

Systemprogrammierung

Grundlagen von Betriebssystemen

Teil B – VI.1 Rechnerorganisation: Virtuelle Maschinen

26. Juni 2025

Rüdiger Kapitza

(© Wolfgang Schröder-Preikschat, Rüdiger Kapitza)



Lehrstuhl für Informatik 4
Systemsoftware



Friedrich-Alexander-Universität
Technische Fakultät

Einführung

Schichtenstruktur

Semantische Lücke

Fallstudie

Mehrebenenmaschinen

Maschinenhierarchie

Maschinen und Prozessoren

Entvirtualisierung

Ausnahmesituation

Zusammenfassung

Einführung

Schichtenstruktur

Semantische Lücke

Fallstudie

Mehrebenenmaschinen

Maschinenhierarchie

Maschinen und Prozessoren

Entvirtualisierung

Ausnahmesituation

Zusammenfassung

- Rechengsysteme als eine **Schichtenfolge** von Maschinen
 - die eine **funktionale Hierarchie** [7] von spezifischen Maschinen zur Ausführung von Programmen darstellt
 - wobei manche dieser Maschinen nicht in Wirklichkeit vorhanden sind, sein müssen oder sein können
 - die somit jeweils als eine **virtuelle Maschine** [11] in Erscheinung treten
- **Abstraktionshierarchie** als Basis Rechengsystemkonstruktionen
 - in der die einzelnen Schichten durch **Prozessoren** implementiert werden, die vor (*off-line*) oder zur (*on-line*) Programmausführungszeit wirken
 - wobei ein Prozessor als **Übersetzer** oder **Interpreter** ausgelegt ist
- Platz für das **Betriebssystem** in dieser Hierarchie ausmachen
 - erkennen, dass ein Betriebssystem ein spezieller Interpreter ist und den Befehlssatz wie auch die Funktionalität einer CPU erweitert
 - die **Symbiose** insbesondere von Betriebssystem und CPU verinnerlichen
- Grundlagen eines „Weltbilds“ legen, das zentral für SP ist

Einführung

Schichtenstruktur

Semantische Lücke

Fallstudie

Mehrebenenmaschinen

Maschinenhierarchie

Maschinen und Prozessoren

Entvirtualisierung

Ausnahmesituation

Zusammenfassung

Schichtenstruktur

Semantische Lücke

Verschiedenheit zwischen Quell- und Zielsprache

Faustregel: $\left\{ \begin{array}{ll} \text{Quellsprache} & \rightarrow \text{höheres} \\ \text{Zielsprache} & \rightarrow \text{niedrigeres} \end{array} \right\} \text{ Abstraktionsniveau}$

Semantische Lücke (*semantic gap*, [14])

The difference between the complex operations performed by high-level constructs and the simple ones provided by computer instruction sets.

It was in an attempt to try to close this gap that computer architects designed increasingly complex instruction set computers.

- Kluft zwischen gedanklich Gemeintem und sprachlich Geäußertem

Schichtenstruktur

Fallstudie

Beispiel: Matrizenmultiplikation

Problemraum



(Mathematik)



Lösungsraum



(Informatik)

- „gedanklich gemeint“ ist ein Verfahren aus der linearen Algebra
- „sprachlich geäußert“ auf verschiedenen Ebenen der **Abstraktion**

- Multiplikation von zwei 2×2 Matrizen:

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

Zwei Matrizen werden multipliziert, indem die Produktsummenformel auf Paare aus einem Zeilenvektor der ersten und einem Spaltenvektor der zweiten Matrix angewandt wird.

- Produktsummenformel für $C = A \times B$: $C_{i,j} = \sum_k A_{ik} \cdot B_{kj}$

- Skalarprodukt oder „inneres Produkt“, Quellmodul (multiply.c):

```
1  typedef int Matrix [N][N];
2
3  void multiply(in Matrix a, in Matrix b, out Matrix c) {
4      unsigned int i, j, k;
5      for (i = 0; i < N; i++)
6          for (j = 0; j < N; j++) {
7              c[i][j] = 0;
8              for (k = 0; k < N; k++)
9                  c[i][j] += a[i][k] * b[k][j];
10         }
11     }
```

- **Konkretisierung** für zwei $N \times N$ Matrizen: $c = a \times b$
 - ausgelegt als Unterprogramm: Prozedur $\mapsto$ C function
- insgesamt sechs Varianten (d.h., Schleifenanordnungen)
 - $\{ijk, jik, ikj, jki, kij, kji\}$: funktional gleich, nichtfunktional ggf. ungleich

Ebene informatischer Sprache: ASM [8, 4]

```
1 .file "multiply.c"
2 .text
3 .p2align 4,,15
4 .globl multiply
5 .type multiply,@function
6 multiply:
7     pushl %ebp
8     movl %esp,%ebp
9     pushl %edi
10    pushl %esi
11    pushl %ebx
12    subl $4,%esp
13    movl 16(%ebp),%esi
14    movl $0,-16(%ebp)
15    .L2:
16    movl 8(%ebp),%edi
17    xorl %ebx,%ebx
18    addl -16(%ebp),%edi
19    .p2align 4,,7
20    .p2align 3
21    .L4:
22    movl 12(%ebp),%eax
23    xorl %edx,%edx
24    movl $0,(%esi,%ebx,4)
25    leal (%eax,%ebx,4),%ecx
26    .p2align 4,,7
27    .p2align 3
28    .L3:
29    movl (%ecx),%eax
30    addl $400,%ecx
31    imull (%edi,%edx,4),%eax
32    addl $1,%edx
33    addl %eax,(%esi,%ebx,4)
34    cmpl $100,%edx
35    jne .L3
36    addl $1,%ebx
37    cmpl $100,%ebx
38    jne .L4
39    addl $400,-16(%ebp)
40    addl $400,%esi
41    cmpl $40000,-16(%ebp)
42    jne .L2
43    addl $4,%esp
44    popl %ebx
45    popl %esi
46    popl %edi
47    popl %ebp
48    ret
49    .size multiply,.-multiply
50    .ident "GCC: (Debian 4.3.2-1.1) 4.3.2"
51    .section .note.GNU-stack,"",@progbits
```

■ Kompilation der Quelle in ein semantisch äquivalentes Programm

- Trick: Übersetzung der Quelle vor dem **Assemblieren** beenden

– Übersetzung¹ von multiply.c mit -DN=100: C function ↦ ASM/x86

¹gcc -O -m32 -fomit-frame-pointer -fno-pic -static -S

Ebene informatischer Sprache: a.out [8, 2]

```
00000000 457f 464c 0101 0001 0000 0000 0000 0000
00000020 0001 0003 0001 0000 0000 0000 0000 0000
00000040 0114 0000 0000 0000 0034 0000 0000 0028
00000060 0009 0006 0000 0000 0000 0000 0000 0000
00001000 8955 57e5 5356 ec83 8b04 1075 45c7 00f0
00001020 0000 8b00 087d db31 7d03 90f0 748d 0026
00001040 458b 310c c7d2 9e04 0000 0000 0c8d 9098
00001060 018b c181 0190 0000 af0f 9704 c283 0101
00002000 9e04 fa83 7564 83e9 01c3 fb83 7564 81d1
00002020 f045 0190 0000 c681 0190 0000 7d81 40f0
00002040 009c 7500 83ae 04c4 5e5b 5d5f 00c3 0000
00002060 4700 4343 203a 4428 6265 6169 206e 2e34
00003000 2e33 2d32 2e31 2931 3420 332e 322e 0000
00003020 732e 6d79 6174 0062 732e 7274 6174 0062
00003040 732e 7368 7274 6174 0062 742e 7865 0074
00003060 642e 7461 0061 622e 7373 2e00 6f63 6d6d
00004000 6e65 0074 6e2e 746f 2e65 4e47 2d55 7473
00004020 6361 006b 0000 0000 0000 0000 0000 0000
00004040 0000 0000 0000 0000 0000 0000 0000 0000
00004060 0000 0000 0000 0000 0000 0000 001b 0000
00005000 0001 0000 0006 0000 0000 0000 0040 0000
00005020 006d 0000 0000 0000 0000 0000 0010 0000
00005040 0000 0000 0021 0000 0001 0000 0003 0000
00005060 0000 0000 00b0 0000 0000 0000 0000 0000
00006000 0000 0000 0004 0000 0000 0000 0027 0000
00006020 0000 0000 0000 0000 0000 0000 0000 0000
00006040 0000 0000 0000 0000 0000 0000 0004 0000
00006060 0000 0000 0000 0000 0000 0000 0000 0000
00006080 0000 0000 0000 0000 0000 0000 0000 0000
000060a0 0000 0000 0000 0000 0000 0000 0000 0000
000060c0 0000 0000 0000 0000 0000 0000 0000 0000
000060e0 0000 0000 0000 0000 0000 0000 0000 0000
00006100 0000 0000 0000 0000 0000 0000 0000 0000
00006120 0000 0000 0000 0000 0000 0000 0000 0000
00006140 0000 0000 0000 0000 0000 0000 0000 0000
00006160 0000 0000 0000 0000 0000 0000 0000 0000
00006180 0000 0000 0000 0000 0000 0000 0000 0000
000061a0 0000 0000 0000 0000 0000 0000 0000 0000
000061c0 0000 0000 0000 0000 0000 0000 0000 0000
000061e0 0000 0000 0000 0000 0000 0000 0000 0000
00006200 0000 0000 0000 0000 0000 0000 0000 0000
00006220 0000 0000 0000 0000 0000 0000 0000 0000
00006240 0000 0000 0000 0000 0000 0000 0000 0000
00006260 0000 0000 0000 0000 0000 0000 0000 0000
00006280 0000 0000 0000 0000 0000 0000 0000 0000
000062a0 0000 0000 0000 0000 0000 0000 0000 0000
000062c0 0000 0000 0000 0000 0000 0000 0000 0000
000062e0 0000 0000 0000 0000 0000 0000 0000 0000
00006300 0000 0000 0000 0000 0000 0000 0000 0000
00006320 0000 0000 0000 0000 0000 0000 0000 0000
00006340 0000 0000 0000 0000 0000 0000 0000 0000
00006360 0000 0000 0000 0000 0000 0000 0000 0000
00006380 0000 0000 0000 0000 0000 0000 0000 0000
000063a0 0000 0000 0000 0000 0000 0000 0000 0000
000063c0 0000 0000 0000 0000 0000 0000 0000 0000
000063e0 0000 0000 0000 0000 0000 0000 0000 0000
00006400 0000 0000 0000 0000 0000 0000 0000 0000
00006420 0000 0000 0000 0000 0000 0000 0000 0000
00006440 0000 0000 0000 0000 0000 0000 0000 0000
00006460 0000 0000 0000 0000 0000 0000 0000 0000
00006480 0000 0000 0000 0000 0000 0000 0000 0000
000064a0 0000 0000 0000 0000 0000 0000 0000 0000
000064c0 0000 0000 0000 0000 0000 0000 0000 0000
000064e0 0000 0000 0000 0000 0000 0000 0000 0000
00006500 0000 0000 0000 0000 0000 0000 0000 0000
00006520 0000 0000 0000 0000 0000 0000 0000 0000
00006540 0000 0000 0000 0000 0000 0000 0000 0000
00006560 0000 0000 0000 0000 0000 0000 0000 0000
00006580 0000 0000 0000 0000 0000 0000 0000 0000
000065a0 0000 0000 0000 0000 0000 0000 0000 0000
000065c0 0000 0000 0000 0000 0000 0000 0000 0000
000065e0 0000 0000 0000 0000 0000 0000 0000 0000
00006600 0000 0000 0000 0000 0000 0000 0000 0000
00006620 0000 0000 0000 0000 0000 0000 0000 0000
00006640 0000 0000 0000 0000 0000 0000 0000 0000
00006660 0000 0000 0000 0000 0000 0000 0000 0000
00006680 0000 0000 0000 0000 0000 0000 0000 0000
000066a0 0000 0000 0000 0000 0000 0000 0000 0000
000066c0 0000 0000 0000 0000 0000 0000 0000 0000
000066e0 0000 0000 0000 0000 0000 0000 0000 0000
00006700 6974 6c70 2e79 0063 756d 746c 7069 796c
00006720 0000
```

■ Assemblieren der kompilierten Quelle und Ausgabebereitstellung

- Hexadezimalcode **ausführbar** – jedoch kein ausführbares Programm!

1. `as multiply.s: ASM/x86 ↦ a.out/x86` (Binde-/Lademodul)
2. `od -x a.out ↪ hexadezimaler (-x) Speicherauszug (dump, od)`

Ebene informatischer Sprache: Binärkode

```
010101011000100111100101010111100101011001010011100000111110110000000100
10001011111001010001000011000111010001011111000000000000000000000000000000
000000001000101111011010000100000110001110110110000001101111011110000
100100001000110101110100001001100000000010001011010001010000110000110001
11010010110001110000010010011110000000000000000000000000000000000000000000
00001100100110001001000010000000110000001110000011001000000000000000000000
0000000000000000000000001111101011110000010010010111100000111100001000000001
000000010000010010011110100000111111010011001000111010111101001100000011
110000110000000110000011111101101100100011101011101000110000000101000101
11110000100100000000000001000000000000000000000000000000000000000000000000
00000000000000000000000010000001011110111110000010000001001110000000000000000
011010110101110100000111100010100000100010110101011110010111101011101
11000011
```

hervorgehobene Bitfolgen repräsentieren (durch `.p2align`, vgl. S. 12) aufgefüllte Nulloperationen

- **Auflösung** des ausführbaren Hexadezimalcodes zur **Bitfolge**
 - die Befehlsverarbeitung geschieht bitweise, nicht byte- oder wortweise

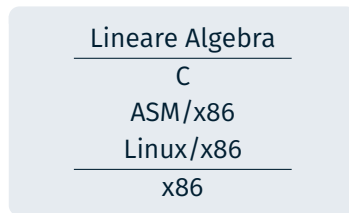
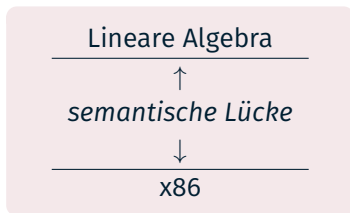
Die für einen Digitalrechner letztendlich benötigte Form — obwohl die CPU wortweise^a auf den Speicher zugreift, mit einer Wortbreite von 8–64 Bits, je nach Prozessortechnologie.

^aGemeint ist das **Maschinenwort**, keine plattformunabhängige Dateneinheit.

Abstraktionshierarchie von Sprachsystemen

- **Modellsprache** (Lineare Algebra) $\leadsto$ 1 Produktsummenformel
- **Programmiersprache** (C) $\leadsto$ 5 Komplexschritte
- **Assemblersprache** (ASM/x86) $\leadsto$ $35+n$ Elementarschritte
- **Maschinensprache** (Linux/x86) $\leadsto$ 109 Bytes Programmtext
(x86) $\leadsto$ 872 Bits

→ eine einzelne komplexe und überwältigende Aufgabe in mehrere kleine und handhabbare unterteilen



Einführung

Schichtenstruktur

Semantische Lücke

Fallstudie

Mehrebenenmaschinen

Maschinenhierarchie

Maschinen und Prozessoren

Entvirtualisierung

Ausnahmesituation

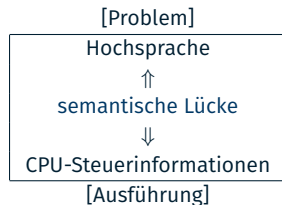
Zusammenfassung

Mehrebenenmaschinen

Maschinenhierarchie

Aufgabenstellung $\mapsto$ Programmlösung

- das Ausmaß der semantischen Lücke gestaltet sich fallabhängig:
 - bei gleich bleibendem Problem mit der Plattform (dem System)
 - bei gleich bleibender Plattform mit dem Problem (der Anwendung)



- der Lückenschluss ist ganzheitlich zu sehen und auch anzugehen
 - Schicht für Schicht die innere (logische) Struktur des Systems herleiten
 - das System, das die Lücke schließen soll, als Ganzes als „Bild“ erfassen
 - hinsichtlich benötigter funktionalen und nicht-funktionalen Eigenschaften
- Kunst der kleinen Schritte: seman. Lücke schrittweise schließen
 - durch hierarchisch angeordnete **virtuelle Maschinen** Programmlösungen auf die reale Maschine herunterbrechen [12]
 - Prinzip *divide et impera* („teile und herrsche“)
 - einen „Gegner“ in leichter besiegbare „Untergruppen“ aufspalten

■ Interpretation und Übersetzung (Kompilation, Assemblieren):

Ebene n	virtuelle Maschine M_n mit Maschinensprache S_n	Programme in S_n werden von einem auf einer tieferen Maschine laufenden Interpreter gedeutet oder in Pro- gramme tieferer Maschinen übersetzt
$\vdots$	$\vdots$	$\vdots$
2	virtuelle Maschine M_2 mit Maschinensprache S_2	Programme in S_2 werden von einem auf M_1 bzw. M_0 laufenden Interpreter gedeutet oder nach S_1 bzw. S_0 übersetzt
1	virtuelle Maschine M_1 mit Maschinensprache S_1	Programme in S_1 werden von einem auf M_0 laufenden Interpreter gedeutet oder nach S_0 übersetzt
0	reale Maschine M_0 mit Ma- schinensprache S_0	Programme in S_0 werden direkt von der Hardware ausgeführt

- Techniken, die einander unterstützend — teils sogar „symbiotisch“
— Verwendung finden, um Programme zur Ausführung zu bringen

- jede einzelne Ebene (d.h., Schicht) in der Hierarchie wird durch einen spezifischen Prozessor implementiert:

Kom|pi|la|tor lat. (Zusammenträger)

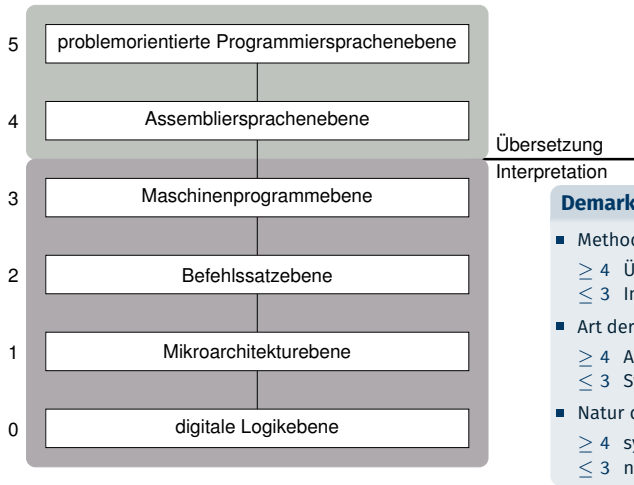
- ein **Softwareprozessor**, transformiert in einer *Quellsprache* vorliegende Programme in eine semantisch äquivalente Form einer *Zielsprache*
 - {Ada, C, C++, Eiffel, Modula, Fortran, Pascal, ...} $\mapsto$ Assembler
 - aber ebenso: C++ $\mapsto$ C $\mapsto$ Assembler

In|ter|pret lat. (Ausleger, Erklärer, Deuter)

- ein **Hard-, Firm- oder Softwareprozessor**, der die Programme direkt ausführt $\leadsto$ ausführbares Programm (*executable*)
 - z.B. Basic, Perl, C, sh(1), x86
- ggf. **Vorübersetzung** durch einen Kompilierer, um die Programme in eine für die Interpretation günstigere Repräsentation zu bringen
 - z.B. Pascal P-Code, Java Bytecode, x86-Befehle
- also abstrakte oder reale Prozessoren, die vor beziehungsweise zur Ausführungszeit des Programms wirken, das sie verarbeiten

Mehrebenenmaschinen

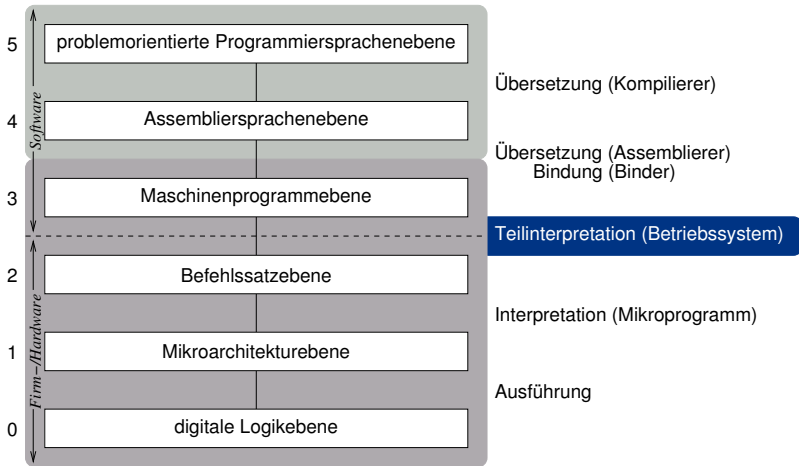
Maschinen und Prozessoren



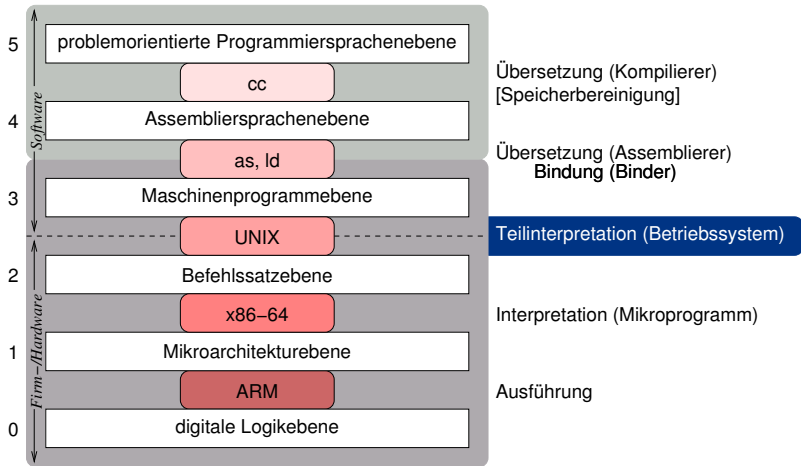
Demarkationslinie

- Methode der Abbildung
 - ≥ 4 Übersetzung vs.
 - ≤ 3 Interpretation
- Art der Programmierung
 - ≥ 4 Anwendung vs.
 - ≤ 3 System
- Natur der Sprache
 - ≥ 4 symbolisch vs.
 - ≤ 3 numerisch

- Schichten der Ebene_[4,5] sind nicht wirklich existent sondern
 - werden durch Übersetzung aufgelöst und auf tiefere Ebenen abgebildet
 - so dass am Ende nur ein Maschinenprogramm (Ebene₃) übrig bleibt

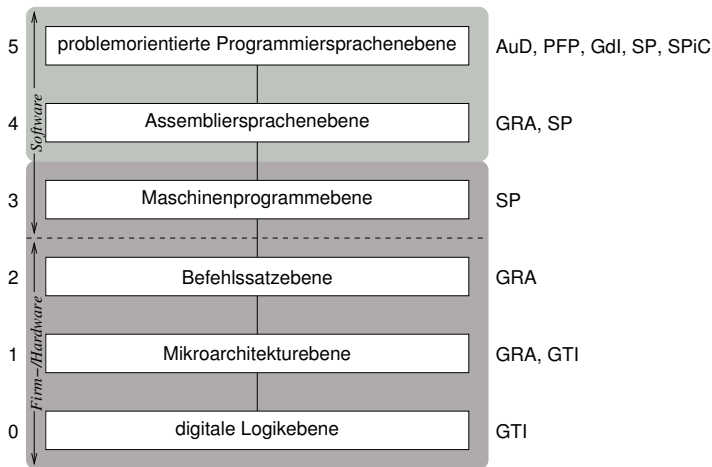


- Schichten der Ebene_[0,2] liegen normalerweise nicht in SW vor
 - sie können aber in Software simuliert/emuliert oder virtualisiert werden
 - dadurch lassen sich Rechnerysteme grundsätzlich **rekursiv** organisieren



- RISC auf Ebene₁ und gegebenenfalls (hier) CISC auf Ebene₂
 - nach außen „complex“, innen aber „reduced instruction set computer“
 - Intel Core oder Haswell ↔ AMD Bulldozer oder Zen (ARM)

Schichtenfolge eingebettet im Informatikstudiengang



- die Schicht auf Ebene₄ ist auch hier eher nur logisch existent ☺
 - Programmierung in Assemblersprache hat (leider) an Bedeutung verloren
 - Prinzipien werden in GRA vermittelt [5], in SP nur bei Bedarf behandelt

Mehrebenenmaschinen

Entvirtualisierung

- Schichten der Ebene_[3,5] repräsentieren **virtuelle Maschinen**, die auf die eine **reale Maschine** (Ebene_[0,2]) abzubilden sind
 - dabei werden diese Schichten „entvirtualisiert“, aufgelöst und zu einem **Maschinenprogramm** „verschmolzen“
 - dieser Vorgang hängt stark ab von der Art einer virtuellen Maschine²
- **Übersetzung**
 - aller Befehle des Programms, das der Ebene_{*i*} zugeordnet ist
 - in eine semantisch äquivalente Folge von Befehlen der Ebene_{*j*}, mit $j \leq i$
 - dadurch **Generierung** eines Programms, das der Ebene_{*j*} zugeordnet ist
- **Interpretation**
 - total**
 - aller Befehle des Programms, das der Ebene_{*i*} zugeordnet ist
 - partiell**
 - nur die Befehle des Programms, die der Ebene_{*i*} zugeordnet sind
 - wobei das Programm der Ebene_{*k*}, $k \geq i$, zugeordnet sein kann
 - durch **Ausführung** eines Programms der Ebene_{*j*}, mit $j \leq i$

²vgl. insb. [10]: die Folien sind Teil des ergänzenden Materials zu SP.

Abbildung durch Übersetzung

Ebene₅ $\mapsto$ Ebene₄ (Kompilation)

- Ebene₅-Befehle „1:N“, $N \geq 1$, in Ebene₄-Befehle übersetzen
 - einen Hochsprachenbefehl als mögliche Sequenz von Befehlen einer Assemblersprache implementieren
 - eine **semantisch äquivalente Befehlsfolge** generieren
- im Zuge der Transformation ggf. Optimierungsstufen durchlaufen

Ebene₄ $\mapsto$ Ebene₃ (Assemblieren)

- Ebene₄-Befehle „1:1“ in Ebene₃-Befehle übersetzen
 - ein **Quellmodul** in ein **Objektmodul** umwandeln
 - mit **Bibliotheken** zum Maschinenprogramm zusammenbinden
 - ein **Lademodul** erzeugen
- dabei den symbolischen Maschinencode (d.h., Mnemone) auflösen
 - in binären Maschinencode umwandeln
 - ADD EAX (Mnemon) $\mapsto$ 05₁₆ (Hexadezimalcode) $\mapsto$ 00000101₂ (Binärkode)
 - hier: Beispiel für den Befehlssatz x86-kompatibler Prozessoren

Abbildung durch Interpretation

$\text{Ebene}_3 \mapsto \text{Ebene}_2$ (*partielle* Interpretation, Teilinterpretation)

- Ebene₃-Befehle typ- und zustandsabhängig verarbeiten:
 - i als Folgen von Ebene₂-Befehlen ausführen
 - **Systemaufrufe** annehmen und befolgen, sensitive Ebene₂-Befehle emulieren
 - synchrone/asynchrone **Unterbrechungen** behandeln
 - ii „1:1“ auf Ebene₂-Befehle abbilden (nach unten „durchreichen“)
- ein Ebene₃-Befehl aktiviert im Fall von i ein Ebene₂-Programm
 - verursacht durch eine **Ausnahmesituation**, die durch Ebene₂ erkannt und zur Behandlung an ein Programm der Ebene₂ „hochgereicht“ wird
 - Ebene₂ stellt eine Falle (*trap*), bedient von einem Ebene₂-Programm

$\text{Ebene}_2 \mapsto \text{Ebene}_1$ (Interpretation)

- Ebene₂-Befehle als Folgen von Ebene₁-Aktionen ausführen
 - **Abruf- und Ausführungszyklus** (*fetch-execute-cycle*) der CPU
- ein Ebene₂-Befehl löst Ebene₁-Steueranweisungen aus

Zeitpunkte der Abbildungsvorgänge

Bezugspunkt ist das jeweils zu „prozessierende“ Programm:

■ **vor Laufzeit** ($\text{Ebene}_5 \mapsto \text{Ebene}_3$) $\leadsto$ **statisch**

- Vorverarbeitung (*preprocessing*)
- Vorübersetzung (*precompilation*)
- Übersetzung: Kompilation, Assemblieren
- Binden (*static linking*)

■ **zur Laufzeit** ($\text{Ebene}_5 \mapsto \text{Ebene}_1$) $\leadsto$ **dynamisch**

- bedarfsorientierte Übersetzung (*just in time compilation*)
- Binden (*dynamic linking*)
- bindendes Laden (*linking loading, dynamic loading*)
- Teilinterpretation
- Interpretation

Betriebssysteme entvirtualisieren zur Laufzeit

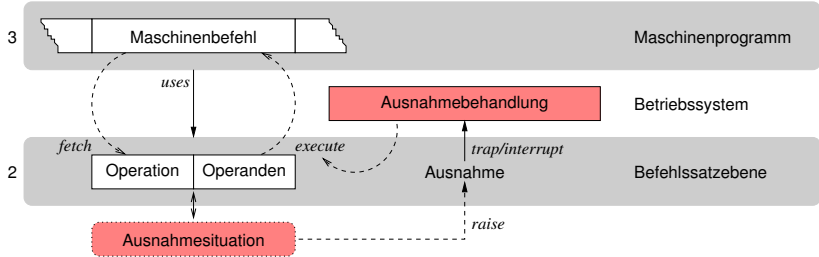
→ dynamisches Binden, bindendes Laden, Teilinterpretation

Mehrebenenmaschinen

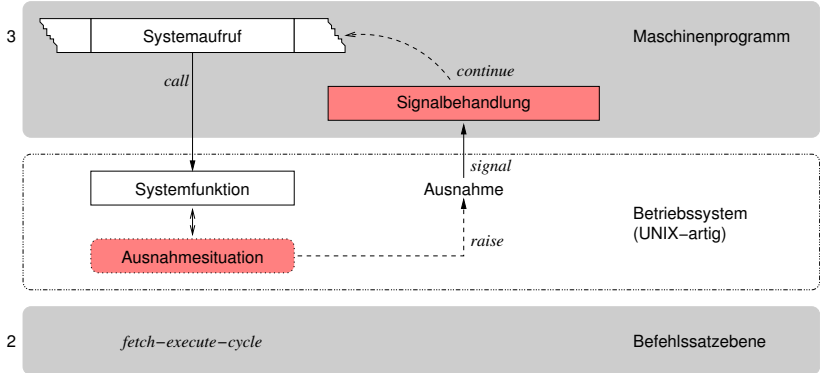
Ausnahmesituation

Abweichung vom normalen Programmablauf

- **Ausnahme** (*exception*), Sonderfall, der die **Unterbrechung** oder den **Abbruch** der Ausführung des Maschinenprogramms bedeutet
 - Feststellung einer **Ausnahmesituation** beim Abruf-/Ausführungszyklus
 - ungültiger Maschinenbefehl oder Systemaufruf
 - Schutz-/Zugriffsverletzung, Seitenfehler, Unterbrechungsanforderung
 - zieht die Reaktion in Form einer **Ausnahmebehandlung** nach sich
 - realisiert durch ein spezielles Programm, einem Unterprogramm ähnlich
 - das durch Erheben (*raise*) einer Ausnahme implizit aufgerufen wird
- die Behandlung eines solchen Sonderfalls verläuft je nach Art und Schwere der Ausnahme nach verschiedenen Modellen:
 - Wiederaufnahme**
 - Ausführungsfortsetzung nach erfolgter Behandlung
 - ↪ Seitenfehler, Unterbrechungsanforderung
 - Termination**
 - Ausführungsabbruch, schwerwiegender Fehler
 - ↪ ungültiger Befehl, Schutz-/Zugriffsverletzung
- manche Programmiersprachen (z.B. Java, C++) bieten Konstrukte zum Umgang mit solchen Ausnahmen



- im Abruf- und Ausführungszyklus interpretiert die CPU den nächsten Maschinenbefehl, führt so das Programm weiter aus
 - ein solcher Befehl hat einen Operations- und ggf. Operandenteil
- bei der Interpretation dieses Befehls tritt eine Ausnahmesituation auf, die CPU erhebt (*raise*) eine Ausnahme
 - die Operation wird abgefangen (*trap*) bzw. unterbrochen (*interrupt*)
- die Ausnahmebehandlung erfolgt durch das Betriebssystem, das dazu durch die CPU aktiviert wird
 - ggf. wird die CPU instruiert, die Operation wieder aufzunehmen



- bei Ausführung der Systemfunktion tritt eine Ausnahmesituation auf, das Betriebssystem erhebt (*raise*) eine Ausnahme
 - die auf ein Signal abgebildet und zur Behandlung hoch gereicht wird
- die Signalbehandlung erfolgt im Kontext des Maschinenprogramms, sie setzt am Ende die Ausführung des Maschinenprogramms fort

Einführung

Schichtenstruktur

Semantische Lücke

Fallstudie

Mehrebenenmaschinen

Maschinenhierarchie

Maschinen und Prozessoren

Entvirtualisierung

Ausnahmesituation

Zusammenfassung

- Rechensysteme zeigen eine bestimmte innere **Schichtenstruktur**
 - die **semantische Lücke** zwischen Anwendungsprogramm und Hardware
 - die Kluft zwischen gedanklich Gemeintem und sprachlich Geäußertem
 - jedes Rechensystem ist als **Mehrebenenmaschine** ausgeprägt
 - eine Hierarchie virtueller Maschinen: **Interpretation** und **Übersetzung**
 - **Demarkationslinie** bzw. ein grundlegender Bruch zwischen Ebene_[3,4]
 - Methode der Abbildung, Art der Programmierung, Natur der Sprache
 - Abbildung der Schichten und Zeitpunkte der Abbildungsvorgänge
 - Betriebssysteme entvirtualisieren zur Laufzeit
 - Kunst der kleinen Schritte: semantische Lücke schrittweise schließen
 - **Ausnahmesituationen** bilden Ebenenübergänge „von unten nach “
 - im Sonderfall bei der Programmausführung kooperieren die Maschinen
 - Analogie zwischen Betriebssystem und CPU: abstrakter/realer Prozessor
- ↪ ergänzend dazu zeigt der Anhang weitere **Interpretersysteme**
- **Virtualisierungssystem** realisiert als VMM (*virtual machine monitor*)

Zusammenfassung

Bibliographie

- [1] APPLE COMPUTER, INC.:
Rosetta.
In: *Universal Binary Programming Guidelines.*
Apple Computer, Inc., Jun. 2006 (Appendix A), S. 65–74
- [2] CHAMBERLAIN, S. ; TAYLOR, I. L.:
Using 1d: **The GNU Linker.**
Boston, MA, USA: Free Software Foundation, Inc., 2003
- [3] CONNECTIX CORP.:
Connectix Virtual PC.
Press Release, Apr. 1997
- [4] ELSNER, D. ; FENLASON, J. :
Using as: **The GNU Assembler.**
Boston, MA, USA: Free Software Foundation, Inc., Jan. 1994

- [5] FEY, D. :
Hardwarenahe Programmierung in Assembler.
In: LEHRSTUHL INFORMATIK 3 (Hrsg.): *Grundlagen der Rechnerarchitektur und -organisation.*
FAU Erlangen-Nürnberg, 2015 (Vorlesungsfolien), Kapitel 2
- [6] GOLDBERG, R. P.:
Architectural Principles for Virtual Computer Systems / Harvard University, Electronic Systems Division.
Cambridge, MA, USA, Febr. 1973 (ESD-TR-73-105). –
PhD Thesis
- [7] HABERMANN, A. N. ; FLON, L. ; COOPRIDER, L. W.:
Modularization and Hierarchy in a Family of Operating Systems.
In: *Communications of the ACM* 19 (1976), Mai, Nr. 5, S. 266–272

- [8] RITCHIE, D. M.:
/* You are not expected to understand this. */.
<http://cm.bell-labs.com/cm/cs/who/dmr/odd.html>, 1975
- [9] ROBIN, J. S. ; IRVINE, C. E.:
Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor.
In: *Proceedings of 9th USENIX Security Symposium (SSYM'00)*,
USENIX Association, 2000, S. 1–16
- [10] SCHRÖDER-PREIKSCHAT, W. :
Virtuelle Maschinen.
Sept. 2013. –
Eingeladener Vortrag, INFORMATIK 2013, Workshop „Virtualisierung:
gestern, heute und morgen“, Koblenz

- [11] SMITH, J. E. ; NAIR, R. :
Virtual Machines: Versatile Platforms for Systems and Processes.
Morgan Kaufmann Publishers Inc., 2005. –
656 S. –
ISBN 9781558609105
- [12] TANENBAUM, A. S.:
Multilevel Machines.
In: Structured Computer Organization[13], Kapitel 7, S. 344–386
- [13] TANENBAUM, A. S.:
Structured Computer Organization.
Prentice-Hall, Inc., 1979. –
443 S. –
ISBN 0-130-95990-1
- [14] <http://www.hyperdictionary.com/computing/semantic+gap>

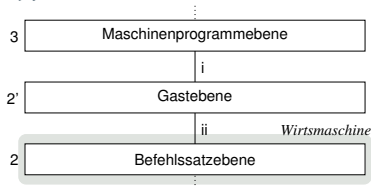
Anhang

Interpretersysteme

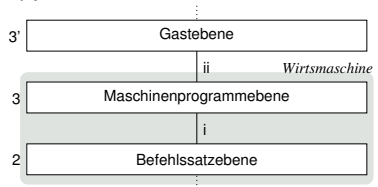
Architektonische Prinzipien virtueller Rechnersysteme

- Schichten der Ebene_[2,3] werden durch reale oder virtuelle Maschinen implementiert, die normalerweise als **Interpreter** fungieren
 - real**
 - beschränkt auf Ebene₂, nämlich die **physische CPU** (z.B. x86)
 - virtuell**
 - für beide jeweils durch ein spezifisches Programm in **Software**
 - im Falle von Ebene₃ das **Betriebssystem** (nur partiell)
 - bezüglich Ebene₂ ein **Virtualisierungssystem** (total/partiell)
- gelegentlich ist aber auch **Binärübersetzung** anzufinden (z.B. [1])
- dabei interpretiert das Virtualisierungssystem alle oder nur einen Teil der Befehle der Programme der virtuellen Maschine
 - total**
 - als **Emulator** der eigenen oder einer fremden realen Maschine [3]
 - „complete software interpreter machine“ (CSIM, [6, S. 21])
 - partiell**
 - als **virtual machine monitor** (VMM, [6, S. 21]), Typ I oder II
 - der nur „sensitive Befehle“ abfängt und (in SW) emuliert
- je nach VMM ist der Übereinstimmungsgrad von virtueller und realer Maschine (Wirt) möglicherweise unterschiedlich [6, S. 17]
 - bei **Selbstvirtualisierung** besteht 100% funktionale Übereinstimmung
 - im Gegensatz zur **Familienvirtualisierung**, bei der die virtuelle Maschine lediglich Mitglied der Rechnerfamilie der Wirtsmaschine ist

■ Typ I VMM



■ Typ II VMM



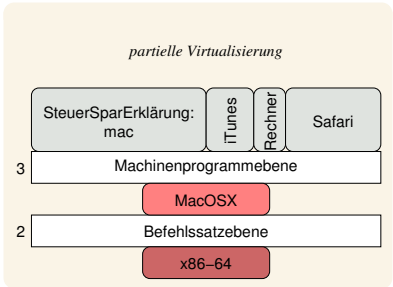
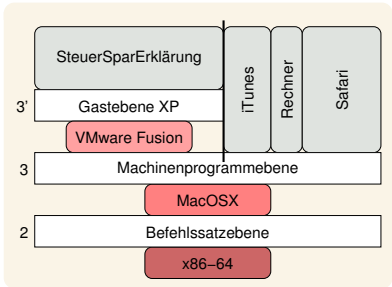
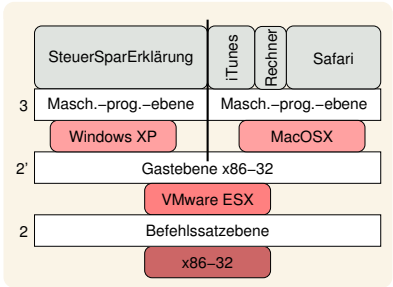
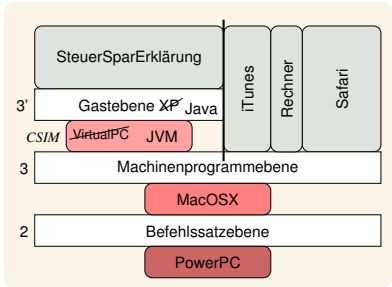
- läuft auf einer „nackten“ Wirtsmaschine
 - unter keinem Betriebssystem
-
- beiden gemeinsames Operationsprinzip ist die **Teilinterpretation**:
 - i durch das Betriebssystem (Typ I) bzw. Wirtsbetriebssystem (Typ II)
 - ii durch den VMM
 - Gegenstand der Teilinterpretation sind **sensitive Befehle**
 - jeder Befehl, dessen direkte Ausführung durch die VM nicht tolerierbar ist
 - privilegierte Befehle ausgeführt im unprivilegierten Modus $\leadsto$ *Trap*
 - aber leider auch unprivilegierte Befehle mit kritischen Seiteneffekten

- typische Anforderungen an die Befehlssatzebene [6, S. 47–53]:
 1. annähernd äquivalente Ausführung der meisten unprivilegierten Befehle im System- und Anwendungsmodus des Rechnersystems
 2. Schutz von Programmen, die im Systemmodus ausgeführt werden
 3. Abfangvorrichtung („Falle“, *trap*) für **sensitive Befehle**:
 - a Änderung/Abfrage des Systemzustands (z.B. Arbeitsmodus des Rechners)
 - b Änderung/Abfrage des Zustands reservierter Register oder Speicherstellen
 - c Referenzierung des (für 2. erforderlichen) Schutzsystems
 - d Ein-/Ausgabe
- **unprivilegierte sensitive Befehle** sind kritisch, **Intel Pentium** [9]:
 - verletzt 3.b** ■ SGDT, SIDT, SLDT; [SMSW;] POPF, PUSHF
 - verletzt 3.c** ■ LAR, LSL, VERR, VERW; POP, PUSH; STR, MOVE
 - CALL, INT *n*, JMP, RET
 - bei Vollvirtualisierung (VMware), ist **partielle Binärübersetzung** eine Lösung, oder eben **Paravirtualisierung** (VM/370, Denali, Xen)
 - in beiden Fällen sind aber Softwareänderungen unvermeidbar, entweder am Maschinenprogramm oder am Betriebssystem

Transparenz für das Betriebssystem

- **Vollvirtualisierung** (Selbstvirtualisierung) ist funktional transparent
 - bis auf Zeitmessung hat das Betriebssystem sonst keine Möglichkeit, in Erfahrung zu bringen, ob es eine virtuelle oder reale Maschine betreibt
 - vorausgesetzt der Abwesenheit (unprivilegierter) sensibler Befehle und damit der Nichterfordernis von Binärübersetzung
- ↪ Betriebssystem und VMM wissen nicht voneinander

- anders verhält es sich mit **Paravirtualisierung** $\leadsto$ intransparent
 - Grundidee dabei ist, dass das Betriebssystem gezielt mit dem VMM in Interaktion tritt und bewusst auf Transparenz verzichtet
 - Hintergrund ist die **Deduplikation** von Funktionen aber auch Daten, die sowohl im Betriebssystem als auch im VMM vorhanden sein müssen
 - Betriebsmittelverwaltung, Gerätetreiber, Prozessorsteuerung, ...
 - weiterer Aspekt ist die damit einhergehende Reduktion von Gemeinkosten (*overhead*) durch Wegfall der Teilinterpretation des Betriebssystems
 - in dem Zusammenhang werden im Betriebssystem ursprünglich enthaltene sensitive Befehle als Elementaroperationen des VMM repräsentiert
- ↪ Betriebssystem und VMM gehen eine Art **Symbiose** ein



partielle Virtualisierung