

Systemprogrammierung

Grundlage von Betriebssystemen

Teil B – VI.3 Betriebssystemkonzepte: Namen

Wolfgang Schröder-Preikschat

24. Juni 2015



Agenda

Einführung

Semiotik

Informatik

Dynamische Namensauflösung

Virtuelle Adressen

Pfadnamen

Internetadressen

Statische Namensauflösung

Übersetzer

Binder

Lader

Zusammenfassung



Gliederung

Einführung

Semiotik

Informatik

Dynamische Namensauflösung

Virtuelle Adressen

Pfadnamen

Internetadressen

Statische Namensauflösung

Übersetzer

Binder

Lader

Zusammenfassung



Definition (Name \mapsto `www.duden.de`)

Kennzeichnende Benennung eines Einzelwesens, Ortes oder Dinges, durch die es von anderen seiner Art unterschieden wird; Eigenname.

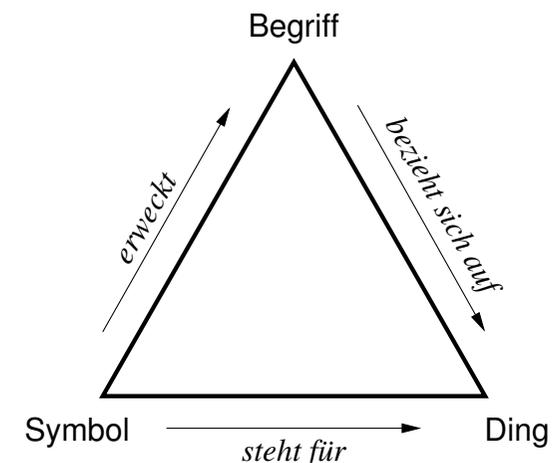
- Thema sind **Namen** zur relativ eindeutigen Bezeichnung von Dingen eines Rechensystems, die nach außen zugänglich sein müssen
 - allgemein Stellen im Haupt-, Arbeits- und Ablagespeicher
 - zu referenzierende Bezugspunkte von Programmtext und -daten
- relativ, da jeder dieser Namen nur innerhalb von einem bestimmten **Kontext** eine wohldefinierte und eindeutige Bedeutung hat
 - innerhalb eines (realen, logischen, virtuellen) Adressraum, Dateisystems
- in dem Zusammenhang werden grundlegende **Abbildungsfunktionen** behandelt, die Namen in **Adressen** umwandeln
 - Seitennummerierung, Segmentierung, seitennummerierte Segmentierung
 - Indexierung bei Dateisystemen: Indexknotentabelle, Verzeichnis, Dateien
- abgerundet durch Verfahren, die vor Programmlaufzeit greifen
 - Abbildungsfunktionen von Kompilierer, Assemblierer, Binder und Lader



Semiotisches Dreieck

Beziehung zwischen Benennung (*Bezeichnung*), Begriff (*Bedeutung*) und Gegenstand (*Bezeichnetes*).

- Benennung meint die Versprachlichung einer Vorstellung
 - ruft Begriffe ins Bewusstsein
 - bezeichnet einen Gegenstand, ein Ding
- Bezeichnung umfasst insbesondere auch nichtsprachliches wie:
 - **Symbole** und **Nummern**
- Bezeichnetes ist ein Objekt — nicht nur im informatischen Sinne
 - mit einem **Namen** versehen, benannt durch ein oder mehrere Wörter
- wichtig ist, die Benennung sprachlich richtig und treffend auszulegen
 - möglichst genau und dabei knapp zugleich
 - am Sprachgebrauch orientieren
 - auf **Fachsprache** bezogen, nicht Flexibilität brauchende Gemeinsprache



```
1  #include <sys/socket.h>
2  #include <netinet/in.h>
3  #include <arpa/inet.h>
4
5  #include <netdb.h>
6  #include <stdlib.h>
7  #include <stdio.h>
8
9  int main(int argc, char **argv) {
10     if (argc == 2) {
11         struct hostent *host = gethostbyname(argv[1]);
12         if (host != NULL) {
13             unsigned int i = 0;
14             printf("%s = ", host->h_name);
15             while (host->h_addr_list[i] != NULL)
16                 printf("%s ", inet_ntoa(*(struct in_addr*)(host->h_addr_list[i++])));
17             printf("\n");
18         }
19     }
20 }
```

■ **Dienstprogramm** (*utility*) zum Nachschlagen einer Internetadresse

- darin werden verschiedene Dinge symbolisch bezeichnet:
 - das Programm `main` samt Variablen `argc`, `argv`, `host` und `i`
 - sowie die Unterprogramme `gethostbyname`, `printf` und `inet_ntoa`
- hinzu kommen Bezeichnungen, die in der Hand der Umgebung liegen:
 - der Dateiname `host.c` und Pfadname `./a.out` bzw. `./host`
 - sowie eine Internetadresse als Programmparameter `argv[1]`



Stufen von Namensauflösungen

- durch Kompilierer, Assemblierer und Binder ($\text{Ebene}_5 \mapsto \text{Ebene}_3$)
 - symbolische (Text, Daten) auf numerische Referenz
- durch Betriebssystem ($\text{Ebene}_3 \mapsto \text{Ebene}_2$)
 - numerische Referenz auf virtuelle, logische oder reale Adresse
 - insbesondere auch Prozesskennung (PID) auf Prozesskontrollblock
 - aber ebenso Dateizeiger (`FILE*`) auf Dateideskriptor
 - allgemein/abstrakt: eine Handhabe (*handle*) auf eine Systemressource
 - symbolische auf numerische Referenz¹
 - Pfadname (Verzeichnis, Datei) auf Indexknotennummer (*inode number*)
 - Internetadresse (URL) auf Netzwerkadresse (IP-Adresse)

Gemeinsamkeit: *Symbol* \mapsto *Nummer*

Dies trifft auch zu auf die Herleitung einer virtuellen, logischen oder realen Adresse aus einer numerischen Referenz. Denn letztere ist als Zwischenschritt zu begreifen, der Dinge eines symbolisch formulierten Programms eine numerische Identität gibt. Daher ist die Abbildung $\text{Ebene}_5 \mapsto \text{Ebene}_2$ ganzheitlich zu sehen:

- symbolische Referenz auf virtuelle, logische oder reale Adresse

¹Die ggf. auf eine Speicheradresse abgebildet wird, wie eben geschildert.



Gliederung

Einführung

Semiotik

Informatik

Dynamische Namensauflösung

Virtuelle Adressen

Pfadnamen

Internetadressen

Statische Namensauflösung

Übersetzer

Binder

Lader

Zusammenfassung



Definition ([1, S. 157–158])

- *An address used by the programmer is called a “name” or a “virtual address,” and the set of such names is called the address space, or name space.*
 - *An address used by the memory is called a “location” or “memory address,” and the set of such locations is called the memory space.*
 - *Since the address space is regarded as a collection of potentially usable names for information items, there is no requirement that every virtual address “represent” or “contain” any information.*
-
- die Deutung ist **kontextabhängig**, ebenso was die Adresse bezeichnet
 - derselbe Name in verschiedenen Namensräumen kann verschiedene Orte (nicht nur in einem Rechensystem) adressieren
 - derselbe Ort kann über verschiedene Namen adressiert werden



Ortstransparente Namen

- Adressen, die **Speicherorte** bezeichnen, können realer, logischer oder virtueller Natur sein
 - real** ■ muss exakt dort liegen, intern
 - logisch** ■ kann woanders liegen, intern
 - virtuell** ■ kann woanders liegen, intern oder extern
- als **interner Ort** ist ein Platz im Hauptspeicher gemeint, identifiziert durch eine Adresse im realen Adressraum
 - insbesondere die „anwesende Seite“ im Falle von Speichervirtualisierung
- demgegenüber drückt **externer Ort** aus, dass der Platz irgendwo im System aber eben nicht im Hauptspeicher liegt
 - im Ablagespeicher oder Hauptspeicher eines anderen Rechensystems
 - beide gegebenenfalls nur indirekt über ein Rechnernetz zugänglich
- dies schließt **speicherabgebildete** (*memory-mapped*) **Dinge** mit ein, also im logischen/virtuellen Adressraum platzierte Objekte
 - wie etwa Gerätereister, Bildspeicher oder **Dateien**
 - der Zugriff läuft dann über Lese-/Schreibaktionen der Befehlssatzebene²

²Partiell interpretiert durch das Betriebssystem.



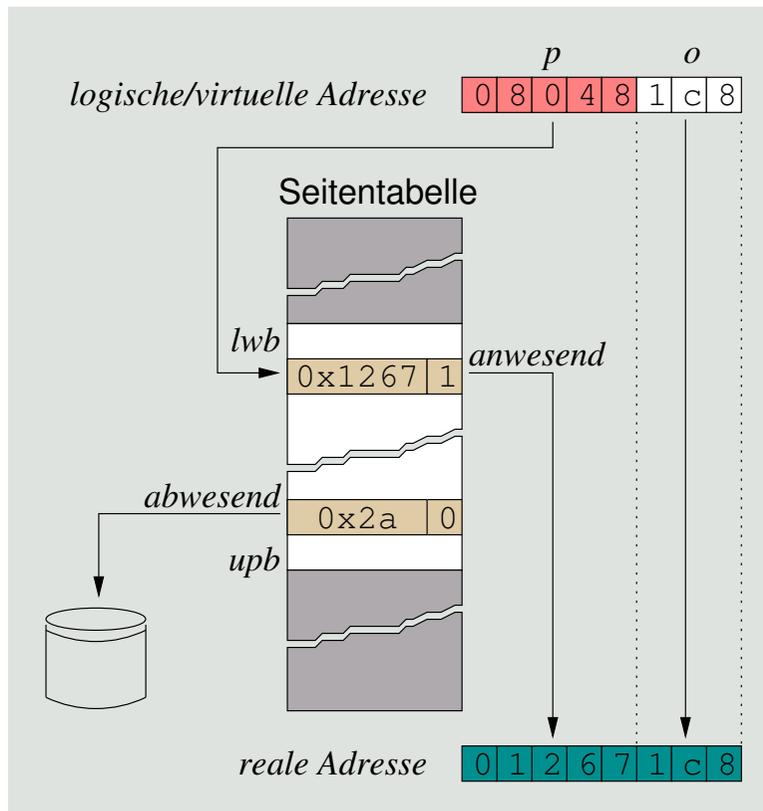
Organisation des Namensraums

- **Seitennummerierung** (*paging*) [4, S. 29–30]
 - jede Adresse wird interpretiert als Tupel $A_p = (p, o)$, wobei
 - Oktettnummer $o = [0, 2^i - 1]$, mit $9 \leq i \leq 30$
 - Seitennummer $p = [0, 2^{n-i} - 1]$, mit $32 \leq n \leq 64$
 - übliche Einkomponentenadresse \rightsquigarrow **eindimensionaler Adressraum**
 - d.h., Oktetts oder Worte in einer Dimension aufgereiht
- **Segmentierung** (*segmentation*)
 - jede Adresse ist repräsentiert als Paar $A_s = (s, a)$, wobei
 - Segmentname $s = [0, 2^m - 1]$, mit $12 \leq m \leq 18$
 - Adresse $a = [0, 2^n - 1]$, mit $32 \leq n \leq 64$
 - Zweikomponentenadresse \rightsquigarrow **zweidimensionaler Adressraum**
 - d.h., Segmente in der ersten und Segmentinhalte in der zweiten Dimension
- **seitennummerierte Segmentierung** (*paged segmentation*)
 - jedes Segment ist seitennummeriert ausgelegt, d.h., a in A_s entspricht A_p



Namensabbildung bei Seitennummerierung

- die Abbildungsfunktion nutzt eine ein-/mehrstufige **Seitentabelle**:



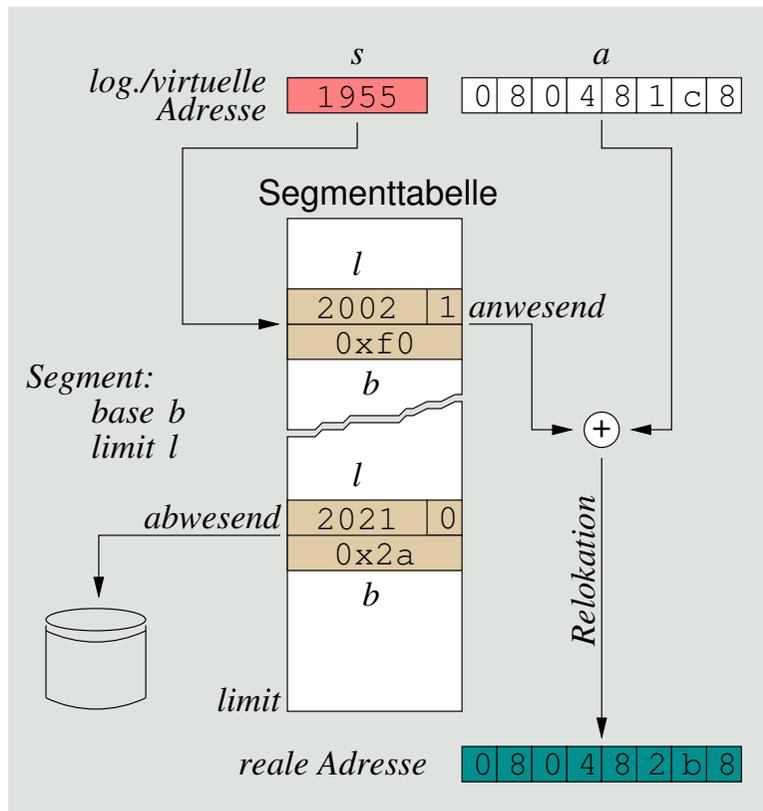
- p ist **Indexwert** für gültige Seiten im Bereich $[P_{lwb}, P_{upb}]$
 - sei $P = 2^{n-i}$ max. Seitenanzahl
 - dann gilt $0 \leq P_{lwb} < P_{upb} \leq P - 1$
- ein möglicher **Indexfehler** muss erkannt werden
 - Tabelle auffüllen mit Einträgen, die Abbildungsfehler erzwingen
 - Grenzwertprüfung auf Basis eines *limit*-Registers ist unüblich
- P hängt ab von Seitengröße 2^i und bestimmt die Stufenanzahl

- ein **Seitenfehler** (*page fault*) bedeutet damit verschiedenerlei:
 - gültig falls $P_{lwb} \leq p \leq P_{upb}$, dann ist p abwesend oder ungenutzt³
 - sonst ungültig: die betreffende Seite gehört nicht zum Prozessadressraum

³Eine ungenutzte Seite ist gültig, sie wurde nur noch nicht abgebildet.



- die Abbildungsfunktion nutzt eine einstufige **Segmenttabelle**:

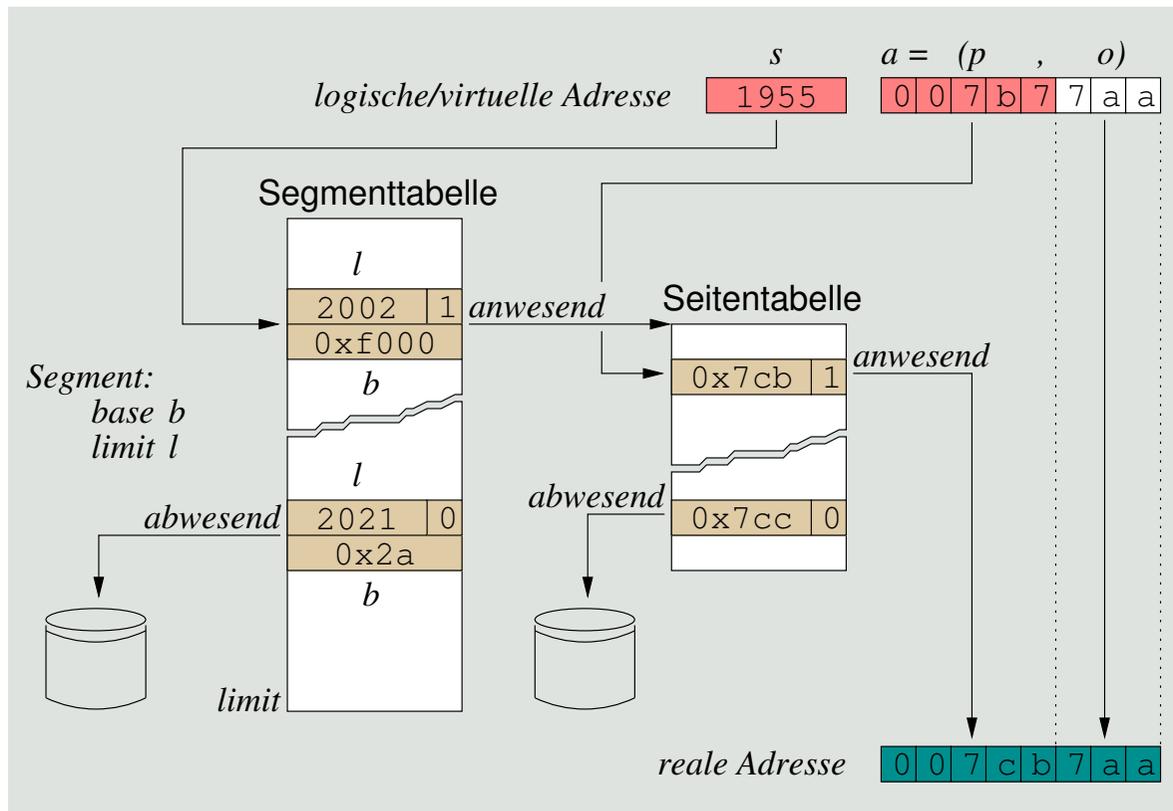


- s ist **Indexwert**, der gültig ist im Bereich $[0, S_{limit}]$
 - sei $S = 2^m$ max. Segmentanzahl
 - dann gilt $0 < S_{limit} \leq S - 1$
- ein möglicher **Indexfehler** muss erkannt werden
 - Grenzwertprüfung auf Basis eines *limit*-Registers (MMU) ist üblich
 - die Tabelle wird (norm.) nicht mit ungültigen Einträgen aufgefüllt
- S hängt ab von der MMU und dem Betriebssystem

- ein **Segmentfehler** (*segment fault*) bedeutet dann verschiedenerlei:
 - gültig falls $0 \leq s \leq S_{limit}$ und $a \leq l$, mit $l = \text{Segmenttabelle}[s].limit$, dann ist s abwesend (*swap-in*) oder ungenutzt (dynamisches Binden)
 - sonst ungültig: Segment s oder Adresse a ist für den Prozess undefiniert



- die Abbildungsfunktion nutzt beide zuvor dargestellten Techniken



- Merkmale:
 - die Seitentabelle ist ein Segment
 - all ihre Einträge sind gültig
- Unterschiede:
 - keine Relokation der Segmente
 - Segmentbasis ist direkte Adresse der Seitentabelle
 - nur der p -Anteil von a ist limitiert

- Segment- und Seitenfehler sind möglich, mit folgender Bedeutung:
 - gültig falls $0 \leq s \leq S_{limit}$ und $p \leq l$, mit $l = Segmenttabelle[s].limit$, dann ist s oder p bzw. beide sind abwesend oder ungenutzt
 - ungültig sonst: Segment s oder Seite p ist für den Prozess undefiniert



Seitennummerierte Segmentierung

- kombiniert die **Vorteile** von Seitennummerierung und Segmentierung
 - einfache Platzierungsstrategie, da die Speicherzuteilung kachelorientiert und damit immer in Einheiten gleicher Größe geschieht (vgl. [4, S. 18])
 - mehrstufige Seitentabellen fallen weg, da alle Tabellen Segmente und so jeweils in ihrer wirklichen Seitenanzahl beschränkt sind
 - bessere Trennung von Belangen, da Segmente und Seiten bzw. Kacheln verschiedenen Zielen dienen

Segment – Abbildung und Erfassung von **Programmstrukturen**
Seite – Optimierung von **Systemfunktionen** der Speicherverwaltung
- seitennummerierte Adressräume zu „segmentieren“, ist etwas anderes
 - also Seiten zu Text-, Daten- oder Stapelsegmenten zusammenstellen⁴
 - Programmstrukturen lassen sich damit im System nicht wirklich abbilden
 - vom Verwaltungsaufwand mehrstufiger Seitentabellen einmal ganz abgesehen
- Segmentierung unterstützt zudem **dynamisches Binden** ungemein
 - die „Bindlinge“ sind symbolisch bezeichnete, **physische Segmente**
 - d.h., Programmstrukturen, Adressräume (Seitentabellen), . . . , Dateien

⁴So, wie es von UNIX-ähnlichen Betriebssystemen (inkl. Linux) bekannt und überhaupt nach Multics [6] eben nur noch gang und gäbe ist.



- gemeint sind Namen von **Datei(standort)en** im **Ablagespeicher** aber auch zur Bezeichnung lokaler **Betriebsmittel** oder Systemstrukturen
 - ursprünglicher Bezugspunkt war die Datei (*file*), d.h., eine abgeschlossene Einheit zusammenhängender Daten
 - dieses **speicherzentrische Betriebsmittel** ist aber nur das Beispiel einer einzelnen Bestandsart, andere sind etwa:⁵
 - Kommunikationsmittel** – Kanal (*pipe*), Sockel (*socket*), Briefkasten
 - Gerät** – zeichen-, block-, stromorientiert
 - Zustandsdaten** – Prozesstabelle, Adressraumbellegung, ...
 - allgemein sind so einige anwendungsrelevante und durch Betriebssysteme bereitgestellte (Exemplare von) **Typen** namentlich zugänglich
- wichtiger Aspekt in dem Zusammenhang ist das **Verzeichnis** solcher Namen und die zugrunde gelegte Organisationsform
 - Struktur eines Namensraums in lokaler und globaler Hinsicht
 - Art der Verknüpfung zwischen Namen und dem benannten Ding
- ist **Persistenz** von Namen, Verzeichnissen und Abbildungen verlangt, bietet ein **Dateisystem** eine adäquate Implementierungsgrundlage

⁵Die Aufstellung ist abhängig vom Betriebssystem.



Benennung

<https://de.wikipedia.org/wiki/Benennung>

Bezeichnung eines Gegenstands durch ein Wort oder mehrere Wörter.

- gedeutet in Bezug auf grundlegende Betriebssystemkonzepte, die mit Multics [6] eingeführt wurden:
 - ein Wort ■ der **Name** relativ zu einem bestimmten Kontext
 - lokal (in seinem Kontext) eindeutig, global mehrdeutig
 - mehrere Wörter ■ der **Pfadname** im Namensraum zum benannten Ding
 - global eindeutige Bezeichnung des Namenskontextes
- die **Mehrwortbenennung** sieht einen „Trenntext“ als **Separator** vor, den die Namensverwaltung im Betriebssystem definiert, z.B.:
 - > ■ Multics (*greater-than*)
 - / ■ UNIX (*slash*)
 - \ ■ Windows (*backslash*)
- wohingegen jedoch die Namen selbst für die Namensverwaltung ohne Bedeutung sind und i.A. von ihr nicht interpretiert werden⁶

⁶ „Name ist Schall und Rauch, Umnebelnd Himmelsglut.“ (Goethe: Faust)



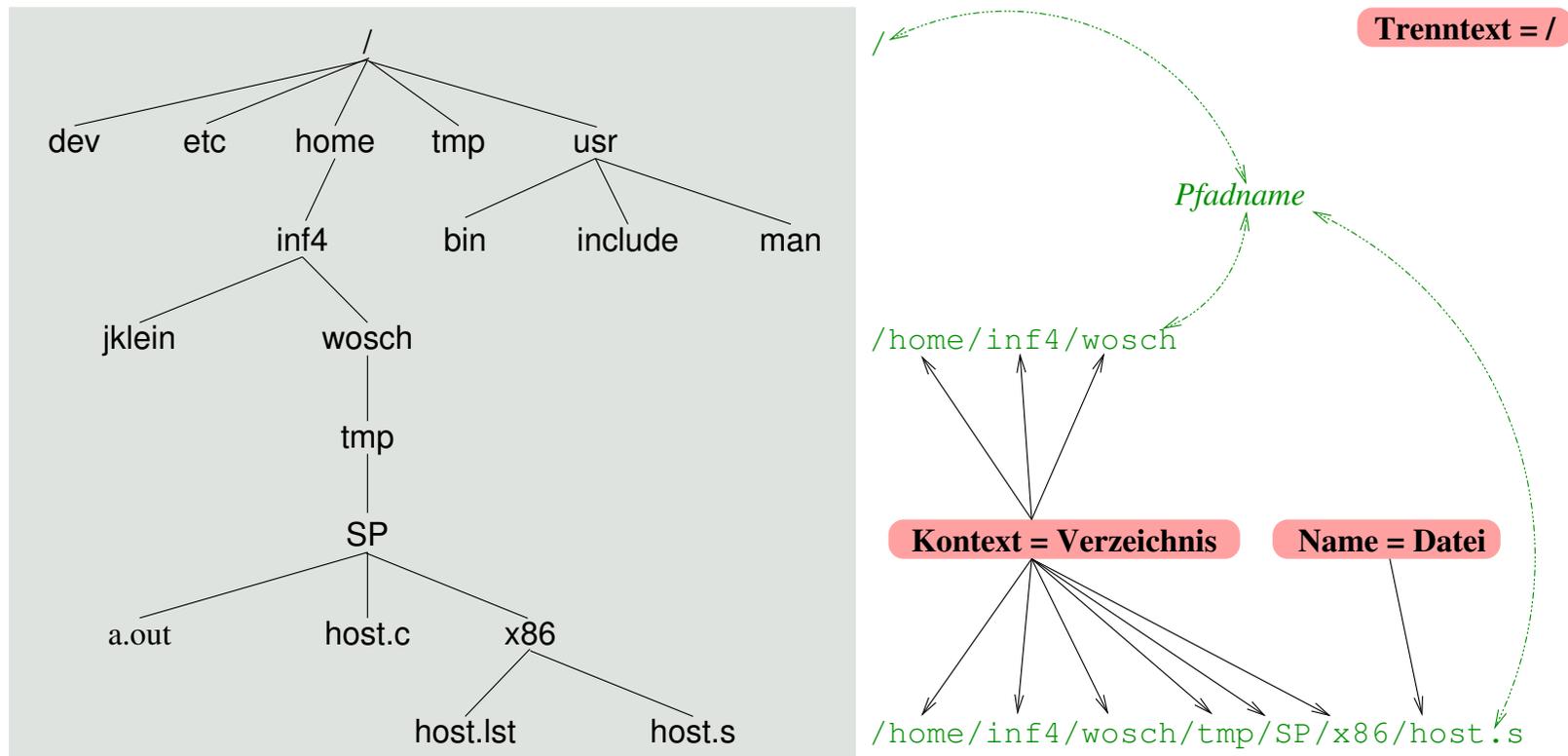
Alphabet und Wortlänge

Jedes Programm (inkl. das Betriebssystem) kann eigenen Ressourcen Zeichenketten zuordnen und Prozessen diese bekanntgeben.

- den **Zeichenvorrat** für Namen und Komponenten eines Pfadnamens gibt die Namensverwaltung des Betriebssystems vor
 - allgemein die Menge der druckbaren Zeichen (ASCII) ohne Trennzeichen⁷
 - je nach Betriebssystem gibt es weitere Ausnahmen (Windows = {", *, /, ?, |})
- genau genommen werden Bezeichnungen aber aus **Ordnungszahlen** gebildet, die ein **Zeichensatz** erst in Zeichen umwandelt
 - z.B. UTF-8 (Linux), latin-1 (MacOSX) oder CP 437 (DOS)
 - nicht jede Ordnungszahl entspricht somit zwingend demselben Zeichen !
- ähnlich uneinheitlich ist die erlaubte Länge einer **Zeichenkette**, um Namen oder Pfadnamen zu formulieren
 - 1–255 Zeichen pro Name, bei UNIX auch pro Pfadnamenskomponekte
 - bis zu $2^{15} - 1$ Zeichen pro Pfadname in Windows

⁷Nach ISO 9660: ausschließlich Großbuchstaben, Ziffern und Unterstrich.





- ein **Kontext** repräsentiert einen Namensraum flacher Struktur
 - darin muss Eindeutigkeit mit der Namenswahl selbst gewährleistet sein
- im Gegensatz zu einem Namensraum hierarchischer Struktur (s. o.)
 - derselbe Name (`tmp`) kann in verschiedenen Kontexten definiert sein
 - durch den Pfadnamen wird sein jeweiliger Standort eindeutig gemacht



- das **Verzeichnis** (*directory*) ist fundamental für die Hierarchiebildung
 - es ordnet einen/mehrere Namen, auch Verzeichnisnamen, listenförmig an
 - ist namentlich selbst in einem **Elterverzeichnis** (*parent directory*) gelistet
 - es gibt mehreren Namen ein gemeinsames Merkmal, denselben Kontext
 - d.h., denselben Standort/Bezugspunkt innerhalb des Namensraums
 - es dient der Umsetzung von symbolischen in numerischen Adressen

Verzeichniseintrag (*directory entry*)

Speichert die Abbildung eines Namens auf eine Informationsstruktur.

- UNIX-artige Betriebssysteme bieten zudem vordefinierte Kontexte:
 - **Wurzelverzeichnis** (*root directory*) des Systems
 - Einstiegspunkt in, aber auch „Steckverbinder“⁸ für, den Namensraum
 - **Heimatverzeichnis** (*home directory*) eines autorisierten Benutzers
 - initiales Arbeitsverzeichnisses nach erfolgter Anmeldung (*login*)
 - **Arbeitsverzeichnis** (*working directory*) eines zugelassenen Prozesses
 - gegenwärtiger, relativer Standort des Prozesses im Namensraum

⁸Womit der Namensraum in einen bestehenden anderen Namensraum an einem Befestigungspunkt (*mount point*) gegebenenfalls eingebunden werden kann.

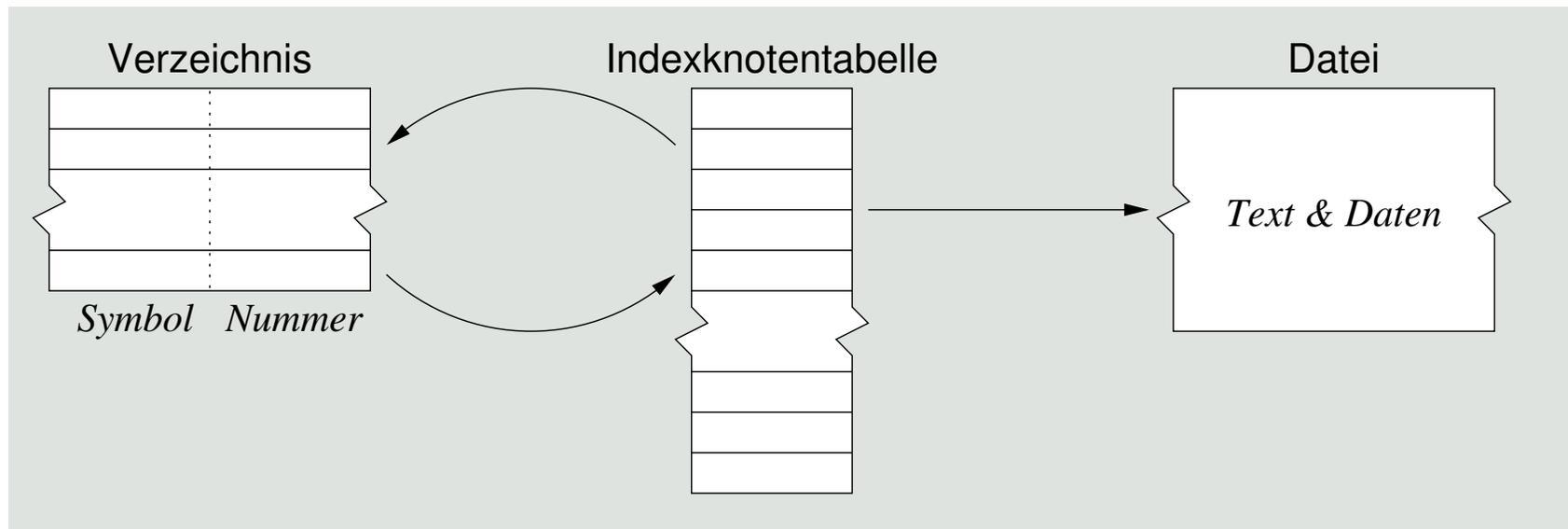


- aggregiert wesentliche **Attribute** eines benannten Gegenstands:
 - Eigentümer (*user ID*)
 - Gruppenzugehörigkeit (*group ID*)
 - Rechte (lesen, schreiben, ausführen: für Eigentümer, Gruppe und Welt)
 - Zeitstempel (letzter Zugriff, letzte Änderung (Typ, Zugriffsrechte))
 - Anzahl der Verweise („*hard link*“-Zähler)
 - Typ:
 - Verzeichnis
 - symbolische Verknüpfung (*symbolic link*)
 - Kommunikationskanal (*pipe*, *named pipe*)
 - Sockel (*socket*) zur Interprozesskommunikation
 - Gerätedatei \rightsquigarrow zeichen-/blockorientiertes Gerät, Pseudogerät, Treiberklasse
 - reguläre Datei \rightsquigarrow Größe in Bytes und Blocknummer(n) im Ablagespeicher
- besitzt in einem Namensraum eine eindeutige **numerische Adresse**

Indexknotennummer (*inode number*)

Hat mit einer logischen Adresse gemeinsam, dass sie nur innerhalb ihres „Adressraums“ (d.h. Namensraums) eindeutig ist.





- die **Indexknotentabelle** (*inode table*) ist ein statisches Feld (*array*) von Indexknoten und die zentrale Datenstruktur
 - ein Indexknoten ist **Deskriptor** insb. eines Verzeichnisses oder einer Datei
- das **Verzeichnis** (*directory*) ist eine **Abbildungstabelle**, es übersetzt symbolisch repräsentierte Namen in Indexknotennummern
 - eine von der Namensverwaltung des Betriebssystems definierte Datei
- die **Datei** (*file*) ist eine abgeschlossene Einheit zusammenhängender Daten beliebiger Repräsentation, Struktur und Bedeutung

⁹Als Einheit auf demselben Medium (z.B. Ablagespeicher) abgelegt.



- die **feste Verknüpfung** (*hard link*) von einem Dateinamen mit einer Indexknotennummer (UNIX V7, `dir.h`):

```
1 typedef unsigned short ino_t;
2
3 #define DIRSIZ 14
4
5 struct direct {
6     ino_t d_ino;
7     char d_name[DIRSIZ];
8 };
```

- eine als **Wertepaar** gespeicherte **surjektive Abbildung**
- mehrere Paare können zum selben Indexknoten (`d_ino`) zeigen
- im selben Verzeichnis jedoch mit verschiedenen Namen (`d_name`)

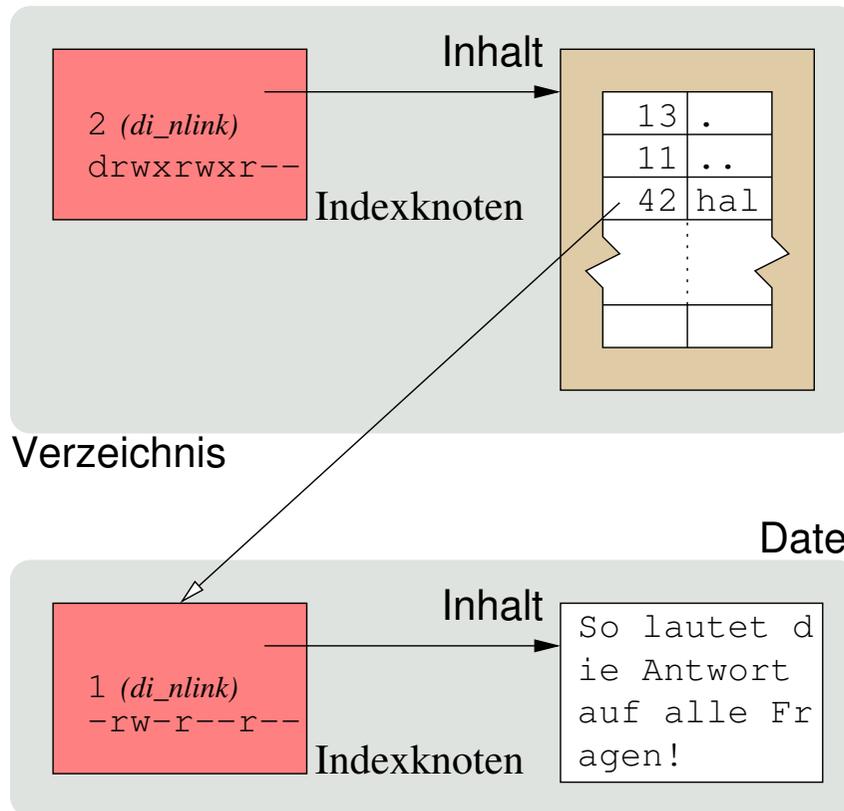
- in einem Indexknoten ist die Anzahl der auf ihn verweisenden Wertepaare desselben Namensraums gespeichert (*reference counter*)
- alle Indexknoten eines Namensraums sind in einer **Indexknotentabelle** (*inode table*) im Namensraum daselbst gespeichert
 - `d_ino` ist der **Indexwert** eines Verzeichniseintrags für diese Tabelle

- Anlegen/Löschen erfordert **Schreibzugriffrecht** auf das Verzeichnis
 - unabhängig von den Zugriffsrechten auf die referenzierte Datei



Verzeichniseintrag II

- ein Namenverzeichnis ist eine **spezielle Datei** der Namensverwaltung



- das selbst einen Namen hat, der einen Indexknoten bezeichnet
- über eine Verknüpfung erreichbar ist aus einem anderen Verzeichnis
- Namen getrennt von eventuellen Dateiinhalten speichert

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

- Selbstreferenz („dot“, 13) und Elterverzeichnis („dot dot“, 11) geben wenigstens zwei Verweise auf ein Verzeichnis
 - auch wenn das Verzeichnis selbst sonst keine weiteren Namen enthält



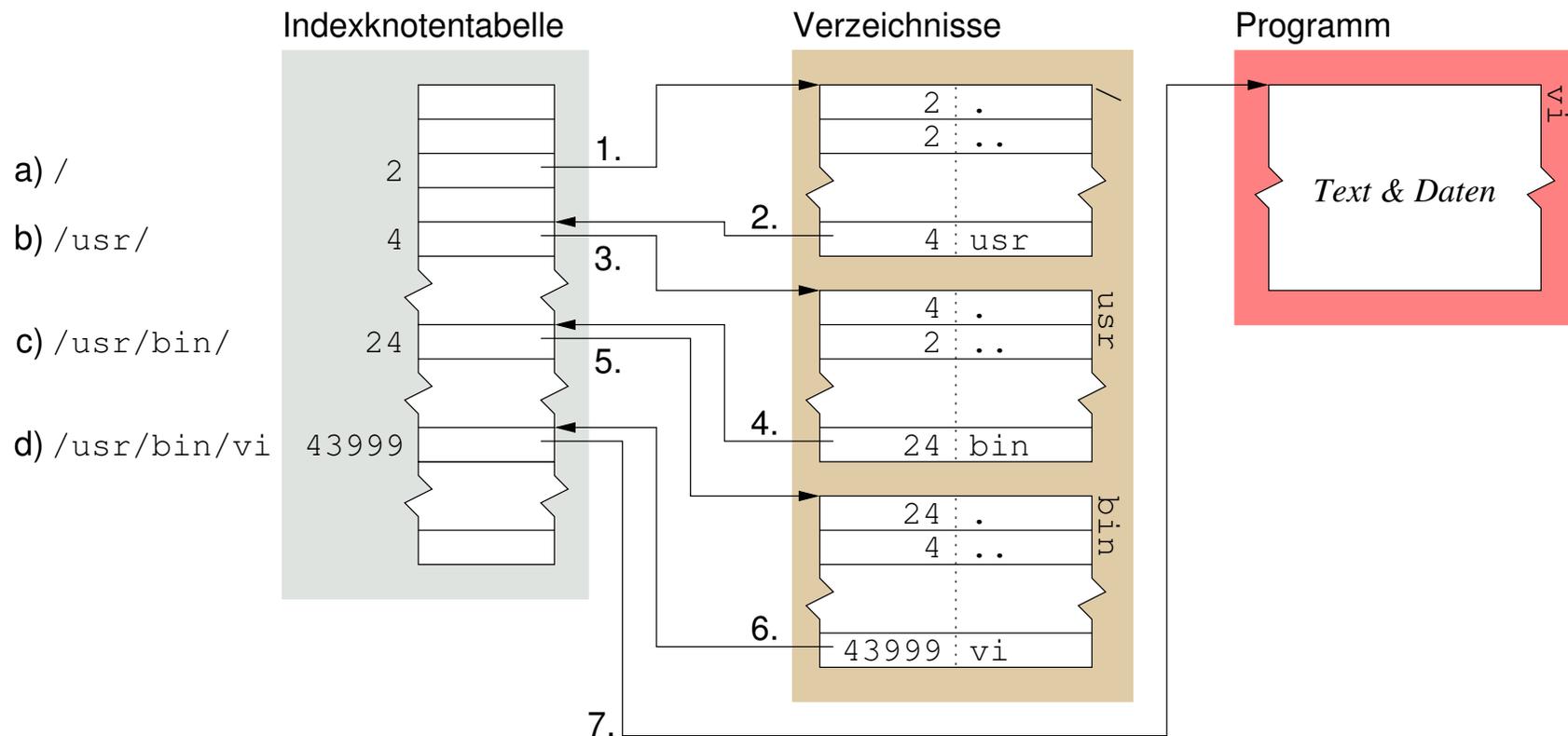
- **Namensbindung** (*name binding*) kommt zuerst, also die **Abbildung** der symbolischen Adresse in eine numerische Adresse
 - geschieht zum **Erzeugungszeitpunkt** eines Datei-/Verzeichnisnamens
 - i diesen mit einem freien/belegten Indexknoten verknüpfen und
 - ii dann in ein Namensverzeichnis eintragen
 - einen Pfadnamen mit einem Indexknoten assoziieren: `creat(2)`, `link(2)`
- **Namensauflösung** (*name resolution*) kommt später, die **Umsetzung** der symbolischen Adresse in eine numerische Adresse
 - geschieht zum **Benutzungszeitpunkt** eines Datei-/Verzeichnisnamens
 - i Verzeichnisse für jeden einzelnen Namen im Pfad durchsuchen und
 - ii schließlich den Dateinamen (Blatt) auffinden
 - einen Indexknoten anhand eines Pfadnamens lokalisieren: `open(2)`

Hinweis

*Der Indexknoten besitzt eine Adresse (a) im Ablagespeicher und (b) im Arbeitsspeicher. Diese kann sehr wohl als eine **virtuelle Adresse** aufgefasst werden, über die Inhalte von Dateien speicherabgebildet im virtuellen Adressraum indirekt zugänglich werden.*



Auflösung am Beispiel von /usr/bin/vi



- Wurzelverzeichnis des Namensraums öffnen
- darin Namenseintrag `usr` suchen, ist ein Verzeichnis, öffnen
- darin Namenseintrag `bin` suchen, ist ein Verzeichnis, öffnen
- darin Namenseintrag `vi` suchen, ist ein Programm, öffnen



- Indexknotennummern sind nur innerhalb ihres Bezugssystems gültig
 - sie teilen sich damit dieselbe Eigenschaft wie logische/virtuelle Adressen
 - sie sind ein **Tabellenindex**, wie der p - bzw. s -Anteil dieser Adressen
 - der Zugriff auf einen anderen Namensraum ist darüber nicht möglich
- der hierarchische Namensraum ist ein **gerichteter azyklischer Graph**
 - feste Verknüpfungen zu Verzeichnissen zerstören die azyklische Struktur:
 - i das Elterverzeichnis (..) eines Verzeichnisses wäre dann uneindeutig
 - ii die Namensraumabsuche (*name space search*) könnte „endlos schleifen“
 - um den **Wurzelbaum** zu erhalten, scheiden solche Verknüpfungen aus
- Abbildung $f : N_{symbolisch}^d \mapsto N_{symbolisch}^z$ hat diese Probleme nicht
 - feste Verknüpfungen sind ununterscheidbar, symbolische nicht: **Dateityp**
 - symbolische Verknüpfungen haben Indexknoten, feste nicht — gelten als:
 - langsam** – weil indirekt gespeichert, in den Datenblöcken als reguläre Datei
 - schnell** – weil direkt gespeichert, im Indexknoten selbst¹⁰
- Ursprung ist der **symbolische Name** (*symbolic name*) in Multics [2]
 - zur dynamischen Bindung von Namen an (besondere) E/A-Geräte

¹⁰Aber nur schnell bei einem Verweis auf einen Eintrag im selben Verzeichnis!



Gegenüberstellung

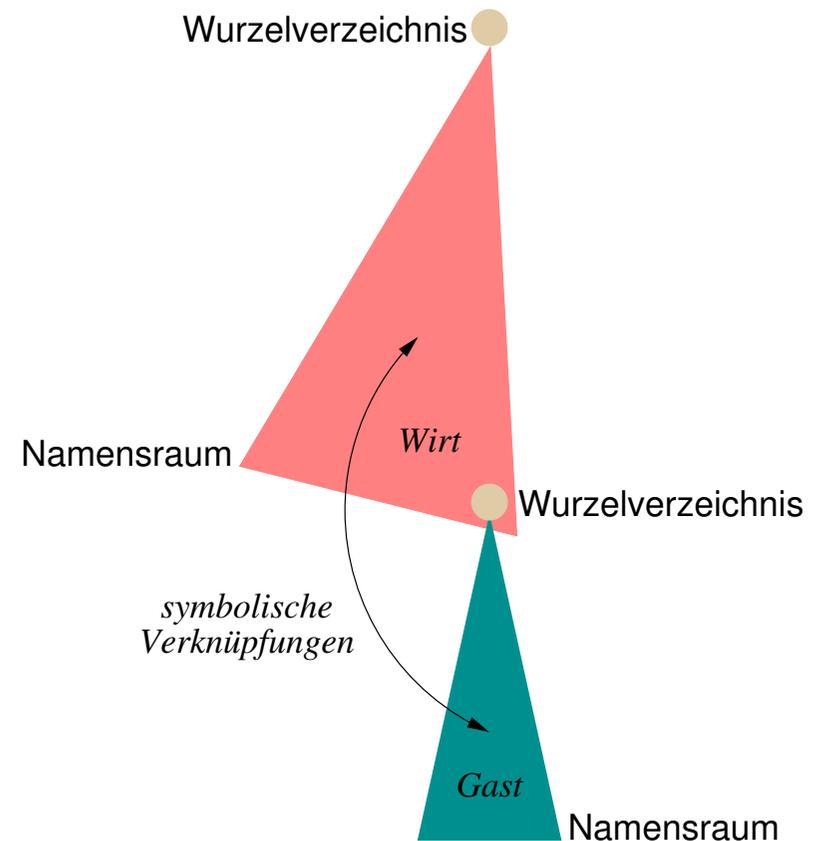
- feste Verknüpfung
 - nicht zu Verzeichnissen oder Dateien anderer Dateisysteme
 - überdauert die Umplatzierung einer Datei
 - bleibt bestehen, nur solange es noch Referenzen gibt
 - hat keinen eigenen Indexknoten
- symbolische Verknüpfung
 - auch zu Verzeichnissen und Dateien anderer Dateisysteme
 - ungültig nach Umplatzierung einer Datei
 - bleibt bestehen, auch wenn es keine Referenzen gibt
 - hat einen eigenen Indexknoten

„Nicht alles, was glänzt, ist Gold“ (Shakespeare, 1600)

```
wosch@lorien 1$ mkdir -p Laptop/faii43w; cd Laptop; ln -s faii43w lorien; ls -l
total 8
drwxr-xr-x  2 wosch  wosch  68 29 Apr 13:01 faii43w
lrwxr-xr-x  1 wosch  wosch   7 29 Apr 13:02 lorien -> faii43w
wosch@lorien 2$ cd lorien
wosch@lorien 3$ cd ..; rmdir faii43w; cd lorien
-bash: cd: lorien: No such file or directory
wosch@lorien 4$ ls -l
total 8
lrwxr-xr-x  1 wosch  wosch   7 29 Apr 13:02 lorien -> faii43w
wosch@lorien 5$ mkdir faii43w; cd lorien
wosch@lorien 6$ ln -s Fata\ Morgana SP1
```



- Namensräume können an einem **Befestigungspunkt** (*mount point*) miteinander verbunden werden
 - ohne damit jedoch eine Erweiterung des Namensraums vorzunehmen
- der Punkt ist ein **Verzeichnis** im Wirtsnamensraum
 - Einhängen (*mount*) blendet den Inhalt des Wirtsverzeichnisses aus
 - der Wurzelverzeichnisinhalt des Gastnamensraums erscheint
 - Aushängen (*unmount*) macht den alten Verzeichnisinhalt sichtbar
- ein **Pfadname** kann dann Wirts- und Gastnamensraum abdecken
 - einerseits streng hierarchietreu, von oben nach unten (*top down*)
 - andererseits quer verweisend, durch **symbolische Verknüpfung**



Definition (Nummerung (DIN 6763))

Bilden, Erteilen, Verwalten und Anwenden von Nummern.

- die **Eindeutigkeit** der Speicher- und Standortadressen ist begrenzt
 - Indexknotennummern durch den **Namensraum** ihres Dateisystems
 - reale, logische und virtuelle Adressen durch ihren (Prozess-) **Adressraum**
 - Prozesskennungen durch den **Nummernraum** ihres Rechensystem
- die **Internetprotokolladresse** (IP-Adresse) ist weltweit eindeutig
 - IPv4** ■ 32 Bit: vier Blöcke zu jeweils drei Dezimalstellen (8 Bit)
 - IPv6** ■ 128 Bit: acht Blöcke zu jeweils vier Hexadezimalstellen (16 Bit)
 - vom ARP (*address resolution protocol*) aufgelöst und umgewandelt in die **Netzwerkadapteradresse** (MAC, *media access control*)
 - rechnerlokal wird das Adressenpaar in der ARP-Tabelle hinterlegt
- verallgemeinerte Form ist der/die **URL** (*universal resource locator*)
 - neben Adressinformationen ist zusätzlich die **Zugriffsmethode** enthalten, die durch `://` von den schemaspezifischen Angaben getrennt ist
 - die Internetadresse identifiziert dabei den **Wirt** (*host*) einer Webanfrage



Gliederung

Einführung

Semiotik

Informatik

Dynamische Namensauflösung

Virtuelle Adressen

Pfadnamen

Internetadressen

Statische Namensauflösung

Übersetzer

Binder

Lader

Zusammenfassung



- am Anfang steht jedoch die **symbolische Adresse** in Programmen, die in einem mehrstufigen Verfahren aufzulösen ist

```
1 int main(int argc, char **argv) {
2     if (argc == 2) {
3         struct hostent *host = gethostbyname(argv[1]);
4         if (host != NULL) {
5             unsigned int i = 0;
6             printf("%s = ", host->h_name);
7             while (host->h_addr_list[i] != NULL)
8                 printf("%s ", inet_ntoa(*(struct in_addr*)(host->h_addr_list[i++])));
9             printf("\n");
10        }
11    }
12 }
```

Kompilierer ■ verteilt Programmtext und -daten auf Programmsegmente

■ generiert dazu Pseudobefehle, die der Assembler deutet

Assembler ■ ordnet Programmsymbolen Werte und Attribute zu

■ generiert Symbol- und Relokationstabellen für den Binder

Binder ■ platziert das gebundene Programm im Adressraum

■ produziert das Lademodul dazu für das Betriebssystem

- zusätzlich erfolgt die Generierung des Maschinencodes, zwei Schritte:

i Kompilierung des Quellcodes in eine andere symbolische Darstellung

ii Assemblierung und Bindung in die benötigte numerische Auslegung



- Programmsymbole auf **Binderabschnitte** (*linker sections*) verteilen

```
1      .file    "host.c"
2      .section          .rodata.str1.1,"aMS",@progbits,1
3
4      .LC0:
5      .string "%s = "
6
7      .LC1:
8      .string "%s "
9      .section          .text.startup,"ax",@progbits
10     .p2align 4,,15
11     .globl  main
12     .type   main, @function
13
14 main:
```

2 ■ beleg- und nur lesbares ("aMS") **Datensegment** (2–6)

■ Symbole .LC0 und .LC1 sind nur lokal definiert

7 ■ beleg- und ausführbares ("ax") **Textsegment** (7–83, vgl. S. 34)

8 ■ Wert der Adresse des Abschnitts soll Vielfaches von 16 sein

9–10 ■ Symbol main ist global definiert, eine Funktion

- die Symbole werden bekannt gemacht und mit Attributen verknüpft

- Text, Daten, Sichtbarkeit, Typ, Benutzungsart

- Adresswerte in Bezug auf den Prozessadressraum stehen noch aus

- konkrete Werte für die Symbole und die einzelnen Maschinenbefehle



■ Abschluss der **Übersetzungseinheit** (*compilation unit*)

```
81 .LFE19:  
82     .size    main,  .-main  
83     .ident   "GCC: (Debian 4.7.2-5) 4.7.2"  
84     .section .note.GNU-stack,"",@progbits
```

82 ■ dem Symbol `main` die Länge des Programmtextes zuweisen

84 ■ das Programmteil benötigt kein ausführbares ("") **Stapelsegment**

■ Ergebnis der Assemblierung ist das Binde- oder **Objektmodul**, das u.a. zwei (vom Assemblierer zusammengestellte) Tabellen enthält

Symboltabelle ■ listet alle in dem Modul definierten Symbole und
■ assoziiert jedes Symbol mit Werten und Attributen

Relokationstabelle ■ listet alle „undefinierten Stellen“ in dem Modul
■ Stellen, an denen definierte Adresswerte noch fehlen

■ jedes Text- und Datenelement besitzt eine **vorläufige Adresse**, die relativ zur Basisadresse 0 ausgelegt/-gerichtet ist

- Stellen mit undefinierten Adressen korrigiert der **Binder** oder der **Lader**
 - d.h., es erfolgt die **Relokation** einer an dieser Stelle liegenden Referenz
- Korrekturmaß ist die Ladeadresse des gebundenen Programms
 - die **Relokationskonstante**, gemäß Adressraummodell des Betriebssystems



- **Auflistung** (*listing*) der Übersetzungseinheit nach der Assemblierung (Ausschnitt des Quellprogramms von S. 32, dort Zeilen 3–6)

```
1 26 0009 488B7E08    movq  8(%rsi), %rdi
2 27 000d E8000000    call  gethostbyname
3 27          00
4 28 0012 4885C0      testq %rax, %rax
5 29 0015 4889C5      movq  %rax, %rbp
6 30 0018 745E        je    .L8
7 31 001a 488B30      movq  (%rax), %rsi
8 32 001d BF000000      movl  $.LC0, %edi
9 32          00
10 33 0022 31C0        xorl  %eax, %eax
11 34 0024 31DB        xorl  %ebx, %ebx
12 35 0026 E8000000      call  printf
13 35          00
```

- typischerweise vier Bereiche:

1. Zeilennummer des Befehls, hier: 26–35
2. relative Adresse im Programm, hier: 0009–0026 (Hexadezimal)
3. numerischer Maschinenkode, hier: 32 Bit (4 Bytes) pro Zeile
4. Mnemonik und Operand(en)

- an folgenden Stellen/relativen Adressen ist **Relokation** erforderlich:

- 2/000e ■ Adresse für gethostbyname, externe Referenz (libc)
- 8/001e ■ Adresse für .LC0, interne Referenz (vgl. S. 33)
- 12/0027 ■ Adresse für printf, externe Referenz (libc)

- korrekte Programmausführung benötigt Korrektur an diesen Stellen

- dort stehende **absolute Adressen** müssen zum Adressraummodell passen



- derselbe Ausschnitt wie zuvor, jedoch dem **Lademodul** entnommen

```
1 0x00000000004003e9 <+9>:    mov     0x8(%rsi),%rdi
2 0x00000000004003ed <+13>:   callq  0x415e60 <gethostbyname>
3 0x00000000004003f2 <+18>:   test   %rax,%rax
4 0x00000000004003f5 <+21>:   mov    %rax,%rbp
5 0x00000000004003f8 <+24>:   je     0x400458 <main+120>
6 0x00000000004003fa <+26>:   mov    (%rax),%rsi
7 0x00000000004003fd <+29>:   mov    $0x48d1a4,%edi
8 0x0000000000400402 <+34>:   xor    %eax,%eax
9 0x0000000000400404 <+36>:   xor    %ebx,%ebx
10 0x0000000000400406 <+38>:   callq  0x401120 <printf>
```

- vorher noch **unaufgelöste Referenzen** haben definierte Werte erhalten:

0x415e60 \rightsquigarrow gethostbyname

0x48d1a4 \rightsquigarrow .LC0

0x401120 \rightsquigarrow printf

- allen Text- und Datenelementen wurden **absolute Adressen** zugewiesen

- der Programmausschnitt liegt im Adressbereich $[4003e9_{16}, 400406_{16}]$

- bei Ausführung nimmt der Befehlszähler u.a. Werte aus diesem Bereich an

- die Adresswerte allein sagen nichts zur Art des Adressraums aus

- es könnte ein realer, logischer oder virtueller Adressraum sein

- Wissen über das **Adressraummodell** des Betriebssystems gibt Aufschluss



■ Adressraumbelegung für dieses „Programm in Ausführung“

```
1  wosch@fau48e 111$ ./a.out faui40.cs.fau.de &
2  [1] 20857
3  faui40.informatik.uni-erlangen.de = 131.188.34.40
4  wosch@fau48e 112$ cat /proc/20857/maps
5  00400000-00401000 r-xp 00000000 00:2a 38020125          /home/inf4/wosch/tmp/a.out
6  00600000-00601000 rw-p 00000000 00:2a 38020125          /home/inf4/wosch/tmp/a.out
7  01ff9000-0201a000 rw-p 00000000 00:00 0              [heap]
8  7f03d63ad000-7f03d6f82000 rw-p 00000000 00:00 0          shared libraries
9  7ffc79532000-7ffc79553000 rw-p 00000000 00:00 0          [stack]
10 7ffc795b8000-7ffc795b9000 r-xp 00000000 00:00 0          [vdso]
11 ffffffff600000-fffffff601000 r-xp 00000000 00:00 0 [vsyscall]
```

- 5 ■ Textsegment, effektiv 4 KiB (eine Seite), insgesamt 2 MiB
- 6 ■ Datensegment, effektiv 4 KiB (eine Seite), insgesamt ≈ 26 MiB
- 7 ■ Haldenspeicher, vorgegeben 132 KiB, erweiterbar
- 9 ■ Stapelspeicher, vorgegeben 132 KiB, erweiterbar

■ die Größenangaben variieren mit dem Programm und dem Prozess !

■ für einen Prozess adressier- aber nicht belegbar sind die Bereiche:

- 8 ■ Gemeinschaftsbibliotheken (*shared libraries*)
- 10/11 ■ Systemaufrufbeschleunigung (vgl. auch [3, S. 21–22])

■ **Ladeadresse** — und somit **Relokationskonstante** — ist $0x400000$

■ der Bereich $[0, 3ffff_{16}]$ ist nicht Teil des Prozessadressraums !



Gliederung

Einführung

Semiotik

Informatik

Dynamische Namensauflösung

Virtuelle Adressen

Pfadnamen

Internetadressen

Statische Namensauflösung

Übersetzer

Binder

Lader

Zusammenfassung



- **symbolische Bezeichnungen** abstrahieren von konkreten Adressen
 - Programmtext- und -datenstellen, Datei-/Pfadnamen, ..., URL
 - üblich ist die mehrstufige statische und dynamische Auflösung
- den Schwerpunkt bildete die **dynamische Namensauflösung**
 - eine (reale, logische, virtuelle) Adresse ist Name einer Speicherstelle
 - je nach Abstraktionsebene der Haupt- oder Arbeitsspeicher
 - Seitennummerierung, Segmentierung, seitennummerierte Segmentierung
 - Datei-/Pfadnamen repräsentieren Adressen im Ablagespeicher
 - der Namens-/Adressraum ist (nicht nur hier) hierarchisch organisiert
 - Orientierung am Dateisystem: Datei, Verzeichnis, Indexknoten, Verknüpfung
 - systemglobale, weltweite Eindeutigkeit liefert die Internetadresse
- Vorarbeit zu all dem leistet die **statische Namensauflösung**:
 - Kompilierer** ■ verteilt Programmtext und -daten auf Programmsegmente
 - Assembler** ■ ordnet Programmsymbolen Werte und Attribute zu
 - Binder** ■ platziert das gebundene Programm im Adressraum
 - Lader** ■ legt den „gefüllten“ Adressraum im Arbeitsspeicher ab
- das Zusammenspiel beider Verfahren ist typisch in Rechensystemen
 - nämlich Abbildungsfunktionen, die vor und zur Laufzeit greifen



Literaturverzeichnis

- [1] DENNING, P. J.:
Virtual Memory.
In: *Computing Surveys* 2 (1970), Sept., Nr. 3, S. 153–189

- [2] FEIERTAG, R. J. ; ORGANICK, E. I.:
The Multics Input/Output System.
In: *Proceedings of the Third ACM Symposium on Operating System Principles (SOSP 1971), October 18–20, 1971, Palo Alto, California, USA*, ACM, 1971, S. 35–41

- [3] KLEINÖDER, J. ; SCHRÖDER-PREIKSCHAT, W. :
Maschinenprogramme.
In: [5], Kapitel 5.2

- [4] KLEINÖDER, J. ; SCHRÖDER-PREIKSCHAT, W. :
Speicher.
In: [5], Kapitel 6.2

- [5] KLEINÖDER, J. ; SCHRÖDER-PREIKSCHAT, W. ; LEHRSTUHL INFORMATIK 4 (Hrsg.):
Systemprogrammierung.
FAU Erlangen-Nürnberg, 2015 (Vorlesungsfolien)

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

