Systemprogrammierung

Grundlagen von Betriebssystemen

Teil B – V.1 Rechnerorganisation: Virtuelle Maschinen

Wolfgang Schröder-Preikschat

2. Juni 2022



Agenda

Einführung

Schichtenstruktur Semantische Lücke Fallstudie

Mehrebenenmaschinen
Maschinenhierarchie
Maschinen und Prozessoren
Entvirtualisierung
Ausnahmesituation

Zusammenfassung



Gliederung

Einführung

Schichtenstruktur Semantische Lücke Fallstudie

Mehrebenenmaschinen
Maschinenhierarchie
Maschinen und Prozessoren
Entvirtualisierung

Ausnahmesituation

Zusammenfassung



Lehrstoff

- Rechensysteme begreifen 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** für Rechensystemkonstruktionen verstehen
 - 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** innerhalb 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 sein wird



Gliederung

Einführung

Schichtenstruktur Semantische Lücke Fallstudie

Mehrebenenmaschinen

Maschinenhierarchie

Maschinen und Prozessoren

Entvirtualisierung

Ausnahmesituation

Zusammenfassung



Verschiedenheit zwischen Quell- und Zielsprache

Faustregel:
$$\left\{ \begin{array}{ll} \text{Quellsprache} & \rightarrow & \text{h\"{o}heres} \\ \text{Zielsprache} & \rightarrow & \text{niedrigeres} \end{array} \right\} \text{ Abstraktionsniveau}$$

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.

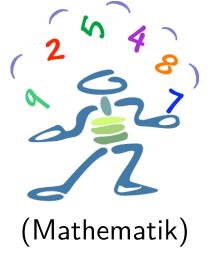
It was in an attempt to try to close this gap that computer architects designed increasingly complex instruction set computers.

Kluft zwischen gedanklich Gemeintem und sprachlich Geäußertem



Beispiel: Matrizenmultiplikation

Problemraum





Lösungsraum



(Informatik)

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



Ebene mathematischer Sprache: Lineare Algebra

Multiplikation von zwei 2×2 Matrizen:

$$\left(\begin{array}{ccc} a_{11} & a_{12} \\ a_{21} & a_{22} \end{array}\right) \cdot \left(\begin{array}{ccc} b_{11} & b_{12} \\ b_{21} & b_{22} \end{array}\right) = \left(\begin{array}{ccc} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{array}\right)$$

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

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



Ebene informatischer Sprache: C

Skalarprodukt oder "inneres Produkt", Quellmodul (multiply.c):

```
typedef int Matrix [N][N];

void multiply(in Matrix a, in Matrix b, out Matrix c) {
    unsigned int i, j, k;
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++) {
        c[i][j] = 0;
        for (k = 0; k < N; k++)
        c[i][j] += a[i][k] * b[k][j];
}

11 }</pre>
```

- **Konkretisierung** für zwei $N \times N$ Matrizen: $c = a \times b$
 - lacktriangle 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]

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

- Kompilation der Quelle in ein semantisch äquivalentes Programm
- Trick: Übersetzung der Quelle vor dem Assemblieren beenden
 - Übersetzung von multiply.c mit -DN=100: C function → ASM/x86



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

```
0000000 457f 464c 0101 0001 0000 0000 0000 0000
                                 0000620 0008 0000 0003 0000 0000 0000 00b0 0000
0000100 8955 57e5 5356 ec83 8b04 1075 45c7 00f0
                                 0000120 0000 8b00 087d db31 7d03 90f0 748d 0026
                                 0000140 458b 310c c7d2 9e04 0000 0000 0c8d 9098
                                 0000160 018b c181 0190 0000 af0f 9704 c283 0101
                                 0001000 0000 0000 0011 0000 0003 0000 0000 0000
                                 0001020 0000 0000 00cf 0000 0045 0000 0000 0000
0000200 9e04 fa83 7564 83e9 01c3 fb83 7564 81d1
0000220 f045 0190 0000 c681 0190 0000 7d81 40f0
                                 0000240 009c 7500 83ae 04c4 5e5b 5d5f 00c3 0000
                                 0001060 0002 0000 0000 0000 0000 0000 027c 0000
                                 0001100 0080 0000 0008 0000 0007 0000 0004 0000
0000260 4700 4343 203a 4428 6265 6169 206e 2e34
0000300 2e33 2d32 2e31 2931 3420 332e 322e 0000
                                 0000320 732e 6d79 6174 0062 732e 7274 6174 0062
                                 0001140 0000 0000 02fc 0000 0015 0000 0000 0000
0000340 732e 7368 7274 6174 0062 742e 7865 0074
                                 0000360 642e 7461 0061 622e 7373 2e00 6f63 6d6d
                                 0000400 6e65 0074 6e2e 746f 2e65 4e47 2d55 7473
                                 0000420 6361 006b 0000 0000 0000 0000 0000 0000
                                 0001240 0000 0000 0000 0000 0003 0001 0000 0000
0001260 0000 0000 0000 0000 0003 0002 0000 0000
0001320 0000 0000 0000 0000 0003 0005 0000 0000
0000520 006d 0000 0000 0000 0000 0000 0010 0000
                                 0001340 0000 0000 0000 0000 0003 0004 000c 0000
0000540 0000 0000 0021 0000 0001 0000 0003 0000
                                 0001360 0000 0000 006d 0000 0012 0001 6d00 6c75
0001400 6974 6c70 2e79 0063 756d 746c 7069 796c
0001420 0000
```

- Assemblieren der kompilierten Quelle und Ausgabeaufbereitung
 - Hexadezimalkode ausführbar jedoch kein ausführbares Programm!
 - 1. as multiply.s: $\overline{\mathsf{ASM}/\mathsf{x86}} \mapsto \mathtt{a.out}/\mathsf{x86}$ (Binde-/Lademodul)
 - 2. od -x a.out \sim hexadezimaler (-x) **Speicherauszug** (dump, od)



Ebene informatischer Sprache: Binärkode

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

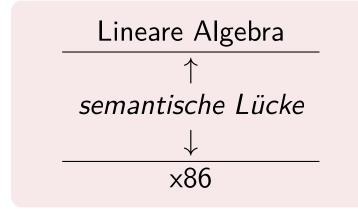
Die für einen Digitalrechner letztendlich benötigte Form — obwohl die CPU wortweise^a auf den Speicher zugreift, mit einer Wortbreite von 8–64 Bits, je nach Prozessortechnologie.

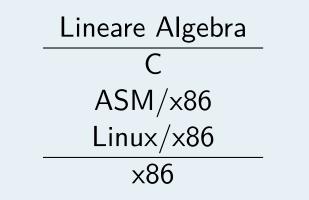
^aGemeint ist das Maschinenwort, keine plattformunabhängige Dateneinheit.



Abstraktionshierarchie von Sprachsystemen

- Modellsprache (Lineare Algebra) \sim 1 Produktsummenformel
- **Assembliersprache** (ASM/ \times 86) \sim 35+n Elementarschritte
- Maschinensprache (Linux/x86) \sim 109 Bytes Programmtext (x86) \sim 872 Bits
- eine einzelne komplexe und überwältigende Aufgabe in mehrere kleine und handhabbare unterteilen







Gliederung

Einführung

Schichtenstruktur
Semantische Lücke
Fallstudie

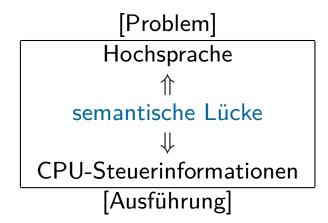
Mehrebenenmaschinen
Maschinenhierarchie
Maschinen und Prozessoren
Entvirtualisierung
Ausnahmesituation

Zusammenfassung



Aufgabenstellung → 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: semantische Lücke schrittweise schließen
 - durch hierachisch angeordnete virtuelle Maschinen Programmlösungen auf die reale Maschine herunterbrechen [12]
 - Prinzip divide et impera ("teile und herrsche")
 - einen "Gegner" in leichter besiegbare "Untergruppen" aufspalten



V.1/15

Hierarchie virtueller Maschinen [13, S. 3]

Interpretation und Übersetzung (Kompilation, Assemblieren):

Ebene <i>n</i>	virtuelle Maschine M_n mit Maschinensprache S_n	Programme in S_n werden von einem auf einer tieferen Maschine laufenden Interpreter gedeutet oder in Programme tieferer Maschinen übersetzt
:	: :	: :
2	virtuelle Maschine M_2 mit	Programme in S_2 werden von einem auf M_1 bzw.
	Maschinensprache S_2	M_0 laufenden Interpreter gedeutet oder nach S_1 bzw. S_0 übersetzt
1	virtuelle Maschine M_1 mit	Programme in S_1 werden von einem auf M_0
	Maschinensprache S_1	laufenden Interpreter gedeutet oder nach S_0
		übersetzt
0	reale Maschine M_0 mit	Programme in S_0 werden direkt von der
	Maschinensprache S_0	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:

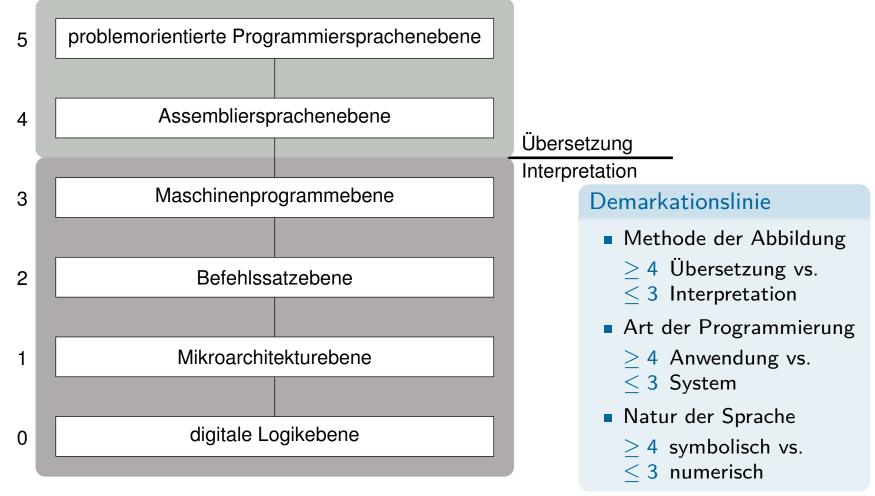
Kom|pi|la|tor lat. (Zusammenträger)

- 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)

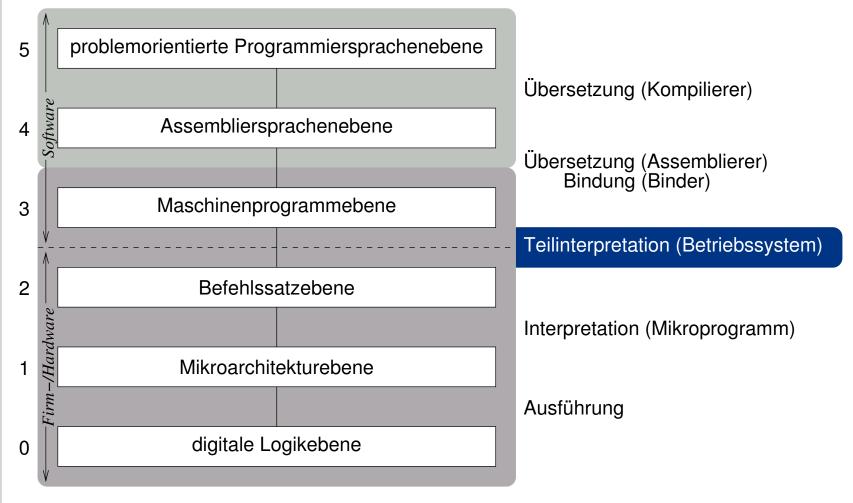
- ein **Hard-, Firm- oder Softwareprozessor**, der die Programme direkt ausführt → ausführbares Programm (*executable*)
 - z.B. Basic, Perl, C, sh(1), x86
- ggf. Vorübersetzung durch einen Kompilierer, um die Programme in eine für die Interpretation günstigere Repräsentation zu bringen
 - z.B. Pascal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





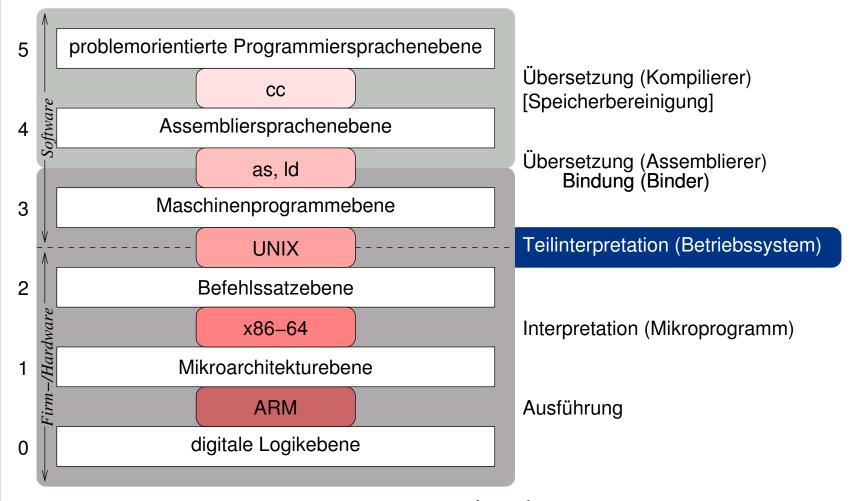
- Schichten der Ebene_[4,5] sind nicht wirklich existent
 - sie werden durch Übersetzung aufgelöst und auf tiefere Ebenen abgebildet
 - so dass am Ende nur ein Maschinenprogramm (Ebene₃) übrigbleibt





- Schichten der Ebene[0,2] liegen normalerweise nicht in Software vor
 - sie können jedoch in Software simuliert, emuliert oder virtualisiert werden
 - dadurch lassen sich Rechensysteme grundsätzlich rekursiv organisieren

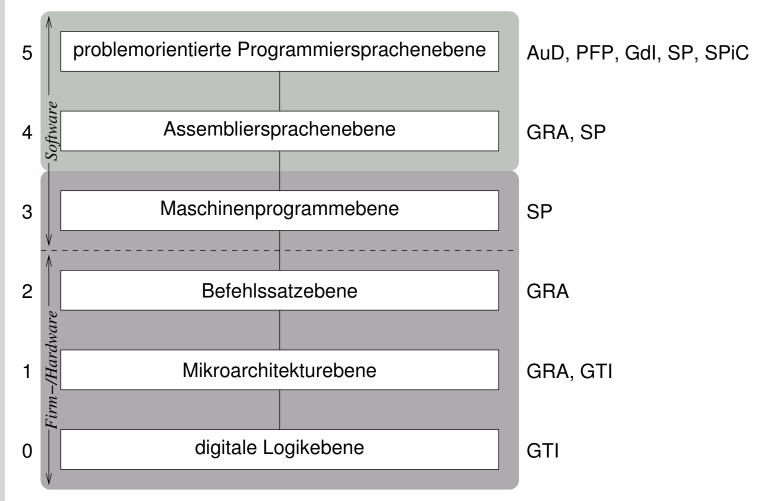




- \blacksquare RISC auf Ebene₁ und gegebenenfalls (hier) CISC auf Ebene₂
 - nach außen "complex", innen aber "reduced instruction set computer"
 - Intel *Core* oder *Haswell* ↔ AMD *Bulldozer* oder *Zen* (ARM)



Schichtenfolge eingebettet im Informatikstudiengang



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



- 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 Maschine¹

Übersetzung

- aller Befehle des Programms, das der Ebene; zugeordnet ist
- in eine semantisch äquivalente Folge von Befehlen der Ebene_j, mit $j \leq i$
- dadurch **Generierung** eines Programms, das der Ebene; zugeordnet ist

Interpretation

- total aller Befehle des Programms, das der Ebene; zugeordnet ist
- partiell nur der Befehle des Programms, die der Ebene; zugeordnet sind
 - wobei das Programm der Ebene_k, $k \geq i$, zugeordnet sein kann
- durch **Ausführung** eines Programms der Ebene_i, mit $j \leq i$



¹vgl. insb. [10]: die Folien sind Teil des ergänzenden Materials zu SP.

Abbildung durch Übersetzung

$\mathsf{Ebene}_5 \mapsto \mathsf{Ebene}_4$ (Kompilation)

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

$Ebene_4 \mapsto Ebene_3$ (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 Maschinenkode (d.h., die Mnemone) auflösen
 - in binären Maschinenkode umwandeln
 - ADD EAX (Mnemon) \mapsto 05₁₆ (Hexadezimalkode) \mapsto 00000101₂ (Binärkode)

3.3 Mehrebenenmaschinen – Entvirtualisierung

hier: Beispiel für den Befehlssatz x86-kompatibler Prozessoren



Abbildung durch Interpretation

- Ebene₃ \mapsto Ebene₂ (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

3.3 Mehrebenenmaschinen – Entvirtualisierung

- Ebene₂ stellt eine Falle (trap), bedient von einem Ebene₂-Programm
- $\mathsf{Ebene}_2 \mapsto \mathsf{Ebene}_1$ (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 (Ebene₅ \mapsto Ebene₃) \rightsquigarrow statisch
 - Vorverarbeitung (preprocessing)
 - Vorübersetzung (precompilation)
 - Übersetzung: Kompilation, Assemblieren
 - Binden (static linking)
- zur Laufzeit (Ebene₅ \mapsto Ebene₁) \rightsquigarrow 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

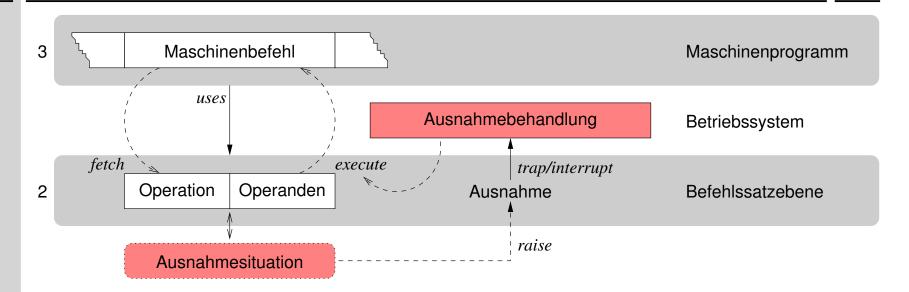


Abweichung vom normalen Programmablauf

- 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

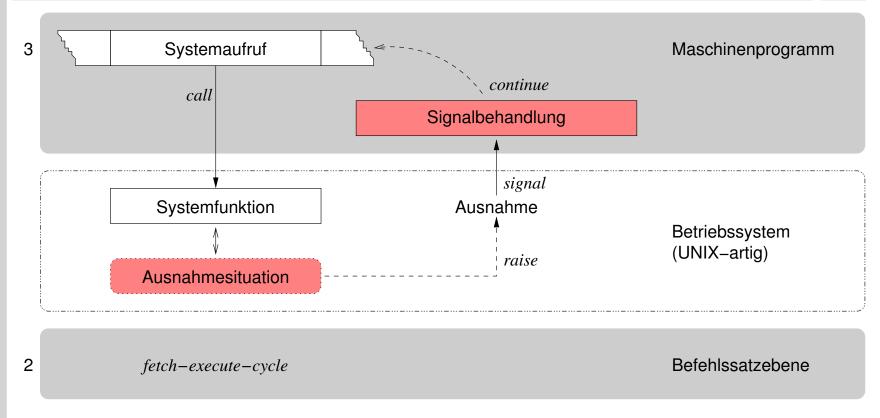
 Ausführungsfortsetzung nach erfolgter Behandlung
 - → Seitenfehler, Unterbrechungsanforderung
 - Termination Ausführungsabbruch, schwerwiegender Fehler
 - → ungültiger Befehl, Schutz-/Zugriffsverletzung
- manche Programmiersprachen (z.B. Java, C++) bieten Konstrukte zum Umgang mit solchen Ausnahmen





- im Abruf- und Ausführungszyklus interpretiert die CPU den nächsten Maschinenbefehl, führt so das Maschinenprogramm 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



Gliederung

Einführung

Schichtenstruktur Semantische Lücke Fallstudie

Mehrebenenmaschinen

Maschinenhierarchie

Maschinen und Prozessoren

Entvirtualisierung

Ausnahmesituation

Zusammenfassung



Resümee

... virtuelle Maschinen existieren vor oder zur Programmlaufzeit

- 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)



Literaturverzeichnis I

[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



Literaturverzeichnis II

[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



Literaturverzeichnis III

[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

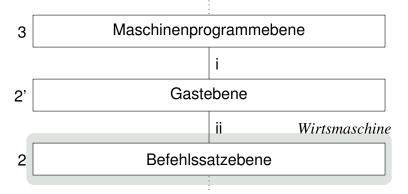


Architektonische Prinzipien virtueller Rechnersysteme

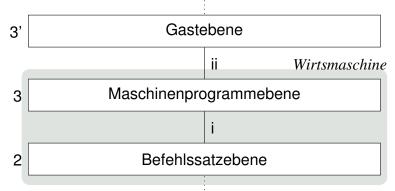
- Schichten der Ebene_[2,3] werden durch reale oder virtuelle Maschinen implementiert, die normalerweise als Interpreter fungieren
 - real beschränkt auf Ebene₂, nämlich die **physische CPU** (z.B. x86)
 - virtuell für beide jeweils durch ein spezifisches Programm in **Software**
 - im Falle von Ebene₃ 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 Software) 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
- läuft auf einer erweiterten Wirtsmaschine
- unter dem Wirtsbetriebssystem
- 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 \sim *Trap*
 - aber leider auch unprivilegierte Befehle mit kritischen Seiteneffekten



Virtualisierbare Reale Maschine

- 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) sensitiver Befehle und damit der Nichterfordernis von Binärübersetzung
 - → Betriebssystem und VMM wissen nicht voneinander
- anders verhält es sich mit **Paravirtualisierung** → 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



Anwendungsbeispiele

$JVM \equiv VirtualPC \equiv CSIM$

