Systemprogrammierung

Rechnerorganisation: Maschinenprogramme

Wolfgang Schröder-Preikschat

Lehrstuhl Informatik 4

11.–17. November 2010
— Selbststudium —

Gliederung

Konventionelle hybride Schicht in einem Rechensystem

Maschinenprogramme enthalten zwei Sorten von Elementaroperationen:

1. **Maschinenbefehle** der Befehlssatzebene (ISA)
   - normalerweise direkt interpretiert durch die Zentraleinheit (Ebene $_2$)
   - ausnahmsweise partiell interpretiert vom Betriebssystem (Ebene $_3$)
2. **Systemaufrufe** an das Betriebssytem
   - das normalerweise nur noch Elementaroperationen der ISA enthält

Hybrid: „etwas Gebündeltes, Gekreuztes oder Gemischtes“ [2]

- ein System, in dem zwei Techniken miteinander kombiniert werden:
  1. Interpretation von Programmen der Befehlssatzebene
  2. partielle Interpretation von Maschinenprogrammen
- ein Maschinenprogramm ist Hybridsoftware, die auf Ebene $_{2,3}$ läuft

Betriebssystem ≡ Programm der Befehlssatzebene

- ein Betriebssystem implementiert die Maschinenprogrammebene
  - es zählt damit selbst nicht zur Klasse der Maschinenprogramme
  - es setzt normalerweise keine Systemaufrufe (an sich selbst) ab
  - es interpretiert eigentümliche Programme nur eingeschränkt partiell

Teilinterpretation von Betriebssystemprogrammen

- bewirkt indirekt rekursive Programmausführungen im Betriebssystem
- erfordert die Fähigkeit zum **Wiedereintritt** (engl. *re-entrance*)[a]
- ab einer bestimmten Ebene im Betriebssystem ist dies unzulässig

[a] Beachte: Teilinterpretation wird durch eine Programmunterbrechung ausgelöst.

- gleichwohl sollten Betriebssysteme es zulassen, in der Ausführung eigentümlicher Programme unterbrochen werden zu können
  - nicht durch Systemaufrufe aber durch *Traps* oder *Interrupts*...

Gliederung

Maschinensprache(n)

Maschinenprogramme setzen sich aus Anweisungen zusammen, die ohne Übersetzung von einem Prozessor ausführbar sind

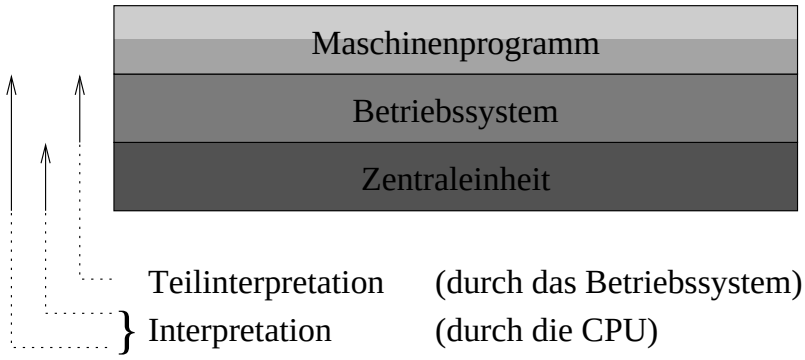
- gleichwohl werden sie (normalerweise) durch Übersetzung generiert
  - nahezu ausschließlich automatisch: Kompilierer, Assemblierer, Binder
  - in seltenen Fällen manuell: natürlicher Kode (engl. *native code*)[1]
- sie repräsentieren sich technisch als **Lademodul** (engl. *load module*)
  - erzeugt durch Dienstprogramme (engl. *utilities*): `gcc(1)`, `as(2)`, `ld(1)`
  - besorgt, verarbeitet und entsorgt durch Betriebssysteme

Grundlage für die Entwicklung von Maschinenprogrammen bilden Hoch- und (vereinzelt) Assembliersprachen — und zwar auf:

1. Anwendungsprogrammebene
2. Laufzeitsystemebene
3. Betriebssystemeebene

[1] Binärkode des realen Prozessors, auch: Maschinenkode.

„Triumvirat“ zur Ausführung von Anwendungsprogrammen



Anwendungsprogrammebene
Laufzeitsystemebene

- Maschinenprogramm = Anwendungsprogramm + Laufzeitsystem
  - beide Ebenen teilen sich denselben Programmadressraum
  - normale Unterprogrammaufrufe aktivieren das Laufzeitsystem
- Ausführungsplattform = Betriebssystem + Zentraleinheit (CPU)
  - zwischen den Ebenen erstreckt sich die Hard-/Softwaregrenze
  - Systemaufrufe (engl. *system calls*) aktivieren das Betriebssystem

Anwendungsprogrammebene: C

Maschinenprogramm realisiert mit Ebene $_5$-Konzepten:

echo.c

```
void echo () {
    char c;
    while (write(1, &c, read(0, &c, 1)) != -1) {}
}
```

Funktion `read(2)` überträgt ein Zeichen von Standardeingabe (`0`) an die Arbeitsspeicheradresse `&c`, deren Inhalt anschließend mit der Funktion `write(2)` zur Standardausgabe (`1`) gesendet wird. Die Schleife terminiert durch Unterbrechung, unter UNIX z.B. nach Eingabe von `^C`.

Anwendungsprogrammebene: ASM

Maschinenprogramm realisiert mit Ebene $_4$-Konzepten:

- `gcc -O6 -fomit-frame-pointer -S echo.c`

`echo () {...`

```
echo:
  pushl %ebx
  subl  $40,%esp
  leal  39(%esp),%ebx
  .p2align 4,,7
  .p2align 3
```

`while (...){}`

```
.L2:
  movl  $1,8(%esp)
  movl  %ebx,4(%esp)
  movl  $0,(%esp)
  call  read
  movl  %ebx,4(%esp)
  movl  $1,(%esp)
  movl  %eax,8(%esp)
  call  write
  addl  $1,%eax
  jne   .L2
```

`...}`

```
  addl  $40,%esp
  popl  %ebx
  ret
```

unaufgelöste Referenzen der Systemfunktionen `read(2)` und `write(2)`

- werden vom Binder `ld(1)` aufgelöst $\mapsto$ `libc.a`

Laufzeitsystemebene: ASM

Maschinenprogramm realisiert mit Ebene $_4$-Konzepten:

1. `gcc -O6 -fomit-frame-pointer -static echo.c`
2. Verwendung der `disassemble`-Operation von `gdb(1)`

```
read:
  push %ebx
  movl 16(%esp),%edx
  movl 12(%esp),%ecx
  movl 8(%esp),%ebx
  mov  $3,%eax
  int  $0x80
  pop  %ebx
  cmp  $-4095,%eax
  jae  __syscall_error
  ret
```

```
__syscall_error:
  neg %eax
  mov %eax,errno
  mov $-1,%eax
  ret

  .comm errno,16
```

```
write:
  push %ebx
  movl 16(%esp),%edx
  movl 12(%esp),%ecx
  movl 8(%esp),%ebx
  mov  $4,%eax
  int  $0x80
  pop  %ebx
  cmp  $-4095,%eax
  jae  __syscall_error
  ret
```

Systemaufrauf durch **synchrone Programmunterbrechung** `int $0x80`

- explizit die Teilinterpretation des Maschinenprogramms anfordern

Betriebssystemebene: ASM

Programm der Befehlssatzebene realisiert mit Ebene$_4$-Konzepten:

- `kernel-source-2.4.20/arch/i386/kernel/entry.S` (Auszug)

Sichern

```
system_call:
  pushl %eax
  cld
  pushl %es
  pushl %ds
  pushl %eax
  pushl %ebp
  pushl %edi
  pushl %esi
  pushl %edx
  pushl %ecx
  pushl %ebx
  ...
```

Interpretieren

```
  ...
  cmpl  $(NR_syscalls),%eax
  jae   badsys
  call  *sys_call_table(,%eax,4)
  movl  %eax,24(%esp)
ret_from_sys_call:
  ...
badsys:
  movl  $-ENOSYS,24(%esp)
  jmp   ret_from_sys_call
```

Wiederherstellen

```
  ...
  popl  %ebx
  popl  %ecx
  popl  %edx
  popl  %esi
  popl  %edi
  popl  %ebp
  popl  %eax
  popl  %ds
  popl  %es
  addl  $4,%esp
  iret
```

Systemaufraufzuteiler (engl. *system call disptachter*): ASM ein Muss...

Betriebssystemebene: Systemaufrufe verarbeiten

Betriebssysteme realisieren einen **Befehlsabruf- und -ausführungszyklus** (engl. *fetch-execute cycle*)[2] zur Ausführung von Systemaufrufen

1. Prozessorstatus des unterbrochenen Programms sichern
   - Aufforderung der CPU zur Teilinterpretation nachkommen
2. Systemaufruf interpretieren (Maschinenprogramm teilinterpretieren):
   (i) Systemaufrufnummer (Operationskode) abrufen
   (ii) auf Gültigkeit überprüfen und ggf. Fehlerbehandlung auslösen
   (iii) bei gültigem Operationskode, zugeordnete Systemfunktion ausführen
3. Prozessorstatus wiederherstellen und zurückspringen
   - Beendigung der Teilinterpretation der CPU „mitteilen“

Notwendigkeit zur Assembliersprache

- Teilinterpretation erfordert kompletten Zugriff auf den CPU-Status
- dieser ist nicht mehr Teil des Programmiermodells einer Hochsprache

[2] Wiederkehrende gleichartige, ähnliche oder vergleichbare Ereignisse oder Prozesse.

Betriebssystemebene: C

Programm der Befehlssatzebene realisiert mit Ebene $_5$-Konzepten:

- `kernel-source-2.4.20/fs/read_write.c` (Auszug)

```
asmlinkage ssize_t sys_read(unsigned int fd, char *buf, size_t count) {
    ssize_t ret;
    struct file *file;

    ret = -EBADF;
    file = fget(fd);
    if (file) {
        ...
    }
    return ret;
}

asmlinkage ssize_t sys_write ...
```

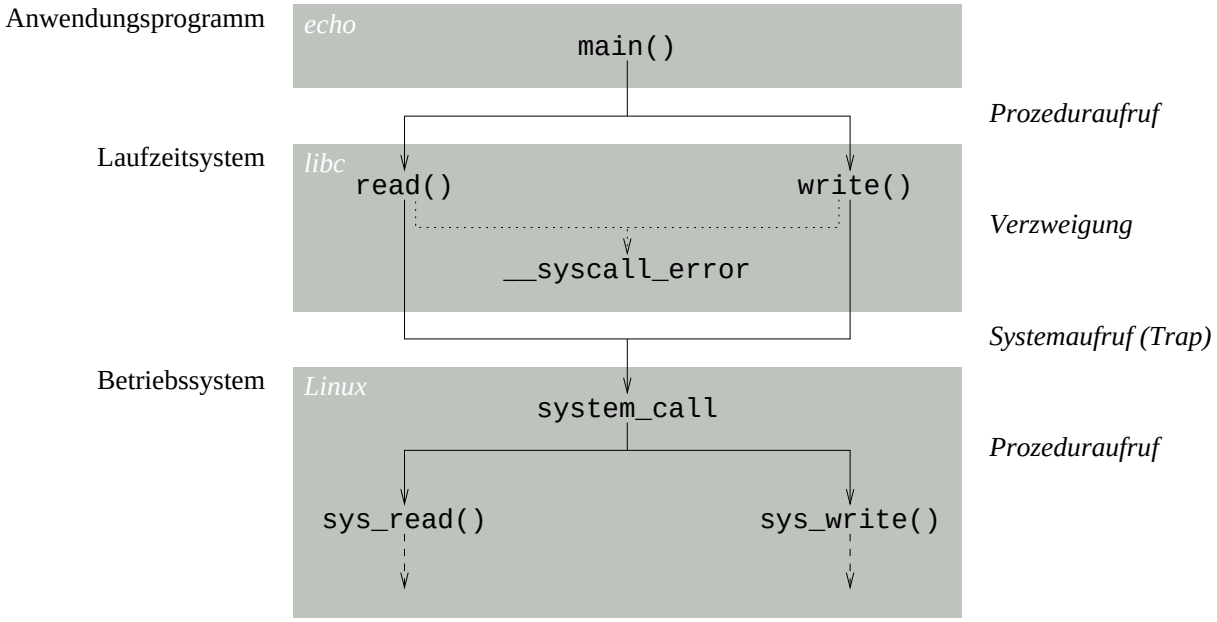
Systemfunktion (Implementierung) innerhalb des Betriebssystems

- aktiviert durch `call *sys_call_table(,%eax,4)` (S. 11)

Gliederung

Zusammenspiel der Ebenen des „echo“-Anwendungsfalls

Aufrufhierarchie



Systemaufrufschnittstelle (engl. *system call interface*)

UNIX Programmers Manual (UPM), Lektion 2 — `man(2)`

```
read:
  push %ebx
  movl 16(%esp),%edx
  movl 12(%esp),%ecx
  movl 8(%esp),%ebx
  mov  $3,%eax
  int  $0x80
  pop  %ebx
  cmp  $-4095,%eax
  jae  __syscall_error
  ret
```

Aufrufstümpfe verbergen die technische Auslegung der Interaktion zwischen Anwendungsprogramm und BS

- „nach außen“ erscheint ein Systemaufruf als normaler Prozeduraufruf
- „nach innen“ setzt ein Systemaufruf eine (synchrone) Programmunterbrechung ab

Systemaufrufe sind spezielle „**Prozedurfernaufrufe**“, die ggf. bestehende Schutzdomänen in kontrollierter Weise überwinden müssen

- getrennte Adressräume für Anwendungsprogramm und BS
- Ein-/Ausgabeparameter in Registern übergeben, „Trap“ auslösen

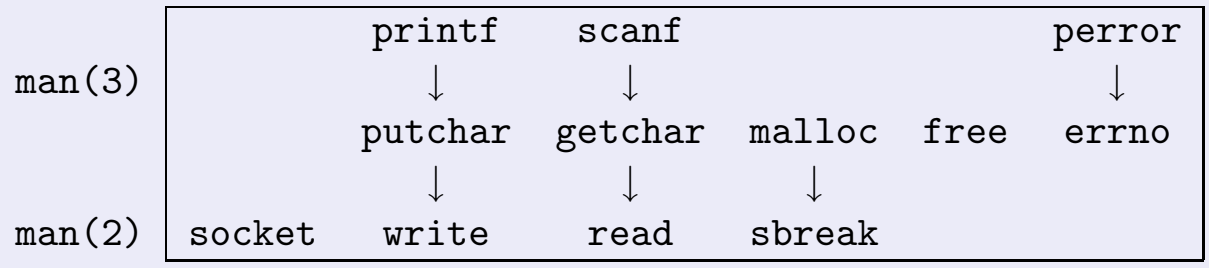
Laufzeitumgebung (engl. *runtime environment*)

UNIX Programmers Manual (UPM), Lektion 3 — `man(3)`

Programmbausteine in Form eines zur Laufzeit zur Verfügung gestellten universellen Satzes von Funktionen und Variablen

- Lesen/Schreiben von Dateien, Ein-/Ausgabegeräte steuern
- Daten über Netzwerke transportieren oder verwalten
- formatierte Ein-/Ausgabe, ...

Laufzeitbibliothek von C unter UNIX (Auszug)



Ensemble problemspezifischer Prozeduren

Anwendungsroutinen (des Rechners)

- bei C/C++ die Funktion `main()` und anderes Selbstgebautes
- setzen u.a. Betriebssystem- oder Laufzeitsystemaufrufe ab

Laufzeitsystemfunktionen (des Kompilierers/Betriebssystems)

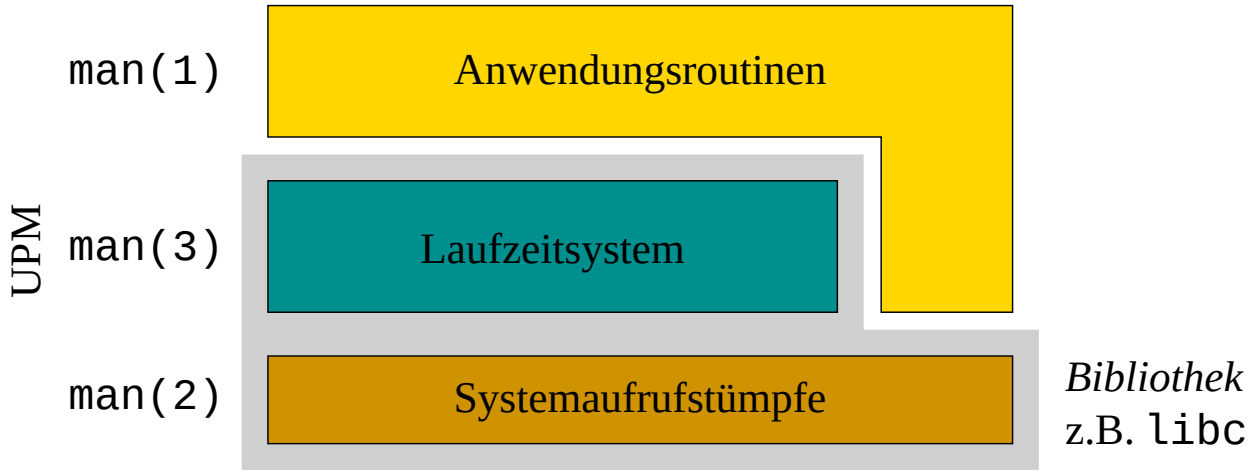
- bei C z.B. die Bibliotheksfunktionen `printf(3)` und `malloc(3)`
- setzt Betriebssystem- oder (andere) Laufzeitsystemaufrufe ab

Systemaufrufstümpfe (des Betriebssystems)

- bei UNIX z.B. die Bibliotheksfunktionen `read(2)` und `write(2)`
- setzen Aufrufe an das Betriebssystem ab
  - Systemaufruf $\mapsto$ synchrone Programmunterbrechung $\sim$ *Trap*

- bilden zusammengebunden das **Maschinenprogramm** (Lademodul)

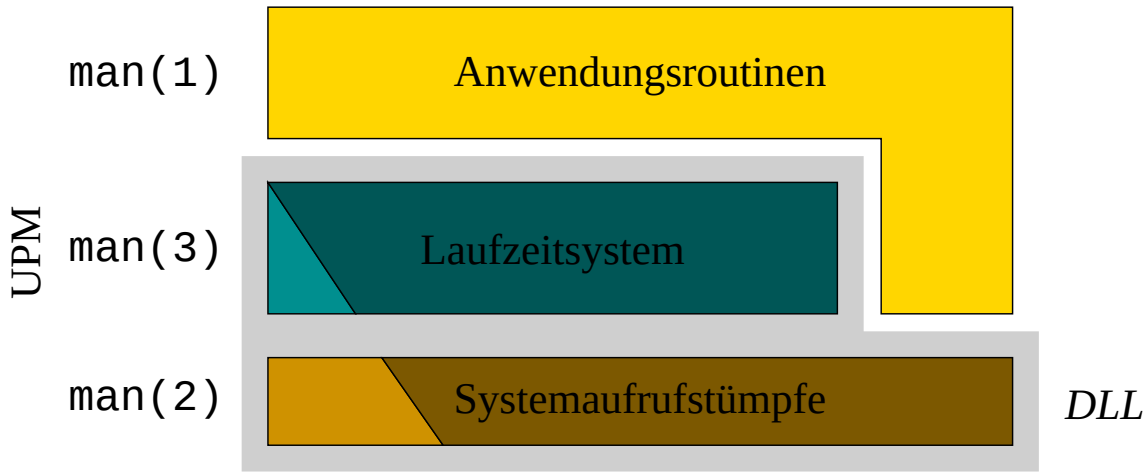
Grobstruktur von Maschinenprogrammen



Statisch gebundenes Programm

- zum Ladezeitpunkt des Programms sind alle Referenzen aufgelöst
  - Kompilierer und Assemblierer lösen lokale (interne) Referenzen auf
  - der Binder löst globale (`extern`, `.globl`) Referenzen auf
- Schalter `-static` bei `gcc(1)` oder `ld(1)`

Grobstruktur von Maschinenprogrammen (Forts.)



Dynamisch gebundenes Programm

- Bibliotheksfunktionen erst bei Bedarf (vom Betriebssystem) einbinden
  - zur Laufzeit, bei erstmaligem Aufruf („*trap on use*“, Multics [1])
  - **bindender Lader** (engl. *linking loader*) im Betriebssystem
- dynamische Bibliothek (*shared library*, *dynamic link library* (DLL))

Gliederung

Resümee

- Maschinenprogramme umfassen zwei Arten Elementaroperationen
  - (1) Maschinenbefehle der Befehlssatzebene und (2) Systemaufrufe
  - Befehle, die ohne Übersetzung von einem Prozessor ausführbar sind
- typisches Programm der Befehlssatzebene ist ein Betriebssystem
  - das einen abstrakten Prozessor (Ebene $_3$) implementiert
  - das einen Befehlsabruf- und -ausführungszyklus realisiert
- Teilinterpretation ist nicht strikt auf Maschinenprogramme bezogen
  - ein Betriebssystem kann eigentümliche Programme teilinterpretieren
    - wenn diese etwa durch *Traps* oder *Interrupts* unterbrochen wurden
  - dazu muss das Betriebssystem allgemein zum Wiedereintritt fähig sein
- Maschinenprogramme als Ensemble problemspezifischer Routinen
  - Anwendungsroutinen, Laufzeitsystemfunktionen, Systemaufrufstümpfe
  - statisch/dynamisch zusammengebunden als Lademodul repräsentiert

Literaturverzeichnis

[1] ORGANICK, E. I.:
The Multics System: An Examination of its Structure.
MIT Press, 1972. –
ISBN 0–262–15012–3

[2] *http://de.wikipedia.org/wiki/Hybrid*