Teil V

Abstraktionen UNIX-ähnlicher Betriebssysteme

Überblick

Adressraumkonzepte und virtuelle Maschinen

physikalischer Adressraum (Hardware) . . . . Ebene $_2$

- ist durch die jeweils gegebene Hardwarekonfiguration definiert
- nicht alle Adressen sind gültig, zur Programmspeicherung verwendbar

logischer Adressraum (Kompilierer, Binder, Betriebssystem) . . . Ebene $_{5/4/3}$

- abstrahiert von Aufbau/Struktur des Haupt- bzw. Arbeitsspeichers
- 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

Physikalischer Adressraum

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.

Logischer Adressraum

Ausführungsdomäne von Prozessen im Mehrprogrammbetrieb

Illusion von einem eigenen (nicht zwingend linearen) Adressraum für jedes im Arbeitsspeicher 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:

Programm	$\mapsto$	logischer Adressraum
logischer Adressraum	$\mapsto$	physikalischer Adressraum

☞ logische Adressen sind mehrdeutig, physikalische dagegen eindeutig

Logischer Adressraum (Forts.)

Abbildungszeitpunkte

Adress(raum)abbildung kann auf verschiedenen Ebenen erfolgen:

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

Zielkonflikt (engl. *trade-off*) in Bezug auf Flexibilität und Effizienz

- je später die Abbildung durchgeführt wird, desto. . .
  - höher das Abstraktionsniveau und geringer die Hardwareabhängigkeit
  - höher der Systemaufwand und geringer der Spezialisierungsgrad

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

Verantwortung trägt allein das Betriebssystem, die MMU führt nur aus

Segmentierung eines logischen Adressraums

Logische Unterteilung zur effektiveren Programmverwaltung

Textsegment (engl. *text segment*)

- Maschinenanweisungen (Ebene $_{2/3}$) und andere Programmkonstanten
- statische oder dynamische Größe, je nach Betriebssystem
- ggf. gemeinsame Verwendung für mehrere Prozesse (engl. *shared text*)

Datensegment (engl. *data segment*)

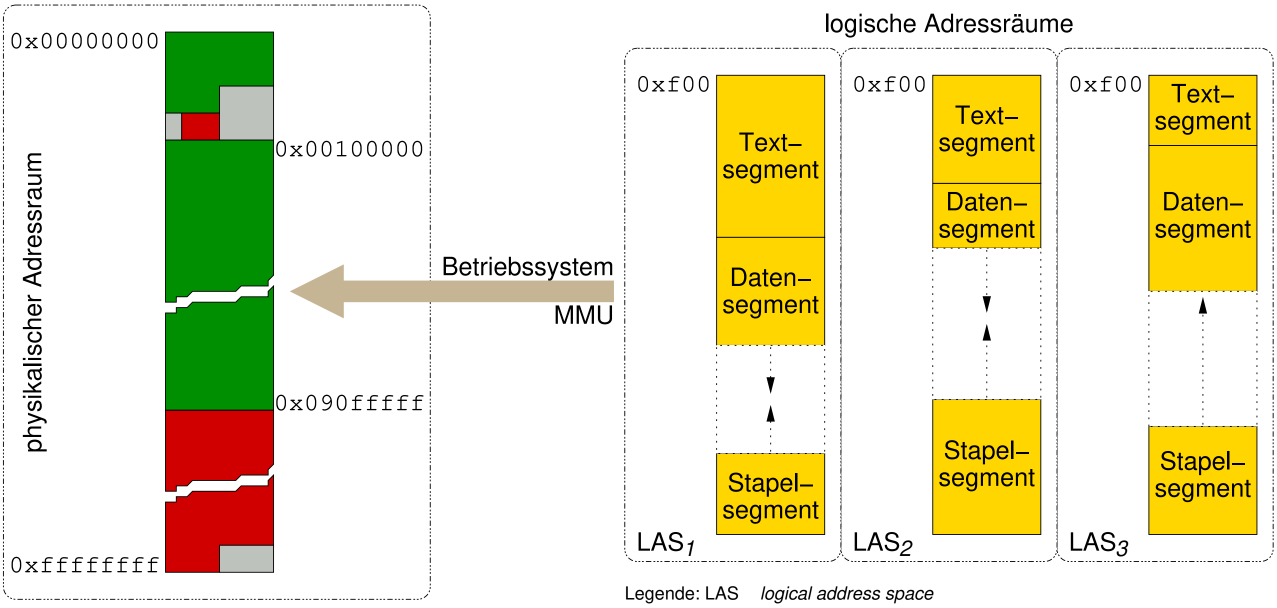
- initialisierte Daten, globale Variablen und ggf. die Halde (engl. *heap*)
- statische oder dynamische Größe, je nach Betriebssystem

Stapelsegment (engl. *stack segment*)

- lokale Variablen, Hilfsvariablen und aktuelle Parameter
- dynamische Größe

Adressraumabbildung auf Ebene 3

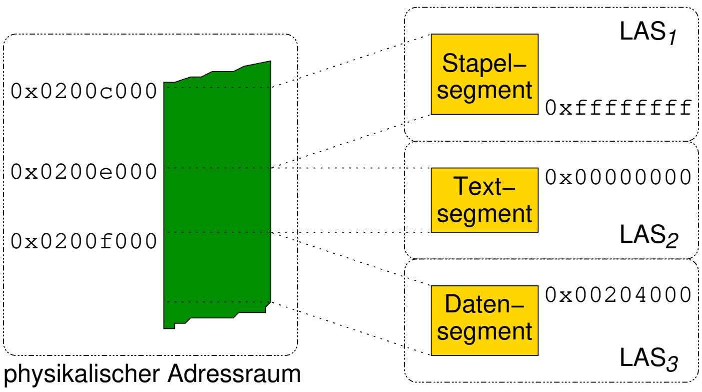
Betriebssystem und MMU implementieren logische Adressräume ☞ V-15



☞ Segmente müssen nicht angrenzend im logischen Adressraum liegen

Disjunktive Abbildung zur Ladezeit

Betriebssystem ordnet Segmente überschneidungsfrei im physikalischen Adressraum an



Kontrolliert vom BS ist eine gemeinsame Verwendung (engl. *sharing*) von Segmenten möglich, nämlich durch absichtliche Überschneidung im Arbeitsspeicher.

Verletzung der Segmentierung (engl. *segmentation violation*) wird durch die MMU verhindert und bewirkt einen Programmabbruch (S. IV-35):

$$0 \leq Adresse_{log} - Adresse_{min} < Länge(Segment), \text{ sonst Trap}$$

- Konstante $Adresse_{min}$ bestimmt den Anfang eines log. Adressraums

Adressrelokation zur Laufzeit

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

Veränderung einer logischen Adresse um eine Relokationskonstante: (Prinzip)

$$Adresse_{phy} = Adresse_{log} - Adresse_{min} + Basis(Segment)$$

- $Basis(Segment)$ ist die Ladeadresse im phys. Adressraum
  - $Adresse_{log} - Adresse_{min}$ relativiert zu Null
    - ist daher auch relative Adresse in Bezug auf $Basis(Segment)$
  - anschließende Addition „verschiebt“ den relativierten Wert
- die Ladeadresse eines Segments ist gleichfalls Relokationskonstante
  - für alle relativ(iert)en Adressen innerhalb von *Segment*

Relokation erfolgt nur bei unverletzter Segmentierung (S. IV-35)

Logischer Adressraum als Schutzdomäne

Robustheit von Softwaresystemen verbessern

Adressraumisolation, eine Maßnahme zur Erhöhung von Sicherheit. . .

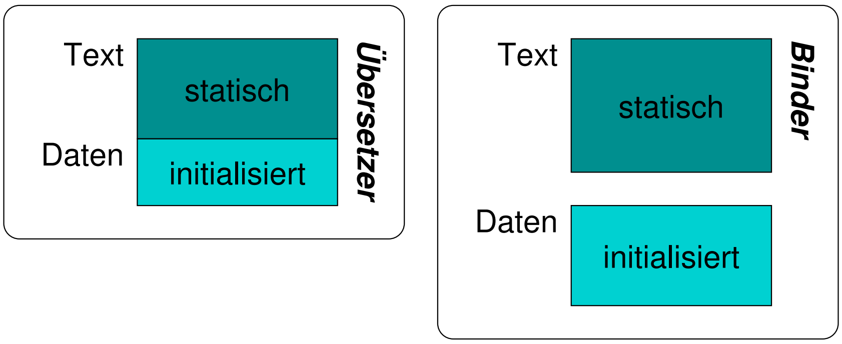
safety Schutz von Menschen und Sachwerten vor dem Versagen technischer Systeme
- Berechnungsfehler oder „Bitkipper“ abfangen
- allgemein (bei BS): Fehlerausbreitung eingrenzen

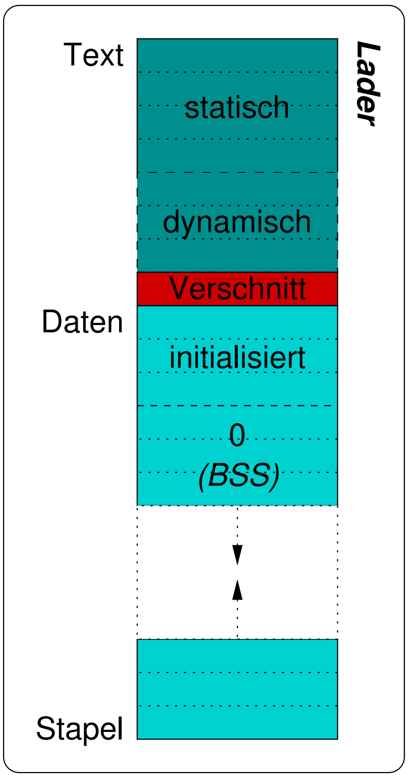
security Schutz von Informationen und Informationsverarbeitung vor „intelligenten“ Angreifern
- Adressraumausbrüche erschweren/verhindern
- allgemein (bei BS): Eindringlinge fern halten

. . . 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 die statischen Segmente in den RAM, fügen dynamische Text-/Stapelsegmente hinzu und setzen BSS (engl. *block started by symbol*) auf 0

Virtueller Adressraum

Grad des Mehrprogrammbetriebs (engl. *degree of multiprogramming*) erhöhen

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

- Erweiterung bzw. Spezialisierung des logischen Adressraums
- meist verbreitet ist die Seitenüberlagerung (S. IV-54)
- Adressraumzugriffe können E/A (Hintergrundspeicher) implizieren

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

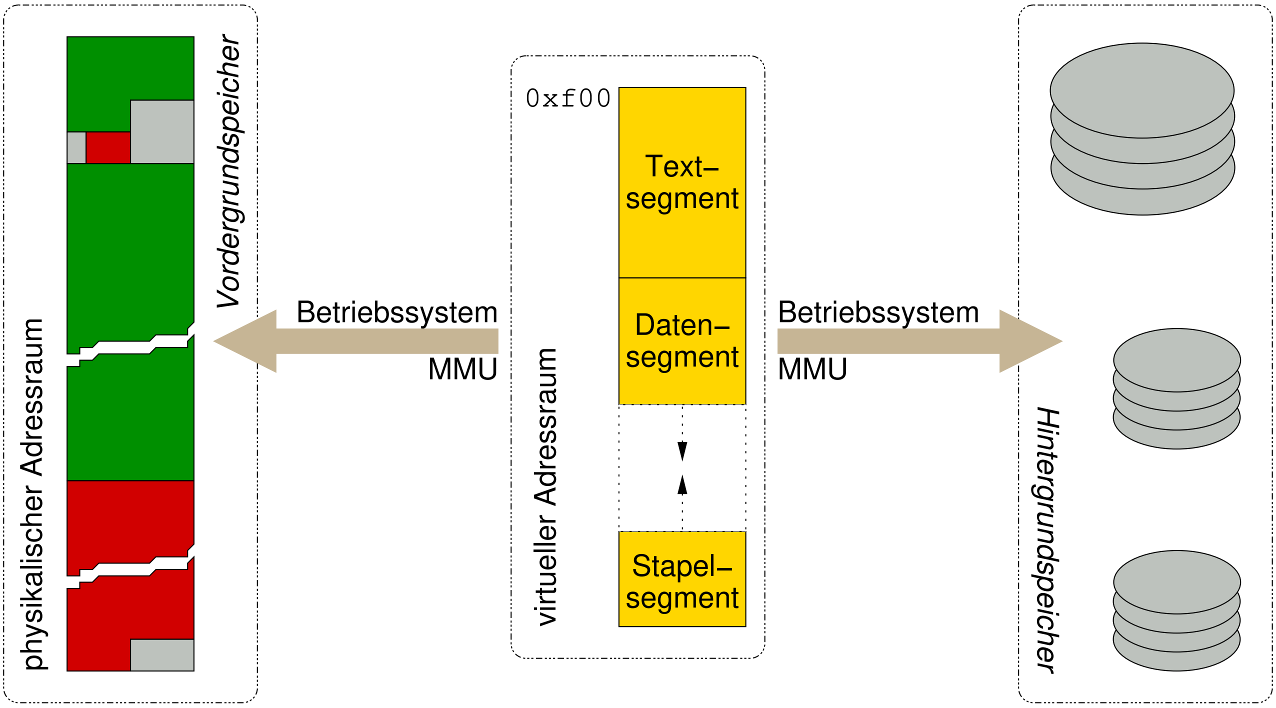
Programm	$\mapsto$	logischer Adressraum
logischer Adressraum	$\mapsto$	virtueller Adressraum
virtueller Adressraum	$\mapsto$	physikalischer Adressraum

☞ virtuelle Adressen „erben“ die Mehrdeutigkeitseigenschaft log. Adressen

Adressraumabbildung auf Ebene 3

Betriebssystem und MMU implementieren virtuelle Adressräume

☞ V-9



Umfang eines virtuellen Adressraums

Adressbreite einer CPU sagt wenig aus über den Arbeitsspeicher eines Rechners

Adressbreite von N Bits...

N	Adressraumgröße (2^N Bytes)	Dimension			
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})

Rechner sind ggf. nur mit einem Bruchteil des von einer CPU adressierbaren Arbeitsspeichers wirklich bestückt!

Intermezzo — 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	×	×

Floppy Disk. Der Zugriff auf das Medium erfolgt sektorweise. Die Größe eines Sektors wird als Zweierpotenz angegeben, die Anzahl der Sektoren kommt als Zehnerpotenz.

Abweichungen

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

Standardisierung [43] erfolgte erst sehr spät (Ende der 90er) — zu spät...

Experiment zur Adressraumgröße

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

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

```
_clear:
    li      r2,0
    li      r0,0
L2:
    stb     r0,0(r2)
    addic.  r2,r2,1
    bne+    cr0,L2
    blr
```

Bei 1 ns Zugriffszeit dauert das Löschen eines Bytes 13 ns!

PowerPC G4: 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 ein Byte mit dem Wert 0 in die nächste Speicherzelle. Jeder Schleifendurchlauf greift somit auf $3 \times 4 + 1 = 13$ Bytes zu.

Experiment zur Adressraumgröße (Forts.)

Virtueller Speicher kann die Programmausführung verlangsamen

Größe	Laufzeit	
2^{16}	851.968 Mikrosekunden	
2^{20}	13.631 Millisekunden	
2^{32}	55.835 Sekunden	
⋮	⋮	← heute
2^{48}	42.352 Tage	
2^{64}	7604.251 Jahre	(ohne Schaltjahre)

Virtueller Speicher: Die zur Zeit nicht benötigten Bereiche eines virtuellen Adressraums liegen im Hintergrundspeicher. Bei Bedarf werden diese „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! Der Löschvorgang eines 2^{32} Bytes umfassenden Adressraums würde somit bereits mehr als 1,5 Stunden dauern!

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()
   ⋮
```

Speicherkonzepte und -medium

Kurz-, mittel- und langfristige Informationsspeicherung

Vordergrundspeicher: Haupt- oder Arbeitsspeicher (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

Virtualisierung kann „Zugriffstransparenz“ mit sich bringen (Multics [14])

Speicherverwaltung (engl. *memory management*)

Symbiose 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

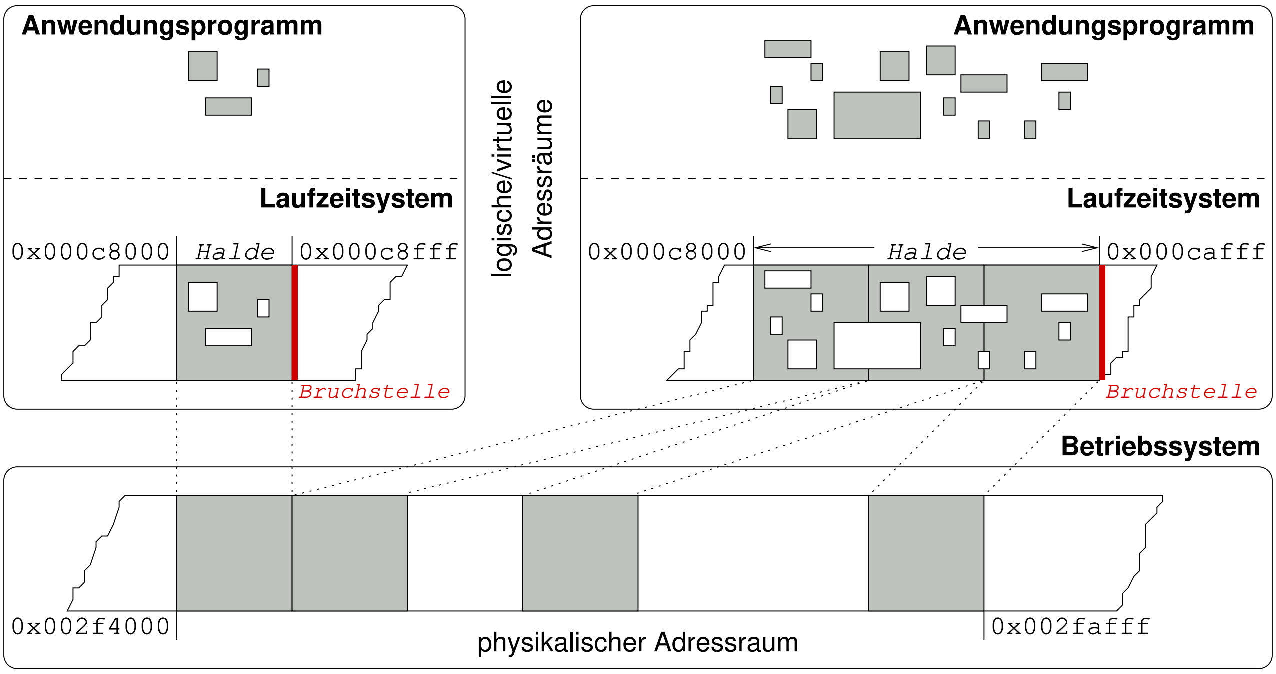
Betriebssystem verwaltet den global vorrätigen Speicher (d.h. den bestückten RAM-Bereich) des physikalischen Adressraums

- 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* [44])

Synergie bei der Speicherverwaltung

Betriebssystemaufruf als „Ausnahme“



UNIX Systemfunktionen

Laufzeitsystem — C Bibliothek

Linux, MacOS, SunOS

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

(☞ Aufgabe 4)

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 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()`

Langfristige Datenspeicherung

Abstraktion von Informationen tragenden Betriebsmitteln

Da'tei (engl. *file*) Sammlung von Daten, eine. . .

- zusammenhängende, abgeschlossene Einheit von Daten
- „beliebige“ Anzahl eindimensional adressierter Bytes

Dauerhaftigkeit von Dateien ist eine Frage des Speichermediums:

nicht-flüchtige Datenträger	Platte, Band, CD, DVD, . . . , EEPROM
flüchtige Datenträger	RAM

- die Datei selbst ist ein durchaus unbeständiges Gebilde

Kommunikationsmittel für kooperierende Prozesse

- im Sinne einer Röhre (engl. *pipe*) zum Informationsaustausch

Arten von Dateien

Unterscheidung von Programmtext und Programmdaten

ausführbare Dateien: Binär- und Skriptprogramme

- von einem Prozessor ausführbarer Programmtext

Binär	$\rightsquigarrow$	CPU, FPU, MCU, JVM, ..., Basic, Lisp, Prolog
Skript	$\rightsquigarrow$	`perl(1)`, `python(1)`, {`a`,`ba`,`c`,`tc`}`sh(1)`, `tcl(n)`

- der Prozessor liegt in Hard-, Firm- und/oder Software vor

nicht-ausführbare Dateien: Text-, Bild- und Tondaten

- von einem Prozessor verarbeitbare Programmdaten
  - .{doc, fig, gif, jpg, mp3, pdf, tex, txt, wav, xls, ...}
  - .{a, c, cc, f, F, h, l, o, p, r, s, S, y, ...}
- der Prozessor liegt in Form von Programmtext vor

Bezeichnung von Dateien

Symbolische und numerische Dateiadressen

Dateien sind „von aussen“ über symbolische Adressen erreichbar...

- benutzerdefinierter Name von beliebiger aber maximaler Länge
- auch als Dateiname (engl. *file name*) bekannt
  - wird ggf. vom Betriebssystem (teilweise) interpretiert

...„nach innen“ besitzt jede Datei eine numerische Adresse

- systemdefinierte Kennung einer Datenstruktur der Dateiverwaltung
- identifiziert den sogenannten Dateikopf (engl. *file head*)

☞ symbolische und numerische Dateiadresse bilden ein (festes) Paar

Erweiterung eines Dateinamens

Anreicherung um semantische Information

Dateinamensuffix (engl. *file extension*): eine meist durch einen Punkt vom Dateinamen abgegrenzte symbolische Erweiterung des Dateinamens

- liefert einen Hinweis auf das Dateiformat bzw. den Dateitypen

`.doc`	Textdokumente	MS-Word	
`.fm`		Framemaker	`maker(1)`
`.tex`		LaTeX	`latex(1)`
`.h`	Programme	Präprozessor	`cpp(1)`
`.c`		Kompilierer	`cc(1)`
`.s`		Assemblierer	`as(1)`
`.o`		Binder	`ld(1)`

- ist Dienstprogrammen und/oder dem Betriebssystem bekannt
  - bei UNIX die Dienstprogramme, bei Windows das Betriebssystem

Dateikopf und Dateikopfnummer

Systeminterne Verwaltungdaten einer UNIX-Datei

(☞ Aufgabe 5)

„*inode*“ (bzw. „*i-node*“, 1. Edition, 1971) enthält Dateiattribute:

- Eigentümer (*user ID*)
- Gruppenzugehörigkeit (*group ID*)
- Typ (reguläre/spezielle Datei)
- Rechte (lesen, schreiben, ausführen; Eigentümer, Gruppe, „Welt“)
- Zeitstempel (letzter Zugriff, letzte Änderung [Typ, Zugriffsrechte])
- Anzahl der Verweise („*hard links*“)
- Größe (in Bytes)
- Adresse(n) der Daten auf dem Speichermedium

„*inode number*“, Index in eine Tabelle von Dateiköpfen („*inode table*“):

- die numerische Adresse der Datei (innerhalb des Dateisystems)

Dateityp

Dateien zur Abstraktion von Daten, Geräten und Kommunikationsmitteln

reguläre Datei (engl. *regular file, ordinary file*)

- problemorientiertes, eindimensionales Bytefeld

spezielle Datei ein „Sammelsurium" von (UNIX) Konzepten:

- Verzeichnis (engl. *directory*)
  - Katalog von regulären und/oder speziellen Dateien
- Gerätedatei (engl. *device file*)
  - Zugang zu zeichen-/blockorientierten Geräte(treiber)n
- symbolische Verknüpfung (engl. *symbolic link*)
  - Abbildung eines Dateinamens auf einen Pfadnamen (S. V-41)
- benannte Röhre (engl. *named pipe*)
  - Kommunikationskanal zwischen unverwandten lokalen Prozessen
- Buchse (engl. *socket*)
  - Endpunkt zur bi-direktionalen Kommunikation zwischen Prozessen

Dateiverzeichnis

Konzept zur Gruppierung von Dateinamen

Katalog (engl. *catalogue*, *directory*) von symbolischen Namen

- definiert einen gemeinsamen Kontext
  - symbolische Adressen sind nur innerhalb ihrer Kontexte eindeutig
- implementiert eine „Umsetzungstabelle":

$$\left\{ \begin{array}{c} \text{symbolische Adresse} \\ \text{Dateiname} \\ \texttt{foo} \end{array} \right\} \rightsquigarrow \left\{ \begin{array}{c} \text{numerische Adresse} \\ \text{Dateikopf} \\ \texttt{4711} \end{array} \right\}$$

- speichert die Abbildung Dateiname $\mapsto$ Dateikopfnummer

Verknüpfung von Dateiname und Dateikopf

UNIX „*hard link*“ (auch kurz: *link*)

Eintrag im Dateiverzeichnis: Dateiname $\mapsto$ Dateikopfnummer

UNIX V7, `dir.h` [45]

```
typedef unsigned short ino_t;

#define DIRSIZ 14

struct direct {
    ino_t d_ino;
    char  d_name[DIRSIZ];
};
```

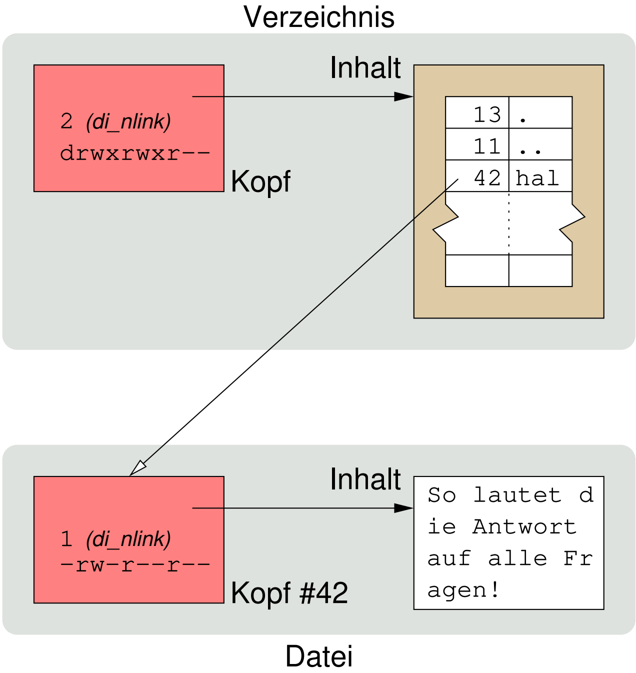
- die Abbildung ist als Wertepaar gespeichert
- mehrere Einträge können auf denselben Dateikopf verweisen
  - identische Dateikopfnummern
  - verschiedene Dateinamen
- ein Referenzzähler im Dateikopf vermerkt die Anzahl der Verweise

Anlegen/Löschen erfordert nur Schreibzugriffsrecht auf das Verzeichnis

- unabhängig von den Zugriffsrechten auf die referenzierte Datei

Verknüpfung von Dateiname und Dateikopf (Forts.)

Einträge anlegen/löschen ist eine Operation auf Verzeichnisse



Verzeichnisse sind „Spezialdateien“

- die selbst einen Namen und Dateikopf haben
- die erreichbar sind über eine Verknüpfung
  - eines anderen Verzeichnisses
- die Namen getrennt von Dateien speichern

Verknüpfungen anlegen/löschen zu können, ist eine Berechtigung, die sich nur auf das Verzeichnis der betreffenden Verknüpfungen bezieht!

Referenzzähler (engl. *reference count*)

☞ V-44

Unterstützung der „Müllsammlung“ (engl. *garbage collection*)

Buchführung über die Anzahl der Verknüpfungen zu einem Dateikopf geschieht über einen Verknüpfungszähler (engl. *link count*; `di_nlink`)

`di_nlink != 0` Datei/Verzeichnis wird referenziert

- der Dateikopf ist (aus Sicht des Systems) in Benutzung

`di_nlink == 0` Datei/Verzeichnis wird nicht referenziert

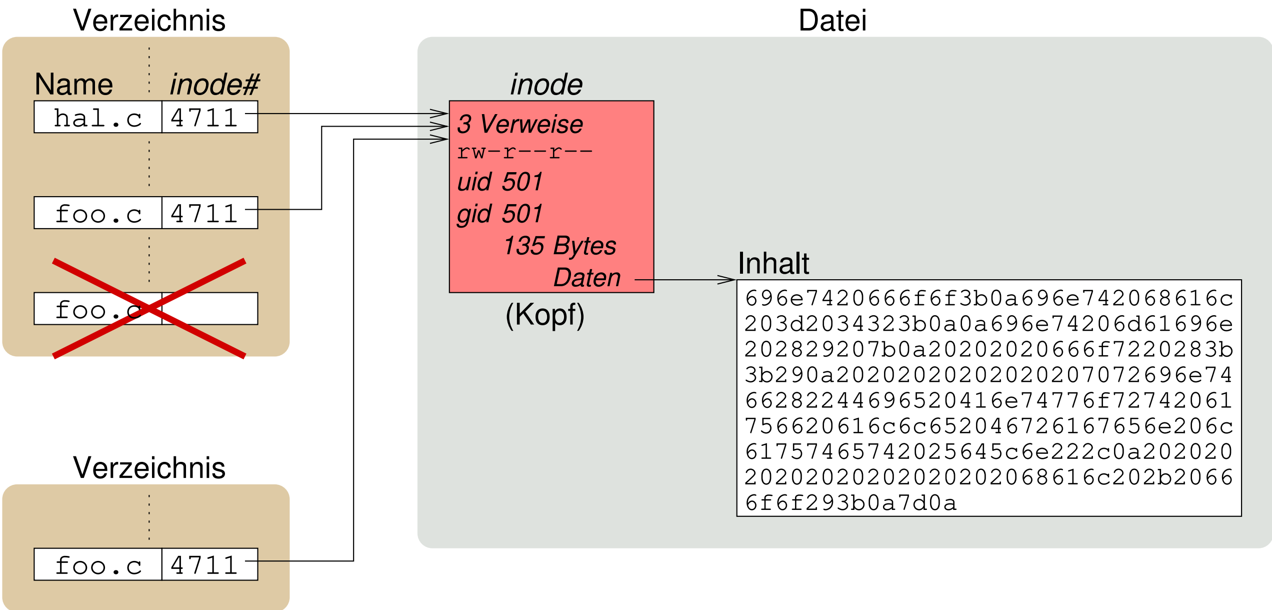
- der Dateikopf samt anhängender Daten kann freigegeben werden

Veränderung des Verknüpfungszählerwertes erfolgt beim Eintragen (++) bzw. Löschen (--) von Verknüpfungen im jeweiligen Verzeichnis:

- Verzeichnisverknüpfung: `mkdir(2)`/`rmdir(2)`
  - auf Verzeichnisse verweisen mindestens zwei Verknüpfungen (S. V-43)
- Dateiverknüpfung: `link(2)`/`unlink(2)`

UNIX Dateiverzeichnis und Datei

Verknüpfung, Dateikopf und Dateiinhalt



UNIX Systemfunktionen

Operationen auf Dateiköpfe

Linux, MacOS, SunOS

```
fd  = open(path, fags, mode)
num = read(fd, buf, nbytes)
num = write(fd, buf, nbytes)
off = lseek(fd, offset, whence)
ok  = close(fd)
ok  = stat(path, buf)
    ⋮
```

(☞ Aufgabe 5)

Dateideskriptor (engl. *file descriptor*)

- von der Dateiverwaltung des Betriebssystems implementierter eindeutiger Bezeichner (engl. *identifier*) einer geöffneten Datei
- meist als Ganzzahl (engl. *integer*) repräsentiert

Bedeutung von Namen

Kontextfreie Namen sind bedeutungslos

Java bedeutet im Kontext. . .

- „Geographie“ eine Insel
- „Genussmittel“ ein Heissgetränk
- „Informatik“ eine Programmiersprache

C bedeutet im Kontext. . .

- „Sprache“ einen Buchstaben
- „Musik“ eine Note
- „Informatik“eine Programmiersprache

Namensräume (engl. *name spaces*) ordnen Namen Bedeutungen zu

Aufbau von Namensräumen

flache Struktur: definiert nur einen einzigen Kontext

- Eindeutigkeit muss mit der Namenswahl selbst gewährleistet werden

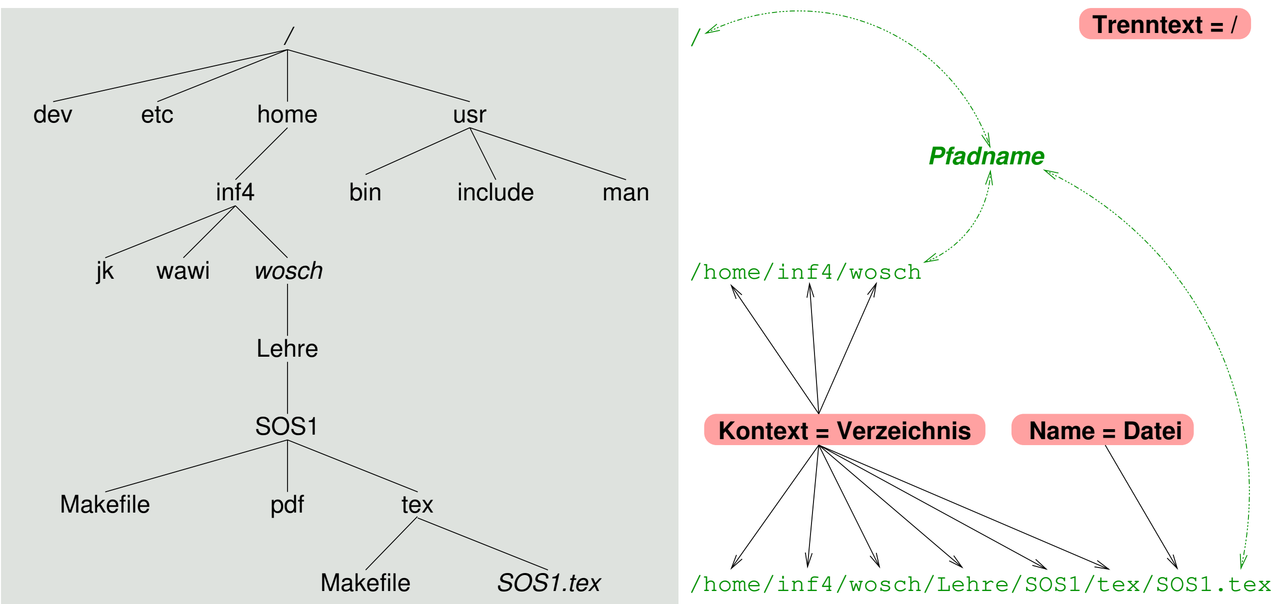
hierarchische Struktur: definiert mehrere Kontexte

- Eindeutigkeit wird durch einen Kontextnamen als Präfix erreicht
  - Kontexte enthalten Namen von Dateien und/oder (anderer) Kontexte
  - der Name einer Datei entspricht dann einem „Blatt“ des Namensbaums
- als Separatoren werden meist Sonderzeichen („Trenntext“) verwendet:

Schrägstrich (*slash*)	⇒ UNIX
zurückgelehnter Schrägstrich (*backslash*)	⇒ Windows

Hierarchischer Namensraum

Dateibaum (engl. *file tree*)



Navigation im Namensraum

Eindeutigkeit der symbolischen Adresse (einer Datei) ist durch einen Pfad (engl. *path*) im Namensraum gegeben

- der Pfadname (engl. *path name*) ist ein vollständiger Dateiname

Formaler Aufbau eines (UNIX) Pfadnamens in EBNF [46]

$$
\begin{aligned}
\textit{pathname} &= \textit{resolver} \mid [\textit{resolver}],\ \{\textit{name},\ \textit{resolver}\},\ \textit{name}; \\
\textit{resolver} &= \{\textit{separator}\}-; \\
\textit{separator} &= \text{“}/\text{“}; \\
\textit{name} &= \{\textit{character}\}-; \\
\textit{character} &= \textit{character set} - \textit{separator}; \\
\textit{character set} &= \text{ASCII};
\end{aligned}
$$

z.B.: `/`, `.`, `..`, `foo`, `foo/bar`, `/foo`, `bar/`, `./bar/..`, `../foo/.//bar//`

Spezielle Kontexte

Sonderverzeichnisse

Wurzelverzeichnis (engl. *root directory*)

- bezeichnet die Wurzel des Dateibaums (solitär '/', bei UNIX)
- wird vom System bzw. Administrator (engl. *super user*) gesetzt
  - `chroot(2)`, privilegierte Operation

Arbeitsverzeichnis (engl. *working directory*)

- die gegenwärtige Position eines Programms/Prozesses im Dateibaum
- ändert sich beim „Durchklettern“ des Dateibaums
  - `chdir(2)`

Heimatverzeichnis (engl. *home directory*)

- das initiale Arbeitsverzeichnis eines Benutzers/Prozesses
- wird vom System gesetzt bei Sitzungsbeginn
  - `login(1)`

Relative Adressierung von Kontexten

Systemdefinierte Verzeichnisnamen

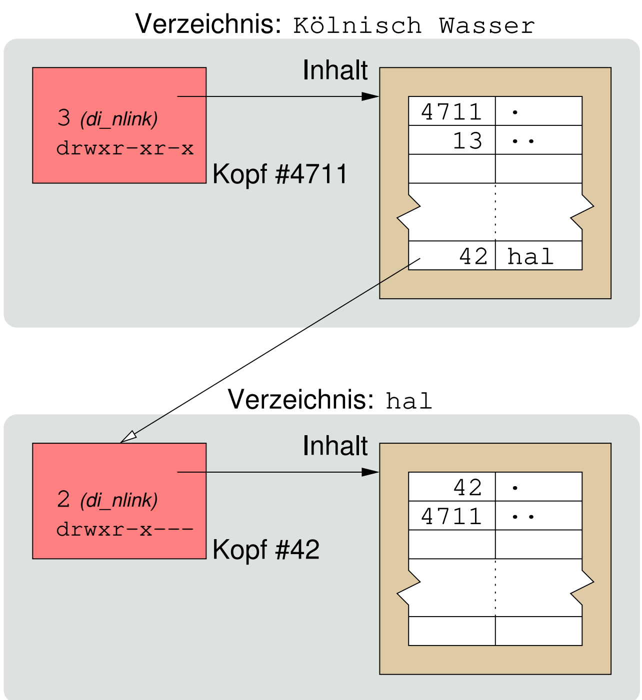
- `.` („*dot*“): das gültige Arbeitsverzeichnis (engl. *current working directory*)
  - benennt die Verknüpfung zu selbigem Verzeichnis (Selbstreferenz)
    - ermöglicht die eindeutige Identifikation eines Arbeitsverzeichnisses, ohne dessen wirklichen Namen kennen zu müssen (`stat(2)`)
    - erzwingt einen lokalen Bezugspunkt (als Namenspräfix „`./`“)
  - erster Eintrag in jedem Verzeichnis
- `..` („*dot dot*“): das gegenwärtige Elternverzeichnis
  - benennt die Verknüpfung zum übergeordneten Verzeichnis, das die Verknüpfung zum Arbeitsverzeichnis enthält
    - entspricht '`.`', falls es kein Elternverzeichnis gibt (Wurzelverzeichnis)
  - zweiter Eintrag in jedem Verzeichnis

☞ `mkdir(2)`

Relative Adressierung von Kontexten (Forts.)

Verknüpfungen für Arbeits- und Elternverzeichnisse

☞ V-35

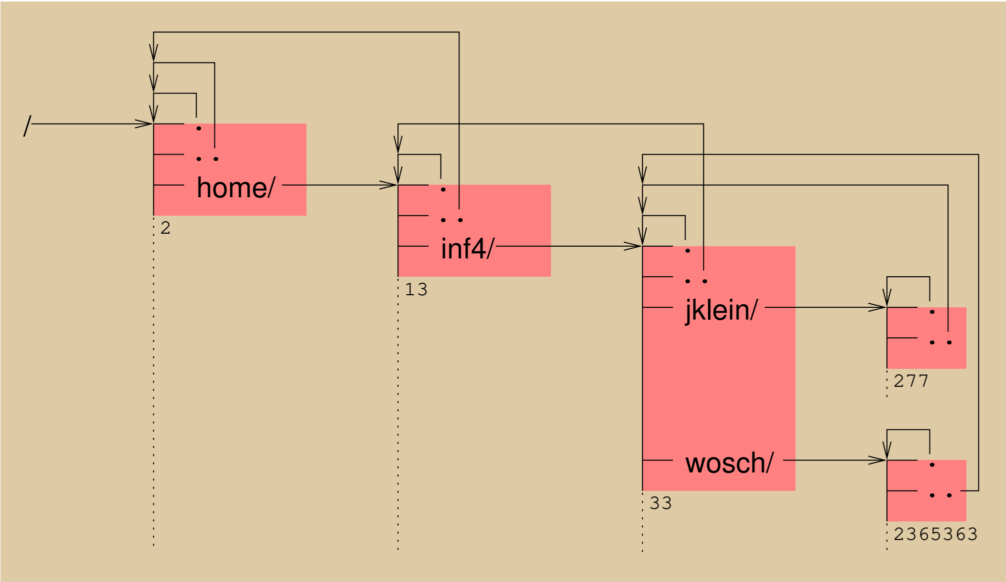


- `.` bezeichnet den Dateikopf, der das Verzeichnis selbst beschreibt
  - Dateikopfnummer des Arbeitsverzeichnisses
- `..` bezeichnet den Dateikopf des Verzeichnisses, das den Verzeichnisnamen speichert
  - Dateikopfnummer des Elternverzeichnisses

`di_nlink` # Verknüpfungen, die `.`

referenzieren SunOS, Linux
speichert MacOSX

Dynamische Datenstruktur „Dateibaum“



Verzeichnisnamen entsprechen einer Vorwärtsverkettung (z.B. `wosch`)
. ist eine Selbstreferenz
. . entspricht einer Rückwärtsverkettung

Arten von Pfadnamen

relativer Pfadname — vom gegenwärtigen Arbeitsverzeichnis ausgehend, z.B. von `/home/inf4/wosch` aus:

- `Lehre/SOS1/SOS1.tex`
- `./Lehre/SOS1/SOS1.tex`
- `../wosch/Lehre/SOS1/SOS1.tex`

. . . oder von `/home/inf4/jk` aus:

- `../wosch/Lehre/SOS1/SOS1.tex`

absoluter Pfadname — vom Wurzelverzeichnis ausgehend:

- `/home/inf4/wosch/Lehre/SOS1/SOS1.tex`

Bindung und Auflösung von Namen bzw. Pfadnamen

Abbildung/Umsetzung: symbolische Adresse $\mapsto$ numerische Adresse

Namensbindung (engl. *name binding*) bedeutet die Abbildung der symbolischen Adresse in eine numerische Adresse

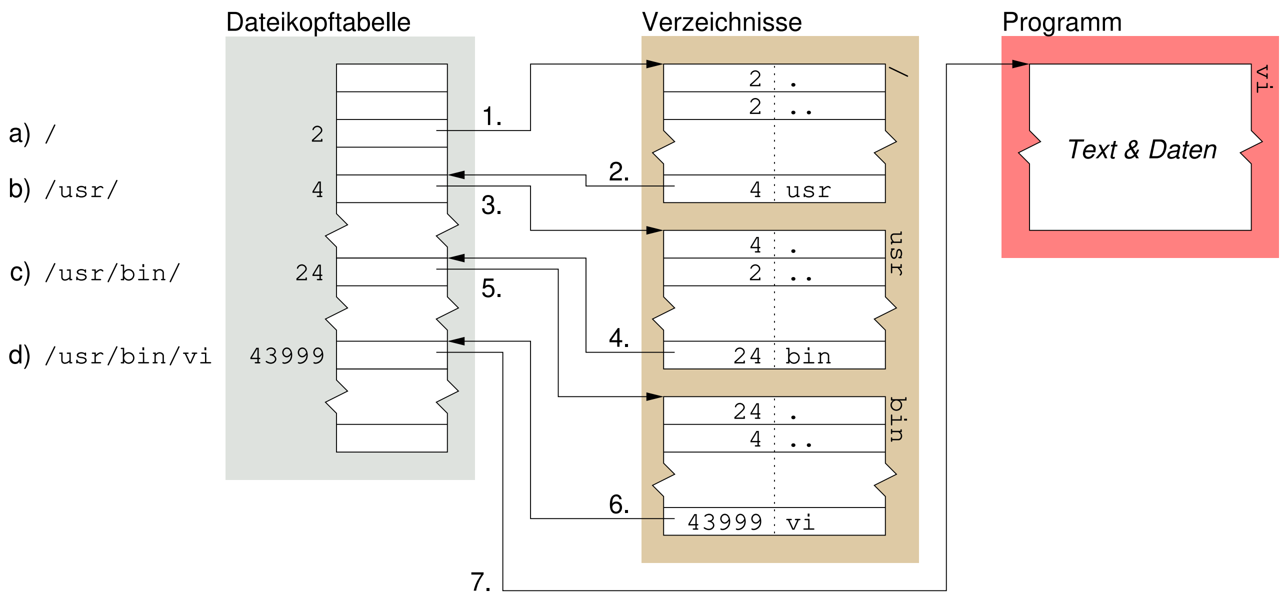
- zum Erzeugungszeitpunkt einen Datei-/Verzeichnisnamen...
  - mit einem freien/belegten Dateikopf verknüpfen
  - in ein Dateiverzeichnis eintragen
- Pfadnamen mit Dateikopf assoziieren: `creat(2)`, `link(2)`

Namensauflösung (engl. *name resolution*) bedeutet die Umsetzung der symbolischen Adresse in eine numerische Adresse

- zum Benutzungszeitpunkt Dateiverzeichnisse durchsuchen...
  - schrittweise für jeden einzelnen Verzeichnisnamen im Pfad
  - schließlich für den Dateinamen
- Dateikopf des Pfadnamens lokalisieren: `open(2)`

Auflösung von Namen bzw. Pfadnamen

Beispiel: `/usr/bin/vi`



Verwaltung von Dateien und Dateibäumen

Dateisystem (engl. *file system*)

Komplex von Datenstrukturen zur Verwaltung von Hintergrundspeicher

- der Dateisystemkopf (UNIX *super block*)
  - speichert Verwaltungsinformationen und Systemparameter
  - legt die Grenzwerte des Dateisystems fest
- die Dateikopftabelle (UNIX *inode table*)
  - zur Beschreibung von Dateien und/oder Verzeichnisse
- die Datenblöcke (engl. *data blocks*)
  - zur Speicherung der Inhalte der Dateien/Verzeichnisse

Beschreibung einer Partition (engl. *partition*) im Hintergrundspeicher

- die logische Unterteilung in einen Satz zusammenhängender Sektoren

Montieren von Dateisystemen

Auf- und Abbau einer Dateisystemhierarchie

Montierpunkt (engl. *mount point*) ist eine Stelle im Wirtsdateisystem, an der ein Gastdateisystem eingebunden werden kann

- ein (beliebiges) Verzeichnis im Wirtsdateisystem
- wird mit der Wurzel (S. V-42) des Gastdateisystems überlagert

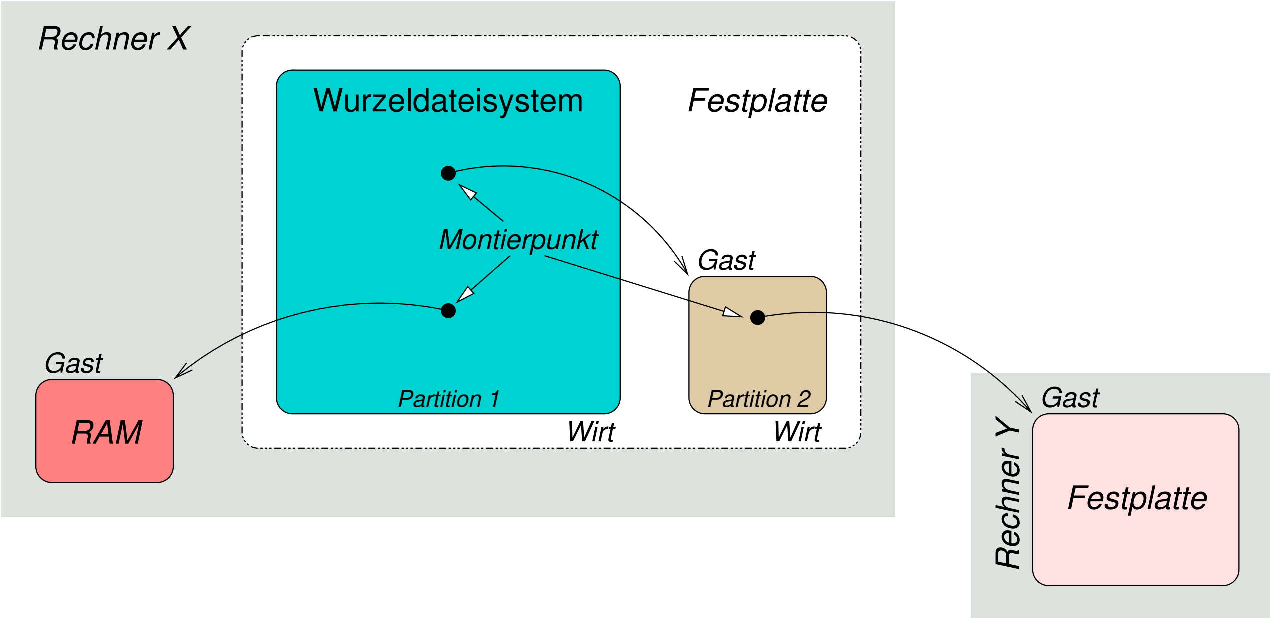
Wirts- und Gastdateisystem bilden jeweils eigene Partitionen. . .

- auf demselben oder einem anderen (dateisystemverträglichen) Gerät
  - z.B. Band, Fest-/Wechselplatte, CD, DVD, EEPROM, . . . , RAM
  - ggf. auch auf unterschiedlichen Rechnern eines Rechnerverbunds
- von gleicher oder verschiedener (logischer) Struktur
  - ggf. ein Mix z.B. von S5FS, UFS, FFS, EXT2 und NTFS

Ausgangspunkt ist das Wurzeldateisystem (engl. *root file system*)

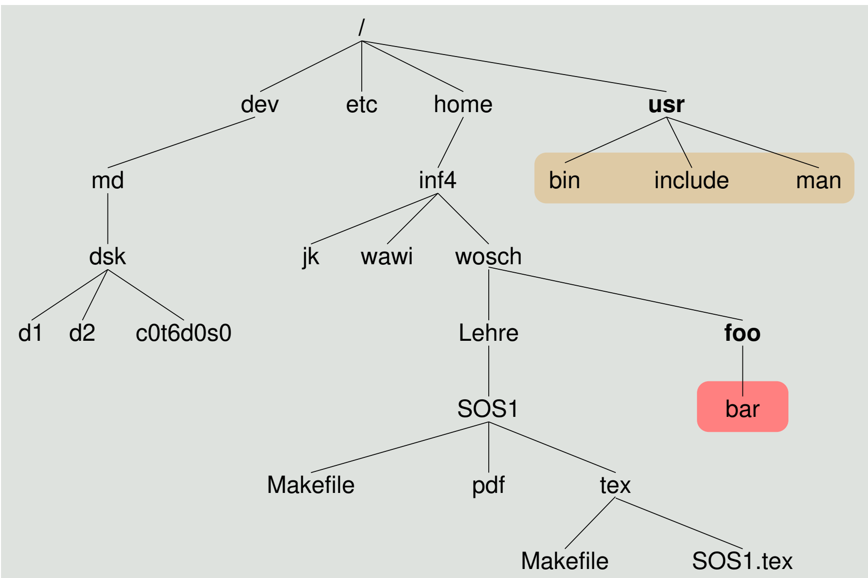
- d.h. das Dateisystem, von dem das Betriebssystem urgeladen wird

Hierarchiebildung von Dateisystemen



Einbindung von Namensräumen

Integration von Dateibäumen montierbarer Dateisysteme



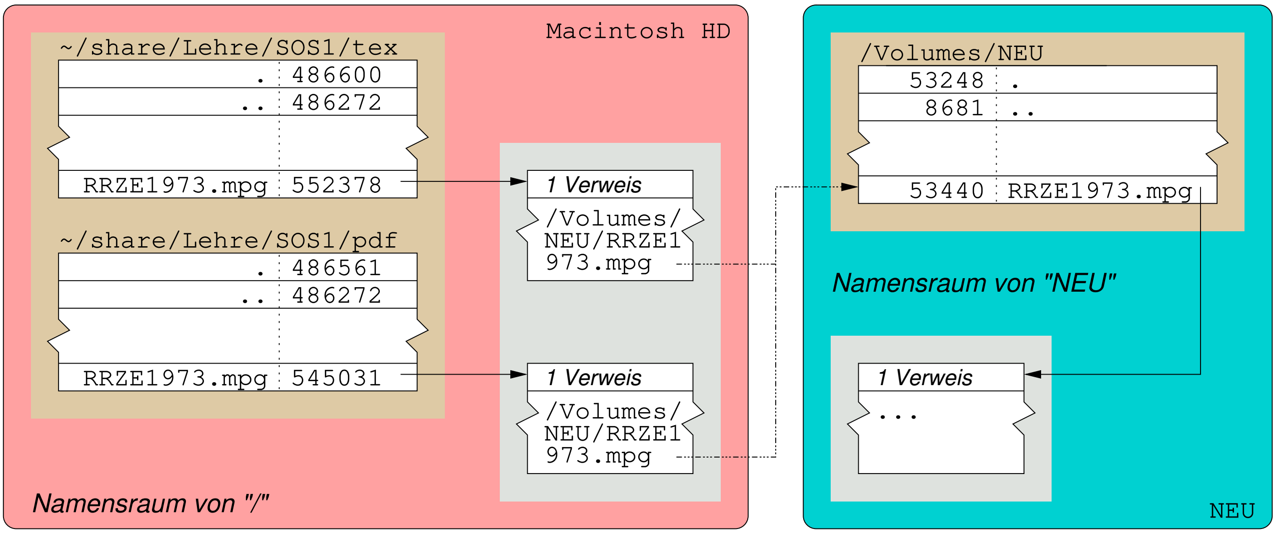
`/   -> /dev/md/dsk/d1`
Wurzel–/Wirtsdateisystem

`usr -> /dev/md/dsk/d2`
Gastdateisystem

`foo -> /dev/md/dsk/c0t6d0s0`
Gastdateisystem

Verweis hinein in einen anderen Namensraum

Symbolische Verknüpfung (engl. *symbolic link*)



Urspung ist der symbolische Name (engl. *symbolic name*) in Multics [47]

- zur dynamischen Bindung von Namen an (besondere) E/A-Geräte

Symbolische Verknüpfungen „*Considered Harmful*“

Namen sind Schall und Rauch . . . auf den Inhalt kommt es an!

```
wosch@lorien 1$ mkdir -p Laptop/faui43w; cd Laptop
wosch@lorien 2$ ln -s faui43w lorien
wosch@lorien 3$ ls -l
total 8
drwxr-xr-x  2 wosch  wosch  68 29 Apr 13:01 faui43w
lrwxr-xr-x  1 wosch  wosch   7 29 Apr 13:02 lorien -> faui43w
wosch@lorien 4$ cd lorien
wosch@lorien 5$ cd ..; rmdir faui43w; cd lorien
-bash: cd: lorien: No such file or directory
wosch@lorien 6$ ls -l
total 8
lrwxr-xr-x  1 wosch  wosch   7 29 Apr 13:02 lorien -> faui43w
wosch@lorien 7$ mkdir faui43w; cd lorien
wosch@lorien 8$ ln -s fata\ morgana SOS1
wosch@lorien 9$
```

UNIX Systemfunktionen
Operationen auf Verzeichnisse

Linux, MacOS, SunOS

```
fd = creat(path, mode)
   ↦ open(path, O_CREAT | O_TRUNC | O_WRONLY, mode)
ok = link(path1, path2)
ok = symlink(path1, path2)
ok = unlink(path)
ok = mkdir(path, mode)
ok = rmdir(path)
ok = mount(type, dir, flags, data)
ok = umount(dir, flags)
   ⋮
```


UNIX Systemfunktionen
Operationen auf Verzeichnisse (Forts.)

Linux, MacOS, SunOS

```
dirp = opendir(path)
dp   = readdir(dirp)
loc  = telldir(dirp)
void   seekdir(dirp, loc)
void   rewinddir(dirp, loc)
ok   = closedir(dirp)
     ⋮
```

Vorsicht ist angebracht: `readdir(3)`'s Implementierung ist eintrittsvariant

- nebenläufige Ausführung kann Nebeneffekte zur Folge haben
- eintrittsinvariant (engl. *reentrant*) dagegen ist `readdir_r(3)`