

# Systemprogrammierung

*Grundlagen von Betriebssystemen*

Teil B – VIII. Zwischenbilanz

Wolfgang Schröder-Preikschat

21. Juli 2020



# Agenda

---

## SP1

Lehrziele

C

UNIX

Einleitung

Rechnerorganisation

Betriebssystemkonzepte

Betriebsarten

## SP2

Ausblick



# Gliederung

---

## SP1

Lehrziele

C

UNIX

Einleitung

Rechnerorganisation

Betriebssystemkonzepte

Betriebsarten

## SP2

Ausblick



## Definition (Systemprogrammierung)

Erstellen von Softwareprogrammen, die Teile eines Betriebssystems sind beziehungsweise mit einem Betriebssystem direkt interagieren oder die Hardware (genauer: Zentraleinheit<sup>a</sup> und Peripherie<sup>b</sup>) eines Rechensystems betreiben müssen.

<sup>a</sup> *central processing unit* (CPU), ein-/mehrfach, ein-, mehr- oder vielkernig.

<sup>b</sup> Geräte zur Ein-/Ausgabe oder Steuerung/Regelung „externer Prozesse“.

Auch schon mal zwischen zwei Stühlen sitzend:

- **Anwendungssoftware** („oben“) einerseits
  - ermöglichen, unterstützen, nicht entgegenwirken
- **Plattformsysteme** („unten“) andererseits
  - anwendungsspezifisch verfügbar machen
  - problemorientiert betreiben, bedingt verbergen
    - nachteilige Eigenschaften versuchen zu kaschieren



Quelle: arcadja.com, Franz Kott



## 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*

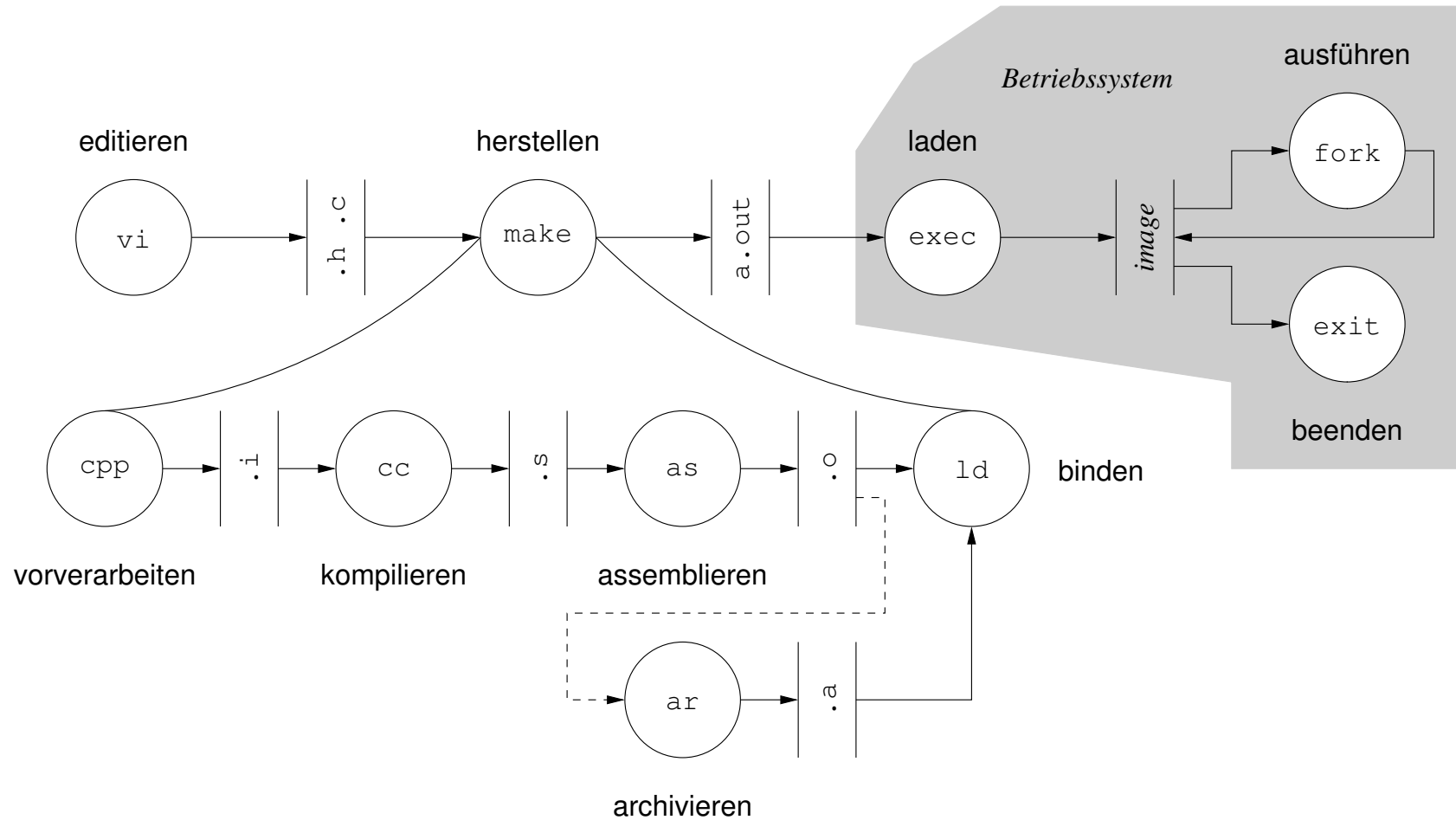
### ■ was geschieht nun?

```
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 $\mapsto$ Prozess

[2] und [3, S. 26ff.]



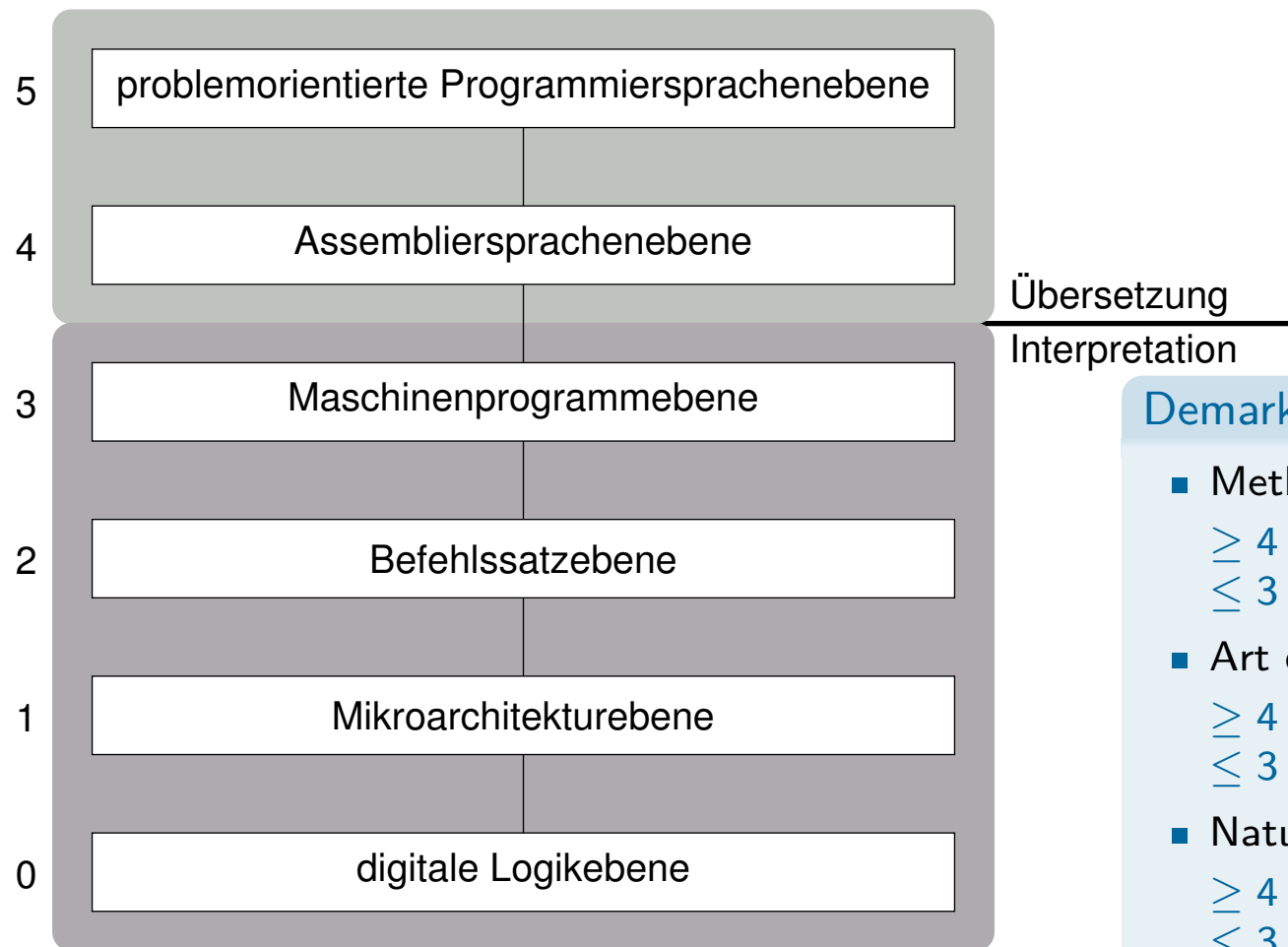
Die Funktionsweise (auch) von Betriebssystemen zu verstehen, hilft **bemerkenswerte Erscheinungen** innerhalb eines Rechensystems zu begreifen und in ihrer Bedeutung besser einzuschätzen.

- **Eigenschaften** (*features*) von Betriebssystemen erkennen:
  - funktionale** ■ Verwaltung der Betriebsmittel (Prozessor, Speicher, Peripherie) für eine Anwendungsdomäne
  - nichtfunktionale** ■ dabei anfallender Zeit-, Speicher-, Energieverbrauch
    - d.h., **Gütemerkmale** einer Implementierung
- aus den funktionalen Eigenschaften resultierendes **Systemverhalten** unterscheiden von Fehlern (*bugs*) des Systems
  - um Fehler kann ggf. „herum programmiert“ werden
  - um zum Anwendungsfall unpassende Eigenschaften oft jedoch nicht

---

<sup>1</sup>Analytische Lernmethode, die die Vermittlung eines Stoffes als Gesamtheit in den Mittelpunkt stellt, um dann konstituierende Elemente weiter zu untersuchen.





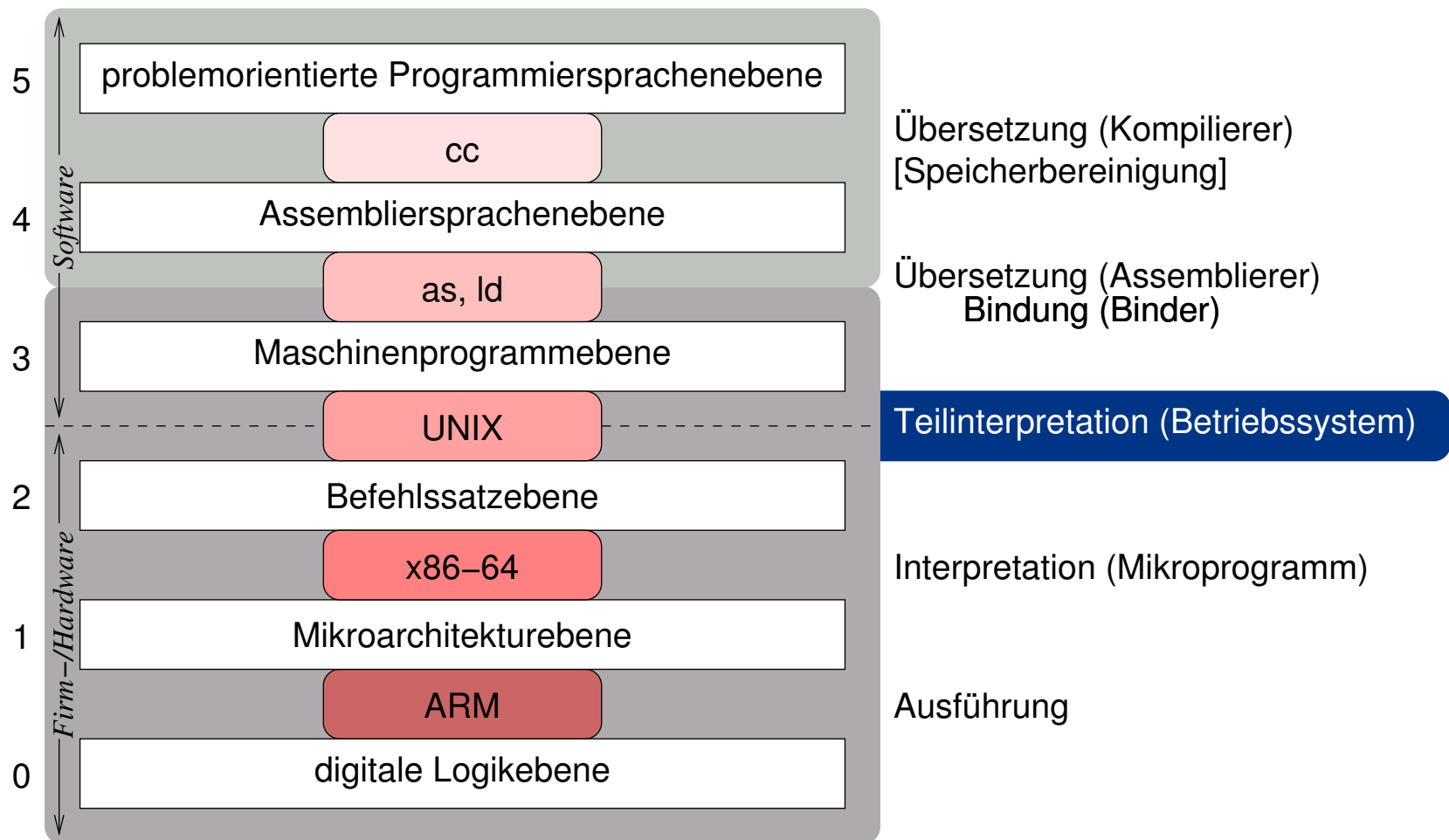
## Demarkationslinie

- Methode der Abbildung
  - $\geq 4$  Übersetzung vs.
  - $\leq 3$  Interpretation
- Art der Programmierung
  - $\geq 4$  Anwendung vs.
  - $\leq 3$  System
- Natur der Sprache
  - $\geq 4$  symbolisch vs.
  - $\leq 3$  numerisch

- 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







- 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)



```
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 einen **Moduswechsel**
- sorgt für **Ortstransparenz** (funktional)
  - die Lokalität der aufgerufenen Systemfunktion 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>2</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)



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
    - wenn das Betriebssystem die dazu nötigen Befehle anbietet!
- 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

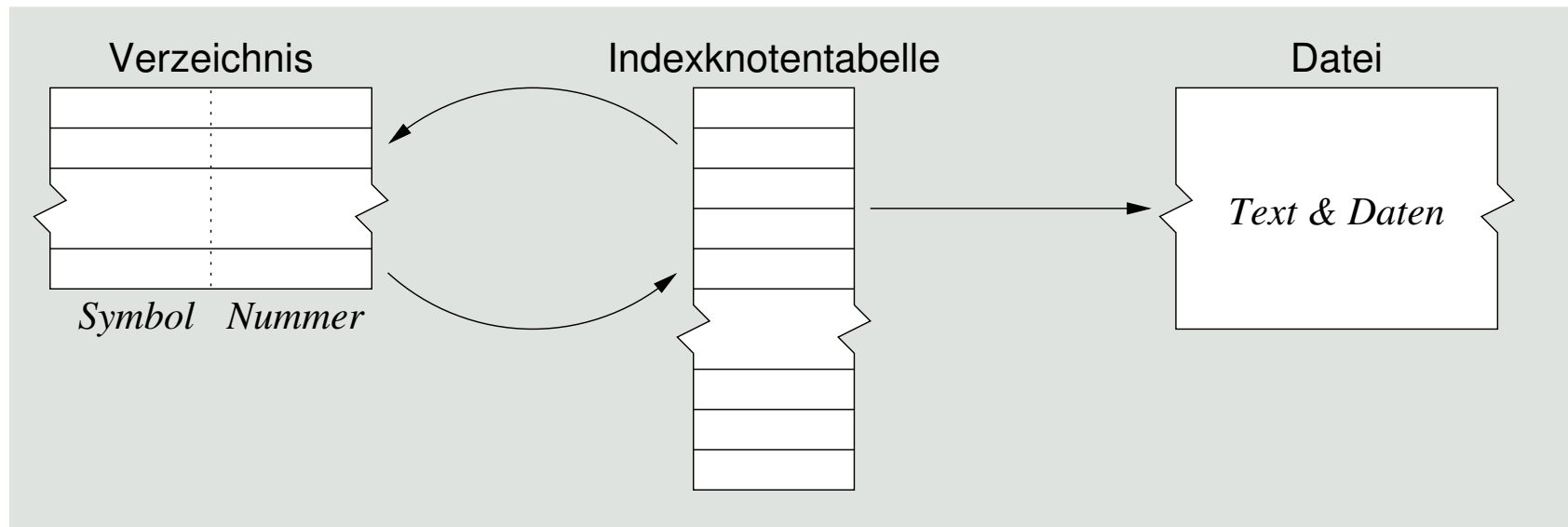


- normalerweise sind die **Verantwortlichkeiten** auf mehrere Ebenen innerhalb eines Rechensystems verteilt
  - Speicherzuteilung**
    - Maschinenprogramm und Betriebssystem
    - Haldenspeicher, Hauptspeicher
  - Speichervirtualisierung**
    - ist allein Aufgabe des Betriebssystems
    - Haupt-/Arbeitsspeicher, Ablage
- das Maschinenprogramm verwaltet den seinem Prozess (-adressraum) jeweils zugeteilten Speicher **lokal** eigenständig
  - stellt dabei **sprachenorientierte Kriterien** in den Vordergrund
    - typisch für den Haldenspeicher  $\leadsto$  malloc/free
- das Betriebssystem verwaltet den gesamten Haupt-/Arbeitsspeicher **global** für alle Prozessexemplare bzw. -adressräume
  - 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



- **Seitenadressierung** (*paging*) mittels **Seitentabelle** [11, S. 29–30]
  - jede von der CPU generierte Adresse wird gedeutet als  $A_p = (p, o)$ , wobei
    - Versatz**  $o = [0, 2^w - 1]$ , mit  $9 \leq w \leq 30$  (*offset*)
    - Seitennummer**  $p = [0, 2^{n-w} - 1]$ , mit  $32 \leq n \leq 64$ , Tabellenindex
  - eine gewöhnliche **lineare Adresse**  $\leadsto$  **eindimensionaler Adressraum**
    - d.h., Oktetts oder Worte in einer Dimension aufgereiht
- **Segmentierung** (*segmentation*) mittels **Segmenttabelle** [3, S. 37]
  - jede Adresse ist repräsentiert als Zweitupel  $A_s = \langle s, d \rangle$ , wobei
    - Segmentname**  $s = [0, 2^m - 1]$ , mit  $12 \leq m \leq 18$ , Tabellenindex
    - Verschiebung**  $d = [0, 2^n - 1]$ , mit  $32 \leq n \leq 64$  (*displacement*)
  - Zweikomponentenadresse  $\leadsto$  **zweidimensionaler Adressraum**
    - d.h., Segmente in der ersten und Segmentinhalte in der zweiten Dimension
- Kombination (vgl. [3, S. 38–40]):
  - **segmentierte Seitenadressierung** (*segmented paging*)
    - die Seitentabellen sind segmentiert, d.h.,  $A_p = (p, o)$  mit  $p = (s, d)$
  - **seitennummerierte Segmentierung** (*paged segmentation*)
    - die Segmente sind seitennummeriert, d.h.,  $A_s = \langle s, d \rangle$  mit  $d = (p, o)$  oder die Segmenteinheit generiert eine lineare Adresse  $A_p$  für die Seiteneinheit





- 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





- **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



- **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] KLEINÖDER, J. :  
Kurzeinführung in C.  
In: [13], Kapitel 2
  
- [2] KLEINÖDER, J. :  
Vom C-Programm zum UNIX-Prozess.  
In: [13], Kapitel 3
  
- [3] KLEINÖDER, J. ; SCHRÖDER-PREIKSCHAT, W. :  
Adressbindung.  
In: [13], Kapitel 6.3
  
- [4] KLEINÖDER, J. ; SCHRÖDER-PREIKSCHAT, W. :  
Betriebssystemmaschine.  
In: [13], Kapitel 5.3
  
- [5] KLEINÖDER, J. ; SCHRÖDER-PREIKSCHAT, W. :  
Dialog- und Echtzeitverarbeitung.  
In: [13], Kapitel 7.2
  
- [6] KLEINÖDER, J. ; SCHRÖDER-PREIKSCHAT, W. :  
Einführung.  
In: [13], Kapitel 4



# Literaturverzeichnis II

---

- [7] KLEINÖDER, J. ; SCHRÖDER-PREIKSCHAT, W. :  
Maschinenprogramme.  
In: [13], Kapitel 5.2
- [8] KLEINÖDER, J. ; SCHRÖDER-PREIKSCHAT, W. :  
Organisation.  
In: [13], Kapitel 1
- [9] KLEINÖDER, J. ; SCHRÖDER-PREIKSCHAT, W. :  
Prozesse.  
In: [13], Kapitel 6.1
- [10] KLEINÖDER, J. ; SCHRÖDER-PREIKSCHAT, W. :  
Rechnerorganisation.  
In: [13], Kapitel 5.1
- [11] KLEINÖDER, J. ; SCHRÖDER-PREIKSCHAT, W. :  
Speicher.  
In: [13], Kapitel 6.2
- [12] KLEINÖDER, J. ; SCHRÖDER-PREIKSCHAT, W. :  
Stapelverarbeitung.  
In: [13], Kapitel 7.1



# Literaturverzeichnis III

---

- [13] KLEINÖDER, J. ; SCHRÖDER-PREIKSCHAT, W. ; LEHRSTUHL INFORMATIK 4 (Hrsg.):  
*Systemprogrammierung*.  
FAU Erlangen-Nürnberg, 2015 (Vorlesungsfolien)

