



## Script generated by TTT

Title: Grundlagen\_Betriebssysteme (30.11.2015)

Date: Mon Nov 30 13:45:23 CET 2015

Duration: 88:07 min

Pages: 31

- Prof. J. Schlichter
  - Lehrstuhl für Angewandte Informatik / Kooperative Systeme, Fakultät für Informatik, TU München
  - Boltzmannstr. 3, 85748 Garching
  - Email: [schlichter@in.tum.de](mailto:schlichter@in.tum.de)
  - Tel.: 089-289 18654
  - URL: <http://www11.informatik.tu-muenchen.de/>

[Übersicht](#)

[Einführung](#)

[Parallele Systeme - Modellierung, Strukturen](#)

[Prozess- und Prozessorverwaltung](#)

[Speicherverwaltung](#)

[Prozesskommunikation](#)

[Dateisysteme](#)

[Ein-/Ausgabe](#)

[Sicherheit in Rechensystemen](#)

[Entwurf von Betriebssystemen](#)

[Zusammenfassung](#)

Generated by Targeteam



## Speicherverwaltung



### Fragestellungen

Dieser Abschnitt beschäftigt sich mit den Adressräumen für Programme und deren Abbildung auf den physischen Arbeitsspeicher einer Rechenanlage:

Programmadressraum vs. Maschinenadressraum.

Direkte Adressierung, Basisadressierung.

Virtualisierung des Speichers; virtuelle Adressierung, insbesondere Seitenadressierung.

### [Einführung](#)

### Speicherabbildungen

Dieser Abschnitt behandelt einige Mechanismen zur Abbildung von Programmadressen auf Maschinenadressen des Arbeitsspeichers.

[Direkte Adressierung](#)

[Basisadressierung](#)

[Seitenadressierung](#)

[Segment-Seitenadressierung](#)

[Speicherhierarchie / Caches](#)

Generated by Targeteam



## Einführung



Die unmittelbare Nutzung des physischen Adressraums (Arbeitsspeichers) bei der Anwendungsentwicklung ist nicht empfehlenswert. Probleme sind folgende:

- Kenntnisse über Struktur und Zusammensetzung des Arbeitsspeichers notwendig.
  - Kapazitätsengpässe bei der Arbeitsspeichergroße.
- ⇒ deshalb Programmerstellung unabhängig von realer Speichergröße und -eigenschaften.

[Adressräume](#)

[Organisation von Adressräumen](#)

[Fragmentierung](#)

[Forderungen an Adressraumrealisierung](#)

Generated by Targeteam



**Maschinenadressraum**

Arbeitsspeicher = Folge von fortlaufend nummerierten Bytes (ab 0). **Maschinenadresse** = Nummer des Bytes.

**Programmadressraum**

**Programmadressen** = Adressen im Programm (z.B. für Variable).

Die Menge der zulässigen Programmadressen ist der **Programmadressraum**. Dieser ist prozessspezifisch, d.h. Programmadressen haben nur Programm-lokale Bedeutung z.B. als Sprungziele.

**Speicherabbildung**

Abbildung der Programmadressen auf Maschinenadressen (durch CPU).

- direkte Adressierung,
- Basisadressierung,
- Seitenadressierung und
- Segment-Seitenadressierung.

Generated by Targeteam



Die unmittelbare Nutzung des physischen Adressraums (Arbeitsspeichers) bei der Anwendungsentwicklung ist nicht empfehlenswert. Probleme sind folgende:

- Kenntnisse über Struktur und Zusammensetzung des Arbeitsspeichers notwendig.
  - Kapazitätsengpässe bei der ArbeitsspeichergroÙe.
- ⇒ deshalb Programmerstellung unabhängig von realer Speichergröße und -eigenschaften.

Adressräume

Organisation von Adressräumen

Fragmentierung

Forderungen an Adressraumrealisierung

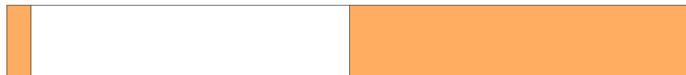
Generated by Targeteam



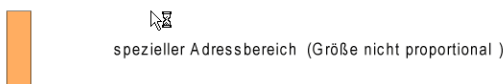
**32 Bit große Adressräume**

Programmcode, statische Daten, Halde und Laufzeitkeller jeweils in einem für die Anwendung zugänglichen Bereich des Adressraums

Windows 32 bit Adressierung



Linux 32 bit Adressierung



Generated by Targeteam



**Single-Threaded Adressraum**



niedrige Adresse

hohe Adresse

**Multi-Threaded Adressraum**



niedrige Adresse

hohe Adresse

Beispiel - Adressräume

Haldenverwaltung

Generated by Targeteam



Dynamisch erzeugte Daten, z.B. Listen und Objekte werden auf der Halde (heap) gespeichert. Die Halde wird nicht nach dem Kellerprinzip verwaltet.

Belegung Beispiel C:

```
void malloc(size_t size) belegt dynamisch einen Speicherbereich und gibt einen Pointer auf diesen Speicherbereich zurück.
```

Der Speicherbereich einer Halde besteht aus zwei Klassen:

**Belegtbereiche** : Speicherbereiche werden für Realisierungen von Datenobjekten verwendet.

**Freibereiche** : Speicherbereiche, die momentan nicht für Realisierungen von Datenobjekten verwendet werden; d.h. sie sind frei.

Buchführung des Speicherbereichs einer Halde mit Hilfe von Belegungs- und Freigabeoperationen.

### [Verwaltung der Freibereiche](#)

#### [Freiliste](#)

#### [Garbage Collection](#)

#### [Animation Haldenverwaltung](#)

Generated by Targeteam

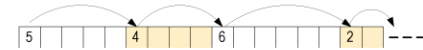


Speicherbereiche (belegt, frei) sind in **Blöcken** unterschiedlicher Länge organisiert. Verwaltung freier **Blöcke**

**implizite Liste** über Längen - verlinkt alle **Blöcke**

jeweils Feststellen, ob der **Block** belegt oder frei ist.

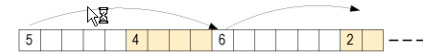
extra Bit im selben Wort wie **Blockgröße**.



Implementierung: sehr einfach; Allokierung in linearer Zeit.

**explizite Liste** (Freiliste) über die freien **Blöcke**

Allokierung: Durchsuchen der Freiliste nach geeignetem **Block**



Generated by Targeteam



Die Freibereiche der Halde werden mit Hilfe einer Liste (z.B. einfach verkettet) verwaltet.

```
public class freibereich {
    int size;
    freibereich next;
    <methodendefinitionen>
}
```

### [Auswahl eines geeigneten Freibereichs](#)

#### Fragmentierung

Problematisch ist die Entstehung von vielen kleinen freien Rest-Freibereichen, die wegen ihrer kleinen Längen als Belegtbereiche ungeeignet sind.

Generated by Targeteam



Bei Erzeugung eines neuen Datenobjektes wird die Operation `belege(int size)` ausgeführt. Für die Auswahl eines geeigneten Freibereichs zur Erfüllung der gestellten Anforderungen existieren verschiedene Verfahren.

#### first-fit-Verfahren

bestimme von Beginn der Freiliste den ersten Speicherbereich, der die Anforderung erfüllt.

#### next-fit-Verfahren

bestimme in der Freiliste den ersten Speicherbereich, der die Anforderung erfüllt. Die Suche wird dort fortgesetzt, wo die letzte Suche beendet wurde.

#### best-fit-Verfahren

bestimmt in der gesamten Freiliste den Speicherbereich, der am besten die gestellte Anforderung erfüllt, d.h. mit möglichst wenig Verschnitt.

#### worst-fit-Verfahren

bestimme in der Freiliste den größten freien Speicherbereich und teile ihn in einen Belegtbereich (zur Erfüllung der Anforderung) und einen verbleibenden Freibereich auf.

Generated by Targeteam



## Freiliste



Die Freibereiche der Halde werden mit Hilfe einer Liste (z.B. einfach verkettet) verwaltet.

```

public class freibereich {
    int size;
    freibereich next;
    <methodendefinitionen>
}

```

### Auswahl eines geeigneten Freibereichs

#### Fragmentierung

Problematisch ist die Entstehung von vielen kleinen freien Rest-Freibereichen, die wegen ihrer kleinen Längen als Belegbereiche ungeeignet sind.

Generated by Targeteam



## Haldenverwaltung



Dynamisch erzeugte Daten, z.B. Listen und Objekte werden auf der Halde (heap) gespeichert. Die Halde wird nicht nach dem Kellerprinzip verwaltet.

Belegung Beispiel C:

`void malloc(size_t size)` belegt dynamisch einen Speicherbereich und gibt einen Pointer auf diesen Speicherbereich zurück.

Der Speicherbereich einer Halde besteht aus zwei Klassen:

**Belegbereiche** : Speicherbereiche werden für Realisierungen von Datenobjekten verwendet.

**Freibereiche** : Speicherbereiche, die momentan nicht für Realisierungen von Datenobjekten verwendet werden; d.h. sie sind frei.

Buchführung des Speicherbereichs einer Halde mit Hilfe von Belegungs- und Freigabeoperationen.

### Verwaltung der Freibereiche

#### Freiliste

#### Garbage Collection

#### Animation Haldenverwaltung

Generated by Targeteam



## Garbage Collection



Java sammelt belegten Speicher, der nicht mehr benötigt wird, auf.

automatische Rückgewinnung von nicht mehr benötigtem Speicher  
üblich in Sprachen wie Java, Lisp, Perl, ....

Woher kennt der Speichermanager nicht mehr benötigte Objekte?

i.a. nicht bekannt

allerdings können gewisse Blöcke nur dann benutzt werden, wenn Pointer zu Ihnen existieren.

Annahmen über Pointer

Speichermanager müssen Pointer von anderen Objekten unterscheiden können.

Alle Pointer müssen auf den Blockanfang zeigen.

### Speicher als Graph

Zeitpunkt der Durchführung von Garbage Collection

wenn kein weiterer Platz mehr vorhanden ist, z.B. `new` gibt Fehler zurück

automatisch im Hintergrund, wenn System nicht ausgelastet ist.

Generated by Targeteam



## Speicher als Graph



Annahme

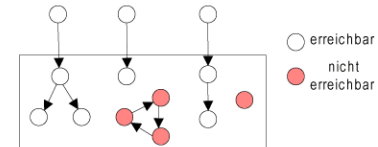
jeder Block ist ein Knoten

jeder Pointer ist eine Kante

Kantenausgangsknoten ist nicht auf der Halde (sondern im Keller)  $\Rightarrow$  Knoten ist Wurzel

Wurzelknoten

Haldenknoten



Bestimmung erreichbarer Knoten.

### Mark and Sweep

Aufnehmen mittels markieren und durchlaufen

Mark: an Wurzeln starten, Pfade durchlaufen und alle erreichbaren Speicherblöcke markieren

Sweep: Scannen aller Blöcke; Freigabe nicht markierter Blöcke

Generated by Targeteam



Java sammelt belegten Speicher, der nicht mehr benötigt wird, auf.  
 automatische Rückgewinnung von nicht mehr benötigtem Speicher  
 üblich in Sprachen wie Java, Lisp, Perl, ....

Woher kennt der Speichermanager nicht mehr benötigte Objekte?  
 i.a. nicht bekannt

allerdings können gewisse Blöcke nur dann benutzt werden, wenn Pointer zu Ihnen existieren.

Annahmen über Pointer  
 Speichermanager müssen Pointer von anderen Objekten unterscheiden können.

Alle Pointer müssen auf den Blockanfang zeigen.

### Speicher als Graph

Zeitpunkt der Durchführung von Garbage Collection  
 wenn kein weiterer Platz mehr vorhanden ist, z.B. `new` gibt Fehler zurück  
 automatisch im Hintergrund, wenn System nicht ausgelastet ist.

*Generated by Targteam*



Die unmittelbare Nutzung des physischen Adressraums (Arbeitsspeichers) bei der Anwendungsentwicklung ist nicht empfehlenswert. Probleme sind folgende:

- Kenntnisse über Struktur und Zusammensetzung des Arbeitsspeichers notwendig.
  - Kapazitätsengpässe bei der Arbeitsspeichergöße.
- ⇒ deshalb Programmerstellung unabhängig von realer Speichergröße und -eigenschaften.

### Adressräume

### Organisation von Adressräumen

### Fragmentierung

### Forderungen an Adressraumrealisierung

*Generated by Targteam*



Die unmittelbare Nutzung des physischen Adressraums (Arbeitsspeichers) bei der Anwendungsentwicklung ist nicht empfehlenswert. Probleme sind folgende:

- Kenntnisse über Struktur und Zusammensetzung des Arbeitsspeichers notwendig.
  - Kapazitätsengpässe bei der Arbeitsspeichergöße.
- ⇒ deshalb Programmerstellung unabhängig von realer Speichergröße und -eigenschaften.

### Adressräume

### Organisation von Adressräumen

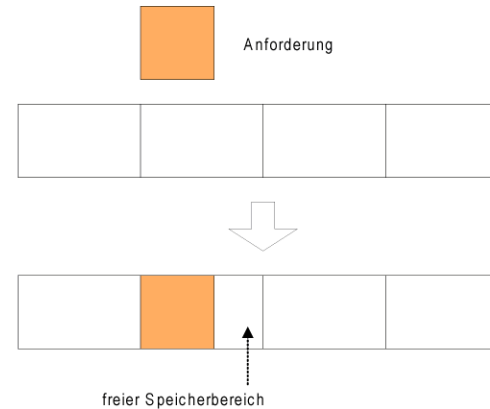
### Fragmentierung

### Forderungen an Adressraumrealisierung

*Generated by Targteam*



Der Speicher ist in Bereiche fester Größe untergliedert und Speicheranforderungen werden nur in Vielfachen dieser festen Grundgröße befriedigt.



*Generated by Targteam*



## Fragmentierung



Unter dem Begriff **Fragmentierung** versteht man verschiedene Formen der Zerstückelung des noch freien und nutzbaren Teils des Adressraums in kleine Bereiche. Unterscheidung zwischen **externer** und **interner** Fragmentierung.

Externe Fragmentierung

Interne Fragmentierung

Generated by Targeteam



## Forderungen an Adressraumrealisierung



Aus der Sicht der Anwendungsprogrammierung können für einen Adressraum eine Reihe wichtiger Forderungen an dessen Realisierung gestellt werden.

Homogene und zusammenhängende Adressbereiche.

Größe des genutzten Adressraums unabhängig von der Kapazität des physischen Adressraums (Arbeitsspeichers).

Erkennen fehlerhafter Zugriffe.

Erkennen von Überschneidungen zwischen Halde und Keller sowie zwischen mehreren Laufzeitkellern.

Schutz funktionstüchtiger Anwendungen gegenüber fehlerhaften Anwendungen.

Kontrollierbares und kontrolliertes Aufteilen der Speicherressourcen auf alle Anwendungen.

Speicherökonomie, minimale Fragmentierung.



Generated by Targeteam



## Einführung



Die unmittelbare Nutzung des physischen Adressraums (Arbeitsspeichers) bei der Anwendungsentwicklung ist nicht empfehlenswert. Probleme sind folgende:

- Kenntnisse über Struktur und Zusammensetzung des Arbeitsspeichers notwendig.
  - Kapazitätsengpässe bei der Arbeitsspeichergöße.
- ⇒ deshalb Programmierstellung unabhängig von realer Speichergröße und -eigenschaften.

Adressräume

Organisation von Adressräumen

Fragmentierung

Forderungen an Adressraumrealisierung

Generated by Targeteam



## Direkte Adressierung



Programmadresse = Maschinenadresse; Probleme

- Verschiebbarkeit,
- Programmgröße,
- Speicherausnutzung.

Verschiebbarkeit

Programmgröße

Programme größer als Arbeitsspeicher; Programmierer zerlegt Programm in Segmente (Nachladen bei Bedarf). Man spricht von der sogenannten **Overlay-Technik** (veraltet).

Forderung

Es muss daher gefordert werden, dass die Programmgröße unabhängig von der realen Arbeitsspeichergöße ist.

Speicherausnutzung

Programme bestehen aus Modulen, die nur zu bestimmten Zeiten verwendet werden.

Forderung

Arbeitsspeicher beinhaltet nur die momentan bzw. in naher Zukunft notwendigen Ausschnitte des Programms.

Nutzen der **Lokalitätseigenschaft** von Programmen:

- durch Datenstrukturen z.B. Arrays, oder
- Programmstrukturen: Prozeduren, Schleifen.

Generated by Targeteam



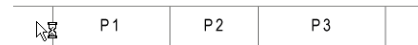
## Verschiebbarkeit



In einem Mehrprozesssystem sind i.d.R. mehrere Programme im Arbeitsspeicher. Bei direkter Adressierung werden die Programme beim Laden fixiert und müssen dann bis zu ihrem Ende an derselben Stelle bleiben.

### Problem

Externe Fragmentierung des Arbeitsspeichers. Sei der Arbeitsspeicher zunächst lückenlos mit Programmen P1, P2, P3 gefüllt.



Nach Beendigung des Programms P2 entsteht eine Lücke.

### Forderung

In Mehrprogramme/Mehrprozesssystemen müssen daher Programme **verschiebbar** sein.

Generated by Targeteam



## Direkte Adressierung



Programmadresse = Maschinenadresse; Probleme

- Verschiebbarkeit,
- Programmgröße,
- Speicherausnutzung.

### Verschiebbarkeit

#### Programmgröße

Programme größer als Arbeitsspeicher; Programmierer zerlegt Programm in Segmente (Nachladen bei Bedarf). Man spricht von der sogenannten **Overlay-Technik** (veraltet).

#### Forderung

Es muss daher gefordert werden, dass die Programmgröße unabhängig von der realen Arbeitsspeichergröße ist.

#### Speicherausnutzung

Programme bestehen aus Modulen, die nur zu bestimmten Zeiten verwendet werden.

#### Forderung

Arbeitsspeicher beinhaltet nur die momentan bzw. in naher Zukunft notwendigen Ausschnitte des Programms.

Nutzen der **Lokalitätseigenschaft** von Programmen:

- durch Datenstrukturen z.B. Arrays, oder
- Programmstrukturen: Prozeduren, Schleifen.

Generated by Targeteam



## Direkte Adressierung



Programmadresse = Maschinenadresse; Probleme

- Verschiebbarkeit,
- Programmgröße,
- Speicherausnutzung.

### Verschiebbarkeit

#### Programmgröße

Programme größer als Arbeitsspeicher; Programmierer zerlegt Programm in Segmente (Nachladen bei Bedarf). Man spricht von der sogenannten **Overlay-Technik** (veraltet).

#### Forderung

Es muss daher gefordert werden, dass die Programmgröße unabhängig von der realen Arbeitsspeichergröße ist.

#### Speicherausnutzung

Programme bestehen aus Modulen, die nur zu bestimmten Zeiten verwendet werden.

#### Forderung

Arbeitsspeicher beinhaltet nur die momentan bzw. in naher Zukunft notwendigen Ausschnitte des Programms.

Nutzen der **Lokalitätseigenschaft** von Programmen:

- durch Datenstrukturen z.B. Arrays, oder
- Programmstrukturen: Prozeduren, Schleifen.

Generated by Targeteam



## Basisadressierung



Die Basisadressierung hat eine einfache Abbildungsvorschrift:

$$\text{Maschinenadresse} = \text{Basisadresse} + \text{Programmadresse}$$

Die Basisadresse ist programmspezifisch.

Die Programmadressen aller Programme beginnen jeweils mit Null.

### Speicherverwaltungsstrategien

Aufgabe: Finden eines zusammenhängenden Arbeitsspeicherbereichs, der groß genug ist, um das Programm zu speichern  $\Rightarrow$  siehe Verfahren zur [Verwaltung der Halde](#).

#### first-fit

Durchsuche die Liste der Freibereiche vom Anfang an und nimm den ersten passenden Frei-Bereich: Spalte nicht benötigten Speicher ab und füge ihn als freien Bereich in die Freibereichsliste ein.

#### next-fit

Durchsuche die Liste der Freibereiche nach first-fit, jedoch beginne die Suche dort, wo man bei der letzten Suche aufgehört hat.

#### best-fit

Durchsuche die Liste der Freibereiche vom Anfang an und nimm den passenden Frei-Bereich, der die Speicheranforderungen des Programms am besten erfüllt: Spalte nicht benötigten Speicher ab und füge ihn als freien Bereich in die Freibereichsliste ein.

#### worst-fit

Durchsuche die Liste der Freibereiche vom Anfang an und nimm den Frei-Bereich, der die Speicheranforderungen des Programms am schlechtesten erfüllt: Spalte nicht benötigten Speicher ab und füge ihn als freien Bereich in die Freibereichsliste ein.

### Buddy-Systeme

Generated by Targeteam



Speicheranforderungen werden in Größen von Zweierpotenzen vergeben, d.h. eine Anforderung wird auf die nächste Zweierpotenz aufgerundet

Anforderung 280 Bytes  $\Rightarrow$  Belegung von 512 Bytes =  $2^9$

am Anfang besteht der Arbeitsspeicher aus einem großen Stück.

Speicheranforderung von  $2^k$ : ist Speicherbereich dieser Größe vorhanden, dann Unterteilung eines Speicherbereichs der Größe  $2^{k+1}$  in 2 Speicherbereiche der Größe  $2^k$  Bytes.

Freigabe eines Speicherbereichs von  $2^k$ : falls auch entsprechendes Partnerstück frei ist, dann Verschmelzung der beiden Speicherbereiche zu einem Speicherstück der Größe  $2^{k+1}$ .

Generated by Targeteam



### Fragestellungen

Dieser Abschnitt beschäftigt sich mit den Adressräumen für Programme und deren Abbildung auf den physischen Arbeitsspeicher einer Rechananlage:

Programmadressraum vs. Maschinenadressraum.

Direkte Adressierung, Basisadressierung.

Virtualisierung des Speichers; virtuelle Adressierung, insbesondere Seitenadressierung.

### Einführung

#### Speicherabbildungen

Dieser Abschnitt behandelt einige Mechanismen zur Abbildung von Programmadressen auf Maschinenadressen des Arbeitsspeichers.

[Direkte Adressierung](#)

[Basisadressierung](#)

[Seitenadressierung](#)

[Segment-Seitenadressierung](#)

[Speicherhierarchie / Caches](#)

Generated by Targeteam



Die virtuelle Adressierung wurde Ende der 50er Jahre eingeführt. Ziel ist

Virtualisierung des Speichers,

Verstecken von realen Beschränkungen, wie Speichergröße,

Speicher als sehr großes Feld gleichartiger Speicherzellen zu betrachten.

Die Seitenadressierung ("paging") ist die Grundform der virtuellen Adressierung.

[Ansatz](#)

[Adressabbildung](#)

[Seiten-Kacheltabelle](#)

[Seitenfehlerbehandlung](#)

[Seitenverwaltungsstrategien](#)

[Linux - Virtuelle Adressierung](#)

Generated by Targeteam



Die Seiten eines Prozesses können im Arbeitsspeicher oder auf dem Hintergrundspeicher (Platte) gespeichert sein.

Die Kacheln nehmen die Seiten der Prozesse auf.

Zugriff auf virtuelle Adresse  $\Rightarrow$  zugehörige Seite muss einer Kachel zugeordnet sein.

Die Zuordnung, welche Seite in welcher Kachel gespeichert ist, und wo sich die Seite auf dem Hintergrundspeicher befindet, erfolgt mittels der **Seiten-Kacheltabelle**, die die **Seitendeskriptoren** enthält.

Seite nicht im Arbeitsspeicher  $\Rightarrow$  **Seitenfehler**  $\Rightarrow$  Einlagerung der Seite bei Bedarf ("Demand Paging").

Falls alle Kacheln belegt  $\Rightarrow$  Auslagern einer Seite gemäß einer **Seitenersetzungsstrategie**.

Generated by Targeteam