Betriebssysteme I

Hochschule München
Sommersemester 2010

Thomas Kolarz

thomas.kolarz@hm.edu

Betriebssysteme I - Inhalt

0. Einführung, Geschichte und Überblick
1. Prozesse und Threads
   (die Abstraktion der CPU)
2. Speicherverwaltung
   (die Abstraktion des Arbeitsspeichers)
3. Dateisysteme
   (die Abstraktion der Platte)
4. Eingabe und Ausgabe
5. Vertiefung am Beispiel von Linux
6. Virtuelle Maschinen und Verteilte Systeme

2. Speicherverwaltung

Grundlagen

- Speicherhierarchien
- Speicherverwaltung mit Bitmaps
- Speicherverwaltung mit verketteten Listen

Virtuelle Speicher

- Virtueller Speicher: Motivation
- Virtueller Speicher: Grundbegriffe
- MMU Adressumsetzung
- Page Fault
- Seitentabellen
- Paging: Beschleunigung
- TLB (Translation Look-Aside Buffer)
- Mehrstufige Seitentabellen
- Invertierte Seitentabellen

Der Paging Prozess

- Paging: Aufgaben des Betriebssystems
- Hintergrundspeicher (Swap-Speicher)

2. Speicherverwaltung

Seitenersetzungsalgorithmen

- Optimale Algorithmus
- Not-Recently-Used
- First-In-First-Out
- Second-Chance
- Clock-Algorithmus
- Least-Recently-Used
- Aging Algorithmus
- Working-Set Algorithmen
- Implementierungsaspekte

Segmentierung

- Segmentierung des virtuellen Adressraums
- Segmentierung beim Pentium

Speicherhierarchie

Der Teil des Betriebssystems, der die Speicherhierarchie verwaltet, heißt **Speicherverwaltung (memory manager)**. Er verfolgt, welche Speicherbereiche gerade genutzt werden, teilt Prozessen Speicher zu, wenn sie ihn benötigen, und gibt ihn anschliessend wieder frei.

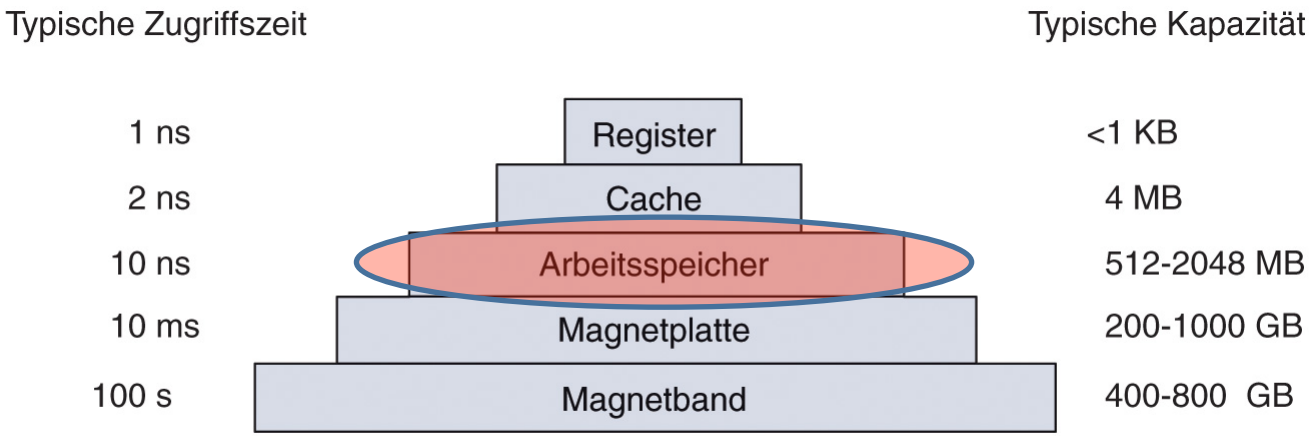
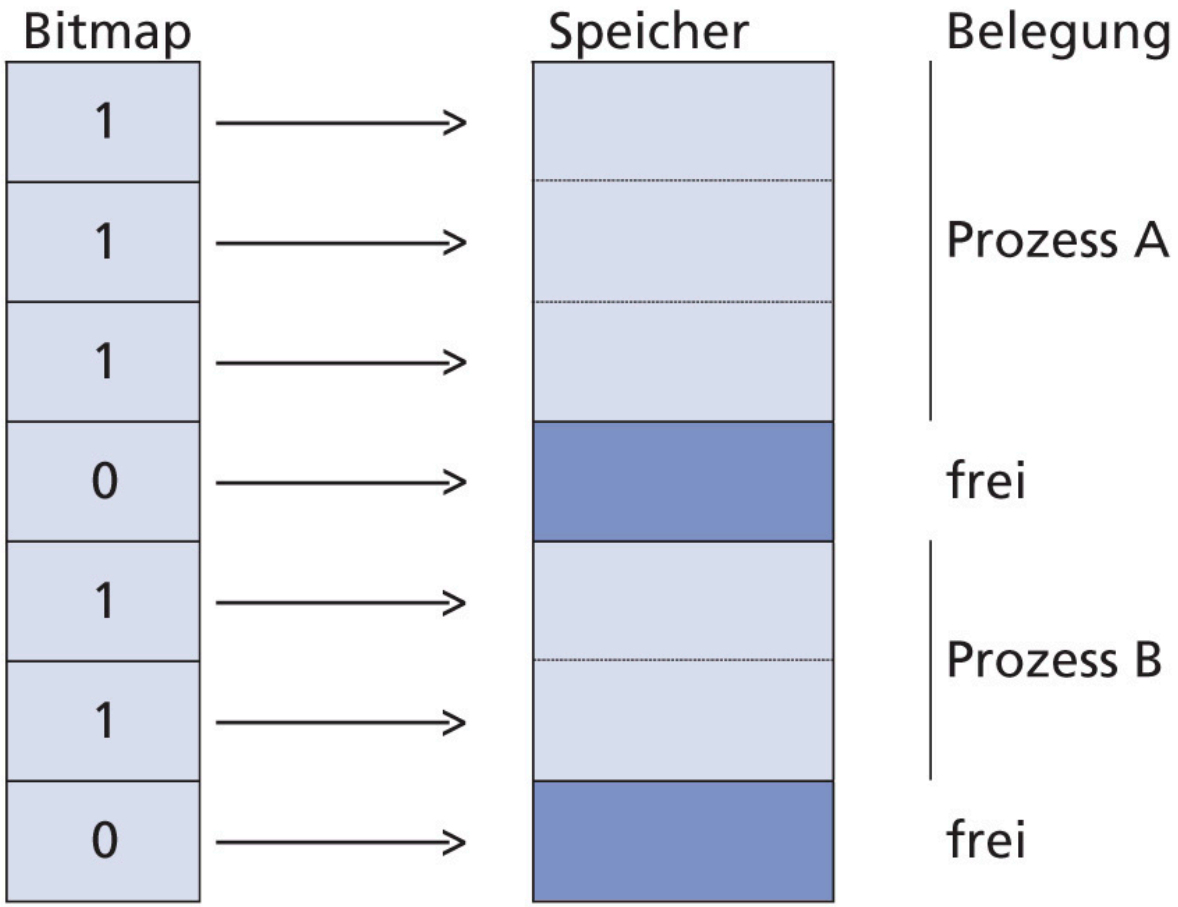


Abbildung 1.9: Eine typische Speicherhierarchie. Die Werte sind sehr grobe Annäherungen.

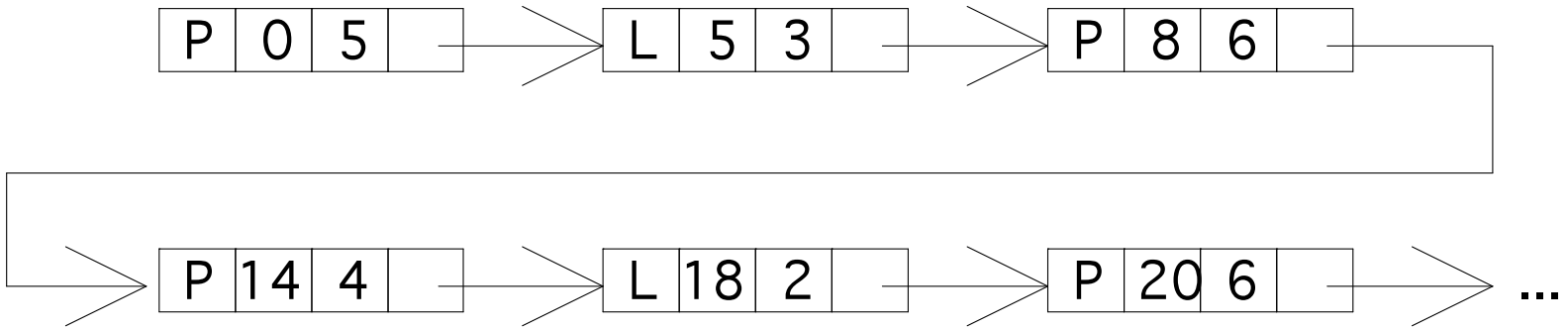
Speicherverwaltung mit Bitmaps



Speicherverwaltung mit Bitmaps - Probleme

- Für einen Prozess, der k Einheiten benötigt, muss die Bitmap nach k aufeinanderfolgenden 0-Bits untersucht werden -> sehr zeitaufwendig
- Gegebenenfalls muss der Speicher verdichtet werden, um freie Bereiche zu schaffen, die groß genug sind -> noch zeitaufwendiger
- Zusätzlich benötigter Speicher während der Ausführung führt zu weiteren Problemen bei der Speicherverwaltung.
- Neue Prozesse können eventuell nicht mehr gestartet werden, falls kein freier Bereich verfügbar ist, der ausreichend groß ist.

Speicherverwaltung mit verketteten Listen



- Die Suche nach freien Speicherbereichen ist schneller und kann noch weiter verbessert werden (z.B. getrennte Listen für Lücken und Prozesse)
- Viele grundsätzliche Probleme, ähnlich wie bei den Bitmaps bleiben jedoch

Virtueller Speicher: Motivation

Ausgangspkt.: mehrere Programme im Arbeitsspeicher ohne Speicherabstraktion

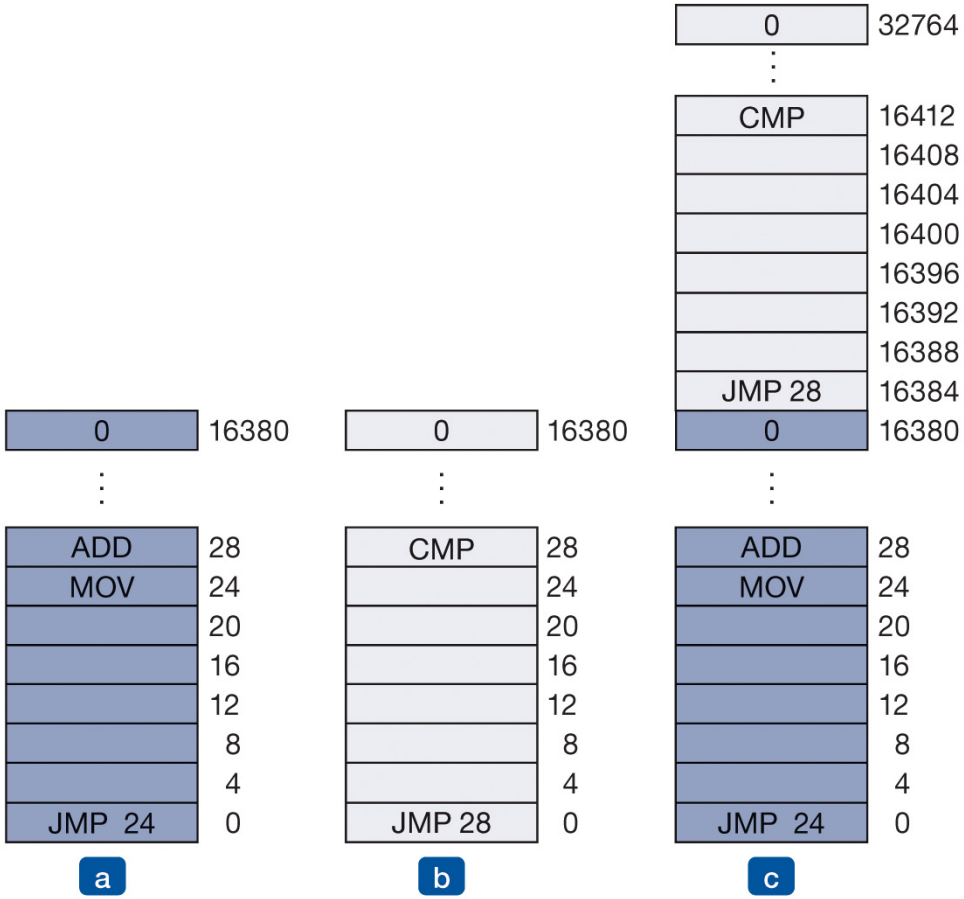


Abbildung 3.2: Darstellung des Relokationsproblems (a) Ein 16-KB-Programm (b) Ein weiteres 16-KB-Programm (c) Die zwei Programme wurden hintereinander in den Speicher geladen.

Virtueller Speicher - Grundidee

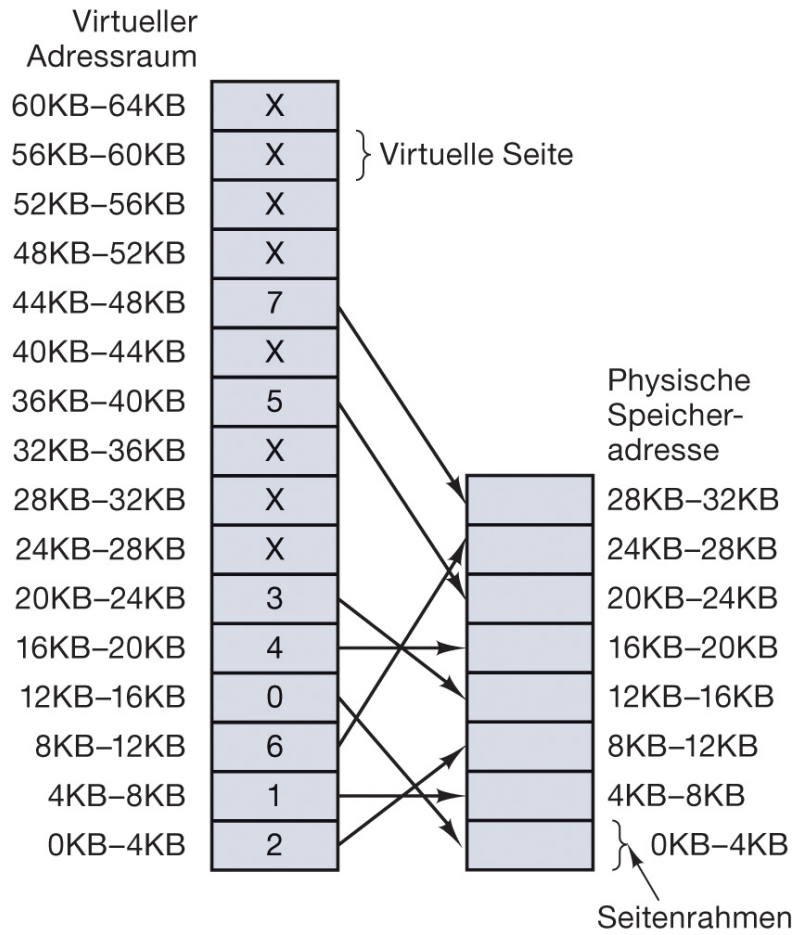


Abbildung 3.9: Die Beziehung zwischen virtuellen und physischen Adressen ist durch eine Seitentabelle vorgegeben. Jede Seite beginnt bei einem Vielfachen von 4.096 und endet 4.096 Adressen höher. 4KB–8KB bedeutet also tatsächlich 4.096–8.191 und 8KB–12KB ist 8.192–12.287.

Virtueller Speicher - Grundbegriffe

- Die vom Prozess generierten Adressen werden **virtuelle Adressen** genannt und bilden den **virtuellen Adressraum**
- Der virtuelle Adressraum ist linear und zusammenhängend (z.B.: 0 bis $2^{32} – 1$) und kann größer sein als der physische Speicher (z.B.: 512 MB)
- Der virtuelle Adressraum ist in Einheiten fester Größe, sogenannte **Seiten (page)** unterteilt (reale Seitengrößen sind beispielsweise: 1 KB , 4 KB oder 8 KB)
- Die entsprechende Einheit im physischen Speicher wird **Seitenrahmen (page frame)** genannt
- Die **MMU** (Memory Management Unit) übersetzt die virtuelle Adresse in eine physische Adresse
- Die Beziehung zwischen virtuellen und physischen Adressen bilden die sog. **Seitentabelle**

MMU Adressumsetzung

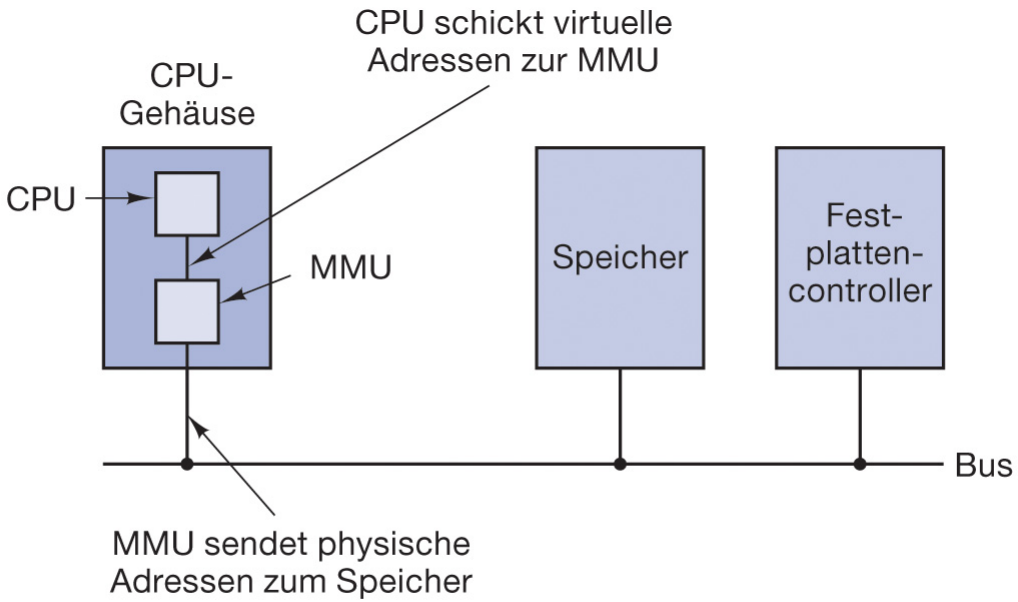


Abbildung 3.8: Position und Funktion der MMU. Die MMU ist hier Teil des CPU-Chips, wie heutzutage üblich. Aus logischer Sicht könnte es aber auch wie in alten Zeiten ein eigenständiger Chip sein.

Beispiele für 2 MMU-Abbildungen

MOV REG, 0	➔	MOV REG, 8192
MOV REG, 8192	➔	MOV REG, 24576

MMU - Adressumsetzungsprinzip

Beispiel:

Seitengröße 4 KB

virtueller Adressraum 16 Bit

physischer Adressraum 15 Bit
(hängt von der Größe des Arbeitsspeichers ab, hier also 32 KByte)

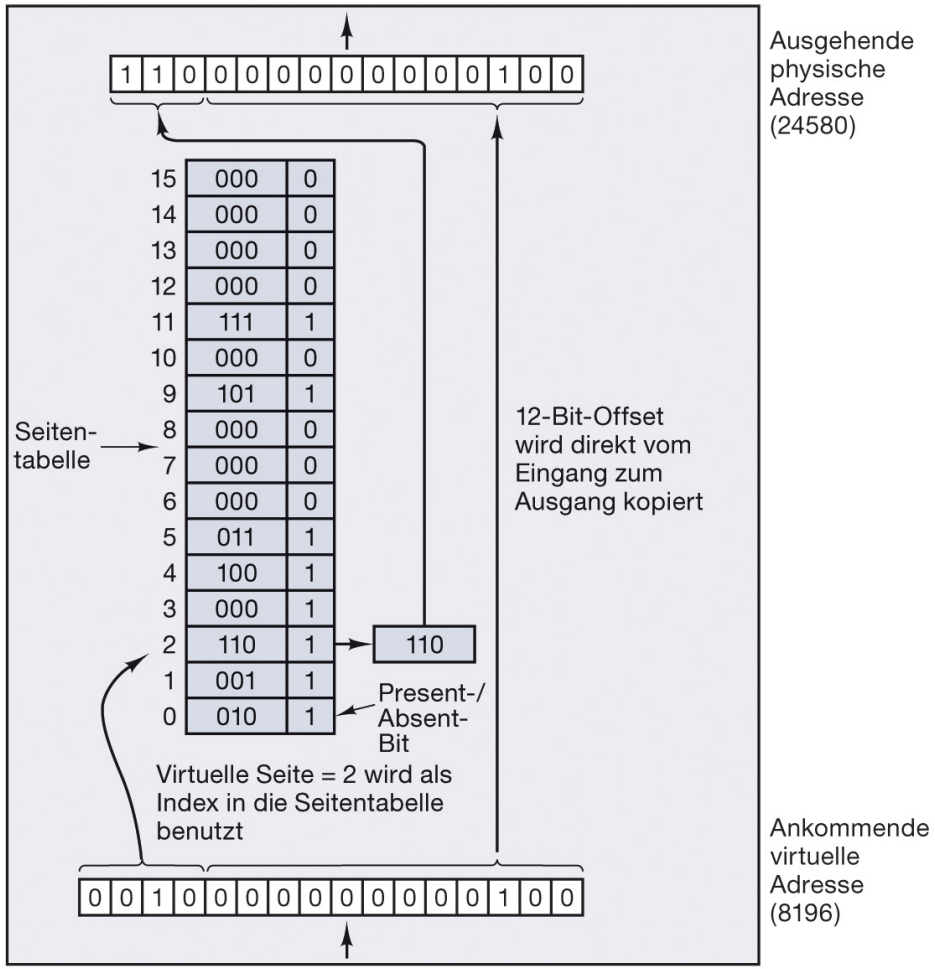


Abbildung 3.10: Interne Operation der MMU mit 16 4-KB-Seiten

Seitentabellen

- Die Abbildung von virtuellen auf physischen Adressen erfolgt durch die Seitentabelle
- Die virtuelle Adresse wird in eine virtuelle Seitennummer (höherwertige Bits) und einen Offset (niederwertige Bits) geteilt
- Der Offset bestimmt die Adresse innerhalb einer Seite – die Anzahl der Bits im Offset ist durch die Seitengröße bestimmt
- Die virtuelle Seitennummer wird als Index genutzt, um in der Seitentabelle den Eintrag für die „physische Seitenrahmennummer“ zu finden.
- Die Seitenrahmennummer wird an das höherwertige Ende des Offset gesetzt und bildet zusammen mit diesem die physische Adresse
- Zu jedem Prozess gibt es eine Seitentabelle (diese ist nicht statisch, sondern kann sich während der Ausführung ändern)

Seitentabelleneintrag

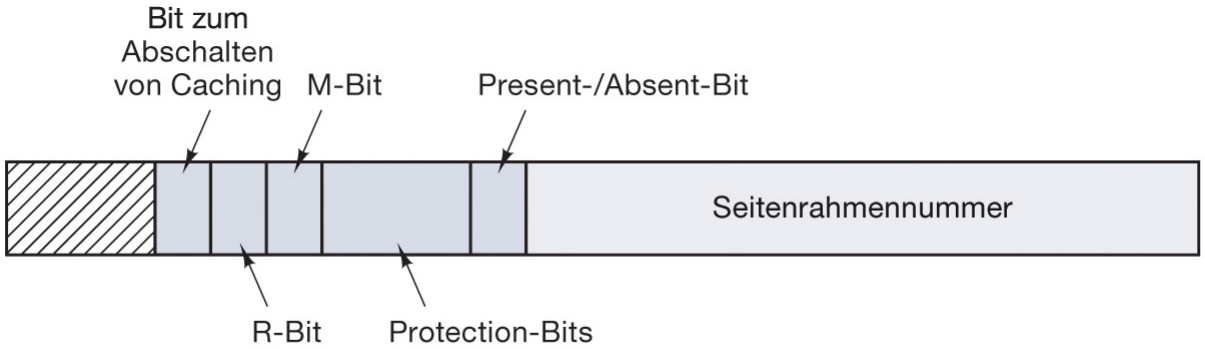


Abbildung 3.11: Ein typischer Seitentabelleneintrag

Page Fault

Present-/Absent – Bit: zeigt an, welche Seiten physisch im Speicher vorhanden sind

Im Fall, dass eine virtuelle Adresse die MMU erreicht, deren Seite ausgelagert ist, wird von der MMU ein Systemaufruf ausgelöst, genannt **page fault (Seitenfehler)**.

Aktionen bei einem Page Fault

- Betriebssystem wählt einen wenig benutzten Seitenrahmen aus und schreibt dessen Inhalt zurück auf die Festplatte (falls notwendig!).
- die angeforderte Seite wird in den Seitenrahmen geladen und die Zuordnungstabelle wird angepasst.

Paging - Beschleunigung

Bei jedem Speicherzugriff wird eine virtuelle Adresse in eine physische umgerechnet. Alle Befehle kommen letztendlich vom Speicher und viele Befehle benötigen auch noch Operanden. Also sind für jeden Befehl ein, zwei, oder mehr Speicherzugriffe nötig

In jedem Paging System gibt es zwei große Probleme

- Die Adressumrechnung muss sehr schnell erfolgen
- Große virtuelle Adressräume führen zu großen Seitentabellen

Beschleunigung des Paging - Motivation

Extrembetrachtung 1: Seitentabelle eines Prozesses in schnellen Registern

- Bei einem virtuellen Adressraum von 32-Bit und 4 KB Seitengröße bräuchte man 1 Million schnelle Register.
- Bei einem Kontextwechsel müsste eine sehr große Seitentabelle ausgetauscht werden.

Extrembetrachtung 2: Seitentabelle im Arbeitsspeicher

- Ein Register zeigt auf den Anfang im Arbeitsspeicher, indem sich die Tabelle befindet.
- Jeder „normale“ Speicherzugriff führt zu einem zusätzlichen Speicherzugriff auf die Tabelle.
- Kontextwechsel gehen schnell, da nur das Register geändert wird.

Beschleunigung des Paging mit TLB

Translation Look-Aside Buffer

- Der TLB ist ein vollassoziativer Hardware Cache, der die zuletzt benutzten Seitentabelleneinträge enthält
- Das Prinzip des TLB basiert darauf, dass Programme dazu neigen, sehr viele Zugriffe auf sehr wenige Seiten auszuführen (Lokalitätsprinzip)
- Der TLB ist Teil der MMU (enthält selten mehr als 64 Einträge); jeder Eintrag enthält eine Zeile aus der Seitentabelle, zusätzlich den zugehörigen Index und ein Gültigkeitsbit
- Bei einem Fehlzugriff (miss) im TLB wird der gesuchte Eintrag in den TLB geladen und verdrängt einen anderen Eintrag; dabei wird das M-Bit des verdrängten Eintrags in die Seitentabelle zurückgeschrieben, da dies evtl. geändert wurde
- Die Behandlung eines TLB-Fehlers wird durch einen Systemaufruf an das BS übergeben und ist nicht einfach. Bei der Behandlung kommen große SW-Caches zum Einsatz (deren TLB-Einträge sind immer im TLB!).

Translation Look-Aside Buffer

Gültig	Virtuelle Seite	Verändert	Schutz	Seitenrahmen
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

Abbildung 3.12: Ein TLB zur Beschleunigung des Paging

Der Inhalt im TLB ist prozessspezifisch! Bei einem Kontextwechsel wird der Inhalt entweder ungültig oder jeder Eintrag enthält zusätzlich die PID, die mit der PID des zugreifenden Prozesses verglichen wird (sog. **Tagged TLB**).

Mehrstufige Seitentabellen

Beisp.: 32-Bit virtueller Adressraum, 4 KB Seitengröße, 4 Byte pro Eintrag

➔ 1 Million Tabelleneinträge mit jeweils 4 Byte = 4 MB pro Prozess

➔ bei einigen hundert aktiven Prozessen wäre allein 1 GB Arbeitsspeicher für die Seitentabellen belegt

- Dieses Problem kann mit mehrstufigen Seitentabellen effektiv gelöst werden.
- Nehmen wir an, ein Prozess belegt die unteren 4 MB für Programmcode, die mittleren 4 MB für Daten und die oberen 4 MB für den Stack.
- Der entscheidende Punkt ist hier, dass nicht mehr alle Seitentabellen eines Prozesses im Arbeitsspeicher gehalten werden müssen, ohne einen Performancenachteil zu haben.

Mehrstufige Seitentabellen

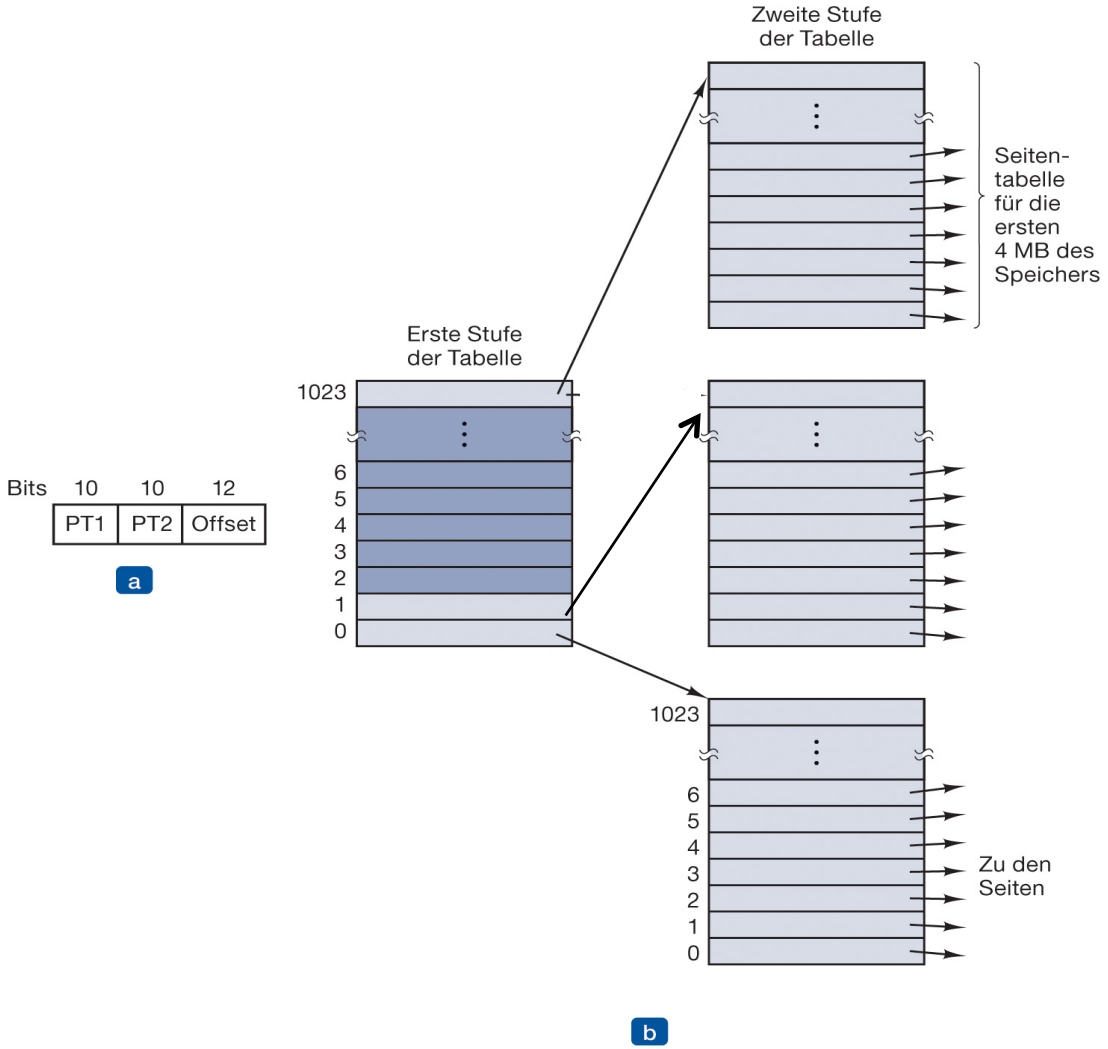
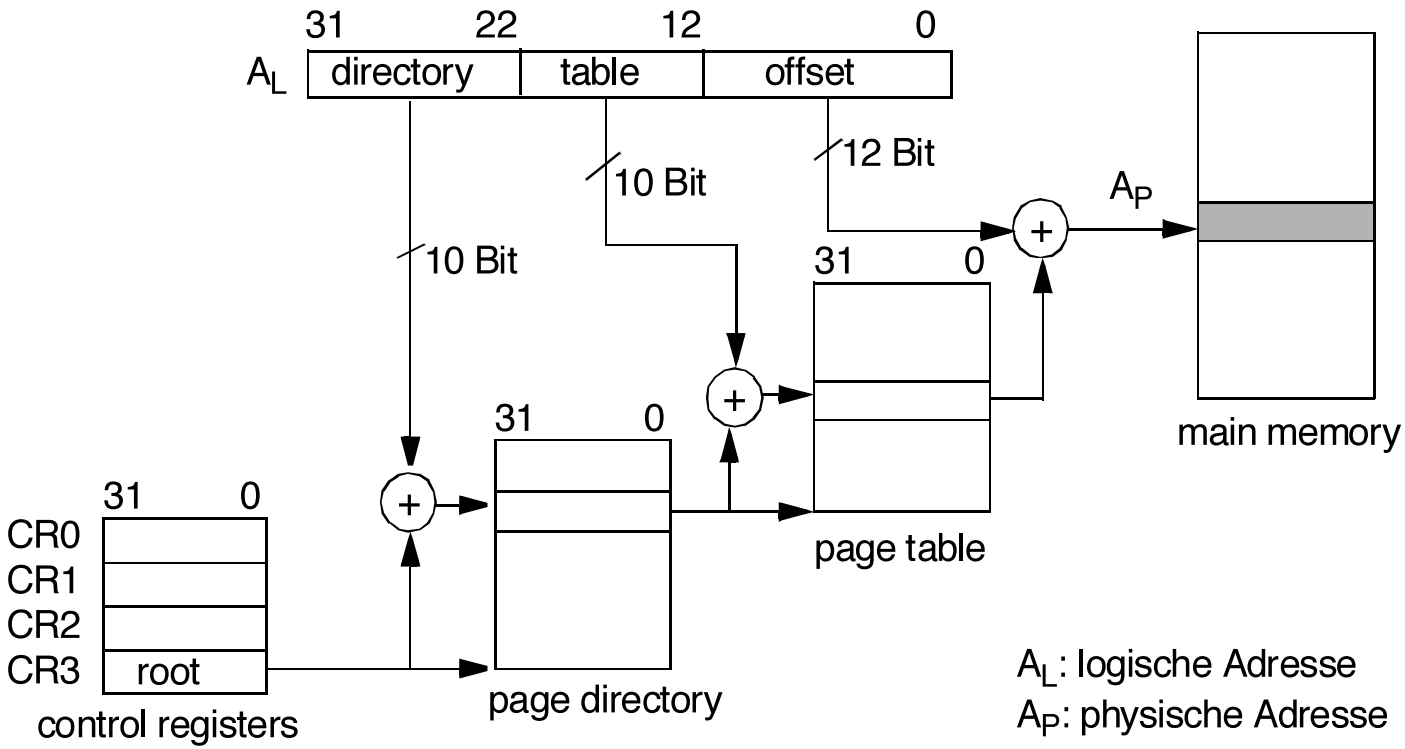


Abbildung 3.13: (a) Eine 32-Bit-Adresse mit zwei 10-Bit-Feldern für Seitennummern (b) Zweistufige Seitentabellen

Mehrstufige Seitentabellen



Mehrstufige Seitentabellen

- Die Seitennummer (Index) in dem Beispiel 20 Bit wird nochmal unterteilt in zwei Felder je 10 Bit.
- Die oberen 10 Bit adressieren einen Eintrag in einer Tabelle der 1.Stufe, deren Einträge auf Tabellen der 2. Stufe verweisen, für die das 2. Feld PT2 als „Offset“ bzw. Index dient.
- In der Tabelle der 2. Stufe findet man die Tabelleneinträge.
- Für den Prozess aus dem obigen Beispiel bräuchte man nur insgesamt 4 Tabellen mit jeweils 4 x 1024 Einträgen im Arbeitsspeicher zu halten (anstelle von 1 M Einträge).
- Von diesen 4096 Einträgen wären wiederum die wichtigsten im TLB!

Invertierte Seitentabellen

Für virtuelle 64-Bit Adressräume mit Paging ist eine andere Lösung nötig:

64-Bit virtueller Adressraum, 4 KB Seiten, 8 Byte pro Eintrag

➔ 2^{52} Einträge in der Seitentabelle

➔ 8 Million Gigabyte Seitentabelle pro Prozess

Invertierte Seitentabellen

- In einer invertierten Seitentabelle gibt es einen Eintrag pro Seitenrahmen, d.h. die Anzahl der Einträge ist unabhängig von der Größe des virtuellen Adressraums.
- Für die Seiteneinträge im TLB ist die Suche genauso schnell, da sämtliche Einträge gleichzeitig verglichen werden.
- Für die restliche Tabelle (die nicht im TLB steht) kann die Suche nach dem richtigen Eintrag nicht wie vorher durchgeführt werden, da jeder Eintrag mit der virtuellen Adresse verglichen werden muss.
- Für die Speicherung der Tabelleneinträge werden Hashtabellen genutzt mit den virtuellen Adressen als Hashwerte.

Der Paging Prozess

Aufgaben des Betriebssystems beim Paging

Es gibt 4 Punkte im Leben eines Prozesses, an denen das BS Arbeit leisten muss im Zusammenhang mit Paging:

1. Erzeugung des Prozesses
2. Ausführung des Prozesses
3. Seitenfehler
4. Terminierung des Prozesses

Aufgaben des Betriebssystems beim Paging

a) Aktuelle Seitentabelle wird auf Seitentabelle des Prozesses gesetzt (inkl. TLB Initialisierung)
b) evt. Prepaging (zumindest wird die Seite mit dem aktuellen Befehl benötigt)

a) Seitentabelle, Seitenrahmen u. Swapbereich freigeben

a) Ermittlung Größe des Programmcodes u. Daten
b) Erzeugung Seitentabelle (Speicher wird zugeteilt – „leere Tab.“)
c) Laden der Seiten in den Swap o. direkt in den Speicher
d) Prozesstabelle erhält die Infos zur Seitentabelle und Swapbereich

a) Feststellen welche Seite benötigt wird
b) Seitenersetzung durchführen
c) Seitentabelle aktualisieren

Hintergrundspeicher (Swap Bereich)

- Seiten, die aus dem Speicher verdrängt werden, werden auf der Swap Partition gespeichert.
- Beim Systemstart ist die Swap Partition leer und wird im Speicher durch einen einzigen Eintrag repräsentiert, der seinen Anfang und seine Größe enthält.
- Im einfachsten Fall reserviert das System beim Start eines Prozesses soviel Platz in der Swap Partition, wie die Gesamtgröße des Prozesses ist.
- Ein anderes Verfahren ist, eine Plattenzuordnungstabelle zu pflegen, in der festgehalten wird wo die Seiten in der Swap Partition liegen.
- Mit jedem Prozess wird die Adresse seines Swap-Bereichs bzw. seiner Plattenzuordnungstabelle verbunden (und in der Prozesstabelle festgehalten).

Hintergrundspeicher (Swap Bereich)

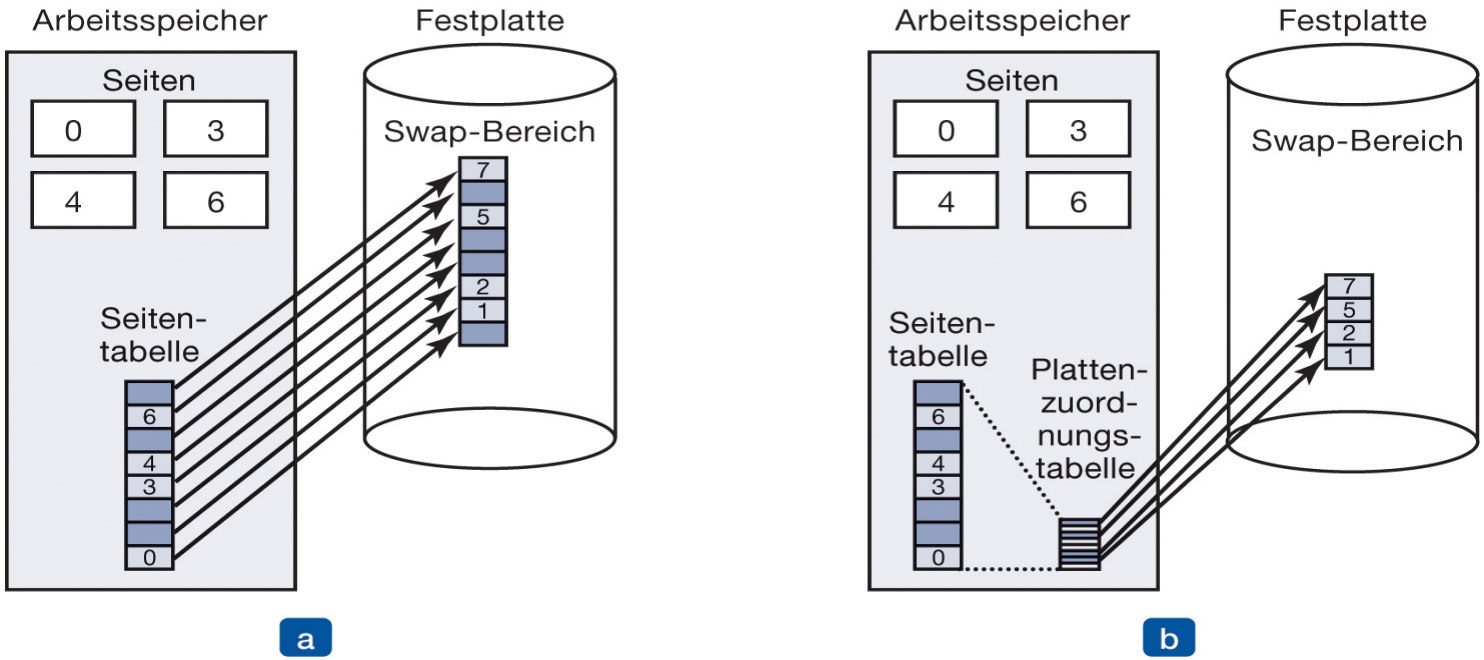


Abbildung 3.29: (a) Paging mit statischem Swap-Bereich (b) Dynamischer Hintergrundspeicher

Hintergrundspeicher (Swap Bereich)

- Wenn keine Plattenpartition zur Verfügung steht, kann eine Datei oder auch mehrere große Dateien als Swap-Bereich hergenommen werden. Windows benutzt diese Methode.
- Für “read-only“ Daten, z.B. Programcode oder Bibliotheken kann als „Swap-Bereich“ die Original-Datei auf der Platte dienen.
- Wenn ein Prozess terminiert, wird sein Swap-Bereich wieder freigegeben.

Zusammenfassung

- Jeder Prozess hat seinen eigenen virtuellen Adressraum (z.B. 32-Bit). In diesem Adressraum arbeitet der Prozess, ohne sich um andere Prozesse kümmern zu müssen -> **Kontextwechsel durch eigene Seitentabelle**.
- Im physischen Speicher befinden sich Seiten, die zu verschiedenen Prozessen gehören. Ein Prozess muss nicht alle Seiten im Speicher haben um lauffähig zu sein und die Seiten eines Prozesses müssen nicht zusammenhängen.
- Benötigt ein Prozess eine Seite, die zur Zeit nicht im physischen Speicher ist, dann wird ein Page-Fault generiert und die Seite wird aus dem Hintergrundspeicher in den Arbeitsspeicher geladen.

Seitenersetzungsalgorithmen

Seitenersetzungsalgorithmen

Optimale Algorithmus

- Lagert die Seite aus, die in der Zukunft am spätesten gebraucht wird. Ist leider in der Praxis nicht realisierbar.

NRU (Not-Recently-Used)

- R-Bit (recently-used) wird nach einem bestimmten Zeitintervall gelöscht und dann erst beim nächsten Zugriff wieder gesetzt. Dadurch ergeben sich die folgenden 4 Klassen:

  Klasse 0: nicht referenziert, nicht modifiziert

  Klasse 1: nicht referenziert, modifiziert

  Klasse 2: referenziert, nicht modifiziert

  Klasse 3: referenziert und modifiziert

- NRU entfernt eine zufällige Seite aus der niedrigsten nicht-leeren Klasse.
- Die Leistung ist in vielen Fällen ausreichend. Timerinterrupt notwendig!

Seitenersetzungsalgorithmen

FIFO (First In First Out)

- Das „Alter“ der Seite ist maßgeblich für die Ersetzung. Wird nur selten eingesetzt. Auf einen Supermarkt übertragen, würde er Dinge wie Schnurrbartwachs entfernen, aber auch Mehl, Salz oder Butter (zumindest kurzzeitig).

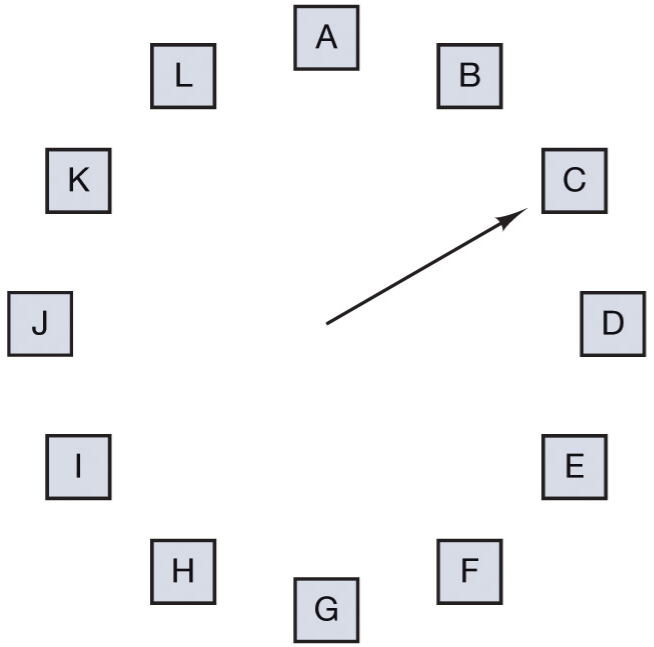
Second-Chance

- Eine FIFO kombiniert mit dem R-Bit. Ist das R-Bit gesetzt, dann wird es gelöscht und die Seite wird ans Ende der FIFO verschoben (bekommt also eine zweite Chance). Die „älteste“ Seite mit nicht-gesetzten R-Bit wird ersetzt.
- Vermindert die große Schwäche von FIFO, auch häufig genutzte Seiten auszulagern. Etwas aufwendig, da er ständig Seiten in seiner Liste verschiebt. Die verbesserte Version nennt sich „**Clock-Algorithmus**“.

Seitenersetzungsalgorithmen

Clock-Algorithmus

- Identisch zu Second-Chance. „Clock“ ermöglicht eine effizientere Implementierung.



Wenn ein Seitenfehler auftritt, wird die Seite untersucht, auf die der Zeiger weist. Die Folgeaktion hängt dann vom R-Bit ab:
R = 0: verdränge die Seite
R = 1: lösche R und rücke Zeiger weiter

Abbildung 3.16: Der Clock-Algorithmus

Seitenersetzungsalgorithmen

LRU (Least-Recently-Used)

- Entferne die Seite, die am längsten unbenutzt ist. Eine der besten Annäherungen an den optimalen Algorithmus.
- Ist zwar theoretisch realisierbar, aber in der Praxis i.a. zu aufwendig, da bei jedem Zugriff die Liste aktualisiert werden muss. Algorithmus muss in HW realisiert sein, um die Befehlsausführung nicht zu verzögern und dies ist i.a. zu teuer.

Aging Algorithmus

- Eine relativ effiziente Annäherung an LRU durch SW. Das „Alter“ der Zugriffe wird durch die Registrierung der gesetzten R-Bits angenähert (Timerinterrupts notwendig), wobei die registrierten Bits altern.

Seitenersetzungsalgorithmen

Aging Algorithmus

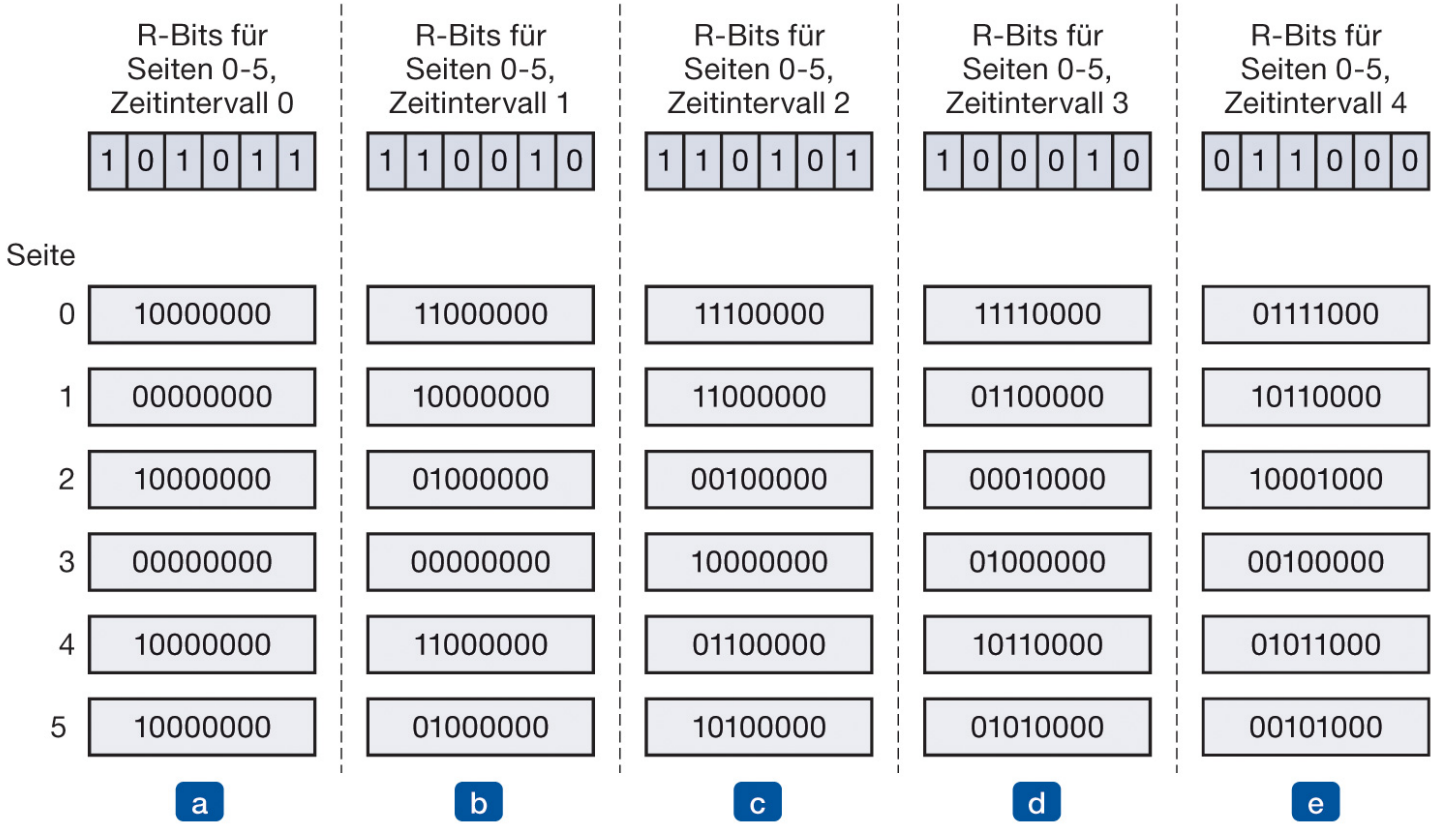


Abbildung 3.18: Der Aging-Algorithmus ist eine Software-Simulation von LRU. Dargestellt werden die Zähler von sechs Seiten für fünf Intervalle. (a) bis (e) zeigen die Zustände nach den Intervallen 1–5.

Seitenersetzungsalgorithmen

Aging Algorithmus vs. LRU

- diskrete Zeitintervalle, in denen nicht zwischen Anfang und Ende eines Intervalls unterschieden wird
- die Anzahl der Bits in den Zählern begrenzt die „zeitliche Registrierungs-Kapazität“.

Seitenersetzungsalgorithmen

Working-Set Algorithmen

- Die Seiten die zu einem Prozess gehören, der sogenannte **Working Set (Arbeitsbereich)**, wird bei Bedarf nach und nach eingelagert **(Demand Paging).**
- Wenn der verfügbare Speicher nicht für alle Seiten des Arbeitsbereichs ausreicht, kann es zum sogenannten **Trashing (Seitenflattern)** kommen, d.h. Seiten müssen ständig ein- und ausgelagert werden.
- Viele BS merken sich den Arbeitsbereich eines Prozesses und lagern diese Seiten ein, bevor sie den Prozess weiterführen. Diesen Ansatz nennt man **Working-Set Modell**. Die Seiten zu laden, bevor sie gebraucht werden, nennt man **Prepaging.**

Seitenersetzungsalgorithmen

Implementierungsaspekte

- **Seitengröße:** große Seiten fördern die sogenannte interne Fragmentierung; kleine Seiten verlängern die Zeit bis der Working Set im Arbeitsspeicher ist
- **Seitenersetzungs-Algorithmus:** Second-Chance, Aging, NFU, etc..
- **Seitenersetzungsstrategie:** lokal oder global (d.h. nur Seiten des aktuellen Prozesses oder alle); Demand Paging oder Prepaging

Segmentierung

Segmentierung des virtuellen Adressraums

- Der bisher behandelte virtuelle Speicher ist eindimensional, weil die virtuellen Adressen linear von 0 bis zu einem bestimmten Maximum wachsen.

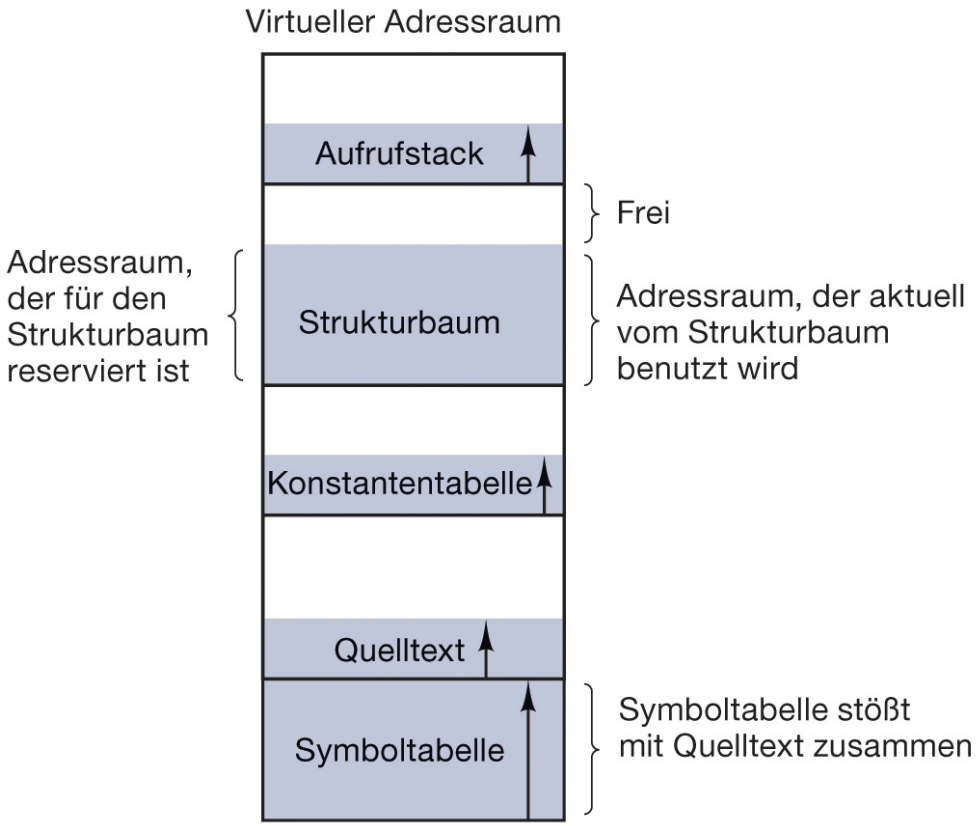
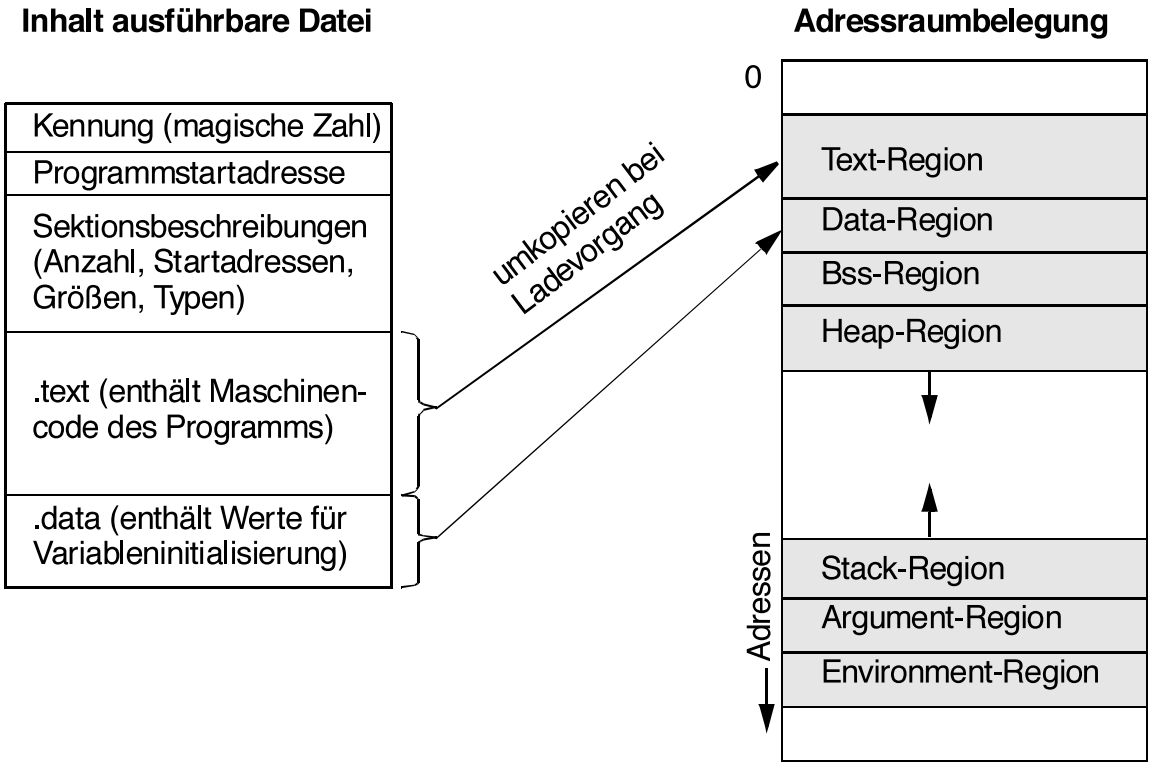


Abbildung 3.31: In einem eindimensionalen Adressraum können wachsende Tabellen kollidieren.

Segmentierung des virtuellen Adressraums

- Die innere Struktur einer ausführbaren Datei wird durch das Objektdateiformat (object file format) festgelegt.
- Daraus ergibt sich die Belegung des virtuellen Adressraums und nach Abbildung durch die Seitentabelle die Belegung im Arbeitsspeicher.



Segmentierung des virtuellen Adressraums

- Ein Segment bildet eine logische Einheit. Der Programmierer muss das Segment entsprechend nutzen.

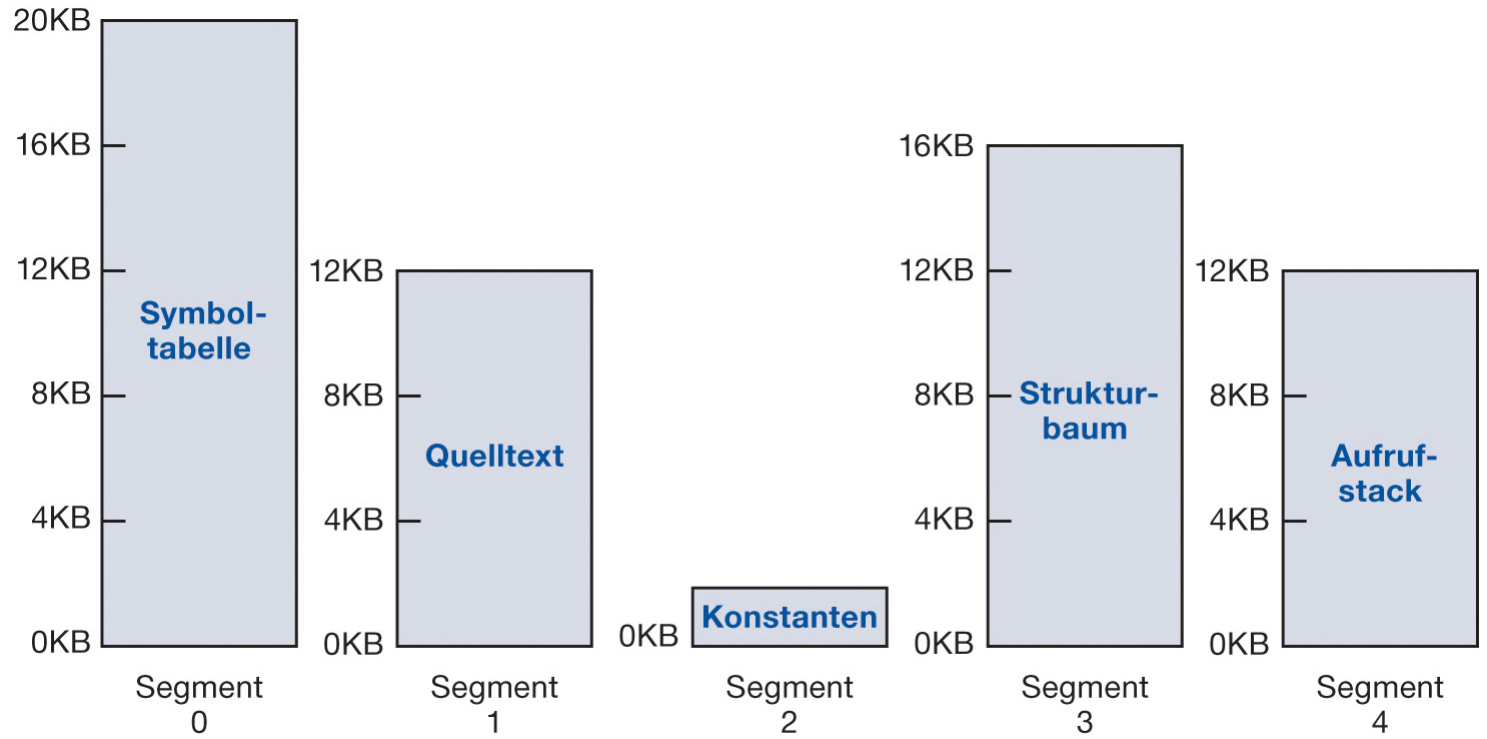


Abbildung 3.32: Segmentierter Speicher ermöglicht Tabellen, die unabhängig voneinander ihre Größe ändern

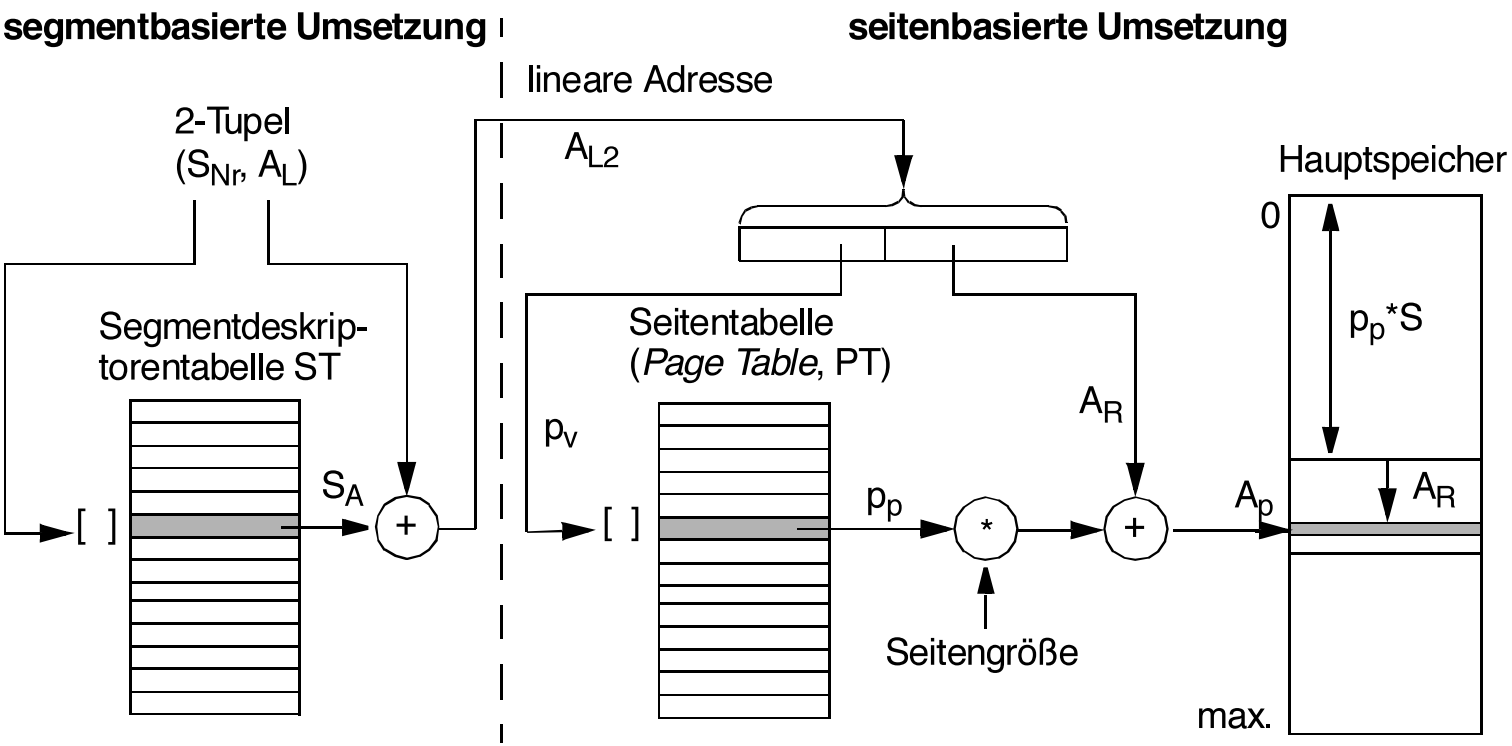
Segmentierung: Vorteile

- Einfachere Verwaltung von Datenstrukturen, die ihre Größe ändern.
- Einfachere Programmierung (z.B. Anfang einer Prozedur im Segment n, liegt immer in (n, 0).
- Modularität
- Speicherschutz (da durch die Segmente die Art des Codes und der Daten bestimmt ist)
- Gemeinsame Nutzung von Speicher wird erleichtert.

Segmentierung beim Pentium

- 2^{14} Segmente unterteilt in 2 Klassen, LDT und GDT (2^{13} Segmente pro Klasse).
- Jedes Programm hat seine eigene **LDT (lokale Deskriptortabelle**). Die **GDT (globale Deskriptortabelle**) beschreibt die Systemsegmente und das BS selbst.
- Jedes Segment hat einen virtuellen Adressraum von 4GB, organisiert in 32-Bit Wörtern, d.h. 2^{30} Adressen in jedem Segment.
- Die Adressumsetzung einer virtuellen Adresse ist logischerweise komplexer als bei einem 1-dimensionalen virtuellen Adressraum.
- Die Seitengröße ist 4KB und die virtuelle Adressumsetzung innerhalb eines Segmentes geschieht 2-stufig (1M 4KB -Seiten sind zu verwalten).
- Der Pentium besitzt 6 Segment-Register und die Umsetzung auf eine physische Adresse wird durch ein Mikroprogramm der HW durchgeführt.

Prinzip der Adressumsetzung



Zusammenfassung