Systemprogrammierung

Betriebssystemkonzepte: Arbeitsspeicher

Wolfgang Schröder-Preikschat

Lehrstuhl Informatik 4

20. Juni 2013

Gliederung

Begriff „Arbeitsspeicher“

Technische Informatik

- zumeist bezeichnet als **flüchtiger Speicher** eines Rechensystems
  - Lese-/Schreibspeicher (engl. *random access memory*, RAM)
  - in dem Programme zur Ausführung bereit vorrätig sind
  - der Text- und Datenbereiche von Programmen zwischenlagert
  - dessen Inhalt nach Anschalten des Rechners zunächst undefiniert ist
- in Teilen auch **nichtflüchtiger Speicher** in Ergänzung zum RAM
  - Festwertspeicher (engl. *read-only memory*, ROM)
  - EEPROM (Abk. engl. *electrically erasable programmable ROM*)[a]
- als **Hauptspeicher** unmittelbar von einer CPU ansprechbar
  - über ein oder mehrere Zwischenspeicher (engl. *caches*)

[a] *Flash*-EEPROM eingeschlossen.

Teilinterpretation von Speicherzugriffen: Massenspeicher

- als **virtueller Speicher** mittelbar durch Betriebssysteme ansprechbar

Gliederung

Adressraumkonzepte und virtuelle Maschinen

physikalischer Adressraum (Hardware) . . . . . Ebene 2
- ist durch die jeweils gegebene Hardwarekonfiguration definiert
- nicht jede Adresse ist gültig, zur Programmspeicherung verwendbar

logischer Adressraum (Kompilierer, Binder, Betriebssystem) . . . Ebene 5/4/3
- abstrahiert von Aufbau/Struktur des Hauptspeichers
- alle Adressen sind gültig und zur Programmspeicherung verwendbar

virtueller Adressraum (Betriebssystem) . . . . . Ebene 3
- auf Vorder- und Hintergrundspeicher abgebildeter log. Adressraum
- erlaubt die Ausführung unvollständig im RAM liegender Programme

Fossil: Toshiba Tecra 730CDT, 1996

Adressbereich	Belegung
00000000–0009ffff	RAM
000a0000–000c7fff	System
000c8000–000dffff	keine
000e0000–000fffff	System
00100000–090fffff	RAM
09100000–fffdffff	keine
fffe0000–ffffffff	System

Je nach Hardwarekonfiguration hat der physikalische Adressraum eines Rechners mehr oder weniger viele bzw. große und nicht verwendbare Lücken.

- der physikalische Adressraum verlangt hardwareabhängige Programme

Ausführungsdomäne für Prozesse

Illusion von einem eigenen (nicht zwingend linearen) Adressraum für jedes im Hauptspeicher vollständig vorliegende Programm
- die Anfangsadressen aller logischen Adressräume sind (meist) gleich
  - festgelegt durch eine Systemkonstante (Übersetzer, Binder, Lader)
- die Endadressen sind variabel, jedoch nach oben begrenzt
  - bestimmt durch die Programmlängen bzw. Hardwarefähigkeiten

Adressabbildung (engl. *address mapping*) erfolgt mehrstufig:

$$\begin{aligned} \text{Programm} &\mapsto \text{logischer Adressraum} \\ \text{logischer Adressraum} &\mapsto \text{physikalischer Adressraum} \end{aligned}$$

- logische Adressen sind mehrdeutig, physikalische dagegen eindeutig

Abbildungszeitpunkte

Adress(raum)abbildung kann auf verschiedenen Ebenen erfolgen:

Entwicklungszeit	Programmierer(in)	Ebene 6	
Übersetzungszeit	Kompilierer, Assemblierer	Ebene 5/4	statisch
Bindezeit	Binder	Ebene 4	
Ladezeit	verschiebender Lader	Ebene 3	
Laufzeit	bindender Lader, MMU	Ebene 3/2	dynamisch

Zielkonflikt (engl. *trade-off*): Flexibilität vs. Effizienz
- je später die Abbildung durchgeführt wird, desto. . . [a]
  - höher das Abstraktionsniveau und geringer die Hardwareabhängigkeit
  - höher der Systemaufwand und geringer der Spezialisierungsgrad

[a] Jeweils in Bezug auf das Maschinenprogramm, das in dem abzubildenden logischen Adressraum residiert.

Verantwortlichkeiten bei der Adressraumabbildung

Zusammenspiel von Betriebssystem und Hardware/MMU

Betriebssystem (Ebene 3): **Adressraumabbildung** zur Ladezeit

- der Lader fordert Betriebsmittel zur Programmausführung an
  - Arbeitsspeicher und Adressraumdeskriptoren, je nach Bedarf/MMU
  - einen Prozess
- Verwaltungsinformationen für die MMU werden aufgesetzt
  - die physikalischen Ladeadressen in die Deskriptoren eintragen
  - ggf. spezielle Attribute (z.B. lesen, schreiben, ausführen) zuordnen
- der neue Prozess wird der Einplanung (engl. *scheduling*) zugeführt

Hardware/MMU (Ebene 2): **Adressumsetzung** zur Laufzeit

- Verwendung der in den Deskriptoren gespeicherten Informationen

Beachte

- Verantwortung trägt ganz allein das Betriebssystem
- die MMU setzt das vom Betriebssystem Vorgegebene „gnadenlos“ um

Segmentierung eines logischen Adressraums

Logische Unterteilung zur effektiveren Programmverwaltung

Textsegment (engl. *text segment*)

- Maschinenbefehle (Ebene 2/3) und andere Programmkonstanten
- statische oder dynamische Größe, je nach Betriebssystem
- ggf. gemeinsam ausgelegt für mehrere Prozesse (engl. *shared text*)

Datensegment (engl. *data segment*)

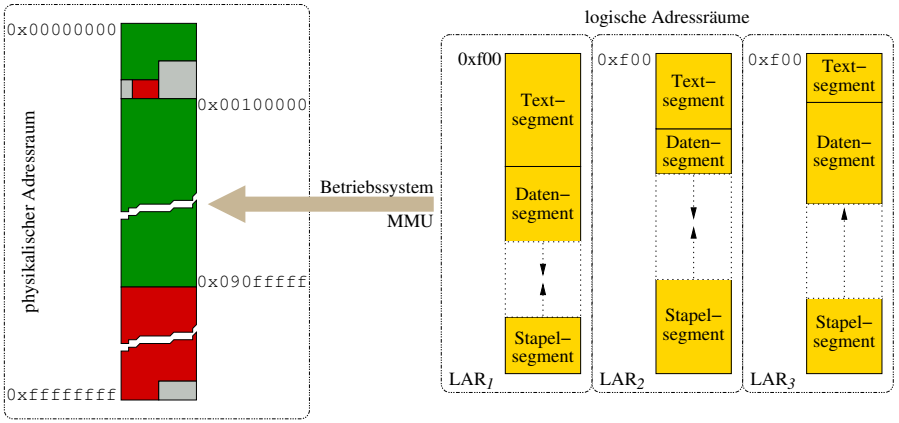
- initialisierte Daten, globale Variablen und ggf. die Halde (engl. *heap*)
- statische oder dynamische Größe, je nach Betriebssystem

Stapelsegment (engl. *stack segment*)

- lokale Variablen, Hilfsvariablen und aktuelle Parameter
- dynamische Größe

Adressraumabbildung auf Ebene 3

Betriebssystem und MMU implementieren logische Adressräume (vgl. S. 19)



Ausrichtung (engl. *alignment*) von Segmenten

- Segmente müssen nicht angrenzend im logischen Adressraum liegen
- ggf. werden sie zur Bindezeit (engl. *link time*) vom Binder ausgerichtet
- Gründe für eine solche Maßnahme:
  - Mitbenutzung (engl. *sharing*) von Segmenten[1] unterstützen
  - differenzierter Schutz (engl. *protection*) bei Mitbenutzung
  - dynamisches Binden oder bindendes Laden von Segmenten
  - Segmente im virtuellen Speicher ablegen und halten

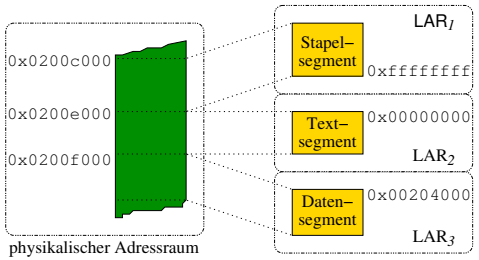
Hardwareabhängigkeit: sei N eine Anzahl von Bytes, größer Null

- die Art der Adress(raum)abbildung der **MMU** bestimmt den Abgleich:
  - nach Seiten
    - N ist die Seitengröße, Zweierpotenz
    - z.B. $N = 2^{12} = 4096$, oder gar $N = 2^{16} = 65\,536$
  - nach Segmenten
    - N ist die Größe eines Segmentbestandteils
    - z.B. $N = 1$, oder gar $N = 16$ bei $x86$ mit $x = 80$
- die Anfangsadresse eines Segments ist dann Vielfaches von N

[1] Durch verschiedene sich in Ausführung befindliche Programme (d.h., Prozesse).

Einrichtung der Segmente erfolgt zur Ladezeit

Betriebssystem platziert Segmente bedarfsorientiert im physikalischen Adressraum



Kontrolliert durch das Betriebssystem ist die Mitbenutzung von Segmenten möglich, indem diese auf einen gemeinsamen Bereich im phys. Adressraum abgebildet werden.

Verletzung der Segmentierung (engl. *segmentation violation*) wird durch die MMU verhindert und bewirkt eine Programmunterbrechung:

$$0 \leq adr_{log} - seg_{log} < size(seg), \text{ sonst Trap}$$

- Konstante seg_{log} definiert den Segmentanfang im log. Adressraum
- der Wert dieser Konstante wurde zuvor passend ausgerichtet (S. 12)

Adressrelokation zur Laufzeit

MMU wandelt jede logische Adresse im Abrufzyklus (engl. *fetch cycle*) der CPU um

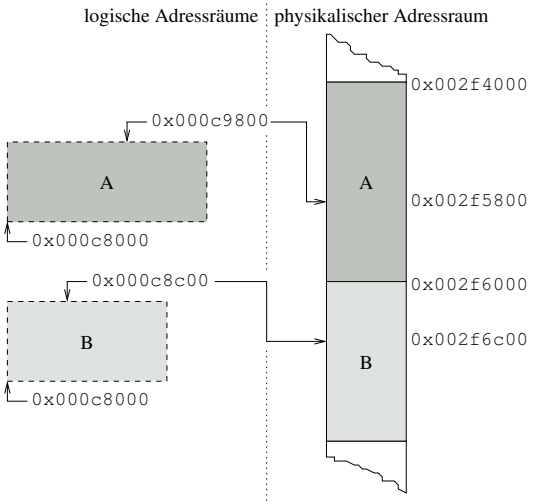
Veränderung einer logischen Adresse um eine Relokationskonstante: (Prinzip)

$$adr_{phy} = adr_{log} - seg_{log} + seg_{phy}$$

- Variable seg_{phy} ist die Ladeadresse im physikalischen Adressraum
  - Ausdruck $adr_{log} - seg_{log}$ relativiert zur log./phys. Segmentbasisadresse
    - er liefert die relative Adresse in Bezug auf den Segmentanfang
  - anschließende Addition „verschiebt“ den relativierten Wert
- die Ladeadresse eines Segments ist gleichfalls Relokationskonstante
  - für alle relativ(iert)en Adressen innerhalb dieses Segments

- Relokation log. Adressen erfolgt nur bei unverletzter Segmentierung

Relokation logischer Adressen



Relokation Segment A

	0x000c9800	adr_{log}
−	0x000c8000	seg_{log}
	0x00001800	*offset*
+	0x002f4000	seg_{phy}
	0x002f5800	adr_{phy}

Relokation Segment B

	0x000c8c00	adr_{log}
−	0x000c8000	seg_{log}
	0x00000c00	*offset*
+	0x002f6000	seg_{phy}
	0x002f6c00	adr_{phy}

- eindeutige Abbildung mehrdeutiger logischer Adressen (Segmente)

Logischer Adressraum als Schutzdomäne

Robustheit von Softwaresystemen verbessern

Adressraumisolation, eine Maßnahme zur Erhöhung von Sicherheit...

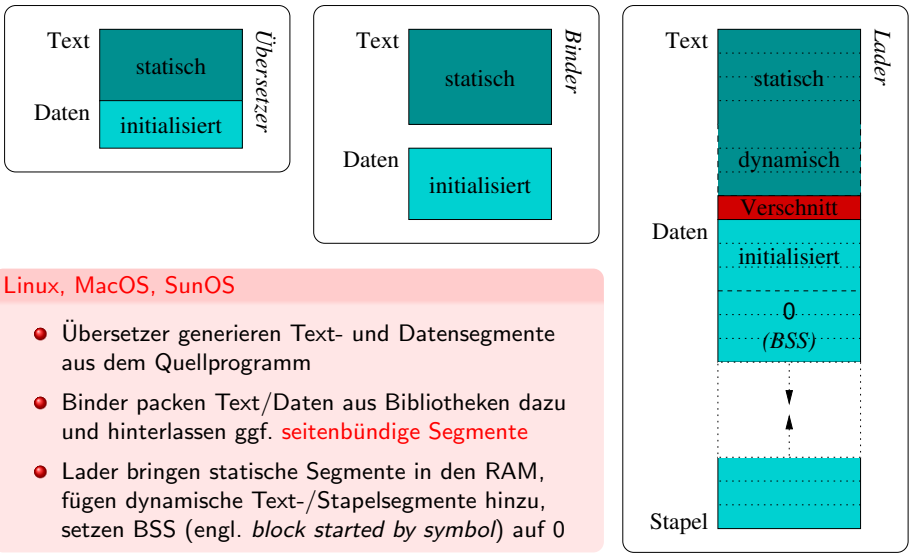
safety Schutz von Menschen und Sachwerten vor dem Versagen technischer Systeme
- Berechnungsfehler oder „Bitkipper“ abfangen
- allgemein (bei BS): Fehlerausbreitung eingrenzen

security Schutz von Informationen und Informationsverarbeitung vor „intelligenten“ Angreifern
- Adressraumausbrüche erschweren/verhindern
- allgemein (bei BS): Eindringlinge fern halten

...in Rechensystemen, die im **Mehrprogrammbetrieb** gefahren werden

UNIX Segmentierung

Dienstprogramm (engl. *utility*) basierter seitennummerierter Ansatz



Linux, MacOS, SunOS

- Übersetzer generieren Text- und Datensegmente aus dem Quellprogramm
- Binder packen Text/Daten aus Bibliotheken dazu und hinterlassen ggf. seitenbündige Segmente
- Lader bringen statische Segmente in den RAM, fügen dynamische Text-/Stapelsegmente hinzu, setzen BSS (engl. *block started by symbol*) auf 0

Virtueller Speicher

Illusion von einem eigenen (nicht zwingend linearen) Adressraum für jedes im Hauptspeicher ggf. unvollständig vorliegende Programm

- Erweiterung bzw. Spezialisierung des logischen Adressraums
- meist verbreitet ist die **Seitenüberlagerung** (engl. *paging*)
- Adressraumzugriffe können E/A (Hintergrundspeicher) implizieren

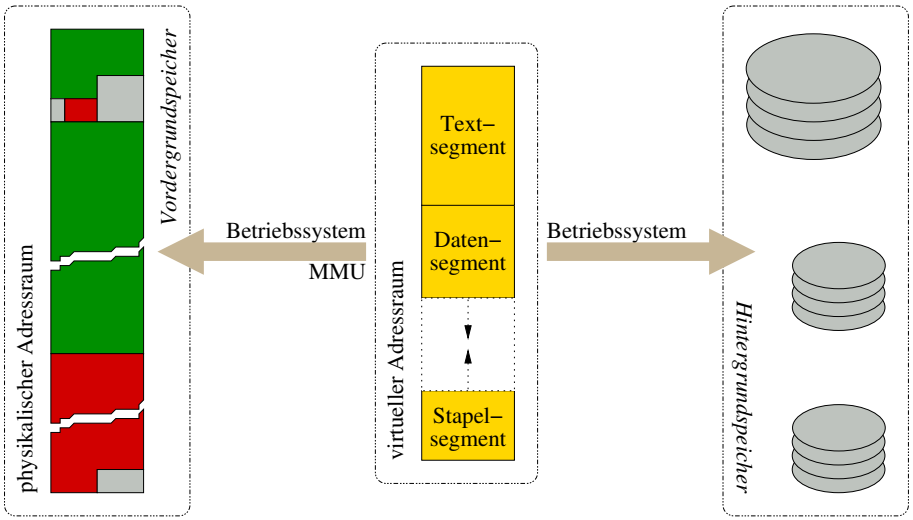
Adressabbildung (engl. *address mapping*) erfolgt mehrstufig:

$$\begin{aligned} \text{Programm} &\mapsto \text{logischer Adressraum} \\ \text{logischer Adressraum} &\mapsto \text{virtueller Adressraum} \\ \text{virtueller Adressraum} &\mapsto \text{physikalischer Adressraum} \end{aligned}$$

- virtuelle Adressen sind ebenso mehrdeutig wie logische Adressen

Adressraumabbildung auf Ebene 3 (Forts.)

Betriebssystem und MMU implementieren virtuelle Adressräume (vgl. S. 11)



Umfang eines virtuellen Adressraums

Adressbreite einer CPU sagt wenig aus über die Hauptspeichergröße eines Rechners

Adressbreite von N Bits...

N	Adressraumgröße (2^N Bytes)	Dimension[2]			
16	65 536	64 kibi	(2^{10})	kilo	(10^3)
20	1 048 576	1 mebi	(2^{20})	mega	(10^6)
32	4 294 967 296	4 gibi	(2^{30})	giga	(10^9)
⋮			⋮		⋮
48	281 474 976 710 656	256 tebi	(2^{40})	tera	(10^{12})
64	18 446 744 073 709 551 616	16 384 pebi	(2^{50})	peta	(10^{15})

Hauptspeicher ⊂ Arbeitsspeicher

Rechner sind im Regelfall nur mit einem Bruchteil des von einer CPU adressierbaren Arbeitsspeichers wirklich bestückt!

[2] Siehe auch Anhang, S. 33.

UNIX Systemfunktionen

Laufzeit- bzw. Betriebssystem

Linux, MacOS, SunOS

```
pa = mmap(addr, len, prot, flags, fd, offset)
ok = munmap(addr, len)
ok = mlock(addr, len)
ok = munlock(addr, len)
ok = mprotect(addr, len, prot)
ok = madvise(addr, len, behav)
ps = getpagesize()
   ⋮
```


Gliederung

Speicherkonzepte und -medium

Kurz-, mittel- und langfristige Informationsspeicherung

Vordergrundspeicher: Hauptspeicher (RAM)

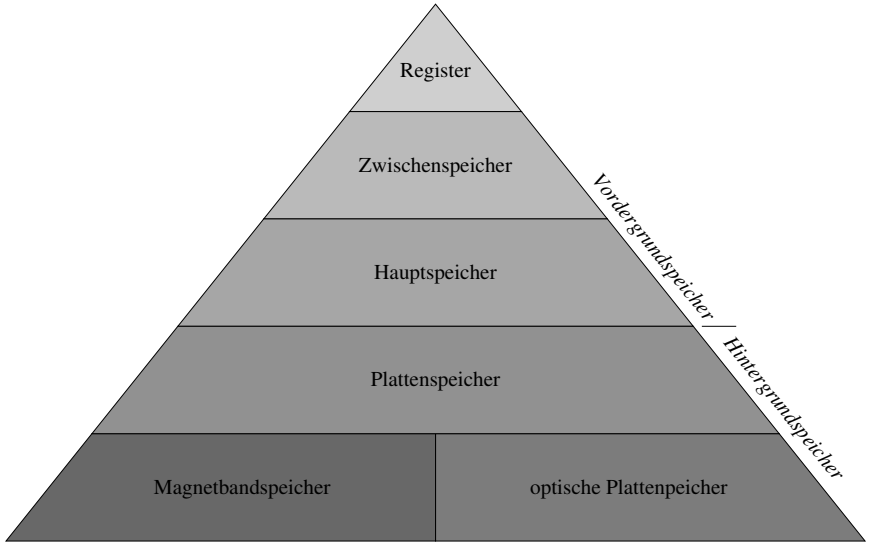
- entsprechend bestückter Bereich im physikalischen Adressraum
- Zentralspeicher zur Programmausführung („von Neumann Rechner“)
- kann phys. Adressraum überschreiten: Speicherbankumschaltung
- kurzfristige Speicherung, Zugriffszeiten im ns-Bereich

Hintergrundspeicher: Massenspeicher (Band, Platte, CD, DVD)

- über Rechnerperipherie (E/A-Geräte) angeschlossene Bereiche
- dient der Datenablage und Implementierung virtueller Adressräume
- ist größer als der phys. Adressraum: Petabytes (2^{50} bzw. 10^{15})
- mittel- bis langfristige Speicherung, Zugriffszeiten im ms-Bereich

- funktional bringt Virtualisierung „Zugriffstransparenz“ (Multics [4])

Speicherhierarchie in Anlehnung an [5]



Arbeitsteilung von Laufzeit- und Betriebssystem

Laufzeitsystem (bzw. Bibliotheksebene) verwaltet den lokal vorrätigen Speicher eines logischen/virtuellen Adressraums (☞ Aufgabe 4)

- Speicherblöcke können von sehr feinkörniger Struktur/Größe sein
  - einzelne Bytes bzw. Verbundobjekte
- Verfahrensweisen orientieren sich (mehr) an Programmiersprachen

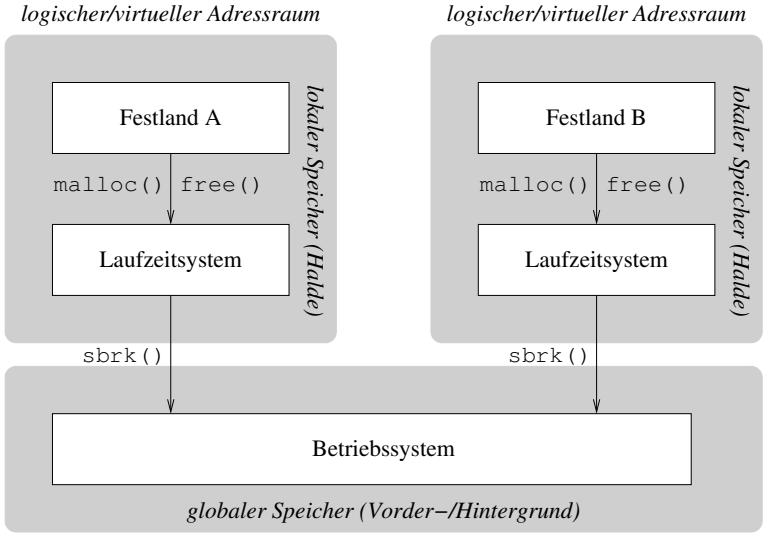
Betriebssystem verwaltet den global vorrätigen Speicher (d.h. den bestückten RAM-Bereich) des physikalischen Adressraums

- Speicherblöcke sind üblicherweise von grobkörniger Struktur/Größe
  - z.B. eine Vielfaches von Seiten
- Verfahrensweisen fokussieren auf Benutzer- bzw. Systemkriterien

Trennung von Belangen (engl. *separation of concerns* [1])

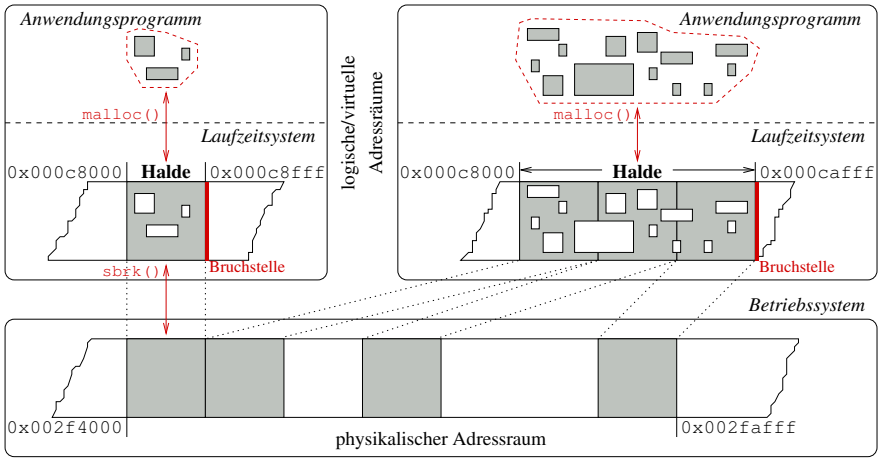
- Aufteilung von Software nach unterschiedlichen Merkmalen, die sich funktional so wenig wie nur möglich überlappen

Zweistufiger Ansatz: Adressraumlokal und -global



Synergie bei der Speicherverwaltung

Adressraumlokal (Halde) im Laufzeitsystem, adressraumglobal durch das Betriebssystem



UNIX Systemfunktionen

Laufzeitsystem — C Bibliothek

Linux, MacOS, SunOS

(☞ Aufgabe 4)

```
ptr = malloc(size)
ptr = valloc(size)
ptr = calloc(count, size)
ptr = realloc(ptr, size)
    ⋮
free(ptr)
```

Freigabe (`free()`) von Speicher hat nur lokale Signifikanz

- keine freiwillige Rückgabe ans Betriebssystem
- die Wiedergewinnung freigegebener Bereiche erfolgt nur bei Beendigung des Programms und/oder auf Basis virtuellen Speichers

UNIX Systemfunktionen

Überbleibsel vergangener Systeme mit nur einem expandierbaren Adressraumsegment

Linux, MacOS, SunOS

```
addr = brk(brkval)
addr = sbrk(incr)
```

Festlegung des Wertes einer neuen „Bruchstelle“ (engl. *break value*) für das Datensegment eines Prozesses

- verändert die diesem Segment zugeordnete Speichermenge
- kann eine vom System vorgegebene Größe nicht überschreiten
- ist die der Endadresse des Datensegments folgende Speicheradresse

- Aufruf erfolgt im Zuge von `*alloc()`, nicht jedoch `free()`

Gliederung

Resümee

- Arbeitsspeicher kann auch Massenspeicher beeinhalten
  - unmittelbar von einer CPU ansprechber: Hauptspeicher
  - mittelbar durch Betriebssysteme ansprechbar: virtueller Speicher
- Adressräume bilden verschiedene Ausführungsdomänen

physikalischer Adressraum	gibt Aufbau/Struktur und Größe des Hauptspeichers vor
logischer Adressraum	abstrahiert von Aufbau/Struktur des Hauptspeichers
virtueller Adressraum	abstrahiert von der Größe des Hauptspeichers

- Speicherverwaltung erfolgt arbeitsteilig auf zwei Ebenen
  - auf der Maschinenprogrammebene durch das Laufzeitsystem
  - auf der Befehlssatzebene durch das Betriebssystem

Literaturverzeichnis

[1] Dijkstra, E. W.:
A Principle of Programming.
Englewood Cliffs, NJ, USA : Prentice-Hall, Inc., 1976

[2] International Electrotechnical Commission (IEC):
Letter Symbols to Be Used in Electrical Technology. Bd. Part 2: Telecommunications and Electronics.
2000

[3] Moore, D. P.:
FORTRAN ASSEMBLY PROGRAM (FAP) for the IBM 709/7090 / IBM Corporation.
New York, NY, USA, 1961 (Form J28-6098-1). –
IBM 709/7090 Data Processing System Bulletin

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

[5] Tanenbaum, A. S.:
Structured Computer Organization.
Prentice-Hall, Inc., 1979. –
443 S. –
ISBN 0–130–95990–1

[6] *Unix — Frequently Asked Questions.*
`http://www.cs.uu.nl/wais/html/na-dir/unix-faq`,

Metrisches System und Informationstechnik

Internationales Einheitensystem (frz. *Système International d'Unités*, SI)

Begriffe des metrischen Systems wurden bedenkenlos übernommen

- noch schlimmer: sie werden inkonsistent verwendet

Medium	Einheit	
	dual	*dezimal*
RAM, ROM, CD	×	
Flash, HD, DVD		×
Floppy[3]	×	×

Abweichungen

kibi ↔ kilo	2,4%
mebi ↔ mega	∼ 4,8%
gibi ↔ giga	∼ 7,3%
tebi ↔ tera	∼ 9,9%
pebi ↔ peta	∼ 12,6%

Standardisierung [2]

- erfolgte erst sehr spät (Ende der 90er Jahre) — zu spät. . .

[3] Zugriff auf das Medium erfolgt sektorweise. Die Größe eines Sektors wird als Zweierpotenz angegeben, die Anzahl der Sektoren kommt als Zehnerpotenz.

Gedankenexperiment zur Adressraumgröße

Aufgabe eines Prozesses soll es sein, seinen Adressraum byteweise zu löschen

```
main () {
    char* p = 0;
    do *p++ = 0;
    while (p);
}
```

Nebenbei gefragt. . .

- geht das so überhaupt?
- ist Selbstauslöschung denn möglich?

PowerPC G5

- Jeder Befehl ist vier Bytes lang.
- Die Löschschleife (L2) umfasst drei Befehle, die von der CPU aus dem Speicher zu lesen sind.
- Der Löschbefehl (`stb r0,0(r2)`) schreibt den Bytewert 0 in die nächste Speicherzelle.
- Jeder Schleifendurchlauf greift somit auf $3 \times 4 + 1 = 13$ Bytes zu.

```
  .machine ppc7400
  .text
  .align 2
  .p2align 4,,15
  .globl _main
_main:
  li r9,0
  li r2,0
  li r0,0
  mtctr r9
  .p2align 4,,15
L2:
  stb r0,0(r2)
  addi r2,r2,1
  bdnz L2
  blr
```

- Bei 1 ns Zugriffszeit dauert das Löschen eines Bytes 13 ns — Fiktion

Dauer des Experiments bei virtuellem Speicher — Fiktion

- Die zur Zeit nicht benötigten Bereiche eines virtuellen Adressraums liegen im Hintergrundspeicher.
- Diese werden bei Bedarf „seitenweise“ in den Vordergrundspeicher eingelagert.
- Angenommen, jede Seite ist 4 KiB groß und die mittlere Zugriffszeit des Hintergrundspeichers (Platte), um eine Seite einzulagern, liegt bei 5 ms.
- Damit kostet ein Bytezugriff durchschnittlich 1,2 μs!

Größe	Laufzeit	
2^{16}	851.968 Mikrosekunden	
2^{20}	13.631 Millisekunden	
2^{32}	55.835 Sekunden	(bereits mehr als 1,5 Stunden!)
⋮	⋮	← heute
2^{48}	42.352 Tage	
2^{64}	7.604.251 Jahre	(ohne Schaltjahre)

Schwergewichtiger Prozess unter SunOS

Prozessinkarnation und Adressraum bilden eine Einheit

```
int foo;
int hal = 42;

int main () {
    for (;;)
        printf("Die Antwort auf alle Fragen lautet %d\n",
            hal + foo);
}
```

Wie ist der Adressraum bzw. Speicher des Prozesses organisiert, der die Ausführung dieses Programms bewirkt?

Organisation des (virtuellen) Adressraums

```
wosch@faui40 40$ gcc -O6 -static -o hal hal.c; ./hal
Die Antwort auf alle Fragen lautet 42 ...^Z
wosch@faui40 41$ ps
   PID TTY      TIME CMD
 28426 pts/4    0:00 hal
   205 pts/4    0:00 ps
 25965 pts/4    0:00 tcsh-6.0
wosch@faui40 42$ pmap -x 28426
28426:  ./hal
 Address  Kbytes     RSS    Anon  Locked Mode   Mapped File
00010000     216     216       -       - r-x--  hal
00054000      16      16       8       - rwx--  hal
00058000       8       8       8       - rwx--    [ heap ]
FFBFE000       8       8       8       - rw---    [ stack ]
-------- ------- ------- ------- -------
total Kb     248     248      24       -
wosch@faui40 43$
```


Pseudobefehle stecken Text-/Datenbereiche ab

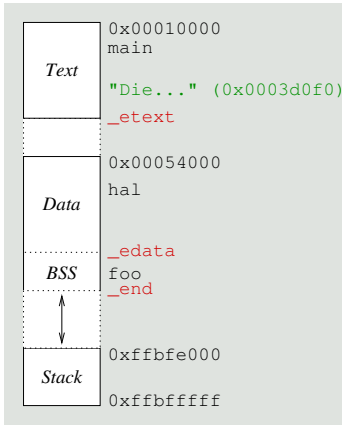
```
    .file   "hal.c"
    .global hal
    .section ".data"
    .align  4
    .type   hal,#object
    .size   hal,4
hal:
    .uaword 42
    .common foo,4,4
    .section ".rodata"
    .align  8
.LLC0:
    .asciz  "Die Antwort auf alle Fragen lautet %d\n"
    .section ".text"
    .align  4
    .global main
    .type   main,#function
    .proc   04
```

```
main:
    !#PROLOGUE# 0
    save    %sp, -112, %sp
    !#PROLOGUE# 1
    sethi   %hi(hal), %l2
    sethi   %hi(foo), %l1
    sethi   %hi(.LLC0), %l0
    ld      [%l2+%lo(hal)], %g1
.LL5:
    or      %l0, %lo(.LLC0), %o0
    ld      [%l1+%lo(foo)], %o3
    call    printf, 0
    add     %g1, %o3, %o1
    b       .LL5
    ld      [%l2+%lo(hal)], %g1
.LLfe1:
    .size   main,.LLfe1-main
    .ident  "GCC: (GNU) 3.0.4"
```


Segmente: Text, Daten, BSS, Stapel

```
wosch@faui40 43> nm -p -g hal
 ⋮
 0000066112 T main     ↦ 0x00010240
 0000352140 D hal      ↦ 0x00055f8c
 0000360336 B foo      ↦ 0x00057f90
 ⋮
 0000286461 D _etext   ↦ 0x00045efd
 0000358433 D _edata   ↦ 0x00057821
 0000361444 D _end     ↦ 0x000583e4
 ⋮
```

Nicht alle Übersetzer/Binder unter UNIX verwenden den Unterstrich ('_'), um die Symbole problemorientierter Programmiersprachen von Symbolen der Assembliersprachen unterscheiden zu können.



Unterteilung des statischen Adressraums

Symbole, die vom Binder definiert und mit Werten belegt werden:

`extern etext`

- die erste Adresse nach dem Programmtext

`extern edata`

- die erste Adresse nach dem initialisierten Datenbereich

`extern end`

- die erste Adresse nach dem uninitialisierten Datenbereich
- entspricht anfangs der „Bruchstelle“ des Programms
  - kann zur Ausführungszeit verschoben werden (`brk(2)`/`sbrk(2)`)
- `sbrk((intptr_t*)0)` liefert den aktuell gültigen Wert

BSS (engl. *block started by symbol*, [3, 6])

- der Binder legt die Segmentgröße fest: [`edata`, `end`[
- der Lader löscht den Segmentinhalt: Vorbelegung mit Null