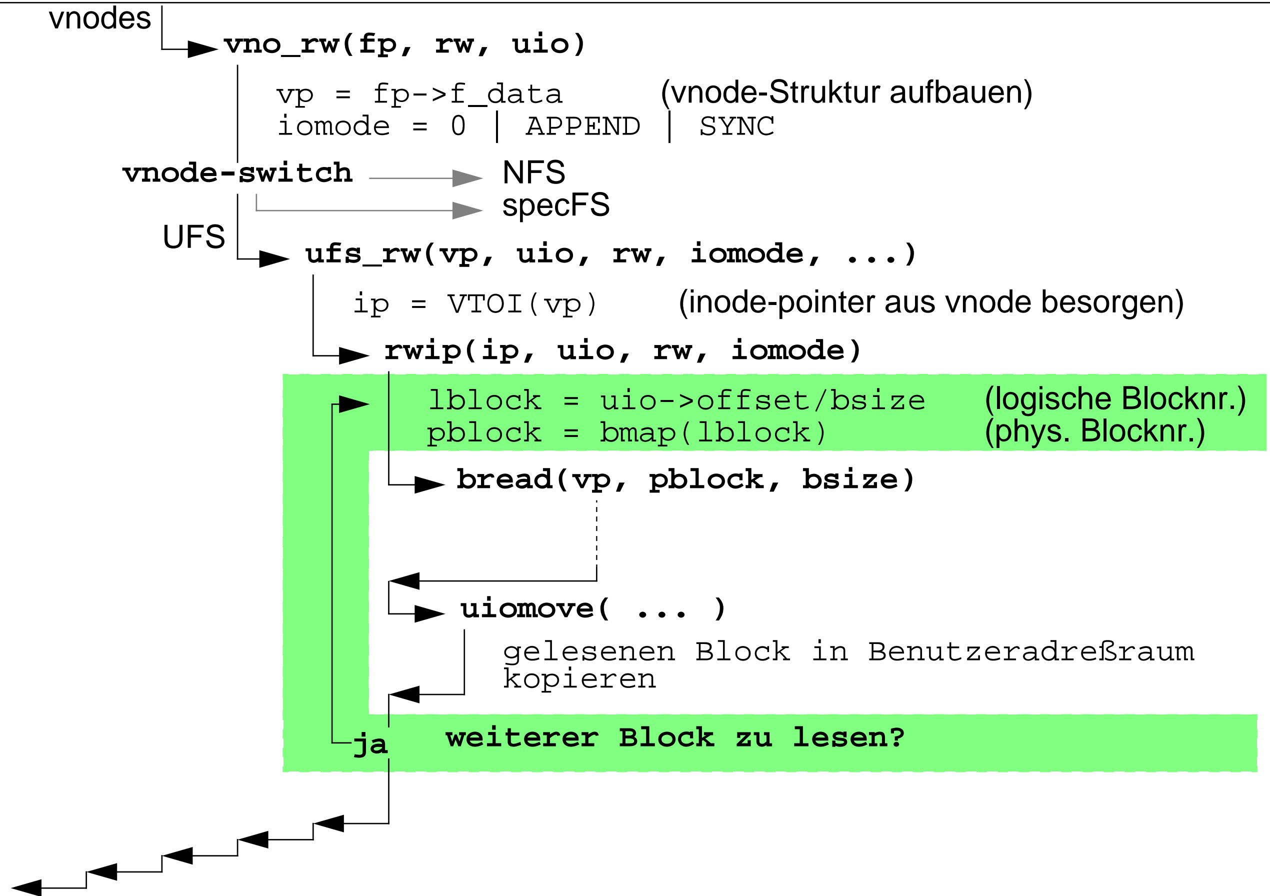
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Dateisystem - Buffer Cache



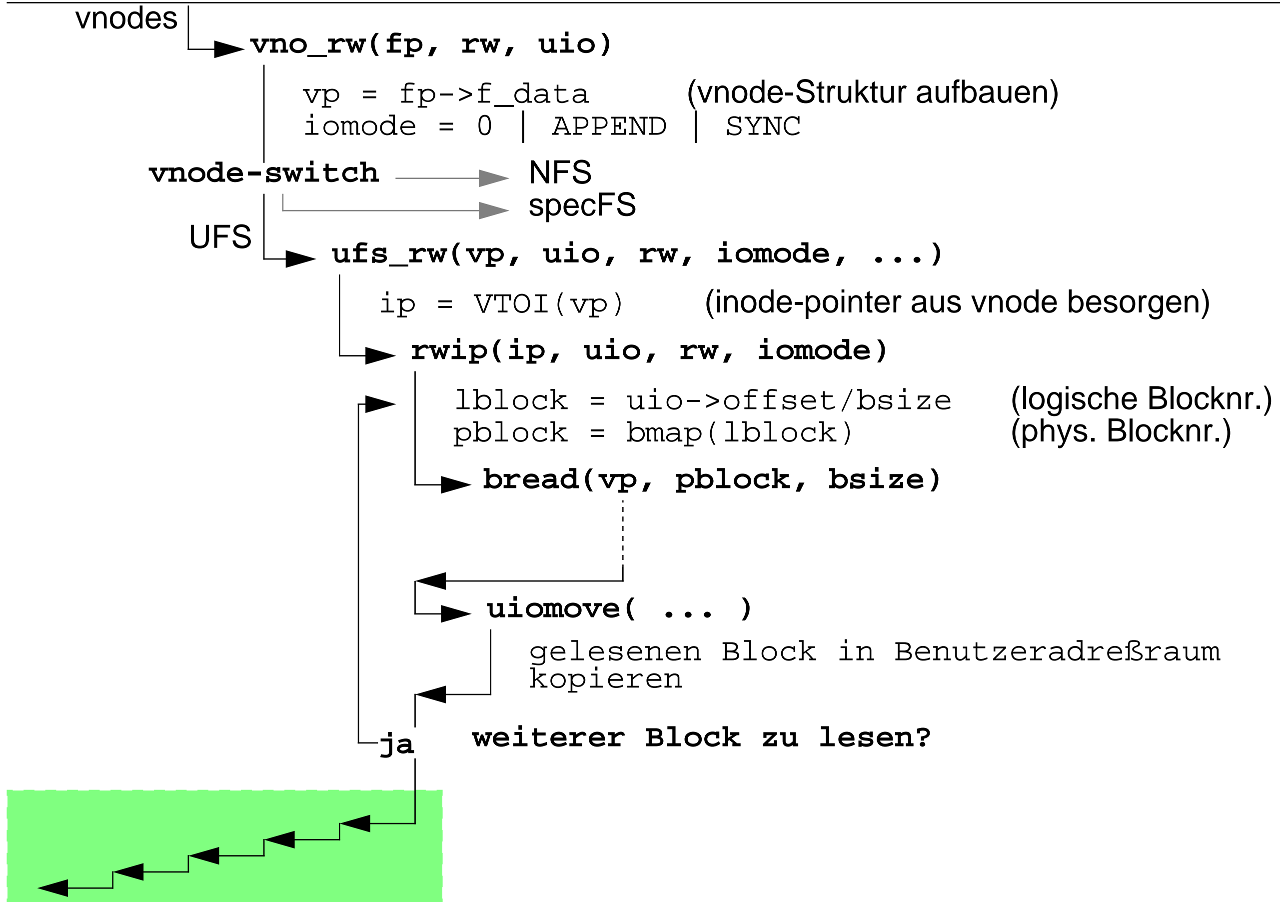
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

I.9 Dateisystem - Buffer Cache



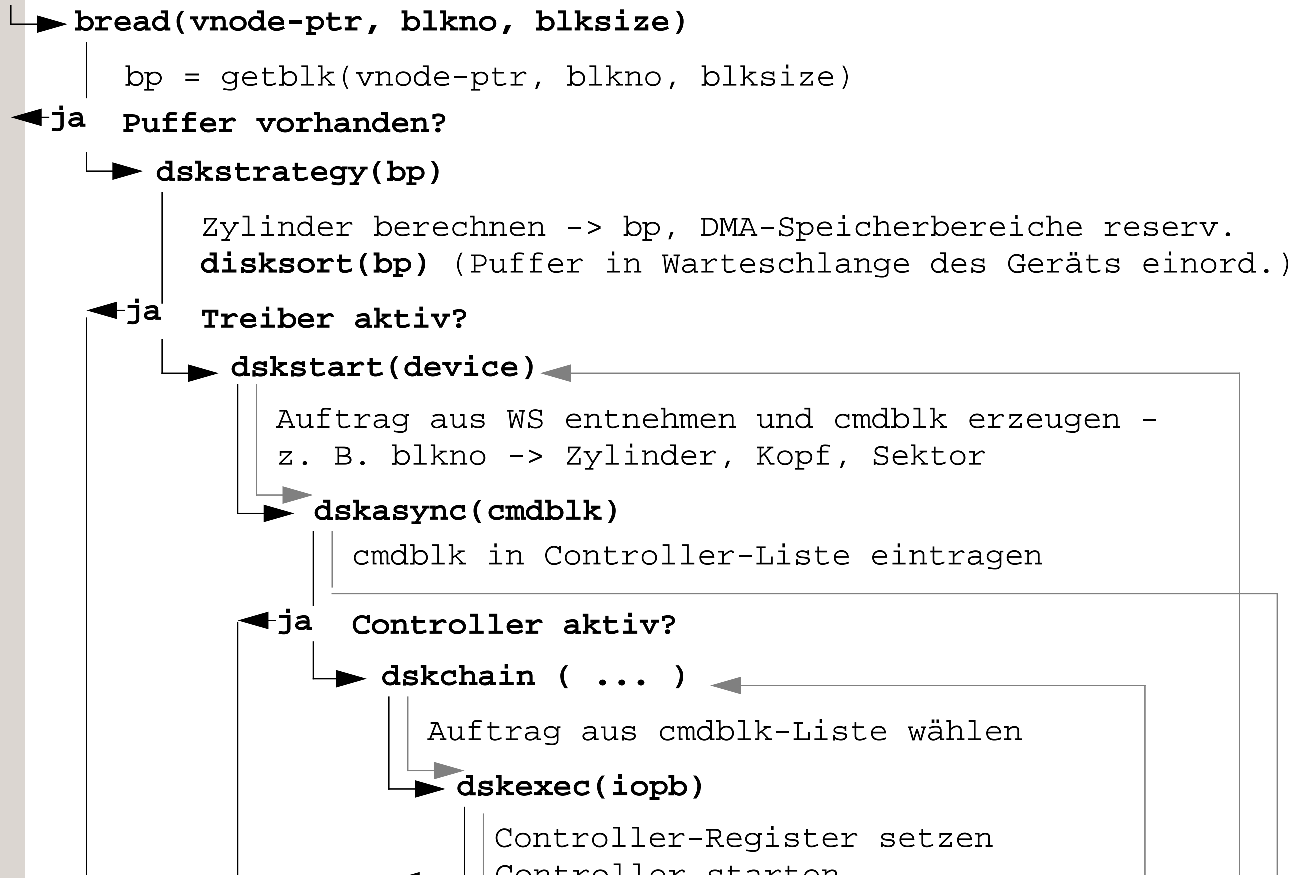
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

I.9 Dateisystem - Buffer Cache



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Kontrollfluß



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Kontrollfluß

```
bread(vnode-ptr, blkno, blksize)
    bp = getblk(vnode-ptr, blkno, blksize)
    Puffer vorhanden?
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Kontrollfluß

```
bread(vnode-ptr, blkno, blksize)
    bp = getblk(vnode-ptr, blkno, blksize)
    Puffer vorhanden?
```


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Kontrollfluß

```
└─► bread(vnode-ptr, blkno, blksize)
      │
      │  bp = getblk(vnode-ptr, blkno, blksize)
◄─ja     Puffer vorhanden?
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Kontrollfluß

```
└──► bread(vnode-ptr, blkno, blksize)
     │
     │    bp = getblk(vnode-ptr, blkno, blksize)
nein Puffer vorhanden?
     └──► dskstrategy(bp)
          │
          │    Zylinder berechnen -> bp, DMA-Speicherbereiche reserv.
          │    disksort(bp) (Puffer in Warteschlange des Geräts einord.)
          │
               Treiber aktiv?
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Kontrollfluß

```
└─► bread(vnode-ptr, blkno, blksize)
      │
      │  bp = getblk(vnode-ptr, blkno, blksize)
nein  Puffer vorhanden?
      └─► dskstrategy(bp)
            │
            │  Zylinder berechnen -> bp, DMA-Speicherbereiche reserv.
            │  disksort(bp) (Puffer in Warteschlange des Geräts einord.)
            │
               Treiber aktiv?
```


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Kontrollfluß

bread(vnode-ptr, blkno, blksize)

bp = getblk(vnode-ptr, blkno, blksize)

nein **Puffer vorhanden?**

dskstrategy(bp)

Zylinder berechnen -> bp, DMA-Speicherbereiche reserv.

disksort(bp) (Puffer in Warteschlange des Geräts einord.)

ja **Treiber aktiv?**

biowait(bp)

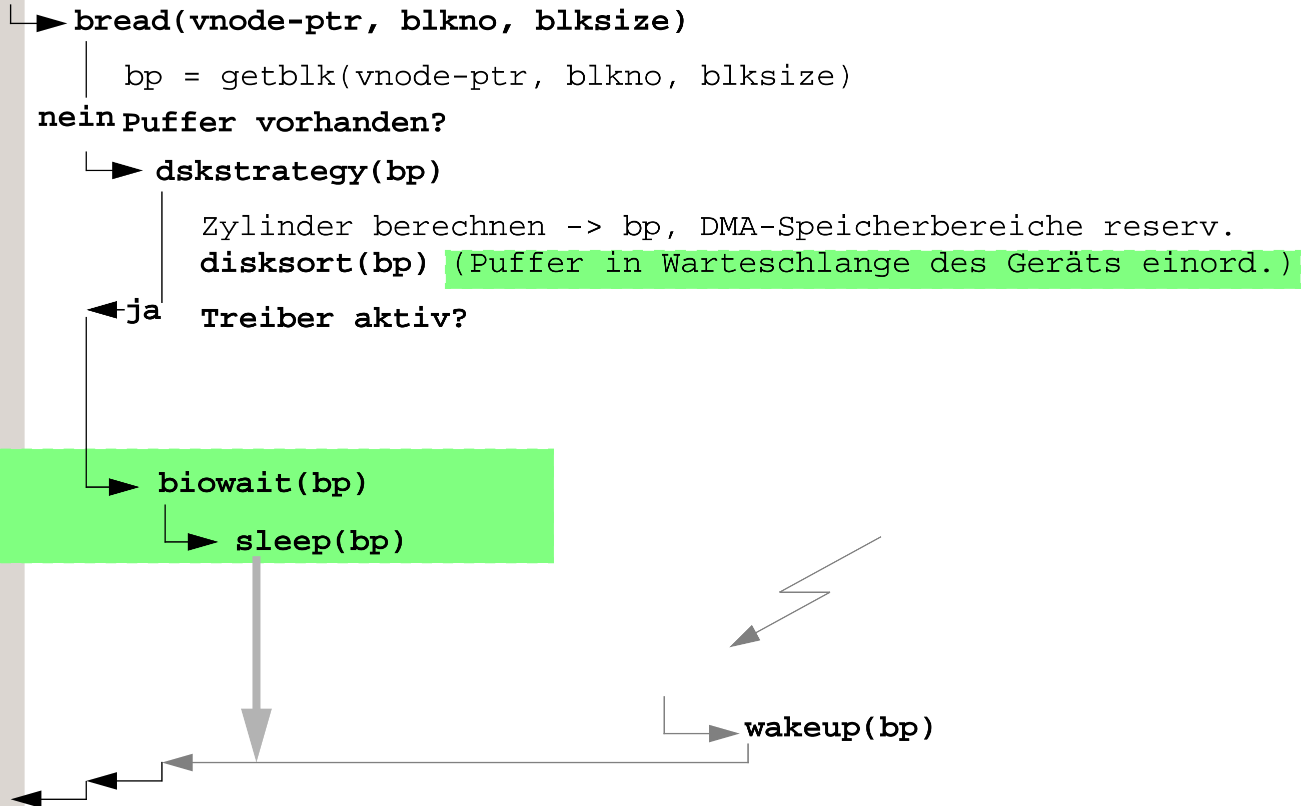
sleep(bp)

wakeup(bp)

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

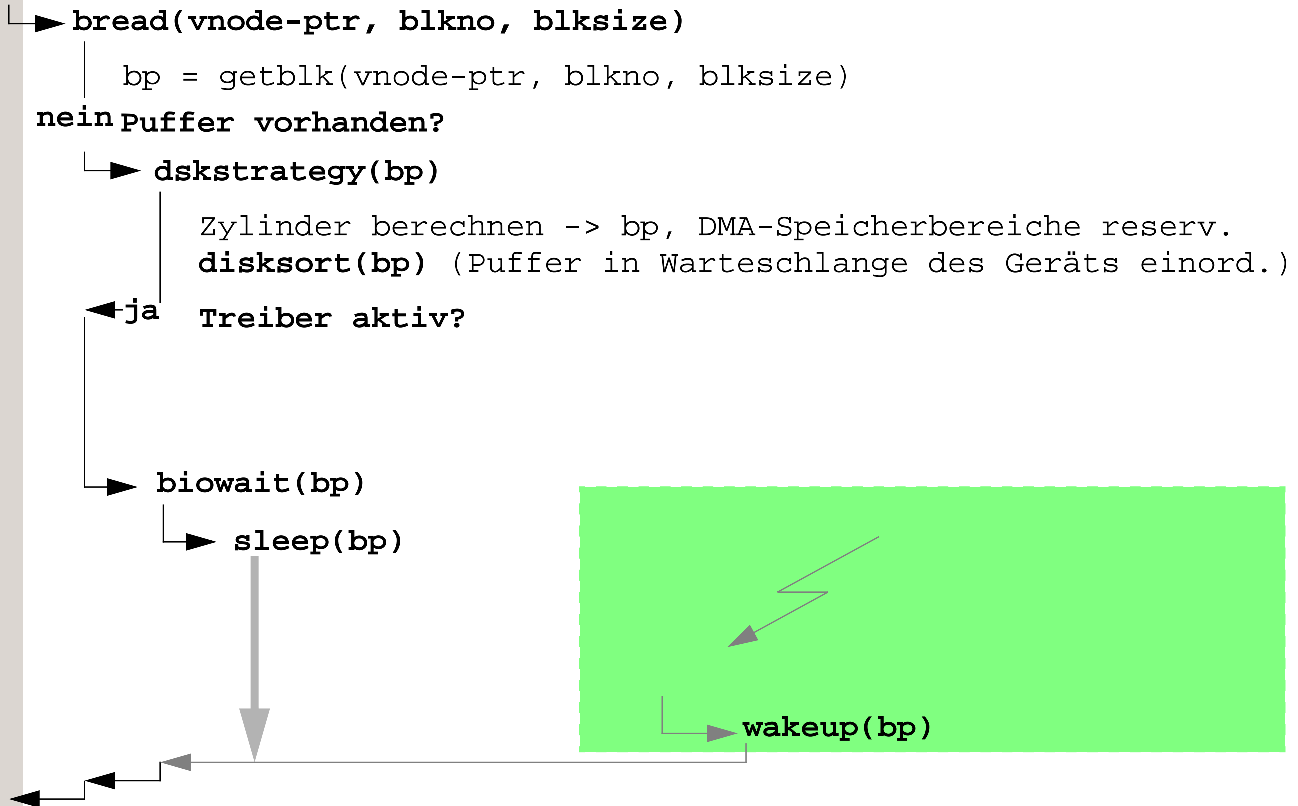
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Kontrollfluß



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

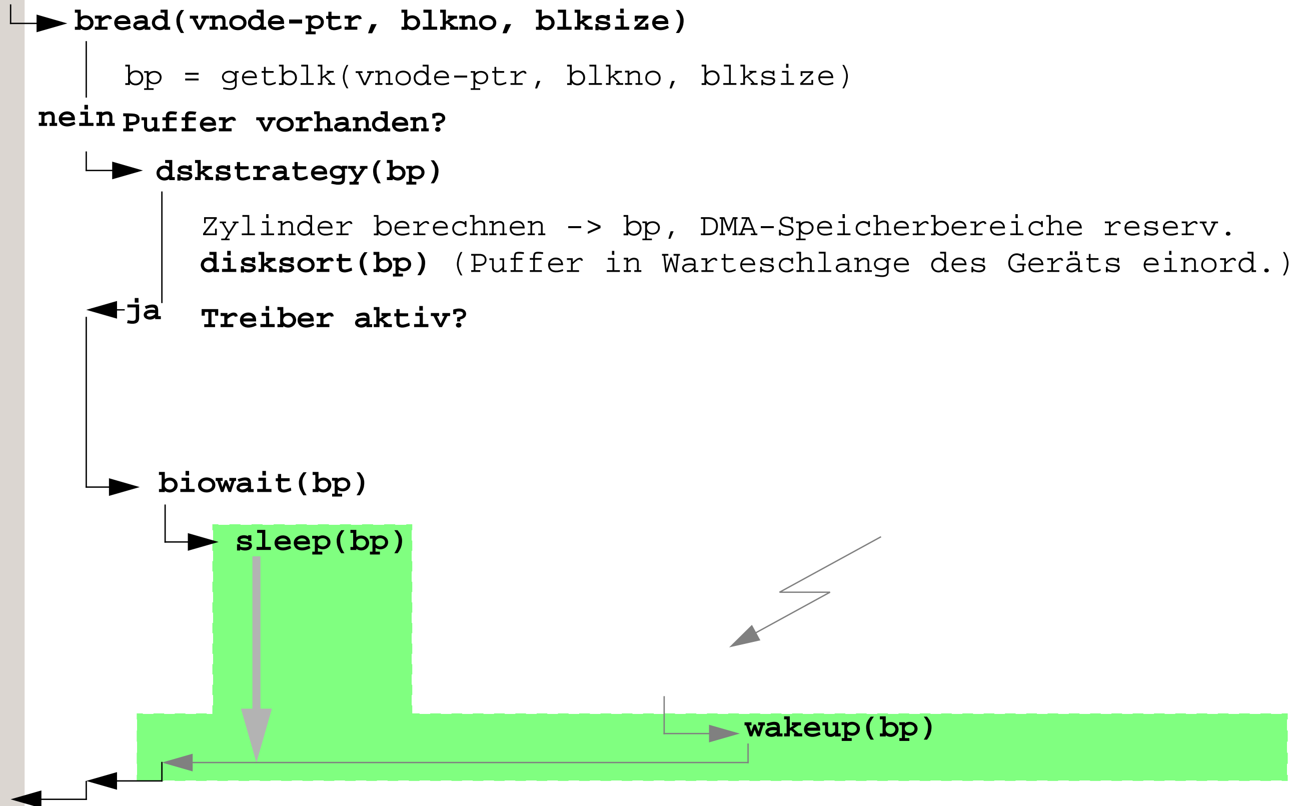
3 Kontrollfluß



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

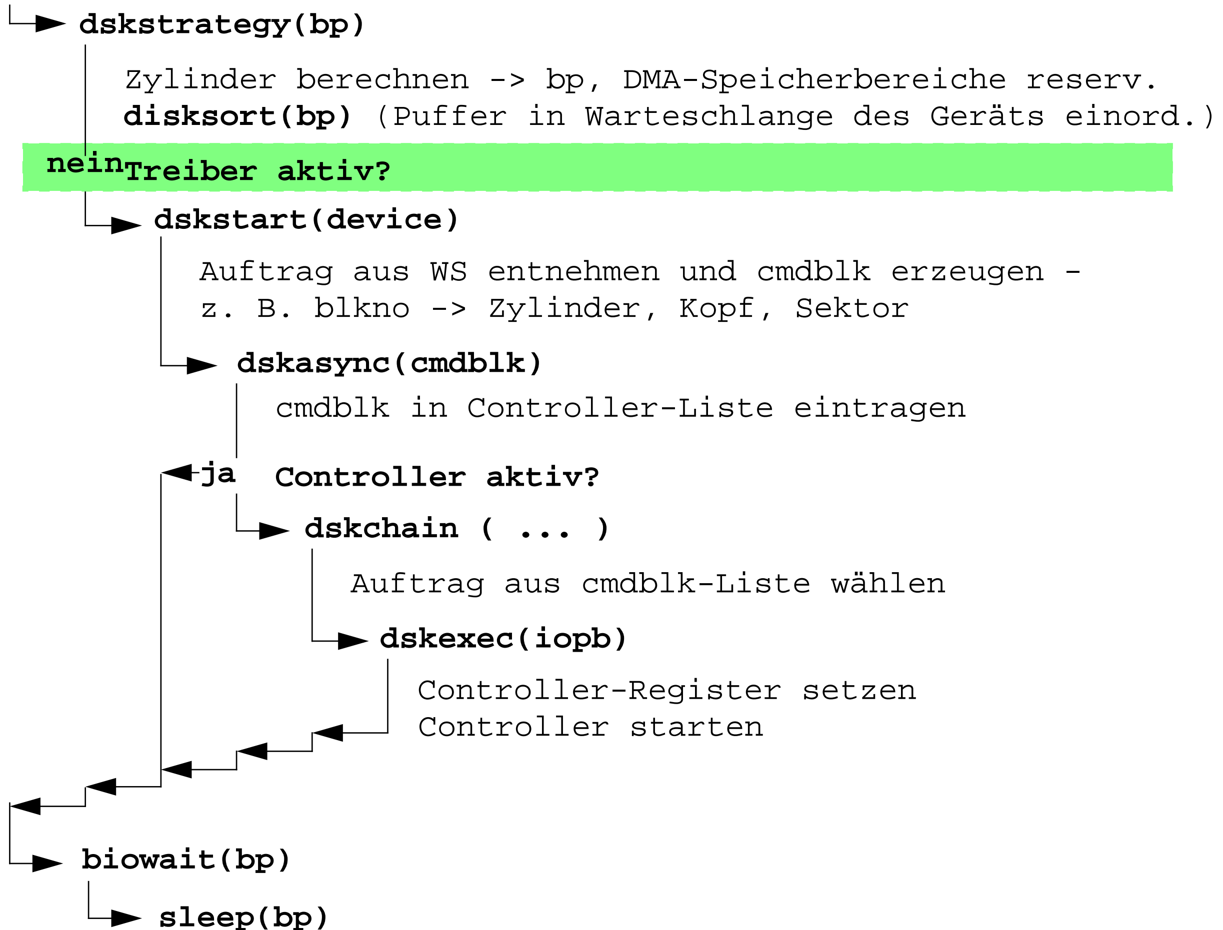
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Kontrollfluß



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

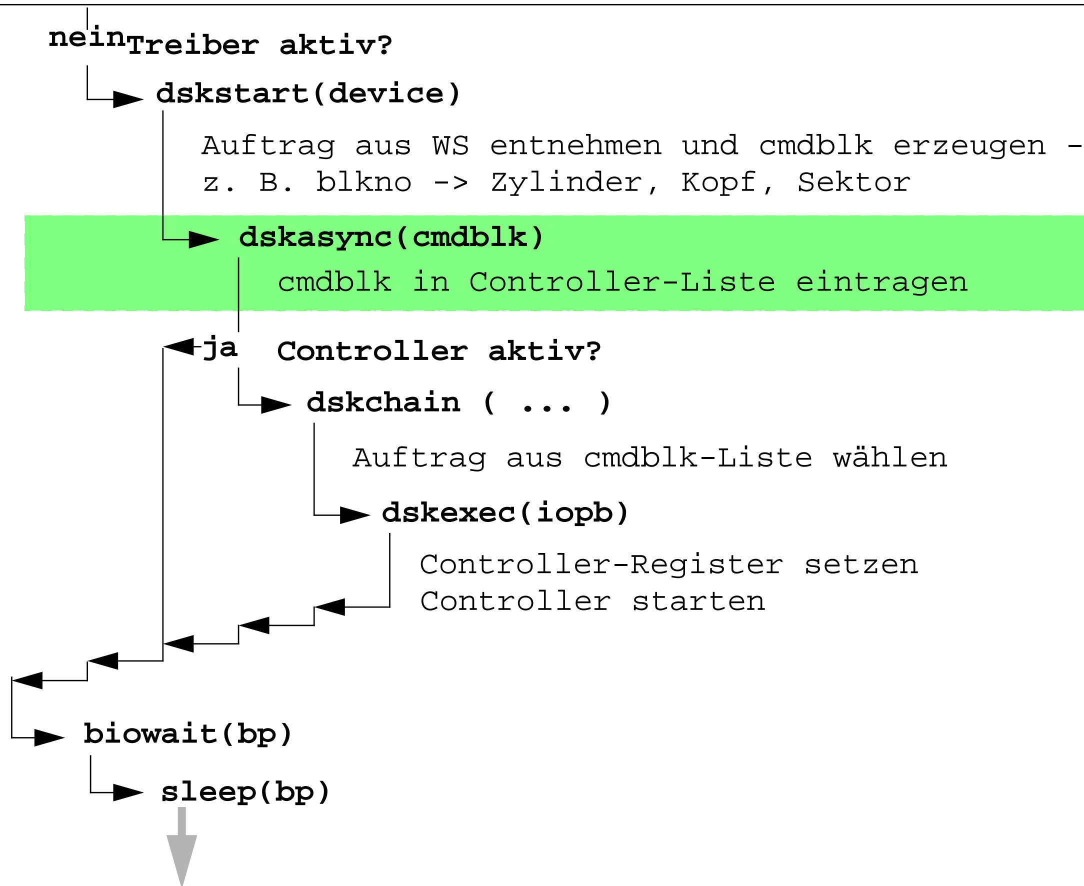
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



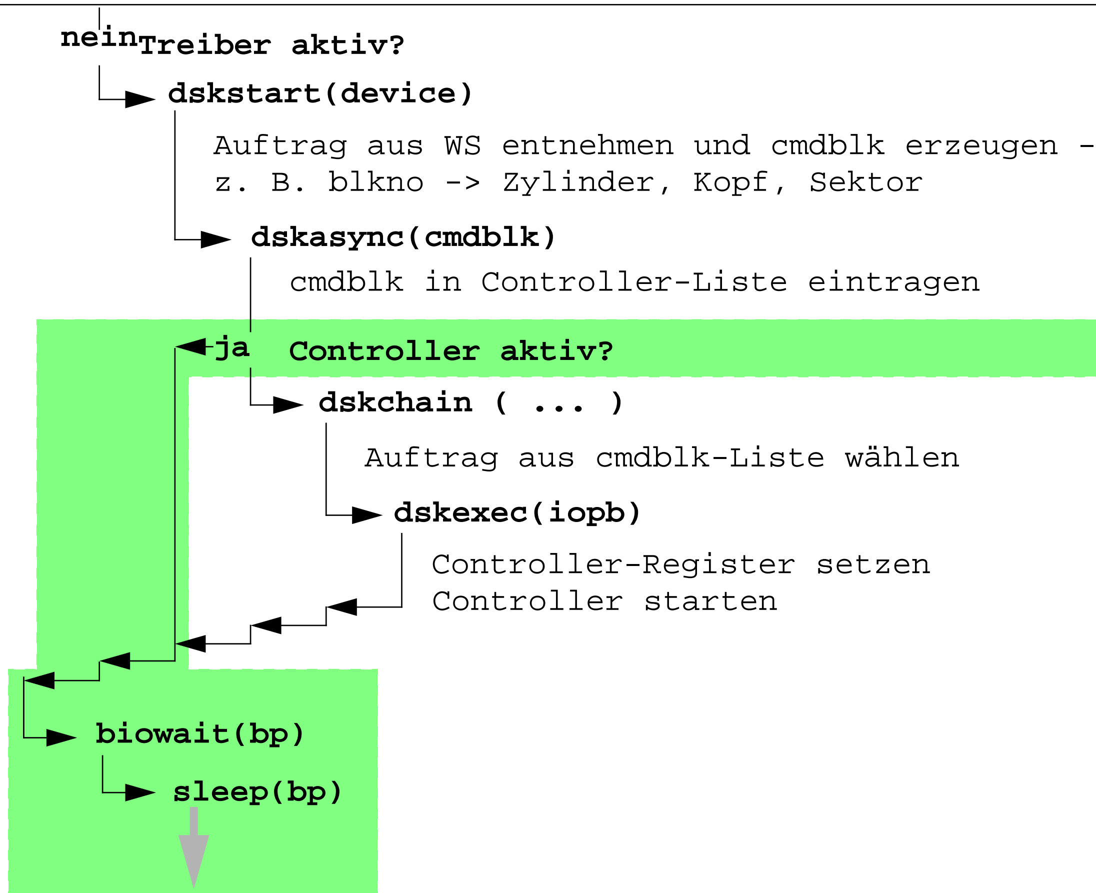
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

```
└─► dskstrategy(bp)
    │
    │  Zylinder berechnen -> bp, DMA-Speicherbereiche reserv.
    │  disksort(bp) (Puffer in Warteschlange des Geräts einord.)
   nein Treiber aktiv?
    └─► dskstart(device)
        │
        │  Auftrag aus WS entnehmen und cmdblk erzeugen -
        │  z. B. blkno -> Zylinder, Kopf, Sektor
        └─► dskasync(cmdblk)
            │
            │  cmdblk in Controller-Liste eintragen
        ◄─ja  Controller aktiv?
            └─► dskchain ( ... )
                │
                │  Auftrag aus cmdblk-Liste wählen
                └─► dskexec(iopb)
                    │
                    │  Controller-Register setzen
                    │  Controller starten

└─► biowait(bp)
    └─► sleep(bp)
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

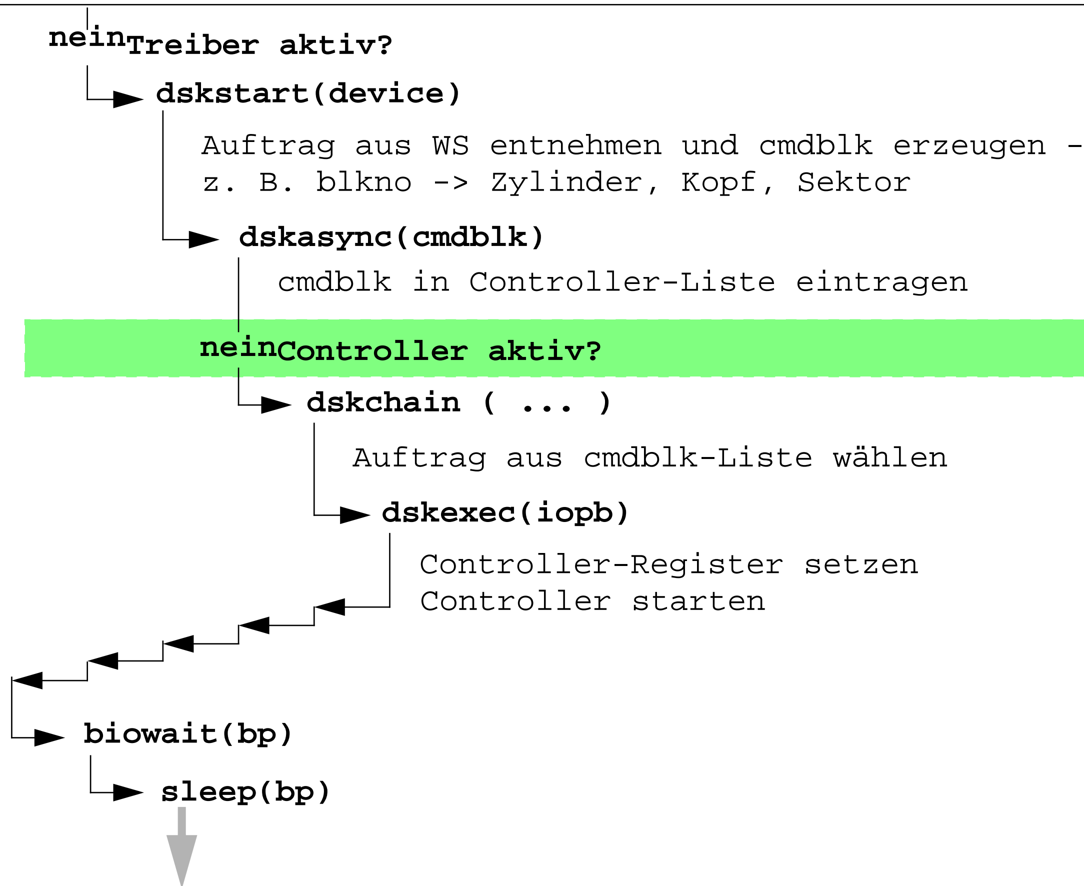


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

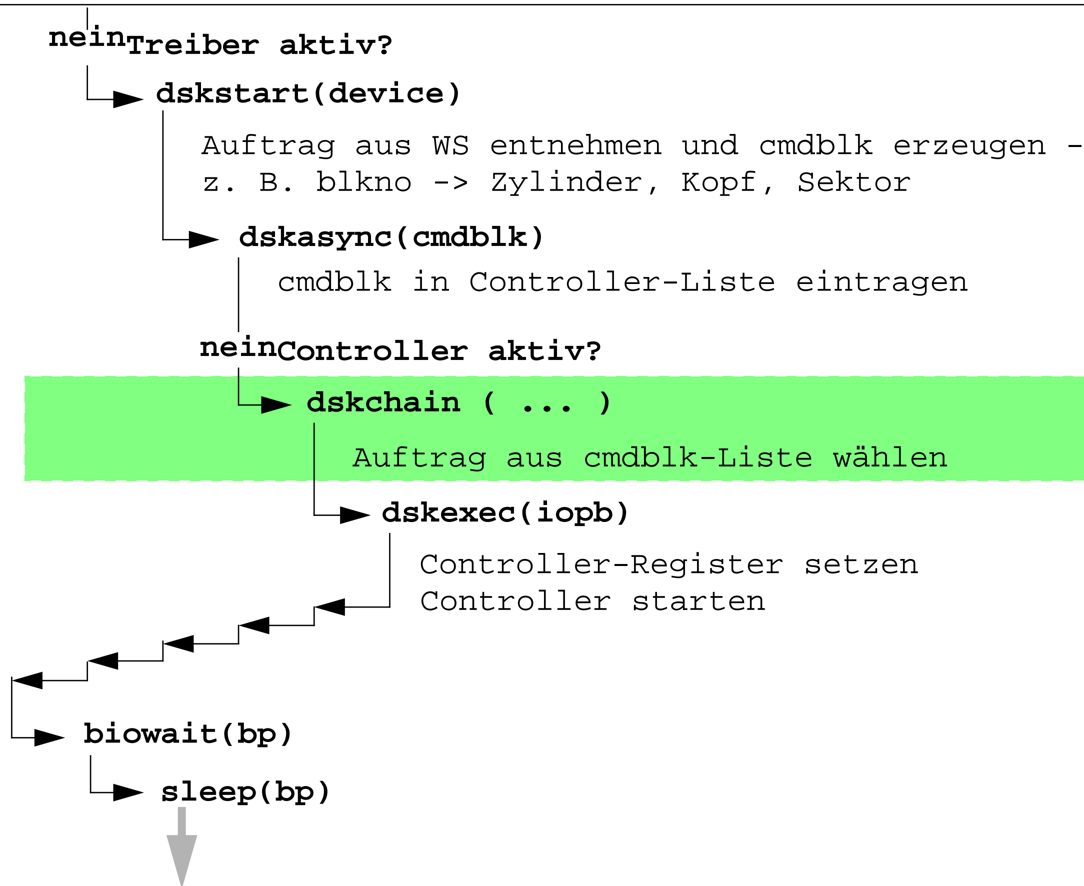
3 Kontrollfluß



© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Kontrollfluß

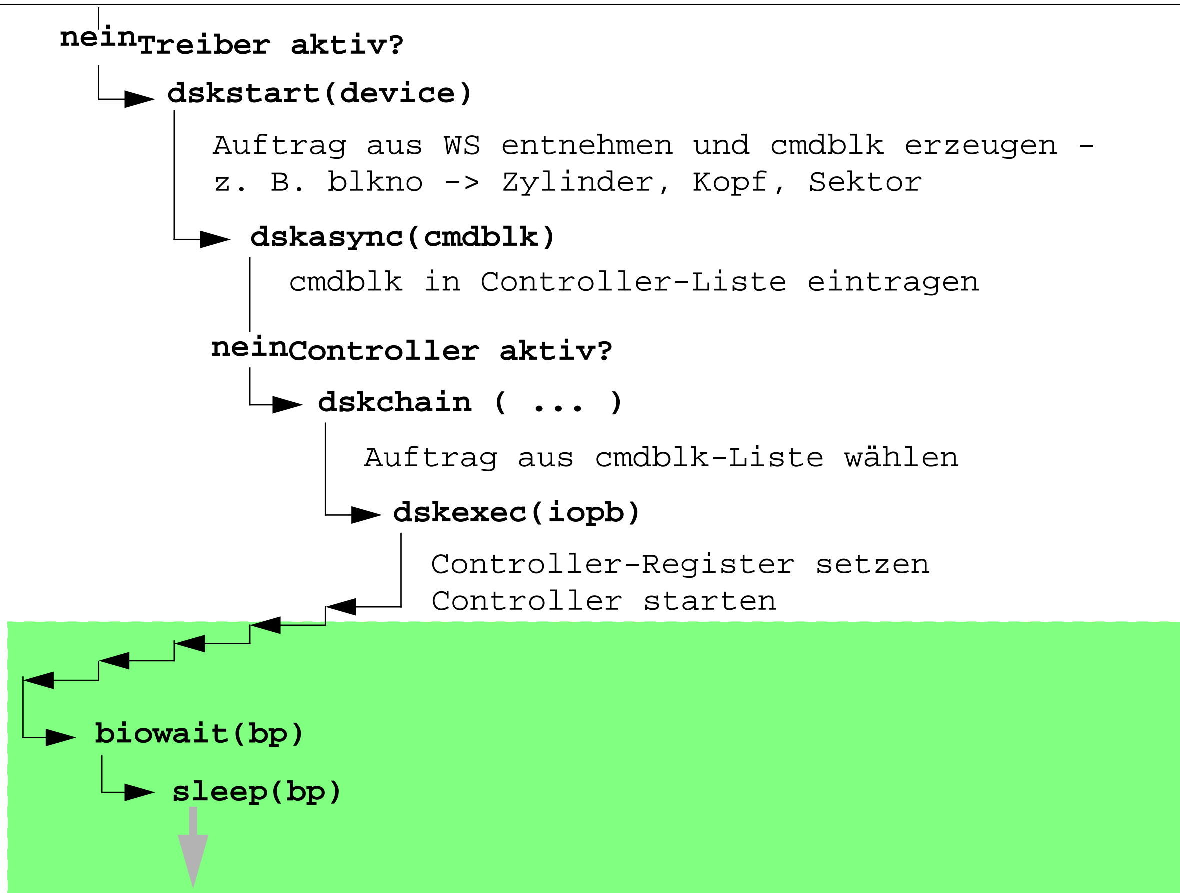


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

```
nein Treiber aktiv?
  └─► dskstart(device)
        Auftrag aus WS entnehmen und cmdblk erzeugen -
        z. B. blkno -> Zylinder, Kopf, Sektor
        └─► dskasync(cmdblk)
              cmdblk in Controller-Liste eintragen
        nein Controller aktiv?
              └─► dskchain ( ... )
                    Auftrag aus cmdblk-Liste wählen
                    └─► dskexec(iopb)
                          Controller-Register setzen
                          Controller starten
└─► biowait(bp)
      └─► sleep(bp)
```


© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

dskstart(device)

Auftrag aus WS entnehmen und cmdblk erzeugen -
z. B. blkno -> Zylinder, Kopf, Sektor

dskasync(cmdblk)

cmdblk in Controller-Liste eintragen

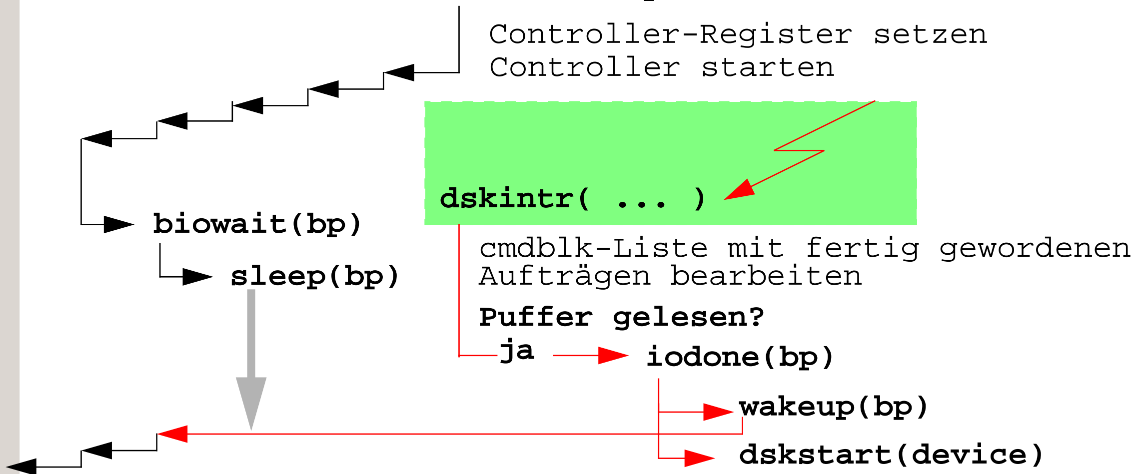
Controller aktiv?

dskchain (...)

Auftrag aus cmdblk-Liste wählen

dskexec(iopb)

Controller-Register setzen
Controller starten



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Kontrollfluß

dskstart(device)

Auftrag aus WS entnehmen und cmdblk erzeugen -
z. B. blkno -> Zylinder, Kopf, Sektor

dskasync(cmdblk)

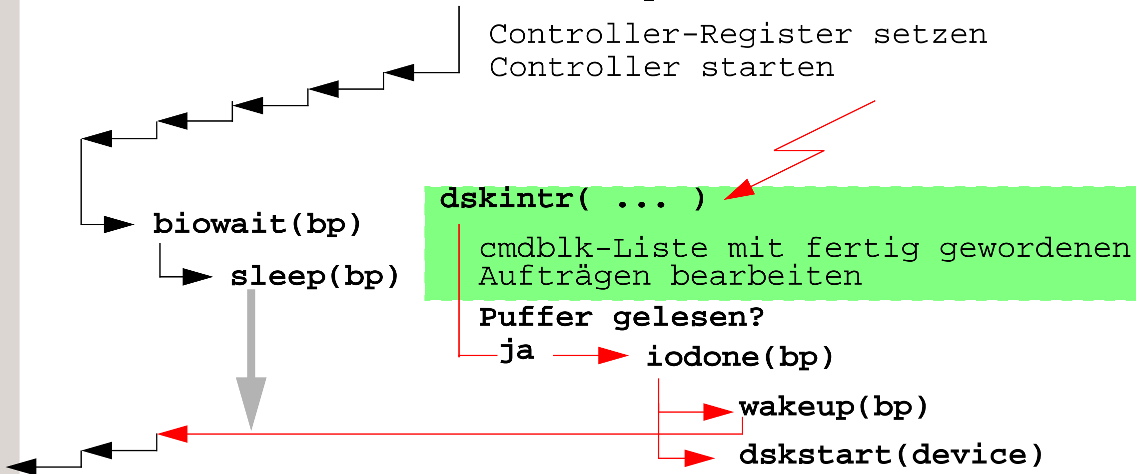
cmdblk in Controller-Liste eintragen

Controller aktiv?

dskchain (...)

Auftrag aus cmdblk-Liste wählen

dskexec(iopb)



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

dskstart(device)

Auftrag aus WS entnehmen und cmdblk erzeugen - z. B. blkno -> Zylinder, Kopf, Sektor

dskasync(cmdblk)

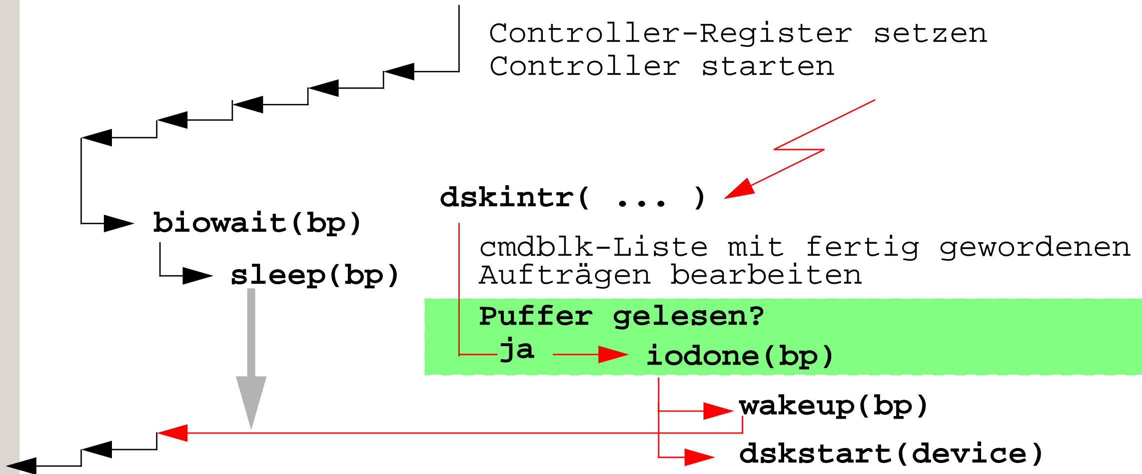
cmdblk in Controller-Liste eintragen

Controller aktiv?

dskchain (...)

Auftrag aus cmdblk-Liste wählen

dskexec(iopb)



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

```
dskstart(device)
    Auftrag aus WS entnehmen und cmdblk erzeugen -
    z. B. blkno -> Zylinder, Kopf, Sektor
      dskasync(cmdblk)
        cmdblk in Controller-Liste eintragen

        Controller aktiv?
          dskchain ( ... )
              Auftrag aus cmdblk-Liste wählen
              dskexec(iopb)
                Controller-Register setzen
                Controller starten

             dskintr( ... )
biowait(bp)
    sleep(bp)    cmdblk-Liste mit fertig gewordenen
                 Aufträgen bearbeiten
                 Puffer gelesen?
                  ja  ->  iodone(bp)
                             wakeup(bp)
                             dskstart(device)
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

dskstart(device)

Auftrag aus WS entnehmen und cmdblk erzeugen -
z. B. blkno -> Zylinder, Kopf, Sektor

dskasync(cmdblk)

cmdblk in Controller-Liste eintragen

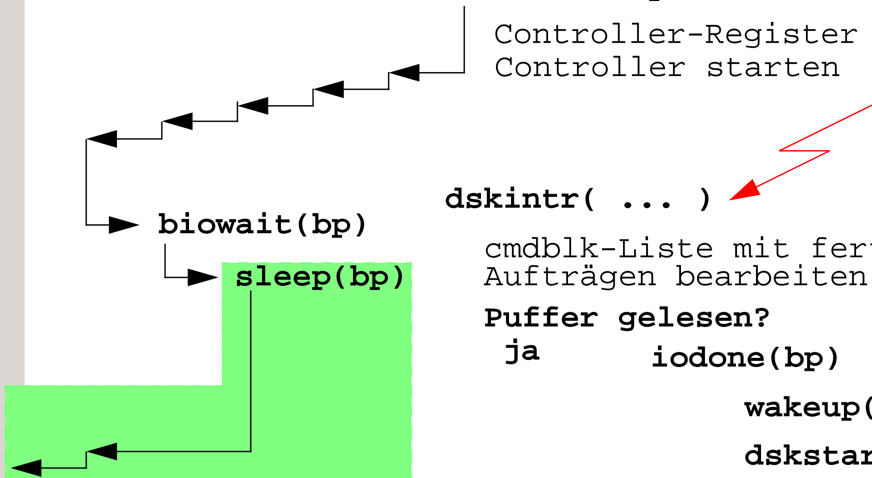
Controller aktiv?

dskchain (...)

Auftrag aus cmdblk-Liste wählen

dskexec(iopb)

Controller-Register setzen
Controller starten



dskintr(...)

cmdblk-Liste mit fertig gewordenen Aufträgen bearbeiten

Puffer gelesen?

ja **iodone(bp)**

wakeup(bp)

dskstart(device)

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Kontrollfluß

dskstart(device)

Auftrag aus WS entnehmen und cmdblk erzeugen - z. B. blkno -> Zylinder, Kopf, Sektor

dskasync(cmdblk)

cmdblk in Controller-Liste eintragen

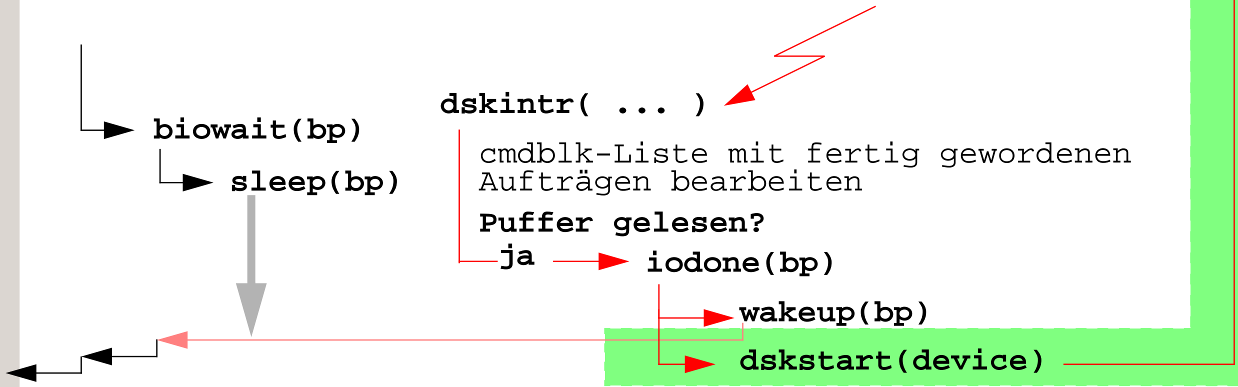
Controller aktiv?

dskchain (...)

Auftrag aus cmdblk-Liste wählen

dskexec(iopb)

Controller-Register setzen
Controller starten



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Kontrollfluß

dskstart(device)

Auftrag aus WS entnehmen und cmdblk erzeugen - z. B. blkno -> Zylinder, Kopf, Sektor

dskasync(cmdblk)

cmdblk in Controller-Liste eintragen

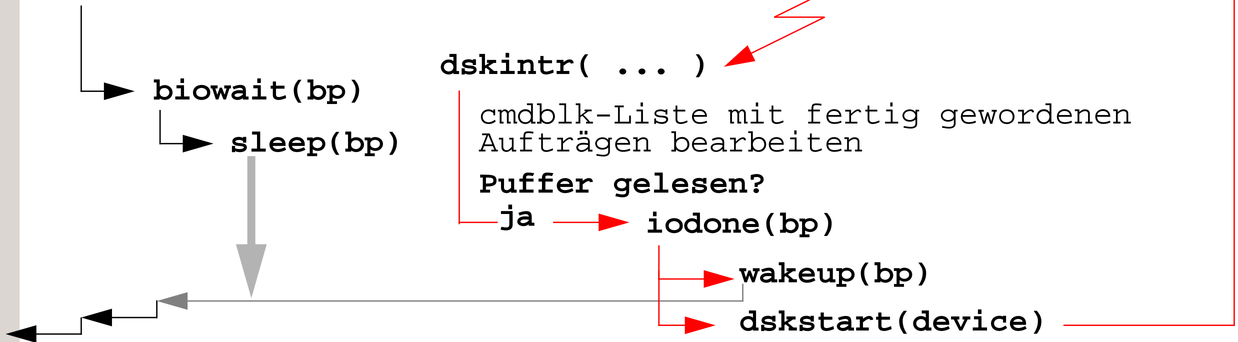
Controller aktiv?

dskchain (...)

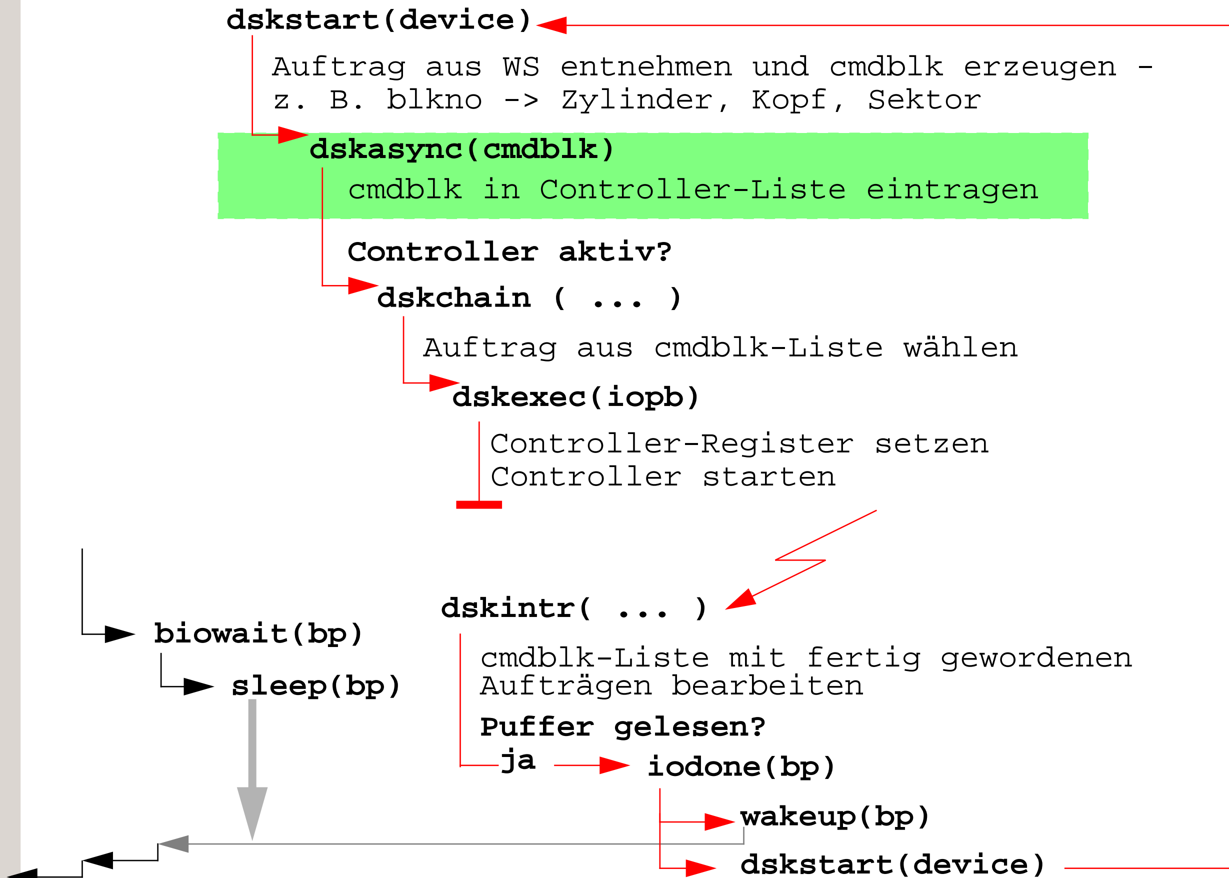
Auftrag aus cmdblk-Liste wählen

dskexec(iopb)

Controller-Register setzen
Controller starten



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Kontrollfluß

```
dskstart(device)
    Auftrag aus WS entnehmen und cmdblk erzeugen -
    z. B. blkno -> Zylinder, Kopf, Sektor
  dskasync(cmdblk)
      cmdblk in Controller-Liste eintragen

      Controller aktiv?
    dskchain ( ... )
        Auftrag aus cmdblk-Liste wählen
      dskexec(iopb)
          Controller-Register setzen
          Controller starten

biowait(bp)            dskintr( ... )
  sleep(bp)              cmdblk-Liste mit fertig gewordenen
                         Aufträgen bearbeiten
                         Puffer gelesen?
                          ja  ->  iodone(bp)
                                    wakeup(bp)
                                    dskstart(device)
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

dskstart(device)

Auftrag aus WS entnehmen und cmdblk erzeugen -
z. B. blkno -> Zylinder, Kopf, Sektor

dskasync(cmdblk)

cmdblk in Controller-Liste eintragen

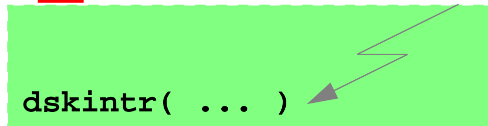
Controller aktiv?

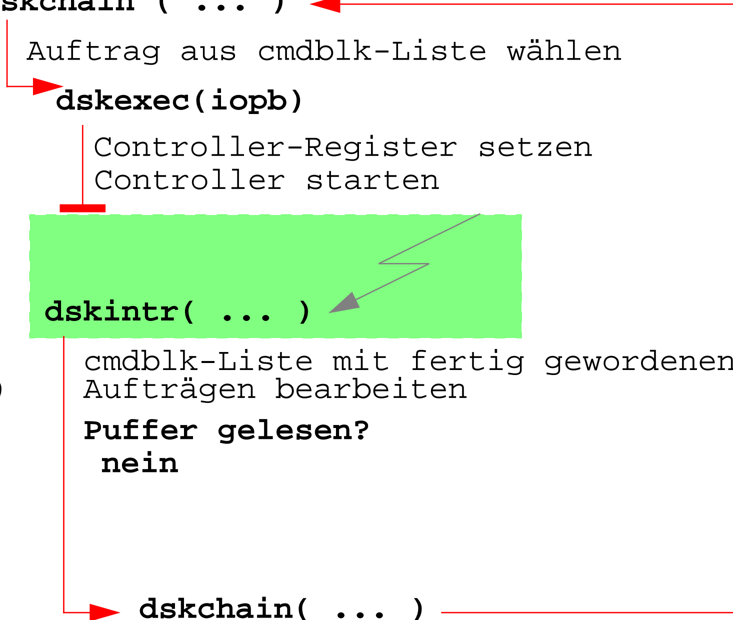
dskchain (...)

Auftrag aus cmdblk-Liste wählen

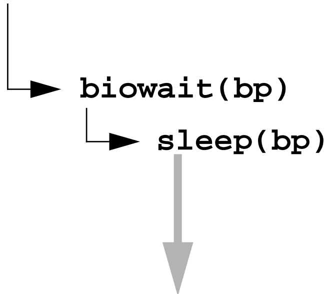
dskexec(iopb)

Controller-Register setzen
Controller starten





dskintr(...)



biowait(bp)

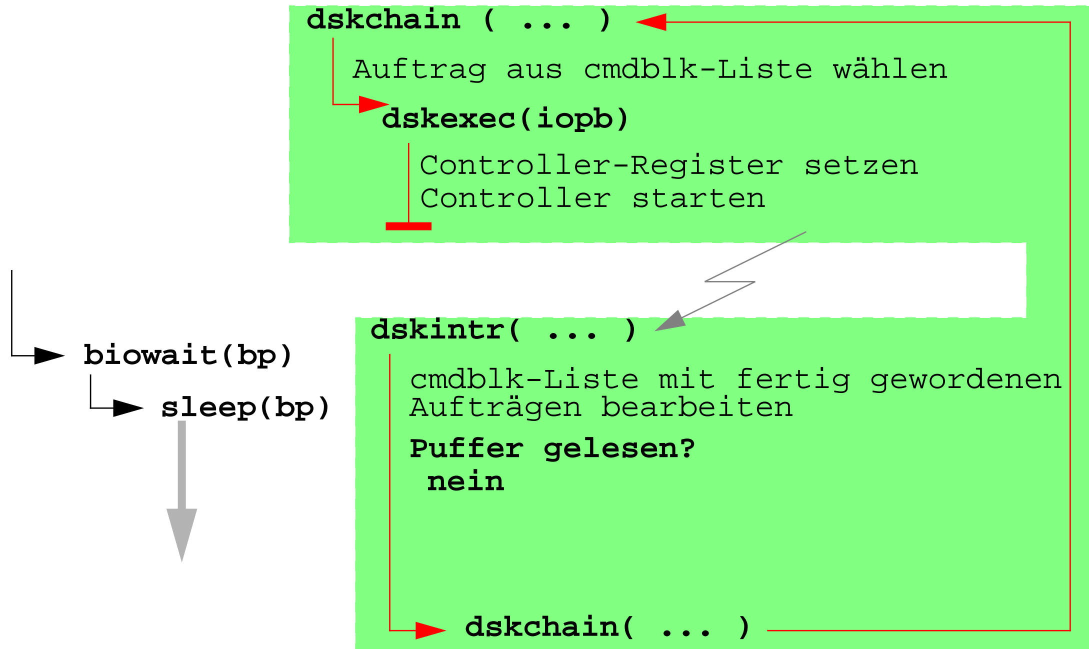
sleep(bp)

cmdblk-Liste mit fertig gewordenen Aufträgen bearbeiten

Puffer gelesen?
nein

dskchain(...)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

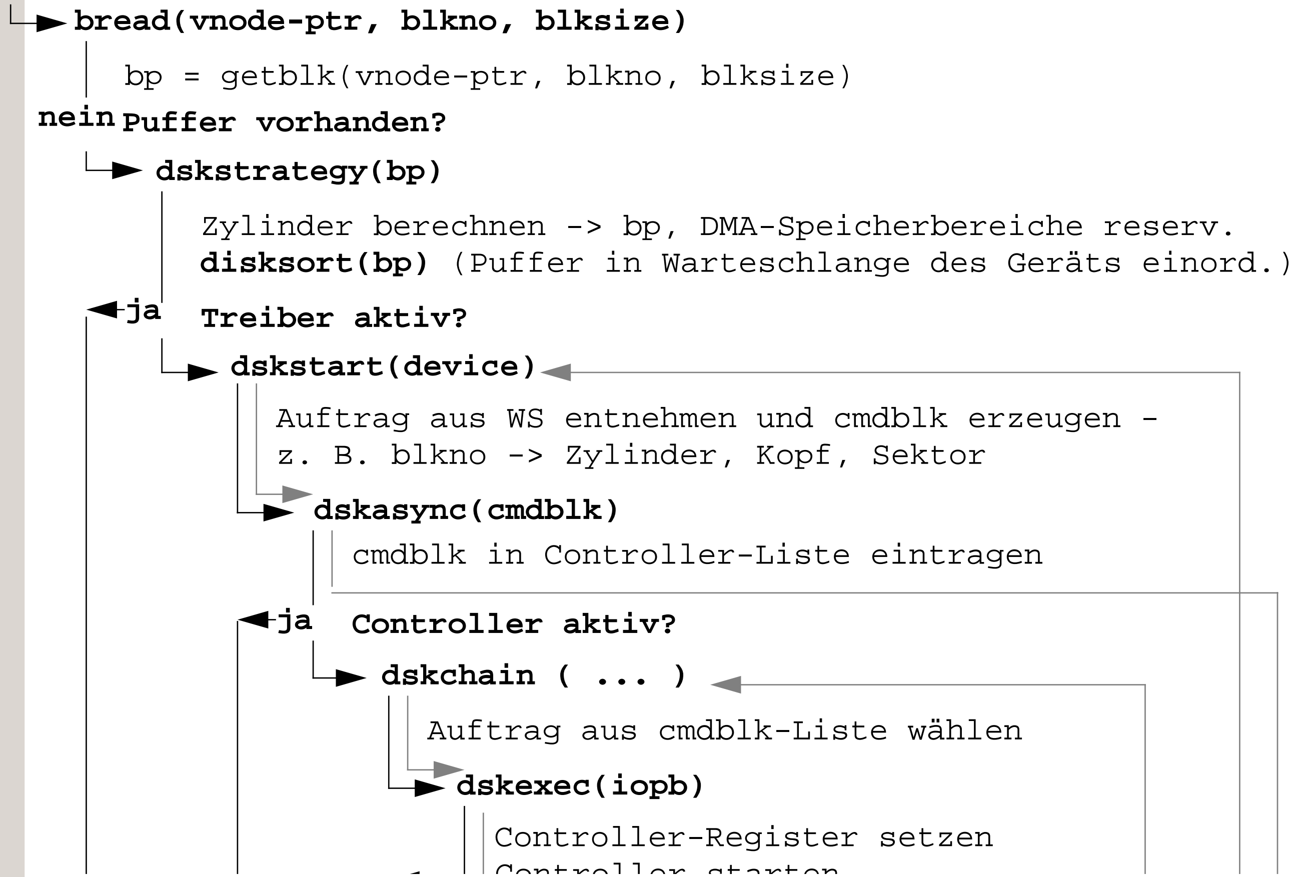
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1999

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Kontrollfluß



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.