



Betriebssysteme I

WS 2019/2020

Roland Wismüller
Betriebssysteme / verteilte Systeme
roland.wismueller@uni-siegen.de
Tel.: 0271/740-4050, Büro: H-B 8404

Stand: 14. November 2019



Betriebssysteme I

WS 2019/2020

1 Einführung

1 Einführung ...



- ➔ Aufgaben eines Betriebssystems
- ➔ Historische Entwicklung der Betriebssysteme
- ➔ Arten von Betriebssystemen
- ➔ Überblick Computer-Hardware
- ➔ Grundlegende Betriebssystemkonzepte
- ➔ Systemaufrufe, Dienste eines Betriebssystems

- ➔ Tanenbaum 1.1 - 1.4.3, 1.5, 1.6

1 Einführung ...

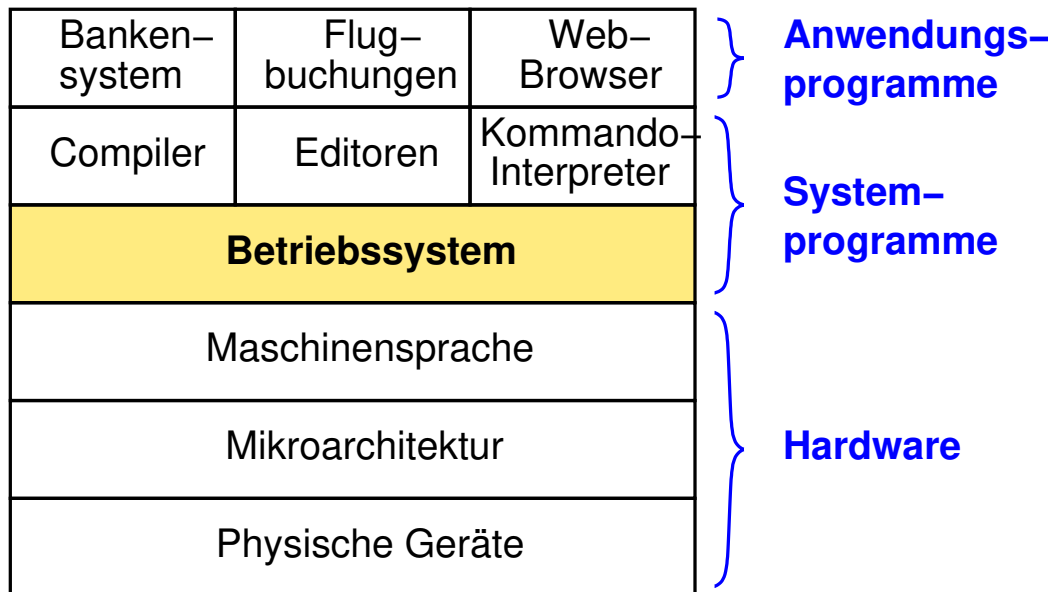


Ein Rechensystem besteht aus:

- ➔ Anwendungssoftware: zur Lösung bestimmter Probleme, z.B.:
 - ➔ Textverarbeitung, Adreßbuch, Datenbank, ...
 - ➔ WWW-Browser, Spiele, ...
 - ➔ Wettervorhersage, CAD, ...
 - ➔ Steuerung eines Kraftwerks, ...
- ➔ Systemsoftware: unterstützt Anwendungen, z.B.:
 - ➔ Übersetzer: erstellt Maschinenprogramme
 - ➔ Betriebssystem: unterstützt **laufende** Anwendungen
 - ➔ Dateimanager (z.B. Windows-Explorer), ...
- ➔ Hardware



Einordnung des Betriebssystems



1.1 Aufgaben eines Betriebssystems



Was soll ein Betriebssystem leisten?

- ➔ Erweiterung (Veredelung) der Hardware
- ➔ Abstraktion der Hardware
- ➔ Verwaltung von Betriebsmitteln

Abkürzung: **BS** = Betriebssystem



Erweiterung / Veredelung der Hardware

- ➔ Hardware muß billig, schnell und zuverlässig sein
 - Beschränkung auf das absolut notwendige
 - Folge: schwierige Programmierung
- ➔ BS stellt komplexe Funktionen bereit, die die Anwendungsprogramme verwenden können
 - Beispiel: Schreiben auf Festplatte
 - BS findet automatisch freie Blöcke auf der Platte, legt Verwaltungsinformation an
 - interne Struktur der Platte (Anzahl Köpfe, Zylinder, Sektoren, etc.) für Anwendung nicht mehr wichtig
 - Folge: erhebliche Vereinfachung der Programmierung



Abstraktion der Hardware

- ➔ Rechner sind trotz ähnlicher Architektur im Detail sehr unterschiedlich, z.B.:
 - Einteilung des Adreßraums (Speicher, E/A-Controller)
 - verschiedenste E/A-Controller und Geräte
- ➔ Fallunterscheidung wird vom BS vorgenommen
- ➔ BS realisiert einheitliche Sicht für Anwendungen
- ➔ Beispiel: Dateien abstrahieren Externspeicher
 - für Anwendungen kein Unterschied zwischen Festplatte, Diskette, CD-ROM, USB-Stick, Netzlaufwerk, ...
 - UNIX: selbst Drucker etc. wie Dateien behandelt
- ➔ BS realisiert eine **virtuelle Maschine**



Verwaltung von Betriebsmitteln

- ➔ **Betriebsmittel (Ressourcen)**: alles was eine Anwendung zur Ausführung braucht
 - ➔ Prozessor, Speicher, Geräte (Festplatte, Netzwerk, ...)
- ➔ Früher: auf einem Rechner lief zu jedem Zeitpunkt nur eine Anwendung eines Benutzers
- ➔ Heute: Rechner im **Mehrprozeß-** und **Mehrbenutzerbetrieb**
 - ➔ mehrere Anwendungen verschiedener Benutzer werden „gleichzeitig“ ausgeführt
- ➔ Notwendig:
 - ➔ Fairness: „gerechte“ Verteilung der Betriebsmittel
 - ➔ Sicherheit: Schutz der Anwendungen und Benutzer voreinander



Verwaltung von Betriebsmitteln ...

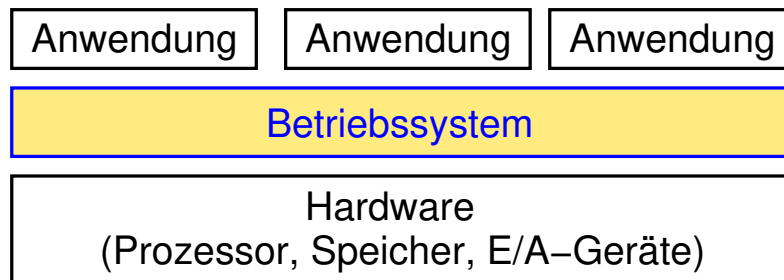
- ➔ Beispiel: Dateien
 - ➔ jeder Datei werden **Rechte** zugeordnet
 - ➔ legen z.B. fest, wer die Datei lesen darf
 - ➔ BS stellt die Einhaltung dieser Rechte sicher
 - ➔ unbefugter Zugriff wird verweigert
- ➔ Beispiel: Drucker
 - ➔ während Max druckt, will auch Moritz drucken
 - ➔ aber nicht auf dasselbe Blatt Papier ...
 - ➔ BS regelt den Zugriff auf den Drucker
 - ➔ der Auftrag von Moritz wird zurückgestellt, bis der von Max beendet ist

1.1 Aufgaben eines Betriebssystems ...



(Animierte Folie)

BS als Mittler zwischen Anwendungen und Hardware



- ➔ Essentiell: Anwendungen können nicht direkt (d.h. **unkontrolliert**) auf die Hardware zugreifen
- ➔ Unterstützende Hardware-Mechanismen:
 - Ausführungsmodi des Prozessors (System- und Benutzermodus, ➔ **1.4.1**)
 - Adreßumsetzung (virtueller Speicher, ➔ **6.3**)

1.2 Entwicklung der Betriebssysteme



1. Generation (-1955): kein Betriebssystem

- ➔ Programm (jedesmal) manuell in Speicher eingeben

2. Generation (-1965): Stapelverarbeitung

- ➔ Lochkarten mit Programmcode (z.B. Assembler, Fortran)
- ➔ BS startet Übersetzer und Programm
- ➔ BS nimmt Ergebnis entgegen, gibt es auf Drucker aus
- ➔ später: auch mehrere Programme (**Jobs**) nacheinander (auf Magnetband): **Stapelbetrieb** (*batch*)
- ➔ Stapelbetrieb auch heute noch teilweise sinnvoll
 - lange, nicht-interaktive Jobs (z.B. Jahresabrechnungen)



3. Generation (-1980):

- ➔ Rechnerfamilien mit gleichem Befehlssatz (z.B. IBM 360)
 - BS abstrahiert Unterschiede der Rechner / Geräte
- ➔ Einführung des Mehrprogrammbetriebs
 - CPU wartet oft (bis zu 90% der Zeit) auf Geräte: Verschwendung!
 - besser: statt zu warten wird ein anderer Job bearbeitet
 - Problem: Verwaltung / Zuteilung der Betriebsmittel
- ➔ Gleichzeitig: interaktive Nutzung der Rechner
 - Terminals statt Lochkarten und Drucker
 - mehrere Benutzer gleichzeitig aktiv
 - gegenseitiger Schutz erforderlich



4. Generation (1980 - heute):

- ➔ Einführung von Mikroprozessoren
 - kleine, billige Rechner: Arbeitsplatzrechner
 - zurück zu Einbenutzersystemen (DOS, Windows 95, ...)
- ➔ Zunehmende Vernetzung der Rechner
 - Client/Server-Systeme: wieder mehrere Benutzer
 - Unix, Linux, Windows (ab NT), ...
- ➔ Trend / Zukunft: verteilte Betriebssysteme
 - mehrere Rechner erscheinen wie ein einziger
 - Ziele: höhere Leistungsfähigkeit und Zuverlässigkeit

1.3 Arten von Betriebssystemen



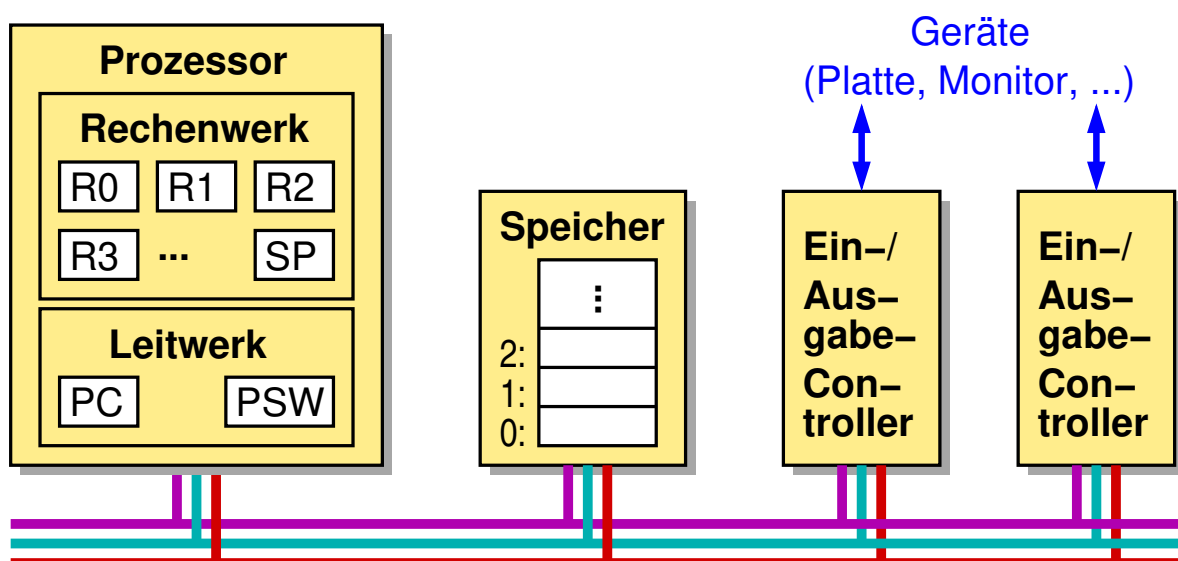
- ➔ Mainframe-BSe
 - ➔ schnelle E/A, viele Prozesse, Transaktionen
- ➔ Server-BSe
 - ➔ viele Benutzer gleichzeitig, Netzwerkanbindung
- ➔ Multiprozessor-BSe
 - ➔ für Parallelrechner
- ➔ PC-BSe
- ➔ Echtzeit-BSe
- ➔ BSe für eingebettete Systeme
- ➔ BSe für Chipkarten

1.4 Computer-Hardware



(Animierte Folie)

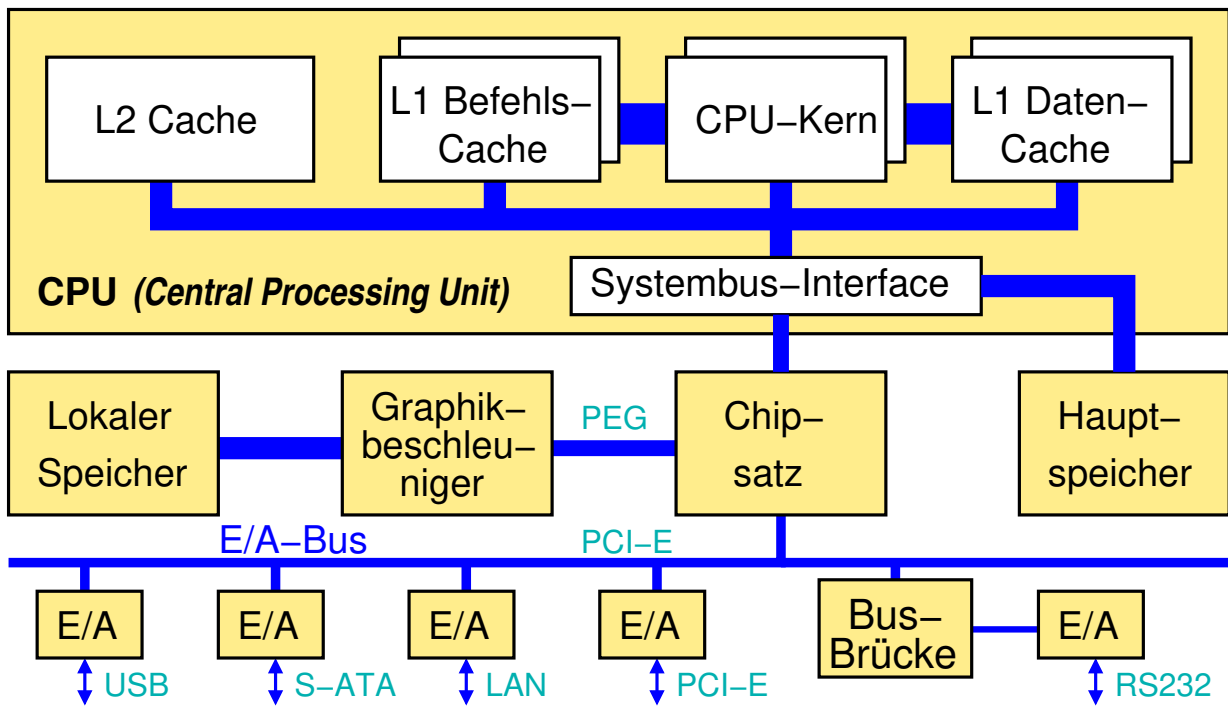
Aufbau eines typischen PCs (stark vereinfacht)



Systembus: Adressen, Daten, Steuersignale



Aufbau eines typischen PCs (realistischer)



Betriebssysteme I

WS 2019/2020

17.10.2019

Roland Wismüller
Betriebssysteme / verteilte Systeme
roland.wismueller@uni-siegen.de
Tel.: 0271/740-4050, Büro: H-B 8404

Stand: 14. November 2019



Multiprozessor-Systeme

- ➔ Heute i.a. Rechner mit mehreren CPUs (bzw. CPU-Kernen)
 - ➔ Multicore-Prozessoren
 - ➔ Server mit mehreren Prozessoren
- ➔ Im Folgenden einheitlich als Multiprozessor-Systeme bezeichnet
 - ➔ Begriff „CPU“ bzw. „Prozessor“ bezeichnet ggf. nur einen CPU-Kern
- ➔ Typische BS-Architektur für Multiprozessorsysteme:
 - ➔ im Speicher: eine Instanz des BSs für alle Prozessoren
 - ➔ jeder Prozessor kann Code des BSs ausführen
 - ➔ **symmetrisches Multiprozessor-System**

1.4.1 Prozessor (CPU)



Elemente einer CPU

- ➔ Register
 - ➔ Ganzzahl-, Gleitkomma-Register
 - ➔ Befehlszähler (PC: *Program Counter*)
 - ➔ Kellerzeiger (SP: *Stack Pointer*)
 - ➔ Statusregister (PSW: *Program Status Word*)
- ➔ Arithmetisch-Logische Einheit (Rechenwerk, ALU)
 - ➔ führt die eigentlichen Berechnungen durch
- ➔ Steuerwerk
 - ➔ holt und interpretiert Maschinenbefehle



Operationen:

- ➔ Lade- und Speicheroperationen
- ➔ Arithmetische und logische Operationen
- ➔ Sprünge (Änderung des Befehlszählers)
 - ➔ bedingt und unbedingt
 - ➔ Unterprogramm-Aufrufe und -Rücksprünge



Ausführungsmodi

- ➔ Eine der Maßnahmen, um den direkten Zugriff auf Systemressourcen durch Anwendungsprogramme zu unterbinden
- ➔ **Systemmodus**: ausgeführtes Programm hat vollen Zugriff auf alle Rechnerkomponenten
 - ➔ für das Betriebssystem
- ➔ **Benutzermodus**: eingeschränkter Zugriff
 - ➔ Speicher nur über Adreßabbildung zugreifbar (☞ 1.5.2, 6.3)
 - ➔ keine **privilegierten Befehle**
 - ➔ z.B. Ein-/Ausgabe, Zugriff auf Konfigurationsregister

1.4.1 Prozessor (CPU) ...



Ausführungsmodi ...

- ➔ Ausführung eines privilegierten Befehls im Benutzermodus führt zu **Ausnahme** (*Exception, Software Interrupt*)
 - ➔ Prozessor bricht gerade ausgeführten Befehl ab
 - ➔ Prozessor sichert PC (Adresse des Befehls) im Keller (Rückkehradresse)
 - ➔ Programmausführung wird an **vordefinierter** Adresse im Systemmodus fortgesetzt
 - ➔ d.h. Verzweigung an eine feste Stelle im BS
 - ➔ BS behandelt die Ausnahme
 - ➔ z.B. Abbruch des laufenden Prozesses
 - ➔ ggf. auch andere Behandlung und Rückkehr in die Anwendung
 - ➔ Rückkehrbefehl schaltet wieder in Benutzermodus

Hardware

BS

1.4.1 Prozessor (CPU) ...



Ausführungsmodi ...

- ➔ Kontrollierter Moduswechsel
 - ➔ spezieller Befehl (**Systemaufruf**, *Trap, System Call*)
 - ➔ bei Ausführung des Befehls:
 - ➔ Prozessor sichert PC im Keller (Rückkehradresse)
 - ➔ Umschalten in Systemmodus
 - ➔ Verzweigung an eine **vordefinierte** Adresse (im BS)
 - ➔ BS analysiert die Art des Systemaufrufs und führt den Aufruf aus
 - ➔ Rückkehrbefehl schaltet wieder in Benutzermodus

Hardware

BS

1.4.1 Prozessor (CPU) ...



Interrupts

- ➔ Prozessor kann auf externe, asynchrone Ereignisse reagieren
 - ➔ spezielles Eingangssignal: Unterbrechungssignal
 - ➔ Signal wird jeweils nach der Abarbeitung eines Befehls abgefragt
 - ➔ falls Signal gesetzt:
 - ➔ Prozessor sichert PC im Keller (Rückkehradresse)
 - ➔ Umschalten in Systemmodus
 - ➔ Verzweigung an eine **vordefinierte** Adresse (im BS) (**Unterbrechungsbehandlungsroutine**, *interrupt handler*)
 - ➔ im BS: Behandlung der Unterbrechung
 - ➔ Rückkehrbefehl schaltet wieder in Benutzermodus
- ➔ Hauptanwendung: Ein-/Ausgabe

Hardware

BS

1.4.1 Prozessor (CPU) ...



Interrupts ...

- ➔ I.d.R. mehrere Interrupts mit verschiedenen Prioritäten
 - ➔ jedem Interrupt kann eine eigene Behandlungsroutine zugewiesen werden
 - ➔ Tabelle von Adressen: **Unterbrechungsvektor**
 - ➔ ggf. Unterbrechung aktiver Behandlungsroutinen
- ➔ Interrupts können **maskiert** („ausgeschaltet“) werden
 - ➔ privilegierter Befehl!

Anmerkungen zu Folie 40:

In Multiprozessor-Systemen (dazu gehören wie erwähnt auch Systeme mit einer Multicore-CPU) kann ein Interrupt von jeder beliebigen CPU (bzw. CPU-Kern) behandelt werden. Damit das funktioniert, muß das Interrupt-Signal durch Hardware an **genau eine** der CPUs weitergeleitet werden. In der Regel will man die Zuordnung zwischen Interrupt und CPU nicht statisch festlegen, sondern dynamisch (lastabhängig).

Diese Aufgabe übernimmt bei der Intel-Architektur der APIC (*Advanced Programmable Interrupt Controller*). Die CPUs geben dem APIC laufend die Priorität der rechnenden Prozesse bekannt. Der APIC leitet einen Interrupt dann an die CPU weiter, deren Prozeß die niedrigste Priorität hat.

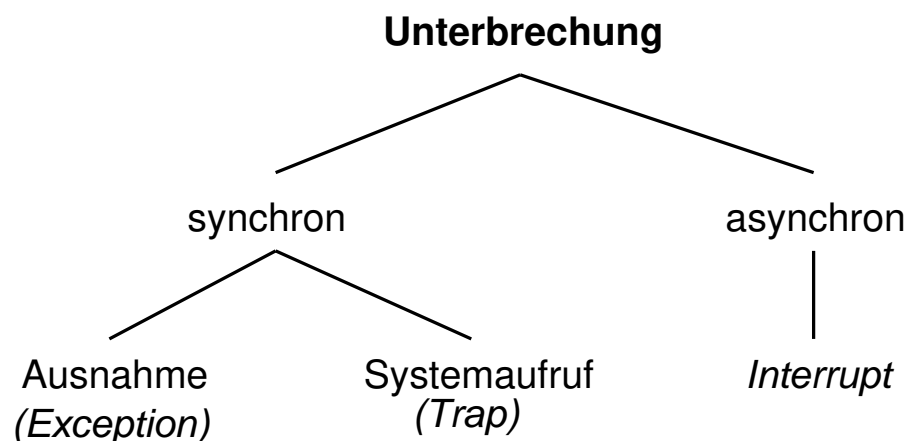
Darüberhinaus ermöglicht der APIC *Cross-CPU Interrupts*, d.h. eine CPU kann gezielt in einer anderen CPU einen Interrupt auslösen, z.B. um ein Scheduling anzustoßen, wenn ein Prozeß rechenbereit wurde.

40-1

1.4.1 Prozessor (CPU) ...



Unterbrechungen des Programmablaufs





Speicherhierarchie

typ. Zugriffszeit

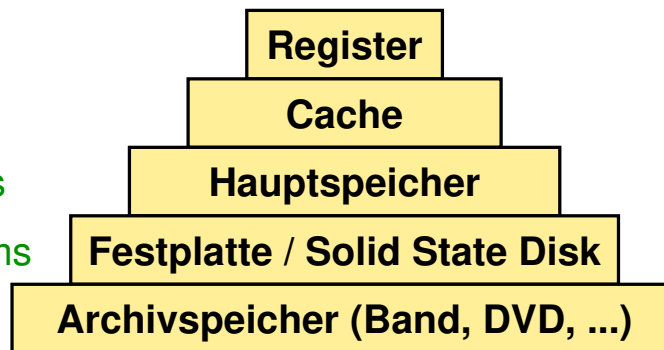
1 ns

2–10 ns

10–100 ns

10 μ s – 10 ms

100 s



typ. Kapazität

< 1 KByte

~ 1 MByte

~ 1 GByte

~ 1 TByte

~ 1 PByte

➔ Verwaltung von Hauptspeicher und Festplatte durch BS

1.4.2 Speicher ...



Caches in Multiprozessorsystemen (incl. Multicore)

- ➔ **Cache**: schneller, prozessornaher Zwischenspeicher
 - ➔ speichert Kopien der zuletzt am häufigsten benutzten Daten aus dem Hauptspeicher
 - ➔ i.a. Blöcke (Cachezeilen) mit 32-64 Byte
 - ➔ falls Daten im Cache: kein Hauptspeicherzugriff nötig
 - ➔ durch Hardware verwaltet, für Programme transparent
- ➔ Caches sind in Multiprozessorsystemen essentiell
 - ➔ Cachezugriff 10-1000 mal schneller als Hauptspeicherzugriff
 - ➔ Entlastung von Hauptspeicher und Bus
- ➔ Aber: Existenz mehrerer Kopien von Daten kann zu Inkonsistenzen führen: **Cache-Kohärenz-Problem**

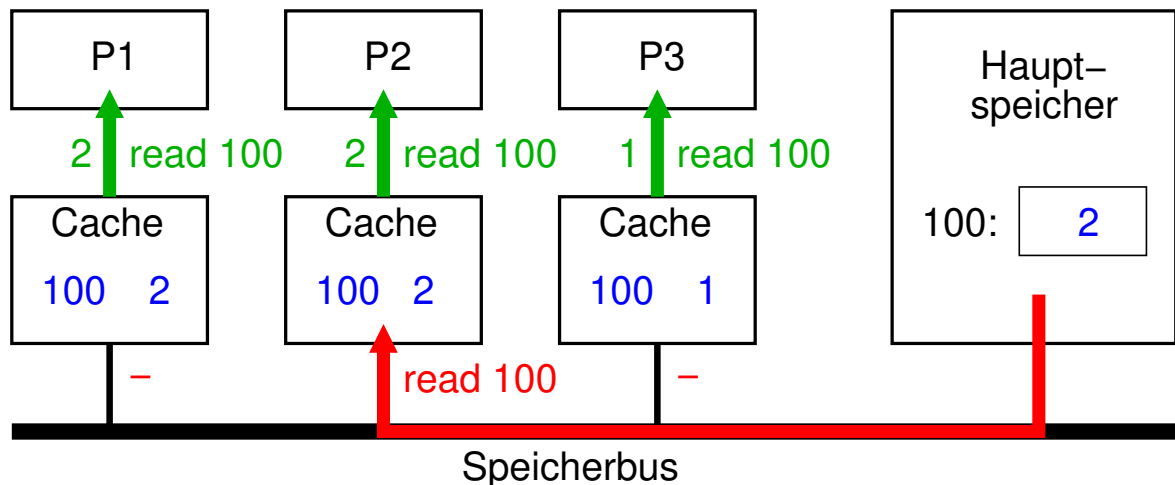
1.4.2 Speicher ...



(Animierte Folie)

Cache-Kohärenz-Problem: Beispiel

- ➔ Annahme: Schreibzugriff aktualisiert auch Hauptspeicher
- ➔ Drei Prozessoren greifen auf dasselbe Speicherwort zu und erhalten verschiedene Ergebnisse!



1.4.2 Speicher ...



Sicherstellung der Cache-Kohärenz

- ➔ Grundprinzip: Bei einem Schreibzugriff müssen alle betroffenen Caches (d.h. Caches mit Kopien) benachrichtigt werden
- ➔ Einfachste Möglichkeit: *Bus Snooping*
 - ➔ Caches „horchen“ auf dem Speicherbus mit
 - ➔ bei Schreibzugriff:
 - ➔ falls Datum im Cache: invalidiere zugehörige Cachezeile
 - ➔ Cachezeile wird beim nächsten Zugriff neu geladen

Anmerkungen zu Folie 45:

Auf dieser und der vorherigen Folie wurde angenommen, daß der Cache im sog. *Write-Through*-Modus arbeitet, d.h. Schreiboperationen sofort auch im Hauptspeicher durchgeführt werden.

I.A. werden heute *Write-Back*-Caches benutzt, bei denen Schreiboperationen zunächst nur im Cache ausgeführt werden. Die geänderten Daten werden erst später bei Bedarf in den Hauptspeicher geschrieben. Dadurch werden die Protokolle für die Cache-Kohärenz etwas komplexer (z.B. das MESI-Protokoll), basieren aber immer noch auf der beschriebenen Idee des „Mithorchens“ auf dem Bus.

45-1

1.4.3 Ein-/Ausgabe



Ansteuerung der Geräte durch *Controller*

- ➔ Spezielle Hardware, oft mit eigenen Mikroprozessoren
- ➔ Steuert Gerät weitgehend autonom
- ➔ Register für Kommandos, Daten, Status
- ➔ Kann Interrupt an CPU senden, falls
 - ➔ Eingabedaten vorhanden
 - ➔ Ausgabeoperation abgeschlossen



Anbindung Controller - CPU

- ➔ Speicherbasierte E/A
 - ➔ Register des Controllers sind in den Speicheradreßraum eingeblendet
 - ➔ Zugriff über normale Schreib-/Lesebefehle
 - ➔ Zugriffsschutz über Adreßabbildung (☞ 1.4.1)
- ➔ Separater E/A-Adreßraum (z.B. bei x86 üblich)
 - ➔ Zugriff auf Controller-Register nur über spezielle (privilegierte) E/A-Befehle
- ➔ Beide Varianten in Gebrauch



Klassen von E/A-Geräten

- ➔ **Zeichen-orientierte Geräte**
 - ➔ Ein-/Ausgabe einzelner Zeichen
 - ➔ z.B. Tastatur, Maus, Modem
- ➔ **Block-orientierte Geräte**
 - ➔ Ein-/Ausgabe größerer Datenblöcke
 - ➔ z.B. Festplatte, Netzwerk
 - ➔ Vorteil: Reduzierung der Interrupt-Rate
 - ➔ wichtig für Programm-Geschwindigkeit



Grundlegende Konzepte von BSen:

- ➔ Prozesse und Threads
- ➔ Speicherverwaltung
- ➔ Ein/Ausgabe
- ➔ Dateiverwaltung

Dazu orthogonale Aufgaben:

- ➔ Sicherheit
- ➔ Ablaufplanung und Ressourcenverwaltung

1.5.1 Prozesse und Threads



Definitionen

- ➔ Anschaulich: ein **Prozeß** ist ein Programm in Ausführung
- ➔ Formaler:
 - ➔ Aktivitätseinheit, gekennzeichnet durch
 - ➔ eine Ausführungsumgebung
 - ➔ Adreßraum (Programmcode und Daten)
 - ➔ Zustandsinformation benutzter Ressourcen (z.B. offene Dateien, Position der Lesezeiger, ...)
 - ➔ ein oder mehrere Aktivitätsträger (**Threads**, „Ablauffäden“)
- ➔ Anmerkungen:
 - ➔ implementierungsnahe Definition
 - ➔ klassischer Prozess enthält genau einen Thread
 - ➔ heute in den meisten BSen mehrfädige Prozesse möglich



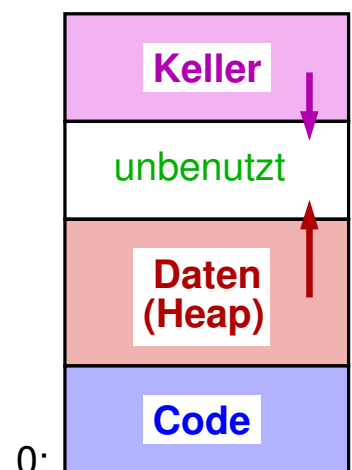
Abarbeitung von Threads: *Timesharing*

- ➔ Threads werden abwechselnd vom Prozessor (bzw. den Prozessoren) bearbeitet
 - BS entscheidet, wer wann wie lange (auf welchem Prozessor) rechnen darf
- ➔ Gründe für Timesharing:
 - Bedürfnisse des Nutzers (mehrere Anwendungen)
 - bessere Auslastung des Rechners
- ➔ **Prozeßwechsel** bedingt Wechsel der Ausführungsumgebung
- ➔ Threads eines Prozesses teilen sich die Ausführungsumgebung
 - Wechsel zwischen Threads desselben Prozesses ist effizienter



(Virtueller) Adreßraum

- ➔ Beginnt bei Adresse 0, durchnummeriert bis Obergrenze
 - linearer Adreßraum
- ➔ Enthält:
 - Programmcode
 - Programmdateien (incl. Heap)
 - Keller (Rückkehradressen)
- ➔ Abstraktion des physischen Speichers
 - unabhängig von Größe und Technologie des physischen Speichers





Ausführungskontext

- ➔ Alle sonstigen Daten, die zur Ausführung gebraucht werden
 - ➔ Prozessorstatus (Datenregister, PC, PSW, ...)
 - ➔ BS-relevante Daten (Eigentümer, Priorität, Eltern-Prozeß, genutzte Betriebsmittel, ...)
- ➔ Aufgeteilt in Prozeß- und Threadkontext
- ➔ Speicherung des Ausführungskontexts:
 - ➔ i.d.R. im Adreßraum des BSs: **Prozeß-** bzw. **Threadtabelle**
 - ➔ alternativ: (geschützter) Teil des Prozeßadreßraums



Prozeß-/Threadinteraktion (Interprozeßkommunikation, IPC)

- ➔ **Kommunikation**: Austausch von Informationen
 - ➔ durch Verwendung eines gemeinsamen Adreßraums (**Speicherkopplung**)
 - ➔ i.d.R. zwischen Threads eines Prozesses
 - ➔ durch explizites Senden/Empfangen von Nachrichten (**Nachrichtenkopplung**)
 - ➔ i.d.R. zwischen verschiedenen Prozessen
- ➔ **Synchronisation**
 - ➔ Steuerung der zeitlichen Reihenfolge von Aktivitäten (meist: Zugriffe auf gemeinsame Ressourcen)
 - ➔ Problem: **Verklemmungen** (**Deadlocks**)
 - ➔ zyklische Wartebeziehungen

Betriebssysteme I

WS 2019/2020

24.10.2019

Roland Wismüller
Betriebssysteme / verteilte Systeme
roland.wismueller@uni-siegen.de
Tel.: 0271/740-4050, Büro: H-B 8404

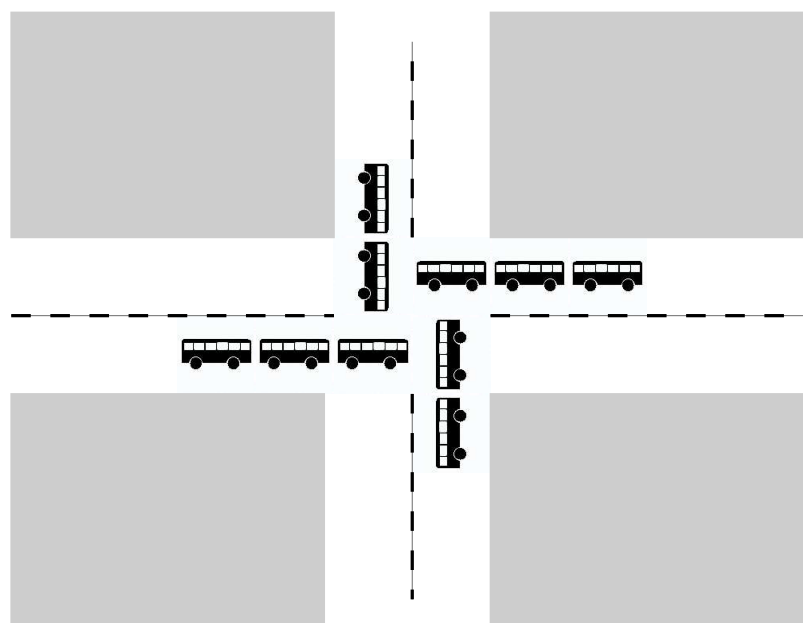
Stand: 14. November 2019

1.5.1 Prozesse und Threads ...



(Animierte Folie)

Eine anschauliche Verklemmung

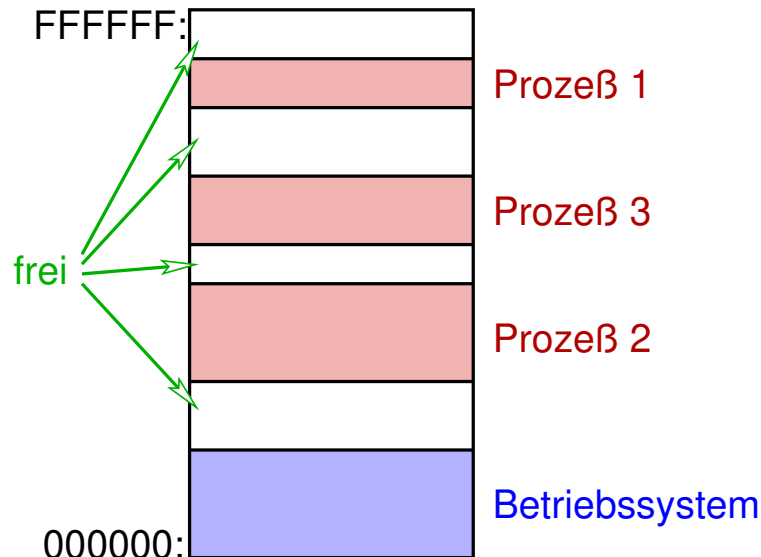


1.5.2 Speicherverwaltung



Situation in Mehrprozeß-Systemen:

➔ Prozesse und BS teilen sich den Hauptspeicher, z.B.:



1.5.2 Speicherverwaltung ...



Fragen / Probleme für das BS:

- ➔ Wie werden die Speicherbereiche an Prozesse zugeteilt?
 - BS muß genügend großes Stück Speicher finden
 - was ist, wenn laufender Prozeß mehr Speicher anfordert?
- ➔ Programme enthalten Adressen
 - kann man Programme an beliebige Adressen schieben?
- ➔ Schutz der Prozesse gegeneinander
 - wie kann man verhindern, daß ein Prozeß auf den Speicher eines anderen Prozesses (oder des BSs) zugreift?
- ➔ Begrenzte Größe des Hauptspeichers
 - was ist, wenn nicht alle Prozesse in den Hauptspeicher passen?



Fragen / Probleme für das BS: ...

- ➔ Transparenz für die Anwendungen
 - ➔ im Idealfall sollte die Anwendung annehmen dürfen, sie hätte den Rechner für sich alleine
 - ➔ Programmieraufwand sollte sich durch Speicherverwaltung nicht erhöhen

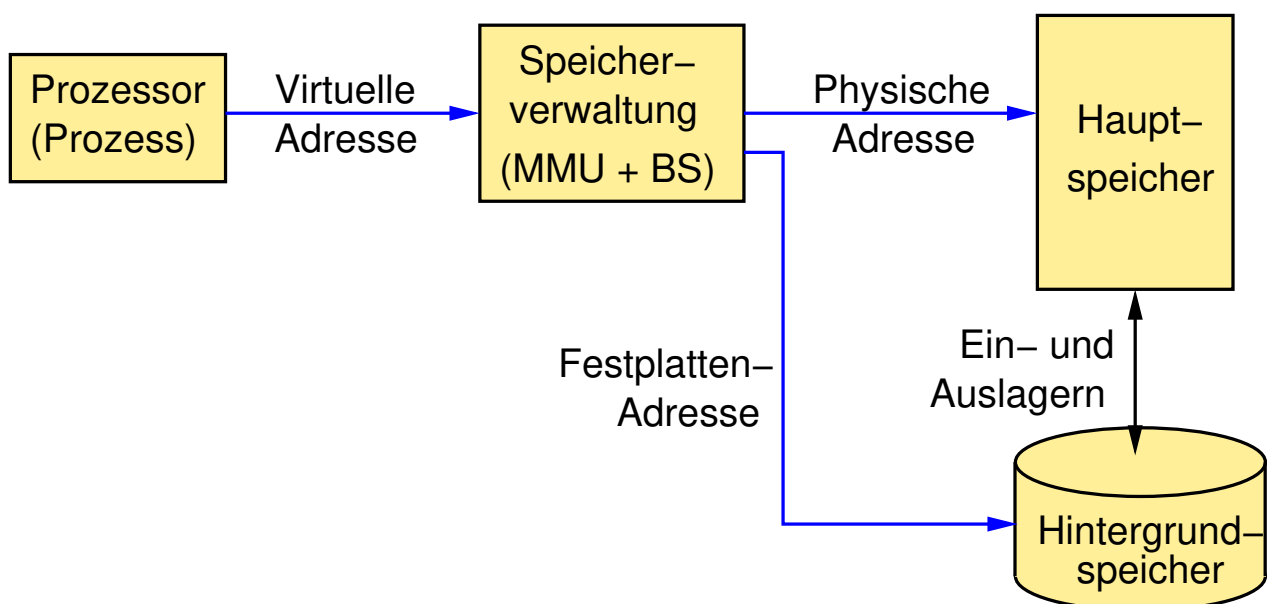
Lösung: Adressabbildung

- ➔ Adressen des Prozesses (**virtuelle Adressen**) werden durch Hardware (**Memory Management Unit, MMU**) auf Speicheradressen (**physische Adressen**) umgesetzt
 - ➔ einfachster Fall: Addition einer Basis-Adresse und Limit
- ➔ Weiterer Vorteil: Adreßraum kann auf Hauptspeicher und Hintergrundspeicher aufgeteilt werden

1.5.2 Speicherverwaltung ...



Adressierung beim virtuellen Speicher



1.5.3 Dateiverwaltung



- ➔ **Datei**: Einheit zur Speicherung von Daten
 - ➔ **persistente Speicherung**: über das Ende von Prozessen (Anwendungen) hinaus
- ➔ Struktur einer Datei:
 - ➔ bei heutigen BSen für PCs und Server: unstrukturierte Folge von Bytes (d.h.: BS kennt Struktur nicht)
 - ➔ in Mainframe-BSen u.a.:
 - ➔ Sequenz von Datensätzen fester Struktur
 - ➔ Index zum schnellen Zugriff auf Datensätze
- ➔ Typen von Dateien (u.a.):
 - ➔ reguläre Dateien (Dokumente, Programme, ...)
 - ➔ Verzeichnisse: Information zur Struktur des Dateisystems
 - ➔ Gerätedateien: falls Geräte ins Dateisystem abgebildet sind

1.5.3 Dateiverwaltung ...



Typische Dateioperationen

- ➔ Öffnen einer Datei; Argumente u.a.:
 - ➔ Lesen/Schreiben/Anfügen?, Erzeugen?, Sperren?, ...
- ➔ Schließen der Datei
- ➔ Lesen / Schreiben von Daten
 - ➔ Zugriff meist sequentiell (historisch bedingt)
 - ➔ daneben auch wahlfreier Zugriff möglich
 - ➔ Offset als Parameter beim Lesen / Schreiben oder explizites Positionieren (*seek*)
 - ➔ Alternative: Einblenden in den Prozeß-Adreßraum
- ➔ Erzeugen, Löschen, Umbenennen
- ➔ Lesen und Schreiben von Dateiattributen (z.B. Zugriffsrechte)



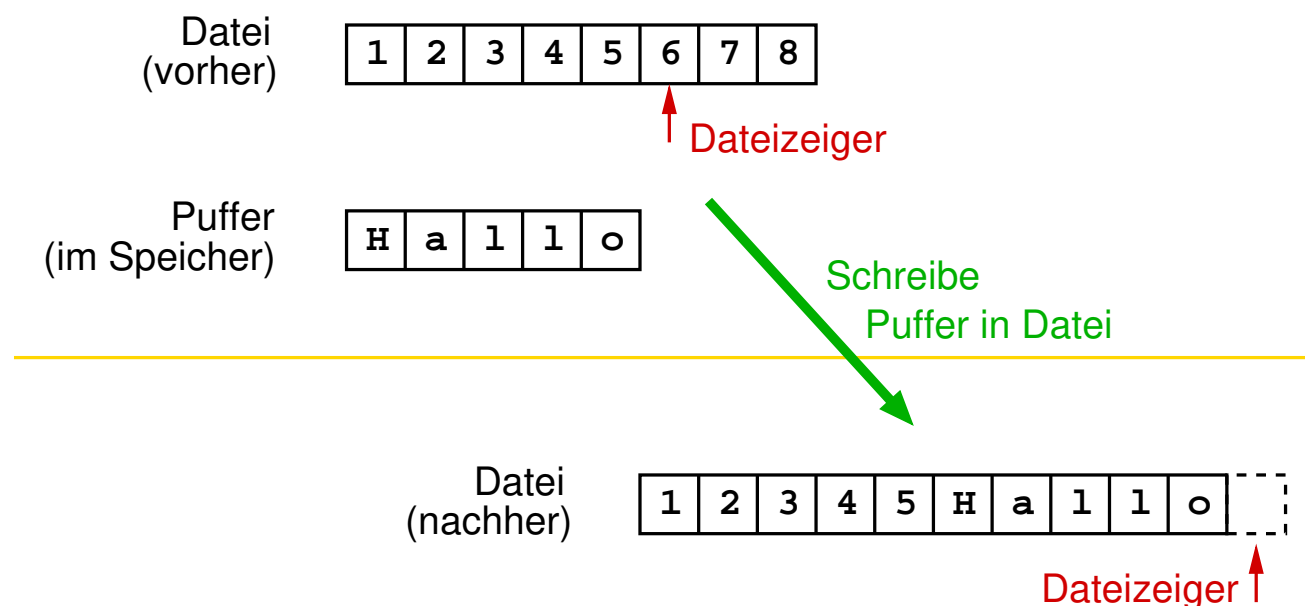
Das sequentielle Dateimodell

- ➔ Datei wird als unstrukturierte Byte-Folge aufgefaßt
- ➔ Nach dem Öffnen einer Datei besitzt ein Prozeß einen privaten **Dateizeiger**
 - ➔ nächstes zu lesendes Byte bzw. Schreibposition
- ➔ Lese- und Schreib-Operationen kopieren einen Datenblock zwischen Hauptspeicher und Datei
 - ➔ Dateizeiger wird entsprechend weitergeschoben
- ➔ Lesen über das Dateiende hinaus (**End-of-file**) ist nicht möglich
- ➔ Schreiben über das Dateiende führt zum Anfügen an Datei
- ➔ Dateizeiger kann auch explizit positioniert werden

1.5.3 Dateiverwaltung ...



Beispiel: Schreiben in eine sequentielle Datei





Verzeichnisse

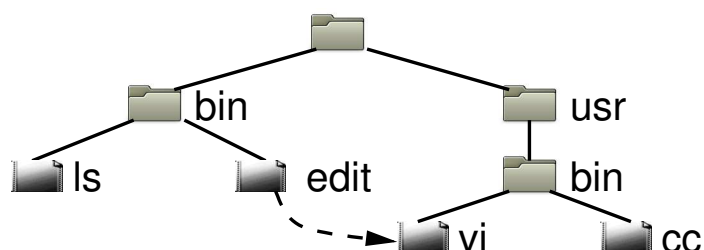
- ➔ Zur hierarchischen Organisation von Dateien
 - Benennung von Dateien über Pfadnamen
 - absolute Pfadnamen (beginnend bei Wurzelverzeichnis)
 - z.B. /home/wismueller/Vorlesungen/BS1/v02.pdf
 - relative Pfadnamen
 - beginnen beim aktuellen Arbeitsverzeichnis
 - z.B. BS1/v02.pdf: selbe Datei wie oben, wenn Arbeitsverzeichnis /home/wismueller/Vorlesungen ist

1.5.3 Dateiverwaltung ...



Verzeichnisse ...

- ➔ Oft: Einführung von Verweisen (*Links*)
 - Datei (bzw. Verzeichnis) kann in mehr als einem Verzeichnis auftreten, ist aber physisch nur einmal vorhanden



/bin/edit führt zur
Datei /usr/bin/vi

- Links sind in der Regel transparent für die Anwendungen



Typische Aufgaben / Konzepte von BSen (Beispiel: Linux)

- ➔ Realisierung von Dateien und Verzeichnishierarchien
 - ➔ Abbildung von Dateien / Verzeichnissen auf das Speichermedium (z.B. Festplatte)
- ➔ Realisierung der Dateioperationen: (Erzeugen, Lesen, ...)
 - ➔ dabei Überprüfung der Zugriffsrechte
- ➔ Einhängen von Dateisystemen in die Verzeichnishierarchie
 - ➔ z.B. Zugriff auf Dateien auf Diskette über den Pfad /media/floppy
 - ➔ einheitliche Sicht auf alle Dateisysteme
- ➔ Spezialdateien (Geräte dateien, Pipes, Prozeßdateisystem, ...)

1.5.4 Sicherheit



Aufgabenbereiche:

- ➔ Zugriffskontrolle
 - ➔ Benutzerzugriff auf Gesamtsystem, Subsysteme, Daten
 - ➔ Prozeßzugriff auf Ressourcen und Objekte des Systems
- ➔ Kontrolle des Informationsflusses
 - ➔ Regulierung des Datenflusses im System / an Benutzer (keine Weitergabe vertraulicher Daten an Unautorisierte)
- ➔ Zertifizierung
 - ➔ Nachweis, daß gewünschte Sicherheitseigenschaften vom System durchgesetzt werden
- ➔ In dieser Vorlesung wird nur Zugriffskontrolle betrachtet!



Beispiel: Zugriffskontrolle unter Linux

- ➔ Einführung von Benutzern und Benutzergruppen
 - spezieller Benutzer `root`: Administrator
- ➔ Prozesse und Dateien haben Eigentümer und Eigentümergruppe
- ➔ Zugriff auf Prozesse nur für Eigentümer (und `root`)
- ➔ Zugriff auf Dateien über 9 Rechtebits festgelegt: `rwX r-x ---`
user group others
 - `r` = Lesen, `w` = Schreiben, `x` = Ausführen
 - im Bsp.: Eigentümer darf alles, Benutzer der Eigentümergruppe darf nicht schreiben, für alle anderen kein Zugriff
- ➔ Rechte werden bei jedem (relevanten) Zugriff vom BS geprüft

1.5.5 Ablaufplanung u. Ressourcenverwaltung



- ➔ Verwaltung / Planung der Ressourcen-Verwendung, z.B.
 - Hauptspeicher: Welcher Prozeß erhält wann wo wieviel Speicher? Wann wird Speicher ausgelagert?
 - Platten-E/A: Reihenfolge der Bedienung? (Optimierung der Armbewegung)
- ➔ Ziele:
 - Fairness: jeder Prozess sollte denselben Anteil der Ressourcen erhalten
 - Differenzierung: Berücksichtigung der unterschiedlichen Anforderungen verschiedener Job-Klassen
 - Effizienz: Durchsatz, Antwortzeit, Anzahl der Benutzer
 - Ziele teilweise widerspüchlich

1.6 Systemaufrufe

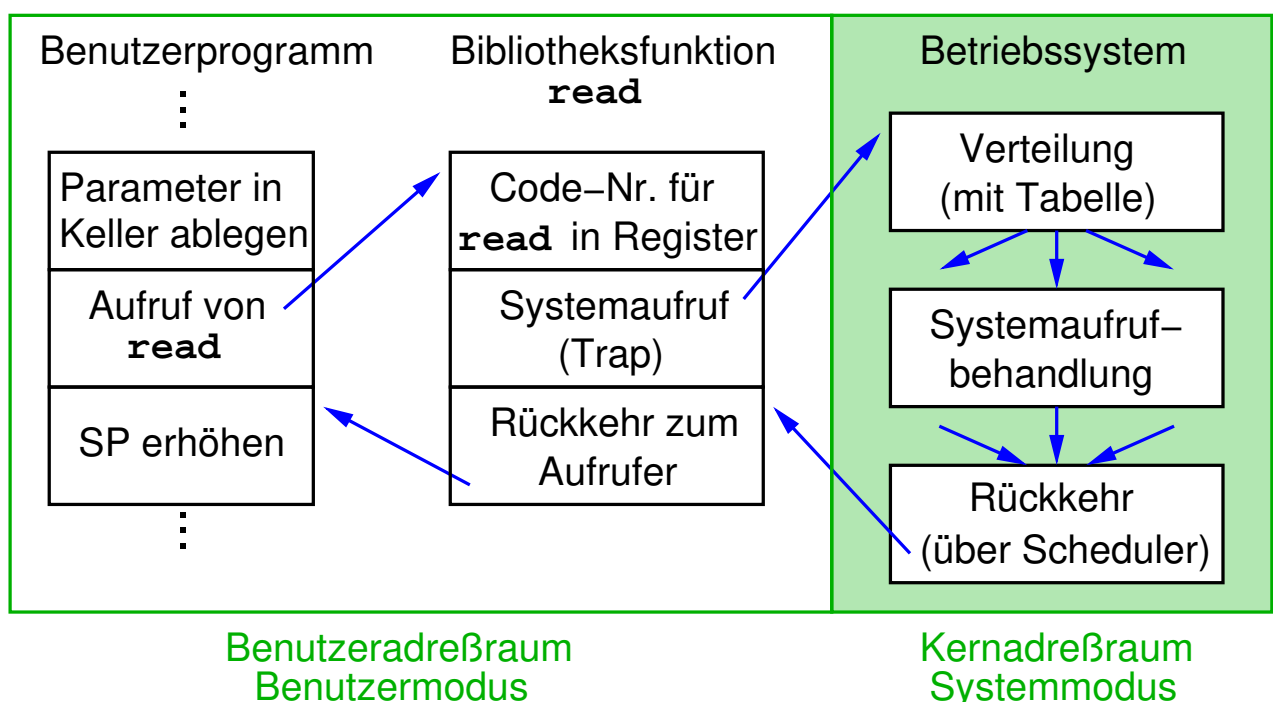


- ➔ Schnittstelle zwischen Benutzerprogrammen und BS
- ➔ Systemaufrufe meist in Bibliotheksfunktionen gekapselt
 - Details des Systemaufrufs sind hardwareabhängig
 - In der Regel:
 - Systemaufrufe sind durchnummeriert
 - Nummer wird vor Ausführung des *Trap*-Befehls in festes Register geschrieben
 - Im BS dann Verzweigung über Funktionstabelle

1.6 Systemaufrufe ...



Ablauf eines Systemaufrufs





Anmerkungen zum Ablauf

- ➔ BS sichert zunächst den vollständigen Prozessorstatus in der Threaddtabelle (bzw. Prozeßtabelle)
- ➔ Aufrufender Thread (bzw. Prozeß) kann blockiert werden
 - ➔ z.B. Warten auf Eingabe
- ➔ Rückkehr aus dem BS erfolgt über den Scheduler
 - ➔ er bestimmt den Thread, der weitergeführt wird
 - ➔ Rückkehr also nicht unbedingt (sofort) zum aufrufenden Thread
- ➔ Bei Rückkehr: Restaurieren des Prozessorstatus des weitergeführten Threads



Typische Systemaufrufe (Beispiel: Linux)

- ➔ Prozeßmanagement
 - fork – Erzeugen eines Kindprozesses
 - waitpid – Warten auf Ende eines Kindprozesses
 - execve – Ausführen eines anderen Programms
 - exit – Prozeß beenden und Statuswert zurückgeben
- ➔ Dateimanagement
 - open – Datei öffnen (Lesen/Schreiben)
 - close – Datei schließen
 - read – Daten aus Datei in Puffer lesen
 - write – Daten aus Puffer in Datei schreiben
 - lseek – Dateizeiger verschieben



Typische Systemaufrufe (Beispiel: Linux) ...

➔ Verzeichnismangement

- `mkdir` – Verzeichnis erzeugen
- `rmdir` – Leeres Verzeichnis löschen
- `unlink` – Datei löschen
- `mount` – Datenträger in Dateisystem einhängen

➔ Verschiedenes

- `chdir` – Arbeitsverzeichnis wechseln
- `chmod` – Datei-Zugriffsrechte ändern
- `kill` – Signal an Prozeß senden
- `time` – Uhrzeit



Verwendung von Systemaufrufen: Prozeßerzeugung

```
#include <sys/types.h>
#include <unistd.h>
```

Deklarationen einbinden

```
main()
{
    if (fork() == 0) {
        printf("Hier ist der Sohn\n");
        exit(0); /* Prozeßende */
    }
    else {
        printf("Hier ist der Vater\n");
        exit(0); /* Prozeßende */
    }
}
```

neuen Prozeß abspalten

vom neuen Prozeß ausgeführt

vom alten Prozeß ausgeführt



Verwendung von Systemaufrufen: Erzeugen einer Datei

```
#include <stdio.h>
#include <fcntl.h>
```

Deklarationen einbinden

```
main()
```

```
{
```

```
    int fd; File-Handle
```

```
    char text[100];
```

```
    fgets(text, 100, stdin);
```

Einlesen eines Strings
von Standard-Eingabe

Dateirechte

Erzeugen und Öffnen der Datei

```
    fd = open("datei.txt", O_CREAT | O_WRONLY, 0600);
```

```
    write(fd, text, strlen(text));
```

Schreiben in die Datei

```
    close(fd); Schließen der Datei
```

```
}
```

1.7 Zusammenfassung / Wiederholung



- ➔ Aufgaben eines BSs
 - ➔ Erweiterung / Abstraktion der Hardware
 - ➔ Verwaltung der Betriebsmittel
 - ➔ Prozessor, Speicher, Platte, E/A, ...
 - ➔ BS liegt zwischen Anwendungen und Hardware
 - ➔ Zugriff auf Hardware nur über BS
- ➔ Entwicklung der BSe
- ➔ Arten von BSen



- ➔ Computer-Hardware
 - ➔ Ausführungsmodi der CPU
 - ➔ Systemmodus für BS
 - ➔ Benutzermodus für Anwendungen
 - ➔ Speicherabbildung, keine privilegierten Befehle
 - ➔ Unterbrechungen:
 - ➔ Systemaufruf (*Trap*)
 - ➔ Ausnahme (*Exception*)
 - ➔ Interrupt



- ➔ Grundkonzepte von BSen
 - ➔ Prozesse
 - ➔ Speicherverwaltung
 - ➔ Ein-/Ausgabe
 - ➔ Dateiverwaltung
- ➔ Konzeptübergreifende Aufgaben
 - ➔ Sicherheit
 - ➔ Ablaufplanung und Ressourcenverwaltung
- ➔ Systemaufrufe