

# Systemprogrammierung

*Grundlage von Betriebssystemen*

## Teil B – VIII. Zwischenbilanz

Wolfgang Schröder-Preikschat

25. Juli 2017



## Agenda

---

### SP1

Lehrziele

C

UNIX

Einleitung

Rechnerorganisation

Betriebssystemkonzepte

Betriebsarten

### SP2

Ausblick



# Gliederung

## SP1

Lehrziele

C

UNIX

Einleitung

Rechnerorganisation

Betriebssystemkonzepte

Betriebsarten

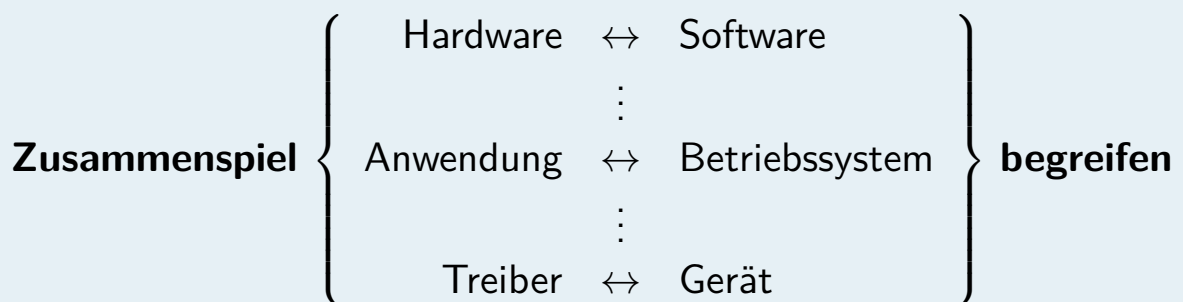
## SP2

Ausblick



# Lernziele

- Vorgänge innerhalb von Rechensystemen **ganzheitlich** verstehen



- imperative Systemprogrammierung (in C) in Grundzügen kennenlernen
  - im Kleinen für **Dienstprogramme** praktizieren
  - im Großen durch **Betriebssysteme** erfahren
- Beziehungen zwischen funktionalen und nicht-funktionalen Systemmerkmalen erfassen



Quelle: fotalia.com



# Kurzeinführung

## Schlüsselwörter

|        |        |          |        |          |          |          |        |
|--------|--------|----------|--------|----------|----------|----------|--------|
| auto   | break  | case     | char   | const    | continue | default  | do     |
| double | else   | enum     | extern | float    | for      | goto     | if     |
| int    | long   | register | return | short    | signed   | sizeof   | static |
| struct | switch | typedef  | union  | unsigned | void     | volatile | while  |

## Operatoren, Selektoren, Klammerungen und andere „Satzzeichen“

|   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ! | " | % | & | ' | ( | ) | * | + | , | - | : | / |
| : | ; | < | = | > | ? | [ | ] | ^ | { | } | ~ |   |

### ■ was macht dieses Programm?

```
1 #include <unistd.h>
2
3 int main() {
4     printf("%d\n", getpid());
5 }
```

### ■ was kann man daraus machen?

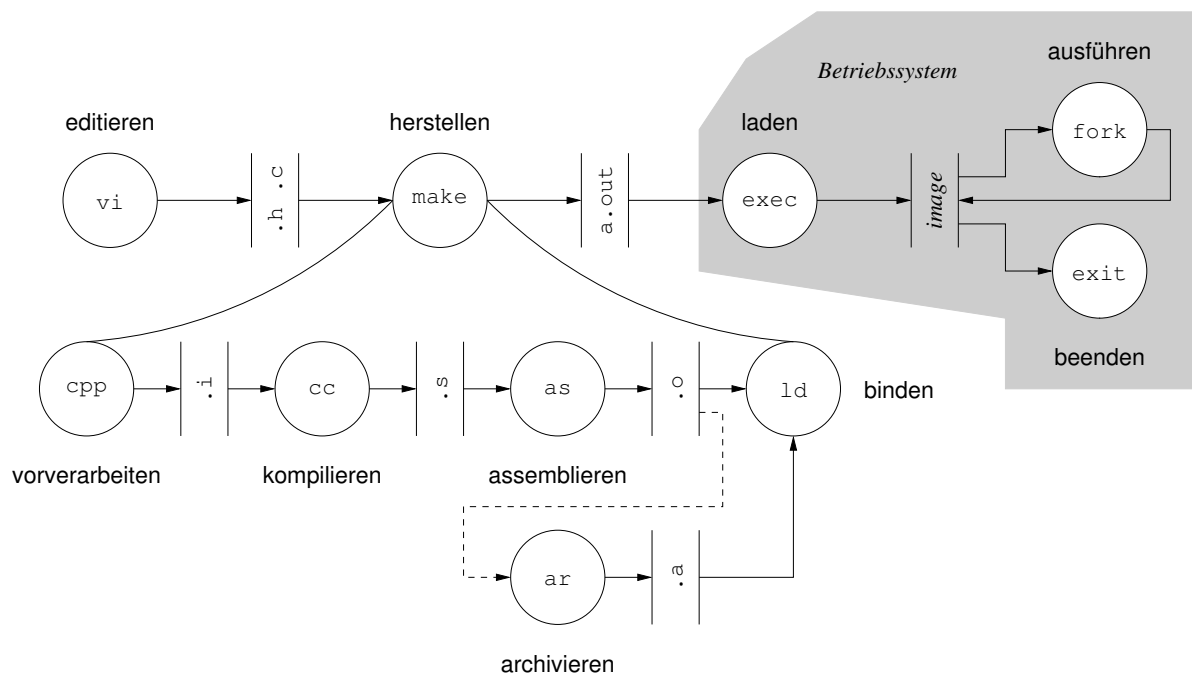
#### ■ *buffer overflow exploit*

### ■ was geschieht nun?

```
6 #include <stdio.h>
7 #include <string.h>
8
9 int getpid() {
10     char buffer[20];
11     gets(buffer);
12     return strlen(buffer);
13 }
```



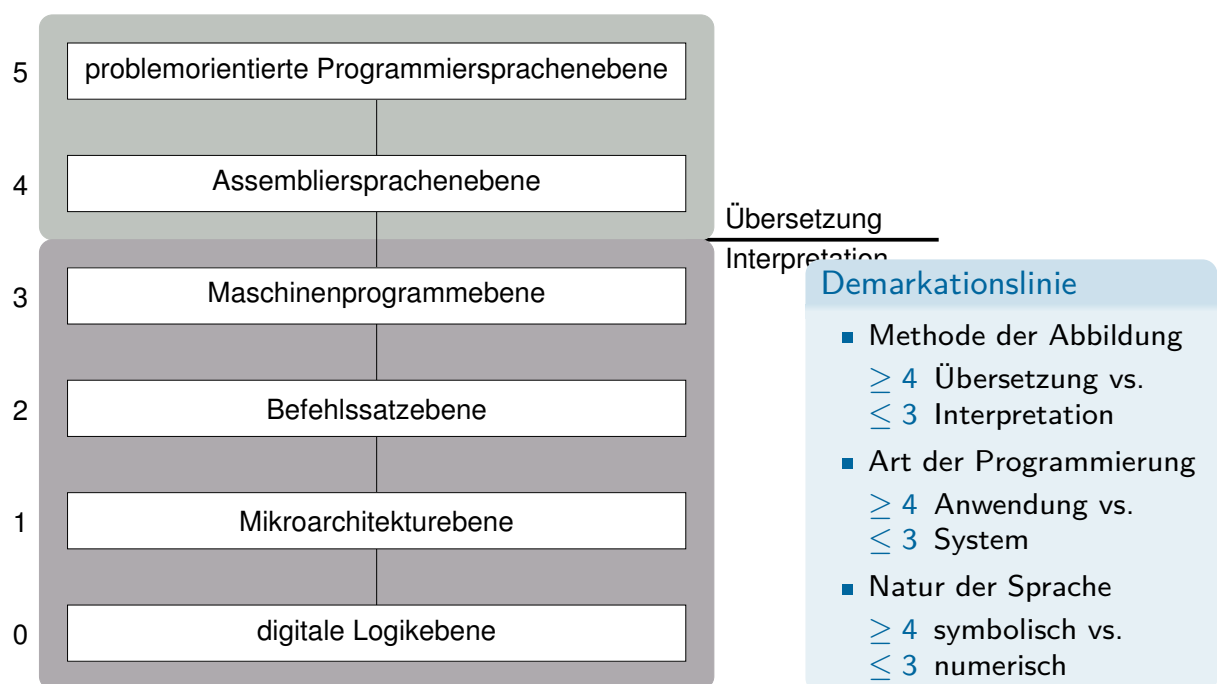
## Quellprogramm → Prozess



- funktionale und nichtfunktionale Eigenschaften von Betriebssystemen werden durch die **Anwendungsdomäne** vorgegeben
    - gelegentlich passen bestehende „unspezifische“ Lösungen (z.B. Linux)
    - viel häufiger sind jedoch **anwendungsspezifische Lösungen** erforderlich
  - Systeme für den **Allgemeinzw**
    - die Domäne der Flach-, Klapp-, Arbeitsplatz-, Grossrechner
    - nur ein vergleichsweise kleiner Anteil an Installationen weltweit (vgl. [1])
  - Systeme für den **Spezialzweck**
    - Rechensysteme zur Steuerung oder Regelung „externer“ Prozesse
- **Echtzeitsysteme**
- mit vorhersagbarem Laufzeitverhalten
- **eingebettete Systeme**
- Mobiltelefon nicht mitgerechnet



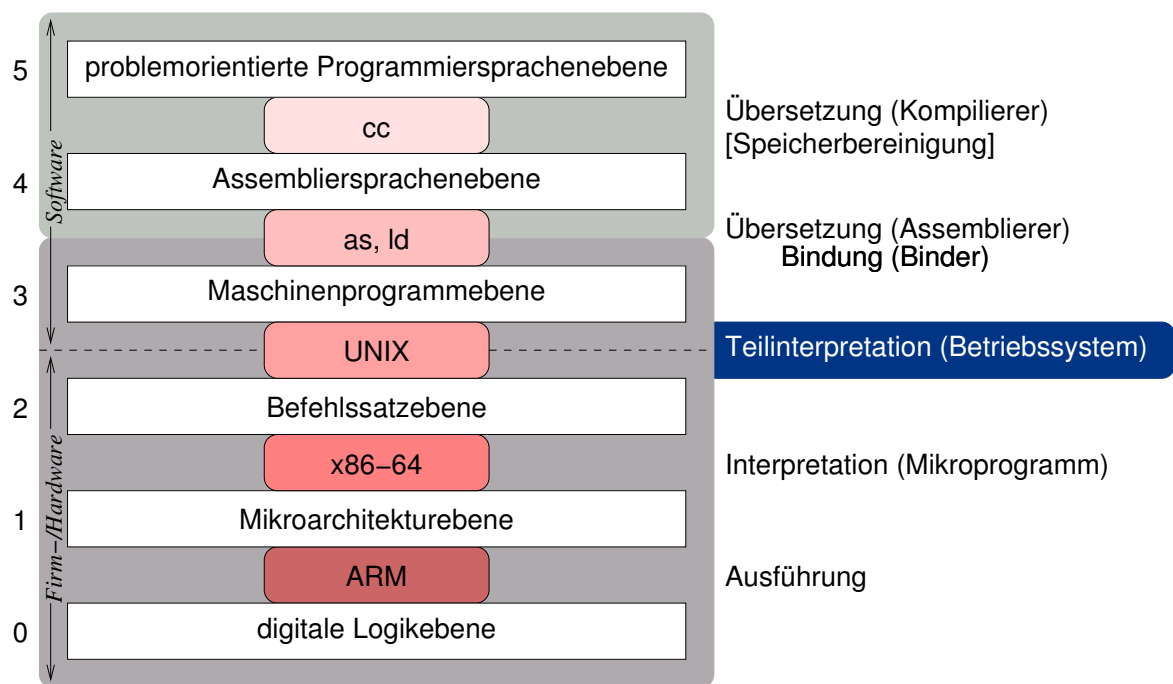
## Schichtenfolge in Rechensystemen I



- Schichten der Ebene<sub>[4,5]</sub> sind nicht wirklich existent
  - sie werden durch Übersetzung aufgelöst und auf tiefere Ebenen abgebildet
  - so dass am Ende nur ein Maschinenprogramm (Ebene<sub>3</sub>) übrigbleibt



# Schichtenfolge in Rechengesystemen II



- RISC auf Ebene<sub>1</sub> und gegebenenfalls (hier) CISC auf Ebene<sub>2</sub>
  - nach außen „complex“, innen aber „reduced instruction set computer“
  - Intel Core oder Haswell ↔ AMD Bulldozer oder Zen (ARM)



## Systemaufrufchnittstelle (system call interface)<sup>1</sup>

```
1 read:
2   push %ebx
3   movl 16(%esp),%edx
4   movl 12(%esp),%ecx
5   movl 8(%esp),%ebx
6   mov $3,%eax
7   int $0x80
8   pop %ebx
9   cmp $-4095,%eax
10  jae __syscall_error
11  ret
```

- „Grenzübergangsstelle“ **Aufrufstumpf**
  - einerseits erscheint ein Systemaufruf als normaler **Prozeduraufruf**
  - andererseits bewirkt der Systemaufruf eine (synchrone) **Ausnahme**
- sorgt für **Ortstransparenz** (funktional)
  - die Lokalität der aufgerufenen Funktion muss nicht bekannt sein

- Systemaufrufe sind **Prozedurfernaufrufe**, um **Prozessdomänen** in kontrollierter Weise zu überwinden

- 3–5 ■ tatsächliche Parameter (Argumente) in Registern übergeben
- 6 ■ Systemaufrufnummer (Operationskode) in Register übergeben
- 7 ■ Domänenwechsel (Ebene<sub>3</sub> ↦ Ebene<sub>2</sub>) auslösen
  - Aufruf abfangen (*trap*) und dem Betriebssystem zustellen
- 9–10 ■ Status überprüfen und ggf. Fehlerbehandlung durchführen



<sup>1</sup>UNIX Programmers Manual (UPM), Lektion 2 — `man(2)`

- Befehle der Maschinenprogrammebene, also Ebene<sub>3</sub>-Befehle sind...
  - „normale“ Befehle der Ebene<sub>2</sub>, die die CPU direkt ausführen kann
    - **unprivilegierte Befehle**, die in jedem Arbeitsmodus ausführbar sind
  - „unnormale“ Befehle der Ebene<sub>2</sub>, die das Betriebssystem ausführt
    - **privilegierte Befehle**, die nur im privilegierten Arbeitsmodus ausführbar sind
- die „aus der Reihe fallenden“ Befehle stellen Adressräume, Prozesse, Speicher, Dateien und Wege zur Ein-Ausgabe bereit
  - Interpreter dieser Befehle ist das Betriebssystem
  - der dadurch definierte Prozessor ist die **Betriebssystemmaschine**
- demzufolge ist ein Betriebssystem immer nur ausnahmsweise aktiv
  - es muss von außerhalb aktiviert werden
    - programmiert im Falle eines Systemaufrufs (CD80: Linux/x86) oder einer sonstigen synchronen Programmunterbrechung (*trap*)
    - nicht programmiert, also nicht vorhergesehen, im Falle einer asynchronen Programmunterbrechung (*interrupt*)
  - es deaktiviert sich immer selbst, in beiden Fällen programmiert (CF: x86)



## Prozess

### Programmablauf, Programm in Ausführung

Betriebssysteme bringen Programme zur Ausführung, in dem dazu Prozesse erzeugt, bereitgestellt und begleitet werden

- im Informatikkontext ist ein Prozess ohne Programm nicht möglich
  - die als Programm kodierte Berechnungsvorschrift definiert den Prozess
  - das Programm legt damit den Prozess fest, gibt ihn vor
  - gegebenenfalls bewirkt, steuert, terminiert es gar andere Prozesse
- ein Programm beschreibt (auch) die Art des Ablaufs eines Prozesses
  - sequentiell** ■ eine Folge von zeitlich nicht überlappenden Aktionen
    - verläuft deterministisch, das Ergebnis ist determiniert
  - parallel** ■ nicht sequentiell
- in beiden Arten besteht ein Programmablauf aus **Aktionen**

### Beachte: Programmablauf und Abstraktionsebene

Ein und derselbe Programmablauf kann auf einer Abstraktionsebene sequentiell, auf einer anderen parallel sein.



- zentrale Aufgabe ist es, über die **Speicherzuteilung** an einen Prozess Buch zu führen und seine Adressraumgröße dazu passend auszulegen  
**Platzierungsstrategie** (*placement policy*)
  - wo im Hauptspeicher ist noch Platz?
- zusätzliche Aufgabe kann die **Speichervirtualisierung** sein, um trotz knappem Hauptspeicher Mehrprogrammbetrieb zu maximieren  
**Ladestrategie** (*fetch policy*)
  - wann muss ein Datum im Hauptspeicher liegen?**Ersetzungsstrategie** (*replacement policy*)
  - welches Datum im Hauptspeicher ist ersetzbar?
- die zur Durchführung dieser Aufgaben typischerweise zu verfolgenden Strategien profitieren voneinander — oder bedingen einander
  - ein Datum kann ggf. erst platziert werden, wenn Platz freigemacht wurde
  - etwa indem das Datum den Inhalt eines belegten Speicherplatzes ersetzt
  - ggf. aber ist das so ersetzte Datum später erneut zu laden
  - bevor ein Datum geladen werden kann, ist Platz dafür bereitzustellen



## Speicherverwaltung II

- normalerweise sind die **Verantwortlichkeiten** auf mehrere Ebenen innerhalb eines Rechensystems verteilt  
**Speicherzuteilung** ■ Maschinenprogramm und Betriebssystem
  - Haldenspeicher, Hauptspeicher**Speichervirtualisierung** ■ ist allein Aufgabe des Betriebssystems
  - Haupt-/Arbeitsspeicher, Ablagespeicher
- das Maschinenprogramm verwaltet den seinem Prozess (-adressraum) jeweils zugeteilten Speicher **lokal** eigenständig
  - nämlich den Haldenspeicher  $\leadsto$  `malloc/free`
  - stellt dabei **sprachenorientierte Kriterien** in den Vordergrund
- das Betriebssystem verwaltet den gesamten Haupt-/Arbeitsspeicher **global** für alle Prozesse bzw. Prozessadressräume
  - stellt dabei **systemorientierte Kriterien** in den Vordergrund
  - hilft, einen Haldenspeicher zu verwalten  $\leadsto$  z.B. `sbrk/mmap`
- Maschinenprogramm und Betriebssystem gehen somit eine **Symbiose** ein, sie nehmen eine **Arbeitsteilung** vor
  - genauer gesagt: das Laufzeitsystem (`libc`) im Maschinenprogramm



### ■ Seitennummerierung (*paging*)

- jede Adresse wird interpretiert als Tupel  $A_p = (p, o)$ , wobei
  - Oktettnummer**  $o = [0, 2^i - 1]$ , mit  $9 \leq i \leq 30$
  - Seitennummer**  $p = [0, 2^{n-i} - 1]$ , mit  $32 \leq n \leq 64$
- übliche Einkomponentenadresse  $\leadsto$  **eindimensionaler Adressraum**
  - d.h., Oktetts oder Worte in einer Dimension aufgereiht

### ■ Segmentierung (*segmentation*)

- jede Adresse ist repräsentiert als Paar  $A_s = (s, a)$ , wobei
  - Segmentname**  $s = [0, 2^m - 1]$ , mit  $12 \leq m \leq 18$
  - Adresse**  $a = [0, 2^n - 1]$ , mit  $32 \leq n \leq 64$
- Zweikomponentenadresse  $\leadsto$  **zweidimensionaler Adressraum**
  - d.h., Segmente in der ersten und Segmentinhalte in der zweiten Dimension

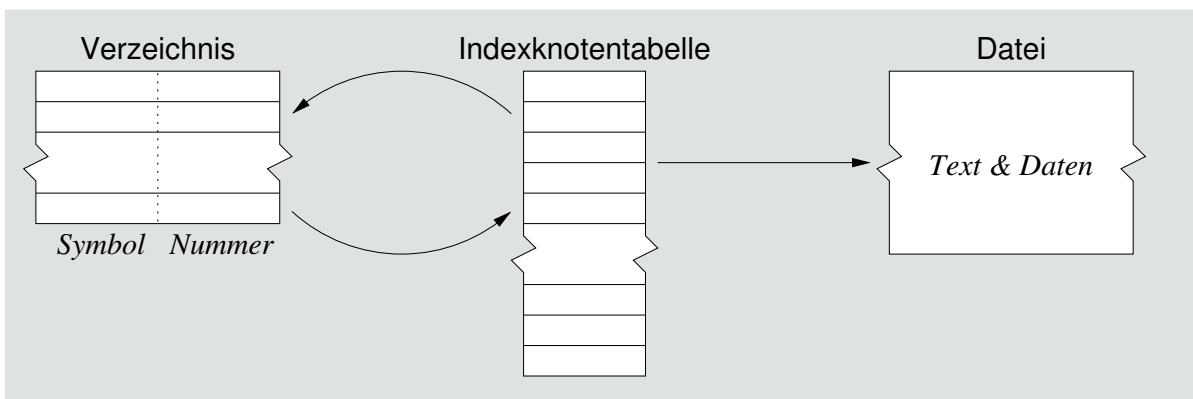
### ■ seitennummerierte Segmentierung (*paged segmentation*)

- jedes Segment ist seitennummeriert ausgelegt, d.h.,  $a$  in  $A_s$  entspricht  $A_p$



# Organisation des Ablagespeichers<sup>2</sup>

## Dateisystem (*file system*)



- die **Indexknotentabelle** (*inode table*) ist ein statisches Feld (*array*) von Indexknoten und die zentrale Datenstruktur
  - ein Indexknoten ist **Deskriptor** insb. eines Verzeichnisses oder einer Datei
- das **Verzeichnis** (*directory*) ist eine **Abbildungstabelle**, es übersetzt symbolisch repräsentierte Namen in Indexknotennummern
  - eine von der Namensverwaltung des Betriebssystems definierte Datei
- die **Datei** (*file*) ist eine abgeschlossene Einheit zusammenhängender Daten beliebiger Repräsentation, Struktur und Bedeutung

<sup>2</sup>Als Einheit auf demselben Medium (z.B. Ablagespeicher) abgelegt.





- **abgesetzter Betrieb:** Satellitenrechner, Hauptrechner
  - Entlastung durch Spezialrechner
- **überlappte Ein-/Ausgabe:** DMA, *Interrupts*
  - nebenläufige Programmausführung
- **überlappte Auftragsverarbeitung:** Einplanung, Vorgriff
  - Verarbeitungsstrom von Aufträgen
- **abgesetzte Ein-/Ausgabe:** *Spooling*
  - Entkopplung durch Pufferbereiche
- **Mehrprogrammbetrieb:** *Multiprogramming*
  - Multiplexen der CPU
- **dynamisches Laden:** Überlagerung (*overlay*)
  - programmiertes Nachladen von Programmbestandteilen



# Mehrzugangsbetrieb

- **Dialogbetrieb:** Dialogstationen
  - mehrere Benutzer gleichzeitig bedienen können
- **Hintergrundbetrieb:** Mischbetrieb
  - Programme **im Vordergrund starten**
- **Teilnehmerbetrieb:** Zeitscheibe, *Timesharing*
  - eigene Dialogprozesse absetzen können
- **Teilhhaberbetrieb:** residente Dialogprozesse
  - sich gemeinsame Dialogprozesse teilen können
- **Multiprozessorbetrieb:** Parallelrechner, SMP
  - Parallelverarbeitung von Programmen
- **Speicheraustausch:** *Swapping, Paging*
  - von ganzen Prozessadressräumen oder einzelnen Bestandteilen



- externe (physikalische) Prozesse definieren, was genau bei einer nicht termingerecht geleisteten Berechnung zu geschehen hat:
  - weich** (*soft*) auch „schwach“
    - das Ergebnis ist weiterhin von Nutzen, verliert jedoch mit jedem weiteren Zeitverzug des internen Prozesses zunehmend an Wert
    - die Terminverletzung ist tolerierbar
  - fest** (*firm*) auch „stark“
    - das Ergebnis ist wertlos, wird verworfen, der interne Prozess wird abgebrochen und erneut bereitgestellt
    - die Terminverletzung ist tolerierbar
  - hart** (*hard*) auch „strikt“
    - Verspätung der Ergebnislieferung kann zur „Katastrophe“ führen, dem internen Prozess wird eine **Ausnahmesituation** zugestellt
    - Terminverletzung ist keinesfalls tolerierbar — aber möglich. . .
- ggf. zusätzlich geforderte Randbedingung ist die Termineinhaltung unter allen Last- und Fehlerbedingungen



## Gliederung

### SP1

Lehrziele

C

UNIX

Einleitung

Rechnerorganisation

Betriebssystemkonzepte

Betriebsarten

### SP2

Ausblick



## ■ Prozessverwaltung

- Einplanung (klassisch, Fallstudien)
- Koroutinen, Programmfäden, Einlastung

## ■ Synchronisation

- ein-/mehrseitig, blockierend/nicht-blockierend
- Verklemmungen (Gegenmaßnahmen, Auflösung)

## ■ Speicherverwaltung

- Adressräume, MMU (Pentium)
- Disziplinen, virtueller Speicher, Arbeitsmenge

## ■ Dateiverwaltung

- Organisation des Hintergrundspeichers
- Datenverfügbarkeit (RAID)



# Literaturverzeichnis I

- [1] TENNENHOUSE, D. :  
Proactive Computing.  
In: *Communications of the ACM* 43 (2000), Mai, Nr. 5, S. 43–50

