

D.9 Zusammenfassung

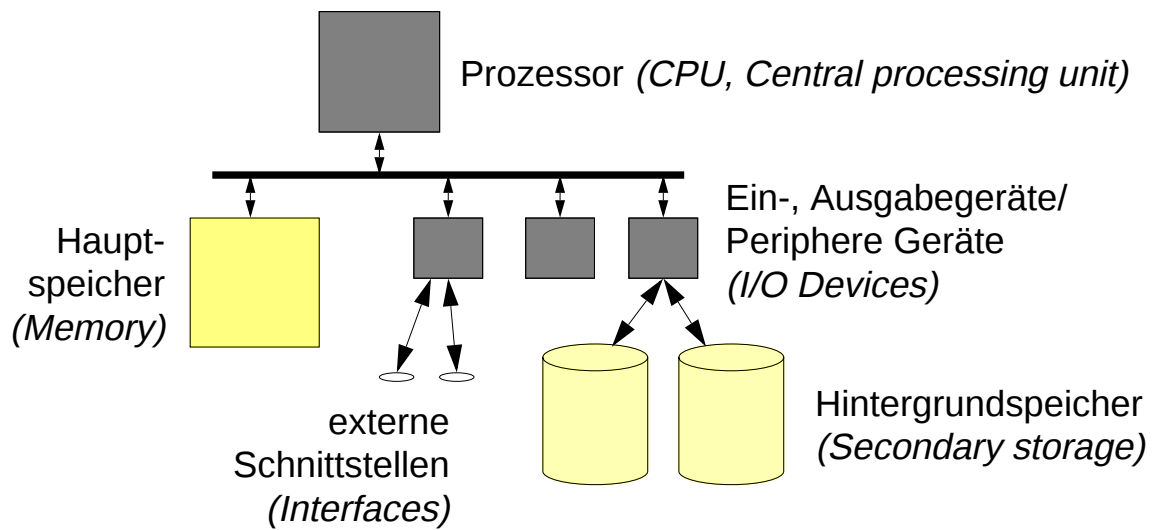
- Programmiermodell: Prozeß
 - ◆ Zerlegung von Anwendungen in Prozesse oder Threads
 - ◆ Ausnutzen von Wartezeiten; Time sharing–Betrieb
 - ◆ Prozeß hat verschiedene Zustände: laufend, bereit, blockiert etc.
- Auswahlstrategien für Prozesse
 - ◆ FCFS, SJF, PSJF, RR, MLFB
- Prozeßkommunikation
 - ◆ Pipes, Queues, Signals, Sockets, Shared memory, RPC
- Koordinierung von Prozessen
 - ◆ Einschränkung der gleichzeitigen Abarbeitung von Befehlsfolgen in nebenläufigen Prozessen/Aktivitätsträgern

D.9 Zusammenfassung (2)

- Gegenseitiger Ausschluß mit Spinlocks
- Klassische Koordinierungsprobleme und deren Lösung mit Semaphoren
 - ◆ Gegenseitiger Ausschluß
 - ◆ Bounded buffers
 - ◆ Leser-Schreiber-Probleme
 - ◆ Philosophenproblem
 - ◆ Schlafende Friseure
- UNIX Systemaufrufe
 - ◆ fork, exec, wait, nice
 - ◆ pipe
 - ◆ msgget, msgsnd, msgrcv
 - ◆ signal, kill, sigaction
 - ◆ semget, semop, semctl

E Speicherverwaltung

■ Betriebsmittel



SP I

Systemprogrammierung I

© Franz J. Hauck, Univ. Erlangen-Nürnberg, IMMD IV, 1998/1999

E-Memory.fm 1998-12-07 10.46

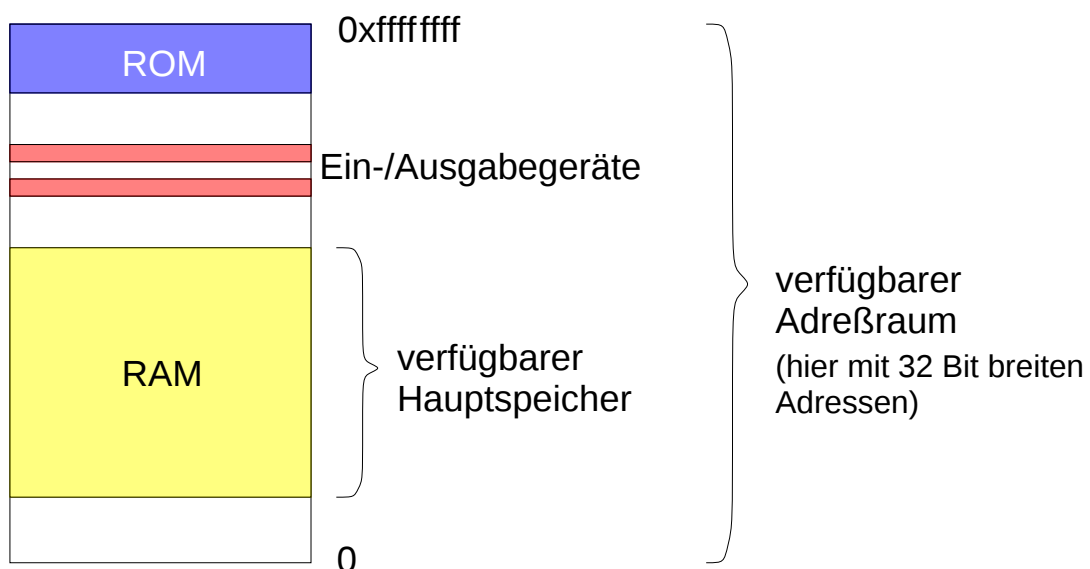
E.1

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

E.1 Speichervergabe

1 Problemstellung

■ Verfügbarer Speicher



SP I

Systemprogrammierung I

© Franz J. Hauck, Univ. Erlangen-Nürnberg, IMMD IV, 1998/1999

E-Memory.fm 1998-12-07 10.46

E.2

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Problemstellung (2)

■ Belegung des verfügbaren Hauptspeichers durch

◆ Benutzerprogramme

- Programmbefehle (Code, Binary)
- Programmdateien

◆ Betriebssystem

- Betriebssystemcode
- Puffer
- Systemvariablen

★ Zuteilung des Speichers nötig

SP I

Systemprogrammierung I

© Franz J. Hauck, Univ. Erlangen-Nürnberg, IMMD IV, 1998/1999

E-Memory.fm 1998-12-07 10.46

E.3

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Statische Speicherzuteilung

■ Feste Bereiche für Betriebssystem und Benutzerprogramm

▲ Probleme:

◆ Begrenzung anderer Ressourcen

(z.B. Bandbreite bei Ein-/Ausgabe wg. zu kleiner Systempuffer)

◆ Ungenutzter Speicher des Betriebssystems kann von

Anwendungsprogramm nicht genutzt werden und umgekehrt

★ Dynamische Speicherzuteilung einsetzen

SP I

Systemprogrammierung I

© Franz J. Hauck, Univ. Erlangen-Nürnberg, IMMD IV, 1998/1999

E-Memory.fm 1998-12-07 10.46

E.4

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Dynamische Speicherzuteilung

■ Segmente

- ◆ zusammenhängender Speicherbereich
(Bereich mit aufeinanderfolgenden Adressen)

■ Allokation (Anforderung) und Freigabe von Segmenten

■ Ein Anwendungsprogramm besitzt üblicherweise folgende Segmente:

- ## ◆ Datensegment

- ◆ Stacksegment (für Verwaltungsinformationen, z.B. bei Funktionsaufrufen)

▲ Suche nach geeigneten Speicherbereichen zur Zuteilung

★ Speicherzuteilungsstrategien nötig

SP I

Systemprogrammierung I

© Franz J. Hauck, Univ. Erlangen-Nürnberg, IMMD IV, 1998 / 1999

E-Memory.fm 1998-12-07 10.46

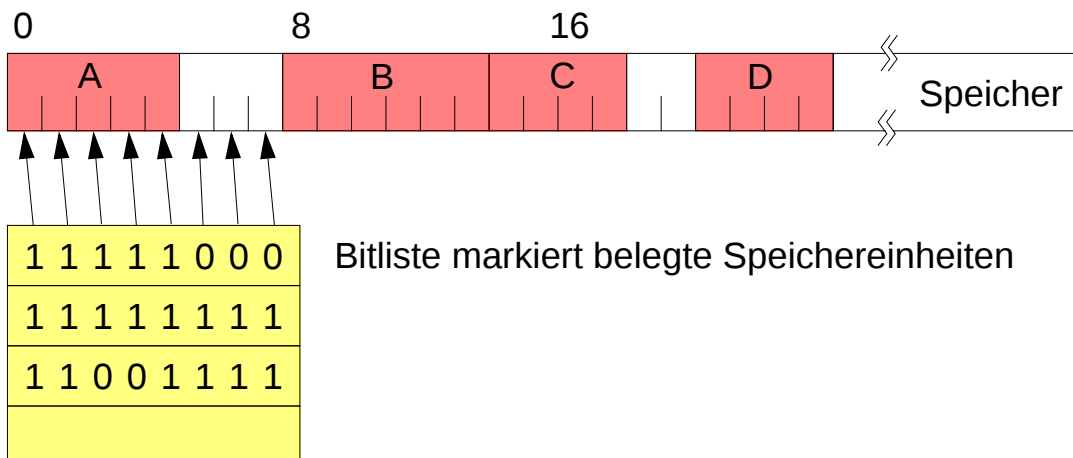
E.5

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Freispeicherverwaltung

- Freie (evtl. auch belegte) Segmente des Speichers müssen repräsentiert werden

■ Bitlisten



Bitliste markiert belegte Speichereinheiten

Speichereinheiten gleicher Größe (z.B. 1 Byte, 64 Byte, 1024 Byte)

SP I

Systemprogrammierung I

© Franz J. Hauck, Univ. Erlangen-Nürnberg, IMMD IV, 1998 / 1999

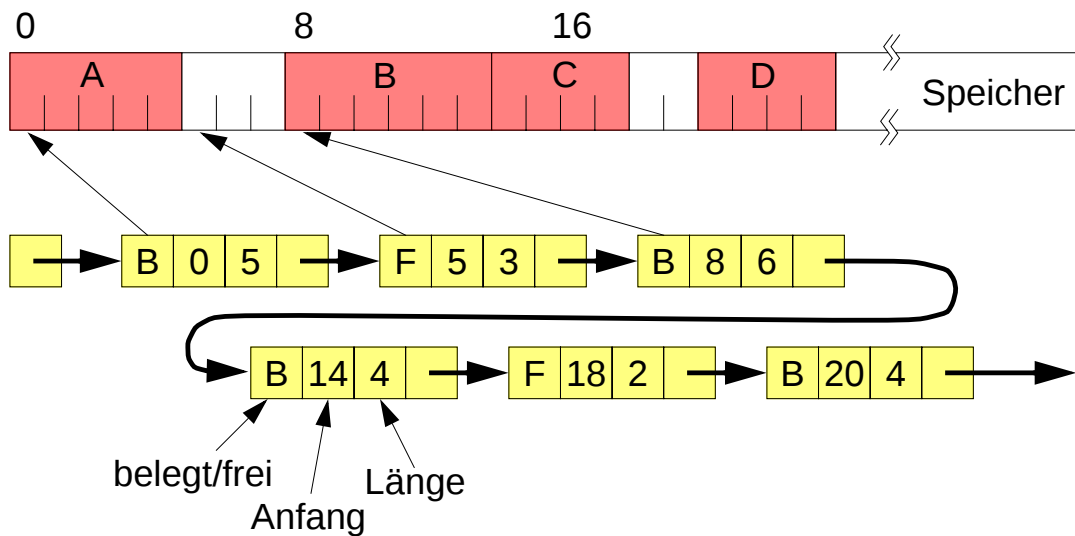
E-Memory.fm 1998-12-07 10.46

E.6

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Freispeicherverwaltung (2)

■ Verkettete Liste



Repräsentation auch von freien Segmenten

SPI

Systemprogrammierung I

© Franz J. Hauck, Univ. Erlangen-Nürnberg, IMMD IV, 1998/1999

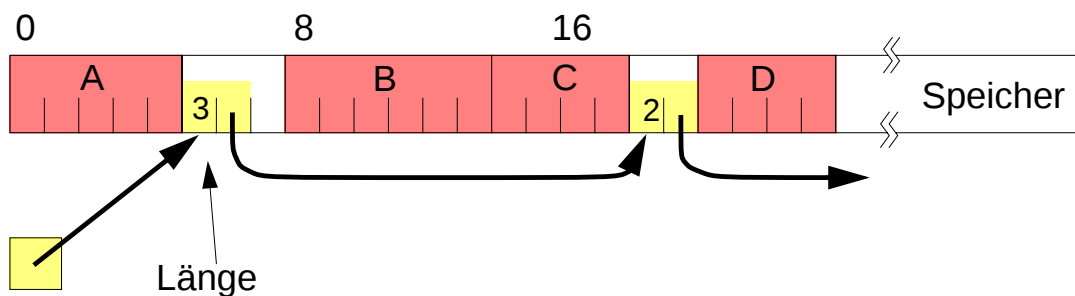
E-Memory.fm 1998-12-07 10.46

E.7

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Freispeicherverwaltung (3)

■ Verkettete Liste in dem freien Speicher



Mindestlückengröße muß garantiert werden

- Zur Effizienzsteigerung eventuell Rückwärtsverkettung nötig
- Repräsentation letztlich auch von der Vergabestrategie abhängig

SPI

Systemprogrammierung I

© Franz J. Hauck, Univ. Erlangen-Nürnberg, IMMD IV, 1998/1999

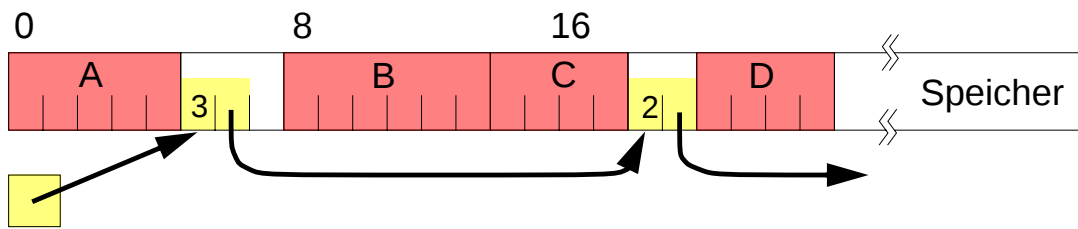
E-Memory.fm 1998-12-07 10.46

E.8

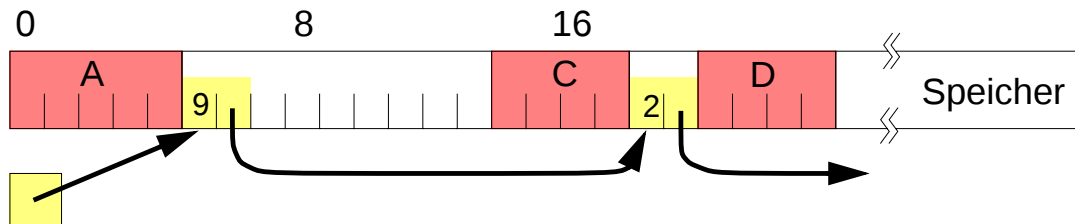
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Speicherfreigabe

■ Verschmelzung von Lücken



nach Freigabe von B:



6 Vergabestrategien

■ First Fit

◆ erste passende Lücke wird verwendet

■ Rotating First Fit / Next Fit

◆ wie First Fit aber Start bei der zuletzt zugewiesenen Lücke

■ Best Fit

◆ kleinste passende Lücke wird gesucht

■ Worst Fit

◆ größte passende Lücke wird gesucht

▲ Probleme:

◆ Speicherverschnitt

◆ zu kleine Lücken

7 Buddy Systeme

- Einteilung in dynamische Bereiche der Größe 2^n

	0	128	256	384	512	640	768	896	1024
									
	1024								
Anfrage 70	A	128	256	512					
Anfrage 35	A	B 64	256	512					
Anfrage 80	A	B 64	C 128	512					
Freigabe A	128	B 64	C 128	512					
Anfrage 60	128	B D	C 128	512					
Freigabe B	128	64 D	C 128	512					
Freigabe D	256	C 128	512						
Freigabe C	1024								

Effiziente Repräsentation der Lücken und effiziente Algorithmen

SPI

Systemprogrammierung I

© Franz J. Hauck, Univ. Erlangen-Nürnberg, IMMD IV, 1998/1999

E-Memory.fm 1998-12-07 10.46

E.11

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

8 Einsatz der Verfahren

- Einsatz im Betriebssystem
 - ◆ Verwaltung des Systemspeichers
 - ◆ Zuteilung von Speicher an Prozesse und Betriebssystem
- Einsatz innerhalb eines Prozesses
 - ◆ Verwaltung des Haldenspeichers (*Heap*)
 - ◆ erlaubt dynamische Allokation von Speicherbereichen durch den Prozeß (**malloc** und **free**)
- Einsatz für Bereiche des Sekundärspeichers
 - ◆ Verwaltung bestimmter Abschnitte des Sekundärspeichers
z.B. Speicherbereich für Prozeßauslagerungen (*Swap space*)

SPI

Systemprogrammierung I

© Franz J. Hauck, Univ. Erlangen-Nürnberg, IMMD IV, 1998/1999

E-Memory.fm 1998-12-07 10.46

E.12

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

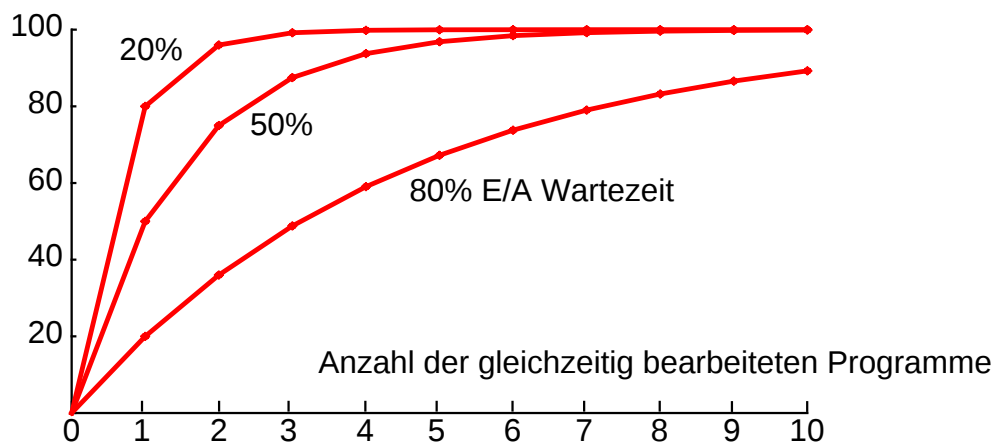
E.2 Mehrprogrammbetrieb

1 Problemstellung

■ Mehrere Prozesse laufen gleichzeitig

- ◆ Wartezeiten von Ein-/Ausgabeoperationen ausnutzen
- ◆ CPU Auslastung verbessern

■ CPU-Nutzung in Prozent, abhängig von der Anzahl der Prozesse



SPI

Systemprogrammierung I

© Franz J. Hauck, Univ. Erlangen-Nürnberg, IMMD IV, 1998/1999

E-Memory.fm 1998-12-07 10.46

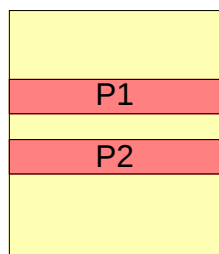
E.13

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Problemstellung (2)

▲ Mehrere Prozesse benötigen Hauptspeicher

- ◆ Prozesse an verschiedenen Stellen im Hauptspeicher liegen
- ◆ Speicher reicht eventuell nicht für alle Prozesse
- ◆ Schutzbedürfnis des Betriebssystems und der Prozesse untereinander



zwei Prozesse und deren Codesegmente im Speicher

★ Relokation von Programmbefehlen (Binaries)

★ Ein- und Auslagern von Prozessen

★ Hardwareunterstützung

SPI

Systemprogrammierung I

© Franz J. Hauck, Univ. Erlangen-Nürnberg, IMMD IV, 1998/1999

E-Memory.fm 1998-12-07 10.46

E.14

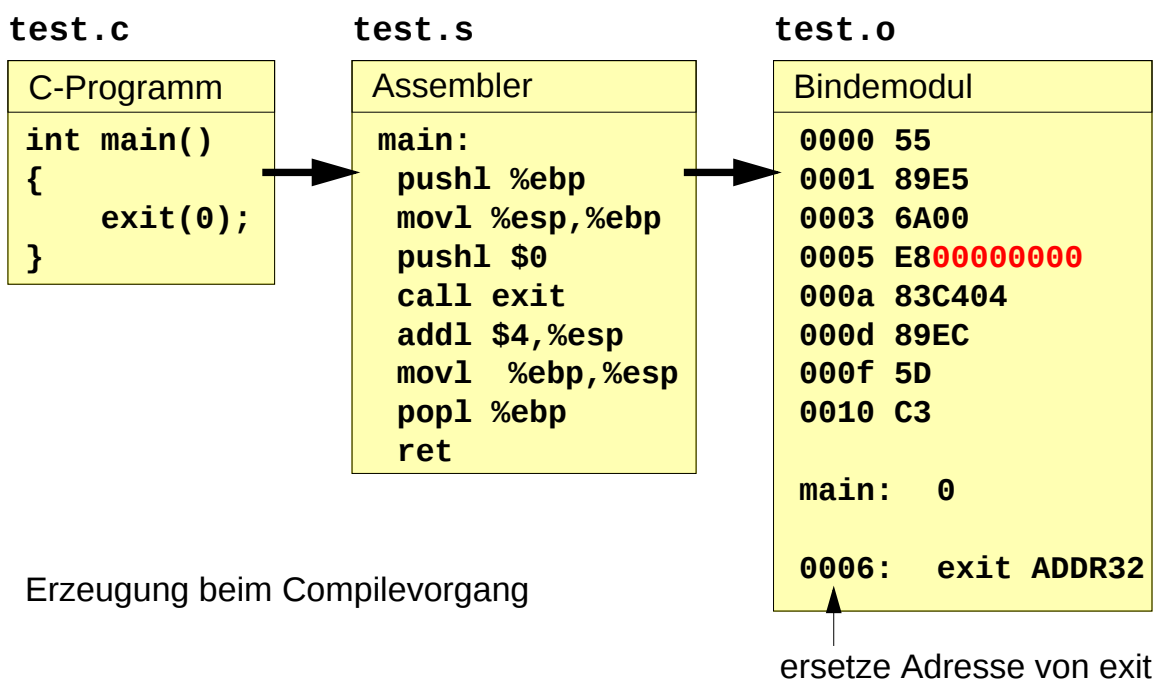
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Relokation

- Festlegung absoluter Speicheradressen in den Programmbefehlen
 - ◆ z.B. ein Sprungbefehl in ein Unterprogramm oder ein Ladebefehl für eine Variable aus dem Datensegment
- Absolutes Binden (*Compile time*)
 - ◆ Adressen stehen fest
 - ◆ Programm kann nur an bestimmter Speicherstelle korrekt ablaufen
- Statisches Binden (*Load time*)
 - ◆ Beim Laden (Starten) des Programms werden die absoluten Adressen angepaßt (reloziert)
 - ◆ Relokationsinformation nötig, die vom Compiler oder Assembler geliefert wird

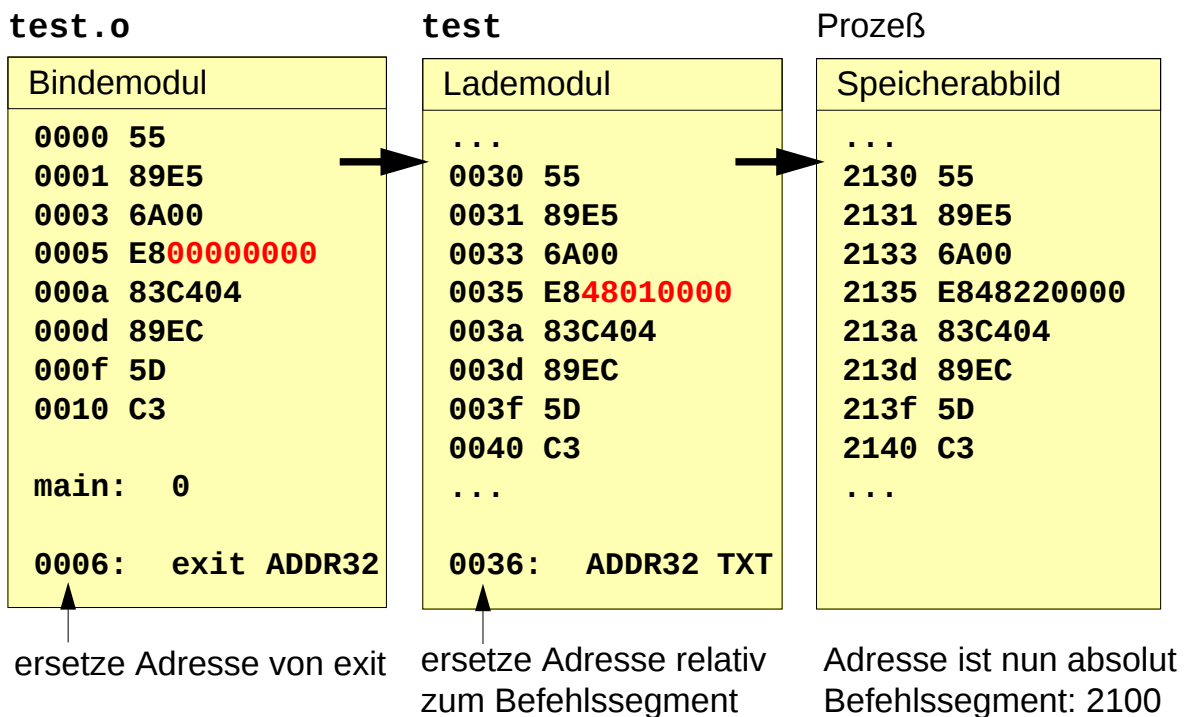
2 Relokation (2)

- Compilervorgang (Erzeugung der Relokationsinformation)



2 Relokation (3)

■ Binde- und Ladevorgang



2 Relokation (4)

■ Relokationsinformation im Bindemodul

- ◆ erlaubt das Binden von Modulen in beliebige Programme

■ Relokationsinformation im Lademodul

- ◆ erlaubt das Laden des Programms an beliebige Speicherstellen
- ◆ absolute Adressen werden erst beim Laden generiert

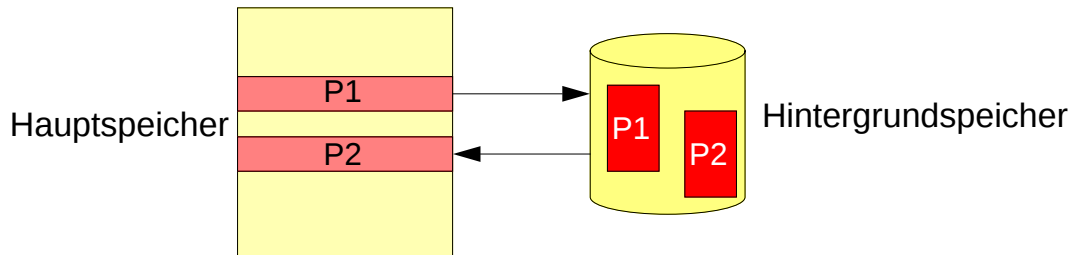
▲ Alternative

- ◆ Programm benutzt keine absoluten Adressen und kann daher immer an beliebige Speicherstellen geladen werden

3 Ein-, Auslagerung (Swapping)

- Segmente eines Prozesses werden auf Hintergrundspeicher ausgelagert und im Hauptspeicher freigegeben
 - ◆ z.B. zur Überbrückung von Wartezeiten bei E/A oder Round-Robin Schedulingstrategie

- Einlagern der Segmente in den Hauptspeicher am Ende der Wartezeit

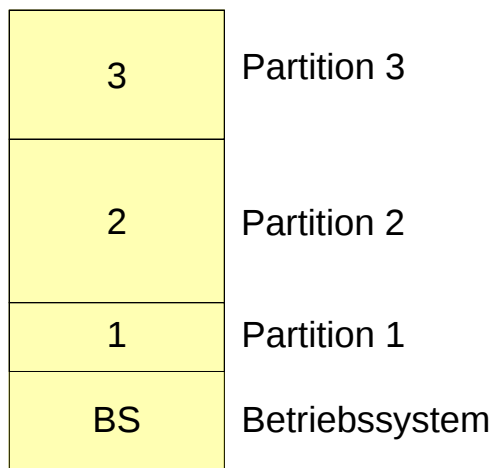


- ▲ Aus-, Einlagerzeit ist hoch
 - ◆ Latenzzeit der Festplatte
 - ◆ Übertragungszeit

3 Ein-, Auslagerung (2)

- ▲ Prozeß ist statisch gebunden
 - ◆ kann nur an gleiche Stelle im Hauptspeicher wieder eingelagert werden
 - ◆ Kollisionen mit eventuell neu im Hauptspeicher befindlichen Segmenten

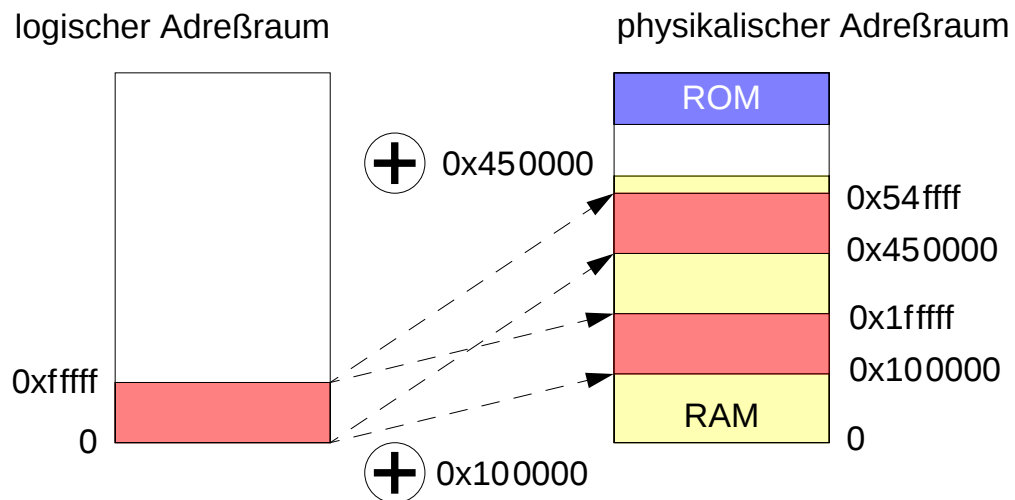
- Mögliche Lösung: Partitionierung des Hauptspeichers



- ◆ In jeder Partition läuft nur ein Prozeß
- ◆ Einlagerung erfolgt wieder in die gleiche Partition
- ◆ Speicher kann nicht optimal genutzt werden

4 Segmentierung

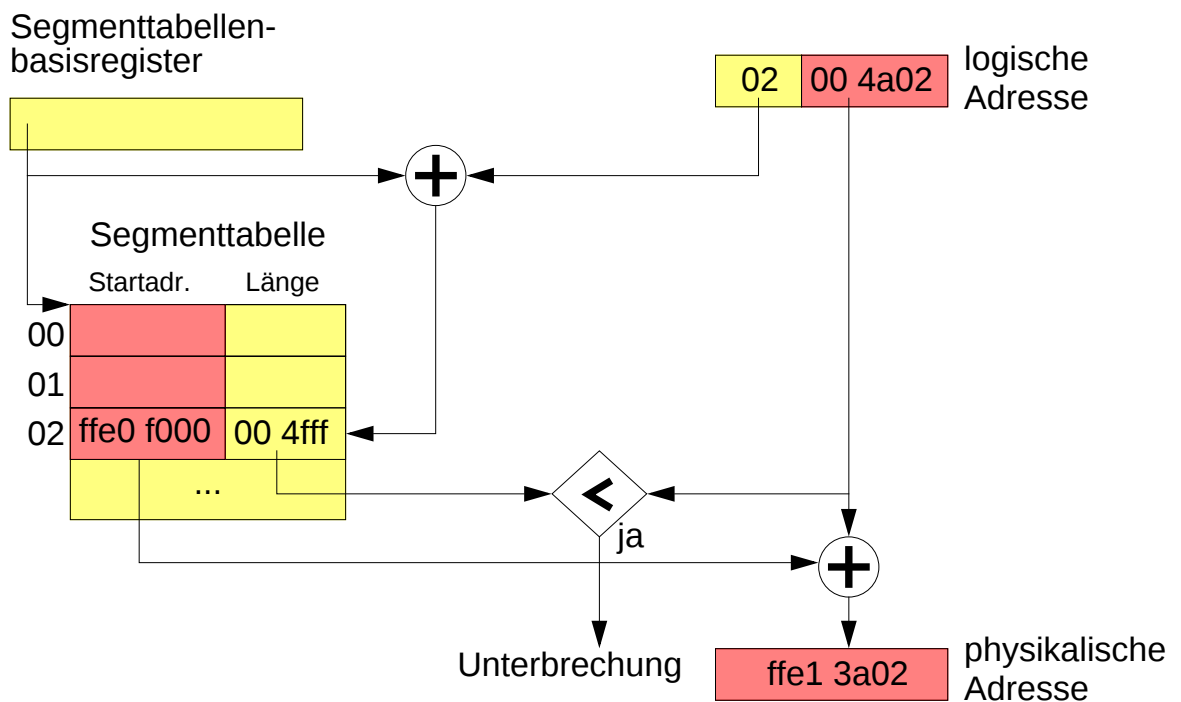
- Hardwareunterstützung: Umsetzung logischer in physikalische Adressen
 - ◆ Prozesse erhalten einen logischen Adreßraum



Das Segment im logischen Adreßraum kann an jeder beliebige Stelle im physikalischen Adressraum liegen.

4 Segmentierung (2)

- Realisierung mit Übersetzungstabelle



4 Segmentierung (3)

- Hardware wird MMU (*Memory management unit*) genannt
- Schutz vor Segmentübertretung
 - ◆ Unterbrechung zeigt Speicherverletzung an
 - ◆ Programme und Betriebssystem voreinander geschützt
- Prozeßumschaltung durch Austausch der Segmentbasis
 - ◆ jeder Prozeß hat eigene Übersetzungstabelle
- Ein- und Auslagerung vereinfacht
 - ◆ nach Einlagerung an beliebige Stelle muß lediglich die Übersetzungstabelle angepaßt werden
- Gemeinsame Segmente möglich
 - ◆ Befehlssegmente
 - ◆ Datensegmente (*Shared memory*)

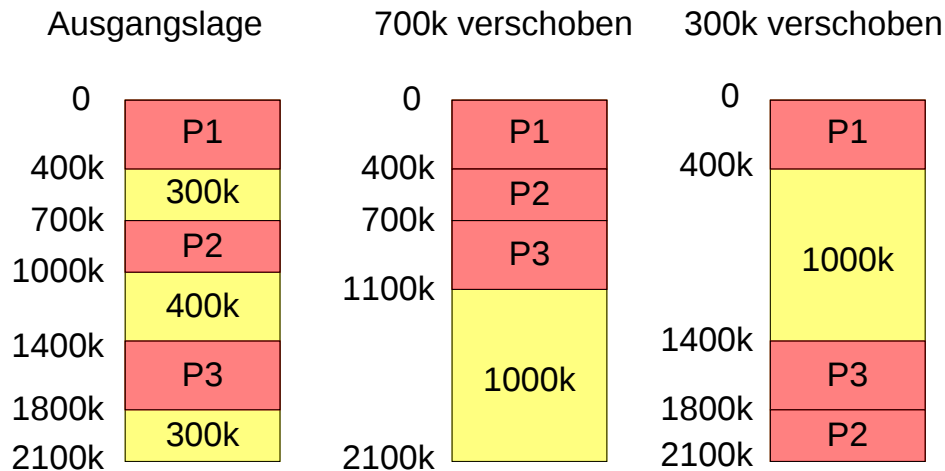
4 Segmentierung (4)

- Zugriffsschutz einfach integrierbar
 - ◆ z.B. Rechte zum Lesen, Schreiben und Ausführen von Befehlen, die von der MMU geprüft werden
- ▲ Fragmentierung des Speichers durch häufiges Ein- und Auslagern
 - ◆ es entstehen kleine, nicht nutzbare Lücken
- ★ Kompaktifizieren
 - ◆ Segmente werden verschoben, um Lücken zu schließen; Segmenttabelle wird jeweils angepaßt
- ▲ lange E/A Zeiten für Ein- und Auslagerung
 - ◆ nicht alle Teile eines Segments werden gleich häufig genutzt

5 Kompaktifizieren

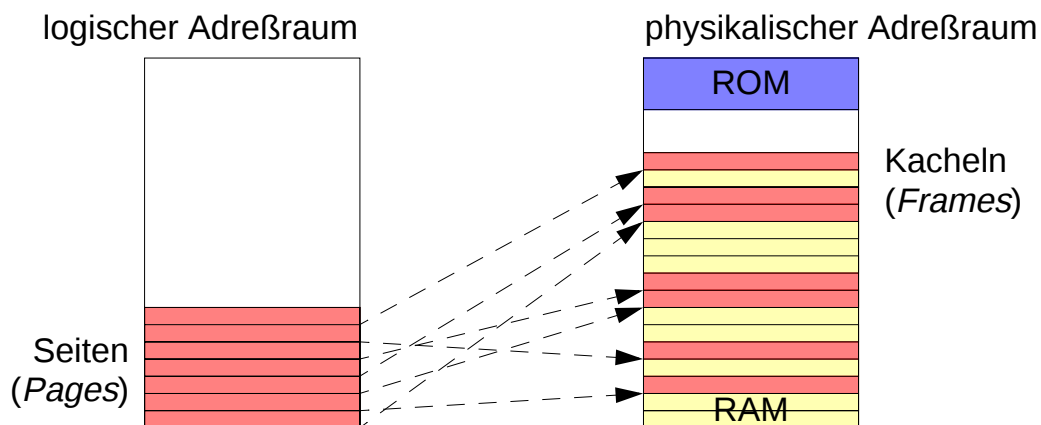
■ Verschieben von Segmenten

- ◆ Erzeugen von weniger aber größeren Lücken
- ◆ Verringern des Verschnitts
- ◆ aufwendige Operation, abhängig von der Größe der verschobenen Segmente



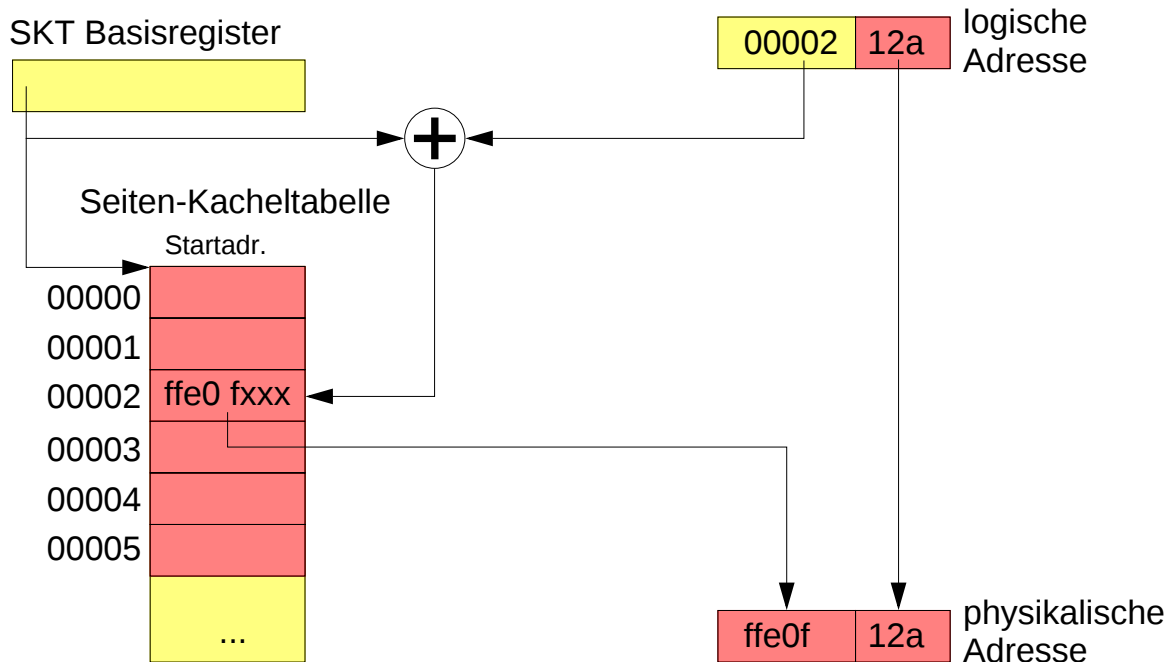
E.3 Seitenadressierung (*Paging*)

- Einteilung des logischen Adreßraums in gleichgroße Seiten, die an beliebigen Stellen im physikalischen Adreßraum liegen können
- ◆ Lösung des Fragmentierungsproblem
- ◆ keine Kompaktifizierung mehr nötig
- ◆ Vereinfacht Speicherbelegung und Ein-, Auslagerungen



1 MMU mit Seiten-Kacheltabelle

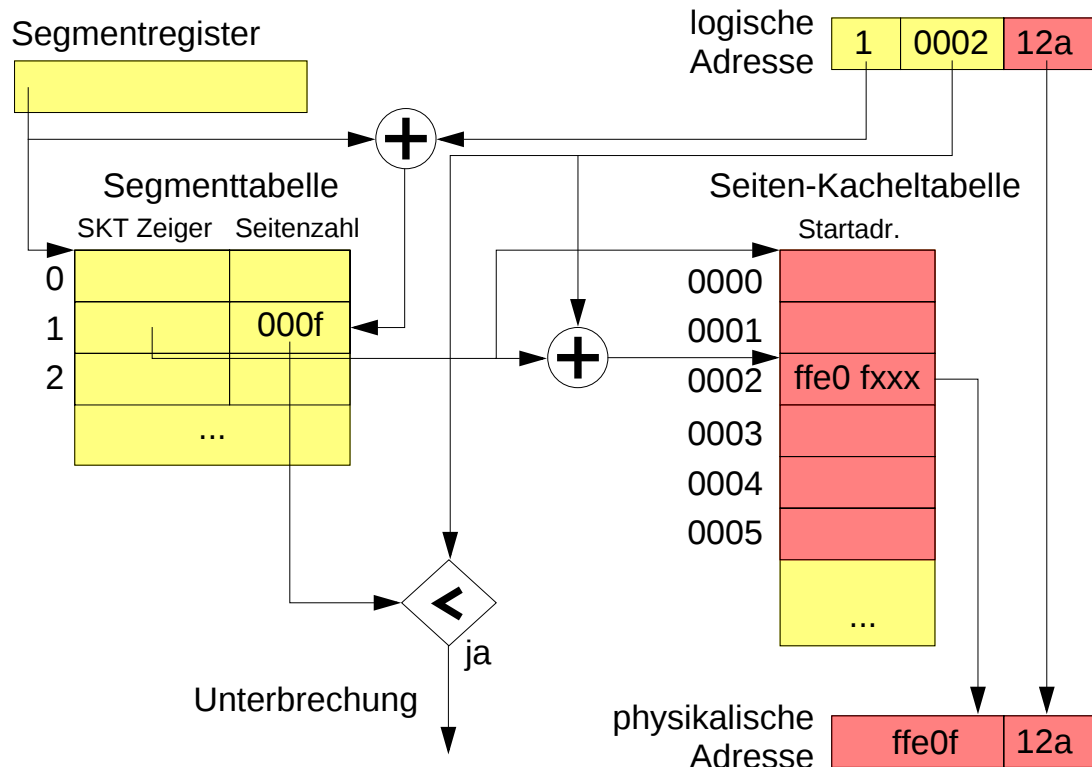
- Tabelle setzt Seiten in Kacheln um



1 MMU mit Seiten-Kacheltabelle (2)

- ▲ Seitenadressierung erzeugt internen Verschnitt
 - ◆ letzte Seite eventuell nicht vollständig genutzt
- Seitengröße
 - ◆ kleine Seiten verringern internen Verschnitt, vergrößern aber die Seiten-Kacheltabelle (und umgekehrt)
 - ◆ übliche Größen: 512 Bytes — 8192 Bytes
- ▲ große Tabelle, die im Speicher gehalten werden muß
- ▲ viele implizite Speicherzugriffe nötig
- ▲ nur ein „Segment“ pro Kontext
- ★ Kombination mit Segmentierung

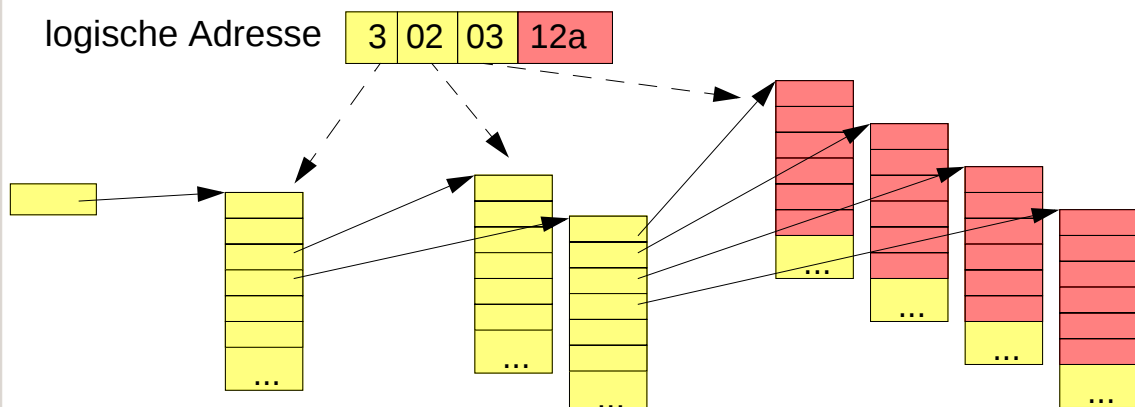
2 Segmentierung und Seitenadressierung



2 Segmentierung und Seitenadressierung (2)

- ▲ noch mehr implizite Speicherzugriffe
- ▲ große Tabellen im Speicher
- ★ Mehrstufige Seitenadressierung mit Ein- und Auslagerung

3 Mehrstufige Seitenadressierung



3 Mehrstufige Seitenadressierung (2)

■ Ein- und Auslagerung von Seiten

Seiten-Kacheltabelle

	Startadr.	Präsenzbit
0000		
0001		
0002	ffe0 fxxx	X
...		

- ◆ Ist das Präsenzbit gesetzt, bleibt alles wie bisher.
- ◆ Ist das Präsenzbit gelöscht, wird eine Unterbrechung ausgelöst (*Page fault*).
- ◆ Die Unterbrechungsbehandlung kann nun für das Laden der Seite vom Hintergrundspeicher sorgen und den Speicherzugriff danach wiederholen (benötigt HW Support in der CPU).

■ Präsenzbit auch für jeden Eintrag in den höheren Stufen

- ◆ Tabellen sind aus- und einlagerbar

▲ Noch mehr implizite Speicherzugriffe

SPI

Systemprogrammierung I

© Franz J. Hauck, Univ. Erlangen-Nürnberg, IMMD IV, 1998/1999

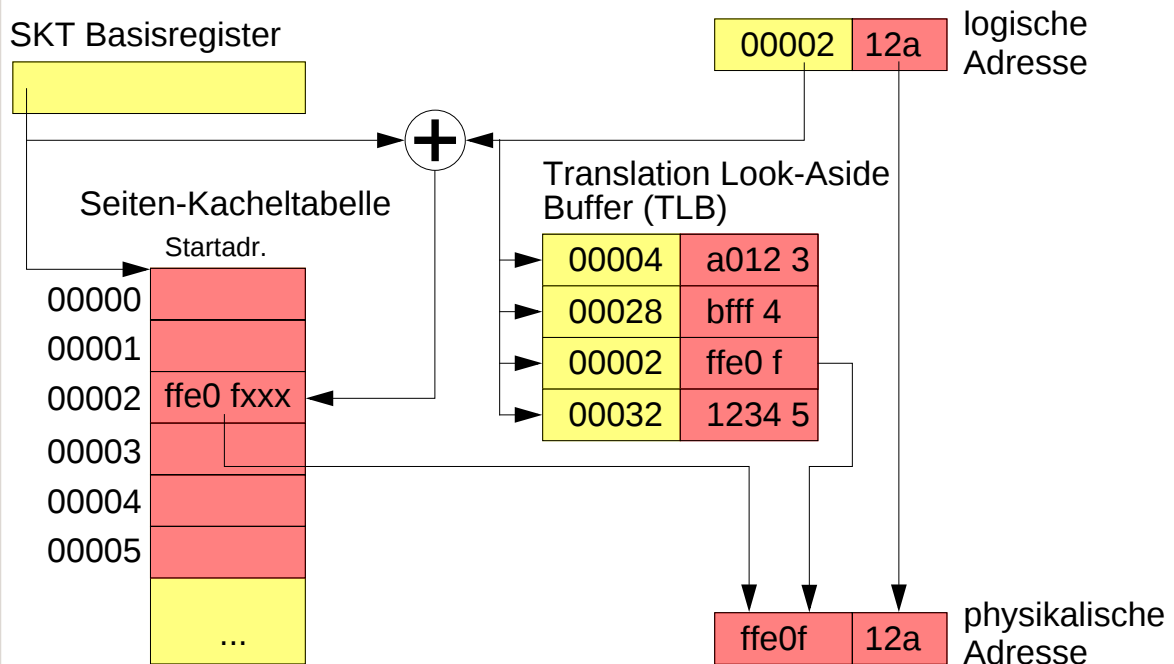
E-Memory.fm 1998-12-07 10.46

E.31

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Translation Look-Aside Buffer

■ Schneller Registersatz wird konsultiert bevor auf die SKT zugegriffen wird



SPI

Systemprogrammierung I

© Franz J. Hauck, Univ. Erlangen-Nürnberg, IMMD IV, 1998/1999

E-Memory.fm 1998-12-07 10.46

E.32

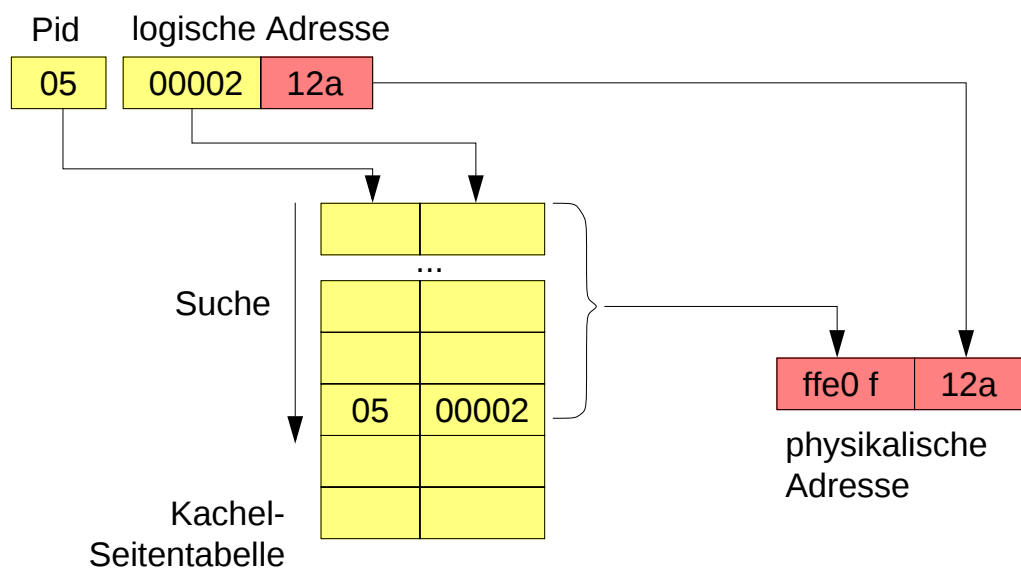
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Translation Look-Aside Buffer (2)

- Schneller Zugriff auf Seitenabbildung, falls Information im voll-assoziativen Speicher des TLB
 - ◆ keine impliziten Speicherzugriffe nötig
- Bei Kontextwechseln muß TLB gelöscht werden (*Flush*)
- Bei Zugriffen auf eine nicht im TLB enthaltene Seite wird die entsprechende Zugriffsinformation in den TLB eingetragen
 - ◆ Ein alter Eintrag muß zur Ersetzung ausgesucht werden
- TLB Größe
 - ◆ Pentium: Daten TLB = 64, Code TLB = 32, Seitengröße 4K
 - ◆ Sparc V9: Daten TLB = 64, Code TLB = 64, Seitengröße 8K
 - ◆ Größere TLBs bei den üblichen Taktraten zur Zeit nicht möglich

5 Invertierte Seiten-Kacheltabelle

- Zum Umsetzen der Adressen nur Abbildung der belegten Kacheln nötig
 - ◆ eine Tabelle, die zu jeder Kachel die Seitenabbildung hält



5 Invertierte Seiten-Kacheltabelle (2)

■ Vorteile

- ◆ wenig Platz zur Speicherung der Abbildung notwendig
- ◆ Tabelle kann immer im Hauptspeicher gehalten werden

▲ Nachteile

- ◆ prozeßlokale SKT zusätzlich nötig für Seiten, die ausgelagert sind
 - diese können aber ausgelagert werden
- ◆ Suche in der KST ist aufwendig
 - Einsatz von Assoziativspeichern und Hashfunktionen

6 Systemaufruf

■ Ermitteln der Seitengröße des Betriebssystems

```
int getpagesize(void);
```

E.4 Fallstudie: Pentium

■ Physikalische Adresse

- ◆ 32 bit breit

■ Segmente

- ◆ CS – Codesegment: enthält Instruktionen
- ◆ DS – Datensegment
- ◆ SS – Stacksegment
- ◆ ES, FS, GS – zusätzliche Segmente
- ◆ Befehle beziehen sich auf eines oder mehrere der Segmente

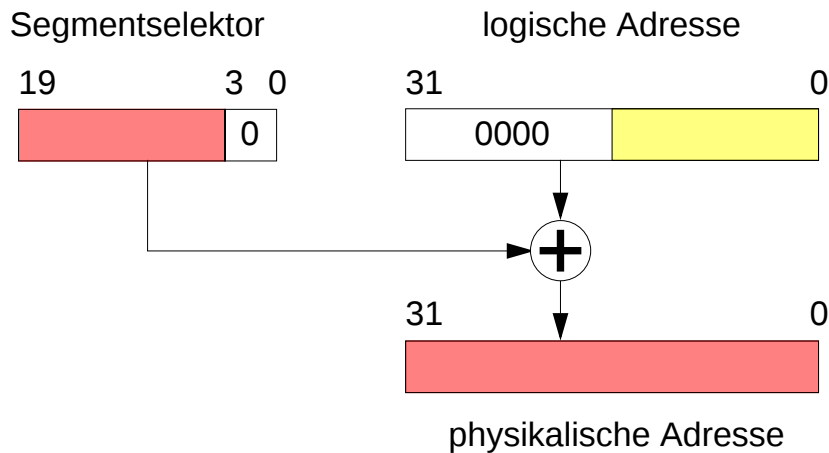
■ Segmentadressierung

- ◆ Segmentselektor zur Auswahl eines Segments:
 - 16 bit bezeichnen das Segment

1 Real Mode Adressierung

■ Adreßgenerierung im Real Mode

- ◆ 16 bit breiter Segmentelektor wird als 20 bit breite Adresse interpretiert und auf die logische Adresse addiert



2 Protected Mode Adressierung

■ Vier Betriebsmodi (Stufen von Privilegien)

- ◆ Stufe 0: höchste Privilegien (privilegierte Befehle, etc.): BS Kern
 - ◆ Stufe 1: BS Treiber
 - ◆ Stufe 2: BS Erweiterungen
 - ◆ Stufe 3: Benutzerprogramme
-
- ◆ Speicherverwaltung kann nur in Stufe 0 konfiguriert werden

■ Segmentelektoren enthalten Privilegiierungsstufe

- ◆ Stufe des Codesegments entscheidet über Zugriffserlaubnis

■ Segmentelektoren werden als Indizes interpretiert

- ◆ Tabellen von Segmentdeskriptoren
 - Globale Deskriptor Tabelle
 - Lokale Deskriptor Tabelle

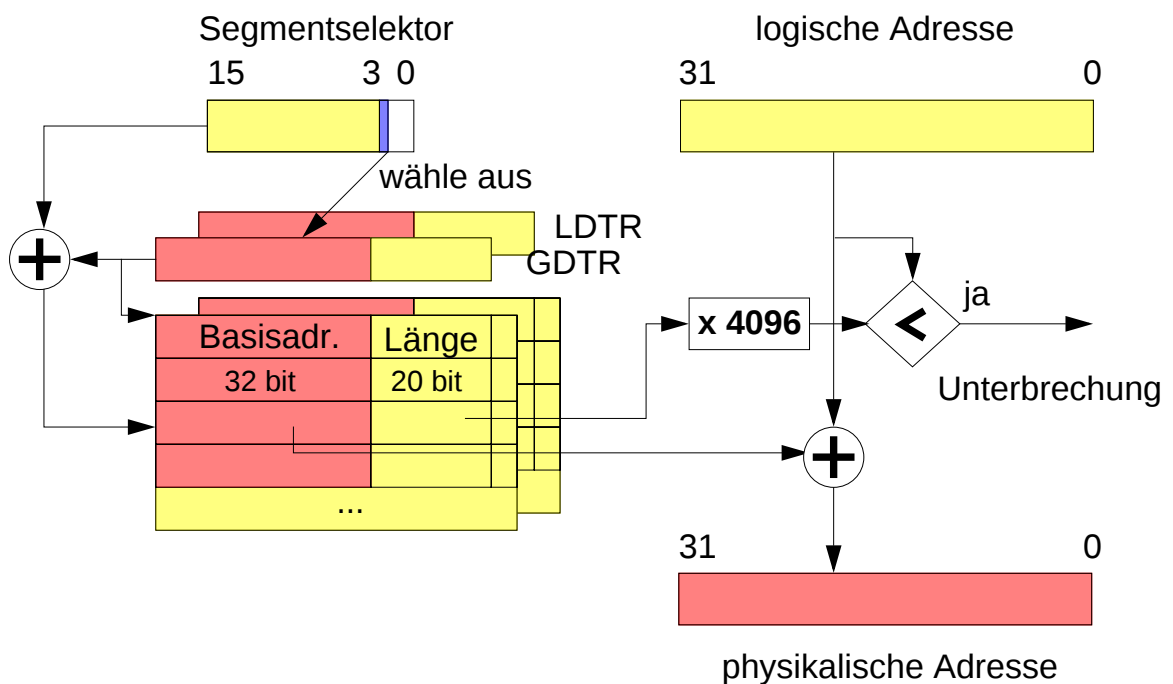
2 Protected Mode Adressierung (2)

■ Deskriptortabelle

- ◆ enthält bis zu 8192 Segmentdeskriptoren
- ◆ Inhalt des Segmentdeskriptors:
 - physikalische Basisadresse
 - Längenangabe
 - Granularität (Angaben für Bytes oder Seiten)
 - Präsenzbit
 - Privilegierungsstufe
- ◆ globale Deskriptortabelle für alle Prozesse zugänglich (Register GDTR)
- ◆ lokale Deskriptortabelle pro Prozeß möglich (Register LDTR gehört zum Prozeßkontext)

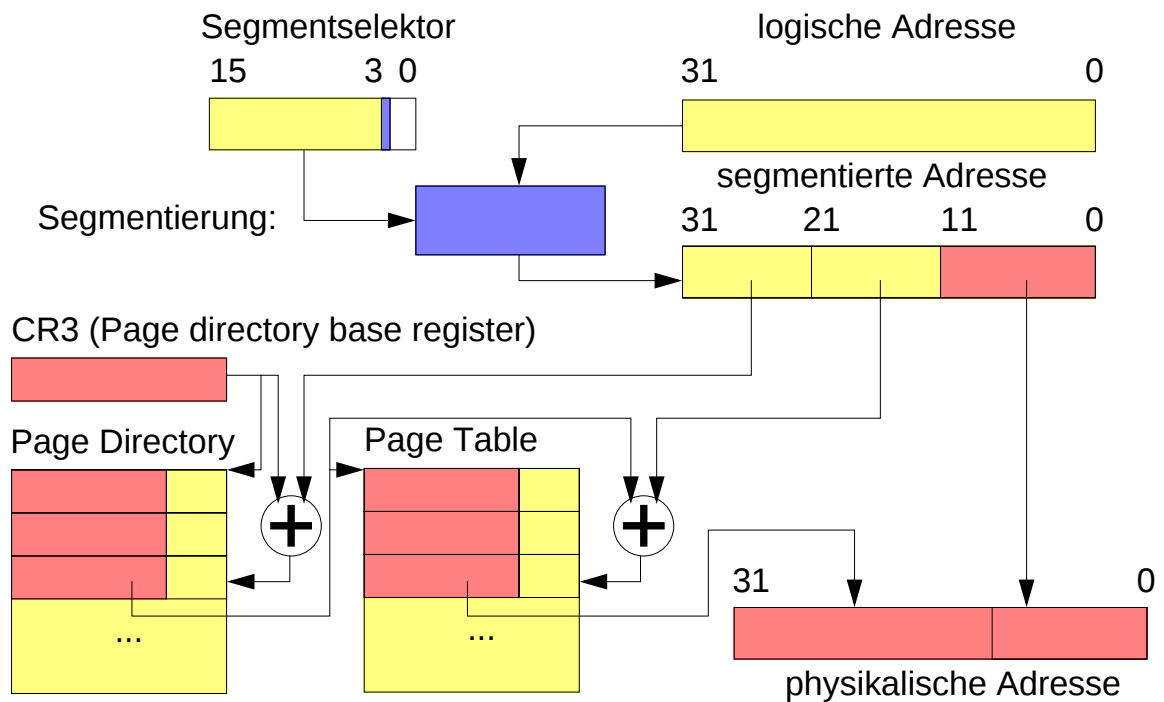
3 Adreßberechnung bei Segmentierung

■ Verwendung der Protected mode Adressierung



4 Adreßberechnung bei Paging

- Seitenadressierung wird der Segmentierung nachgeschaltet



SPI

Systemprogrammierung I

© Franz J. Hauck, Univ. Erlangen-Nürnberg, IMMD IV, 1998/1999

E-Memory.fm 1998-12-07 10.46

E.41

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Adreßberechnung bei Paging

- Zweistufige Seitenadressierung
 - ◆ Directory — Page table
 - ◆ Seitengröße fest auf 4096 Bytes
- Inhalt des Seitendeskriptor
 - ◆ Kacheladresse
 - ◆ Dirty Bit: Seite wurde beschrieben
 - ◆ Access Bit: Seite wurde gelesen oder geschrieben
 - ◆ Schreibschutz: Seite nur lesbar
 - ◆ Präsenzbit: Seite ausgelagert (31 Bits für BS-Informationen nutzbar)
 - ◆ Kontrolle des Prozessorcaches
- Getrennte TLBs für Codesegment und Datensegmente
 - ◆ 64 Einträge für Datenseiten; 32 Einträge für Codeseiten

SPI

Systemprogrammierung I

© Franz J. Hauck, Univ. Erlangen-Nürnberg, IMMD IV, 1998/1999

E-Memory.fm 1998-12-07 10.46

E.42

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

E.5 Gemeinsamer Speicher (*Shared Memory*)

- Speicher, der mehreren Prozessen zur Verfügung steht
 - ◆ gemeinsame Segmente (gleiche Einträge in verschiedenen Segmenttabellen)
 - ◆ gemeinsame Seiten (gleiche Einträge in verschiedenen SKTs)
 - ◆ gemeinsame Seitenbereiche (gemeinsames Nutzen einer SKT bei mehrstufigen Tabellen)
- Gemeinsamer Speicher wird beispielsweise benutzt für
 - ◆ Kommunikation zwischen Prozessen
 - ◆ gemeinsame Befehlssegmente

E.5 Gemeinsamer Speicher (2)

- Systemaufrufe unter Solaris 2.5
 - ◆ Erzeugen bzw. Holen eines gemeinsamen Speichersegments

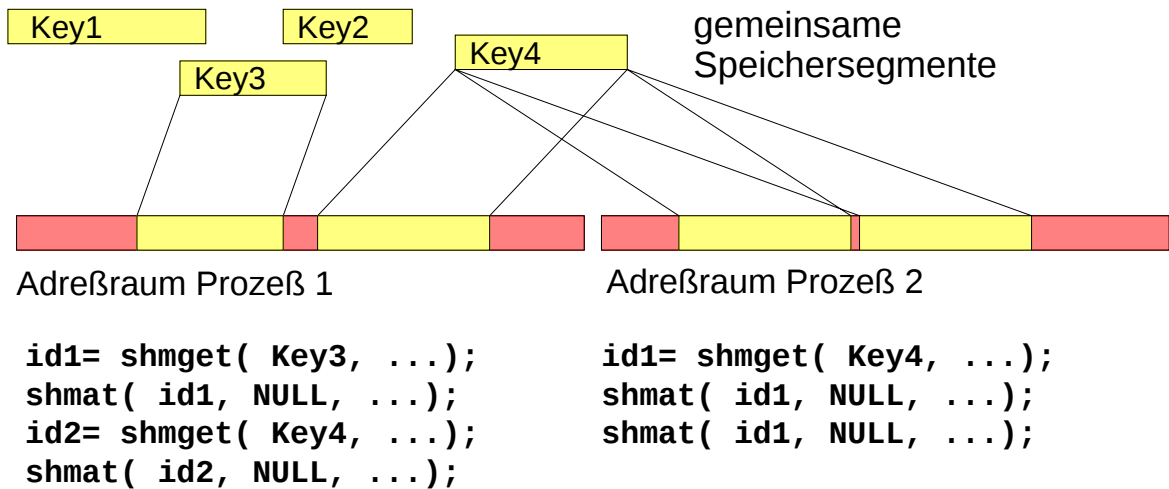
```
int shmget( key_t key, int size, int shmflg );
```
 - ◆ Einblenden und Ausblenden des Segments in den Speicher

```
void *shmat( int shmid, void *shmaddr, int shmflg );
int shmdt( void *shmaddr );
```
 - ◆ Kontrolloperation

```
int shmctl( int shmid, int cmd, struct shmid_ds *buf );
```

E.5 Gemeinsamer Speicher (3)

■ Prinzip der shm* Operationen



E.5 Gemeinsamer Speicher (4)

■ Verwendung des Keys

- ◆ Alle Prozesse, die auf ein Speichersegment zugreifen wollen, müssen den Key kennen
- ◆ Keys sind eindeutig innerhalb eines (Betriebs-)Systems
- ◆ Ist ein Key bereits vergeben, kann kein Segment mit gleichem Key erzeugt werden
- ◆ Ist ein Key bekannt, kann auf das Segment zugegriffen werden
 - gesetzte Zugriffsberechtigungen werden allerdings beachtet
- ◆ Es können Segmente ohne Key erzeugt werden (private Segmente)

■ Keys werden benutzt für:

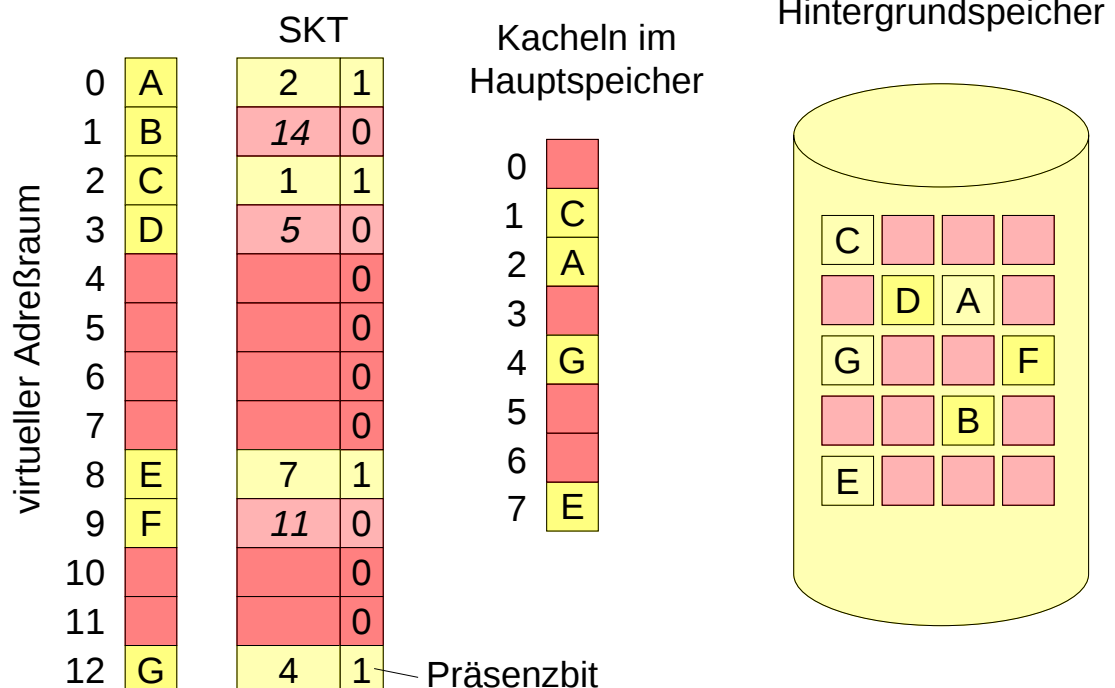
- ◆ Queues
- ◆ Semaphore
- ◆ Shared memory segments

E.6 Virtueller Speicher

- Entkoppelung des Speicherbedarfs vom verfügbaren Hauptspeicher
 - ◆ Prozesse benötigen nicht alle Speicherstellen gleich häufig
 - bestimmte Befehle werden selten oder gar nicht benutzt (z.B. Fehlerbehandlungen)
 - bestimmte Datenstrukturen werden nicht voll belegt
 - ◆ Prozesse benötigen evtl. mehr Speicher als Hauptspeicher vorhanden
- Idee
 - ◆ Vortäuschen eines großen Hauptspeichers
 - ◆ Einblenden benötigter Speicherbereiche
 - ◆ Abfangen von Zugriffen auf nicht-eingeblendete Bereiche
 - ◆ Bereitstellen der benötigten Bereiche auf Anforderung
 - ◆ Auslagern nicht-benötigter Bereiche

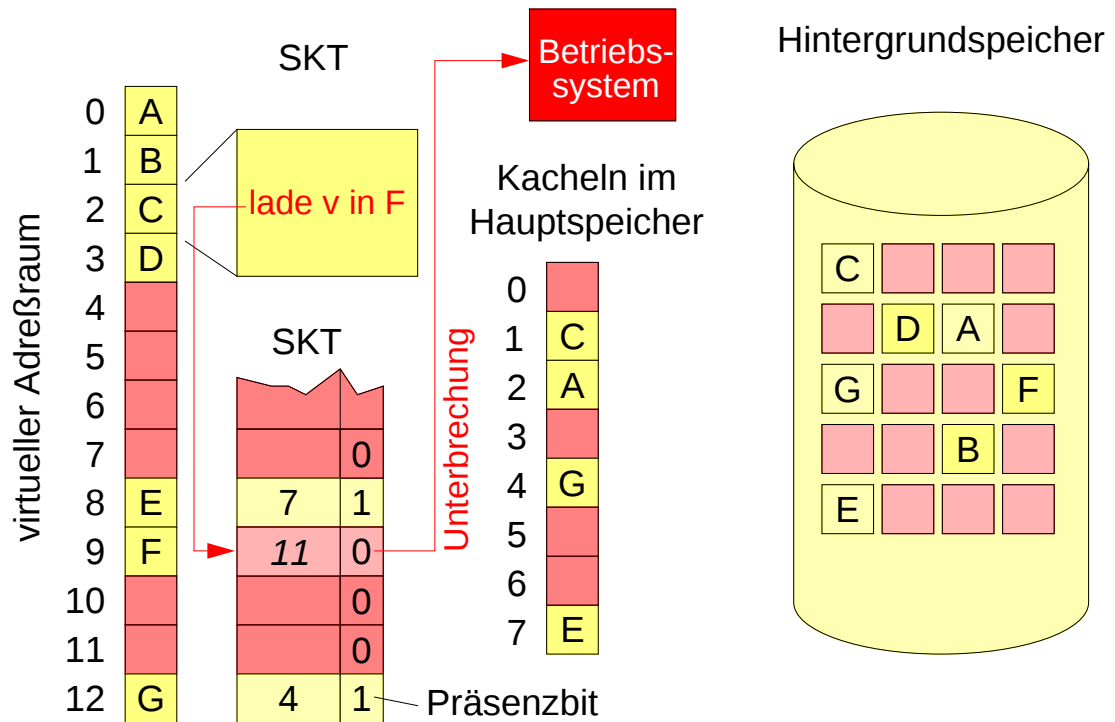
1 Demand Paging

- Bereitstellen von Seiten auf Anforderung



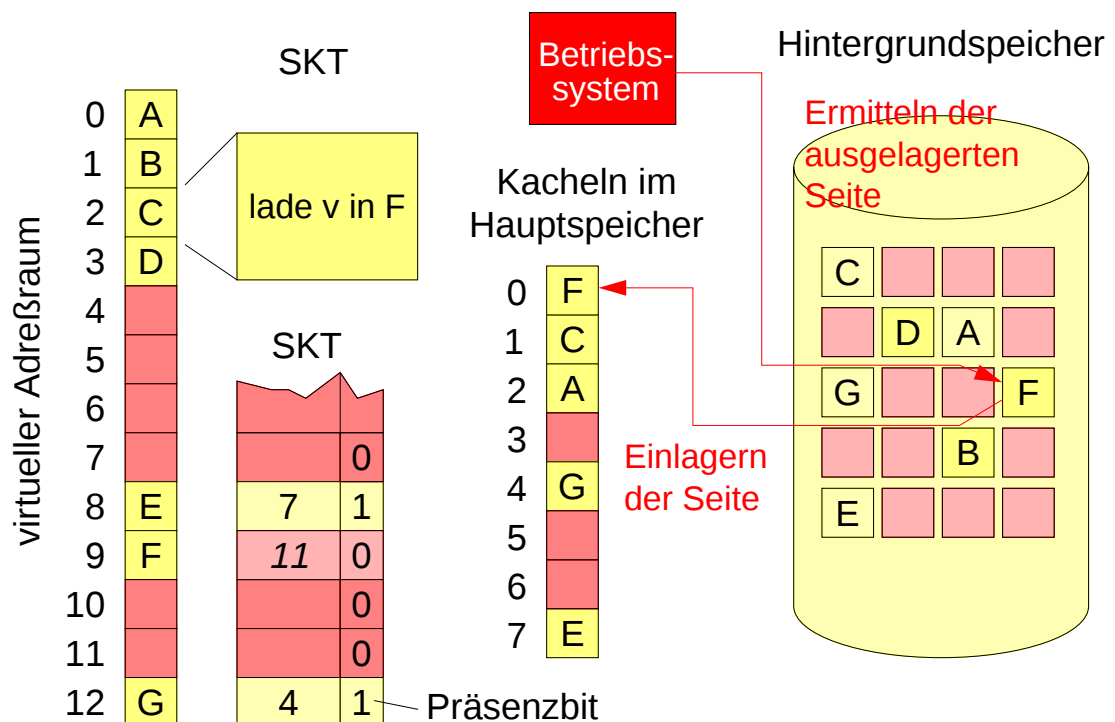
1 Demand Paging (2)

■ Reaktion auf Seitenfehler



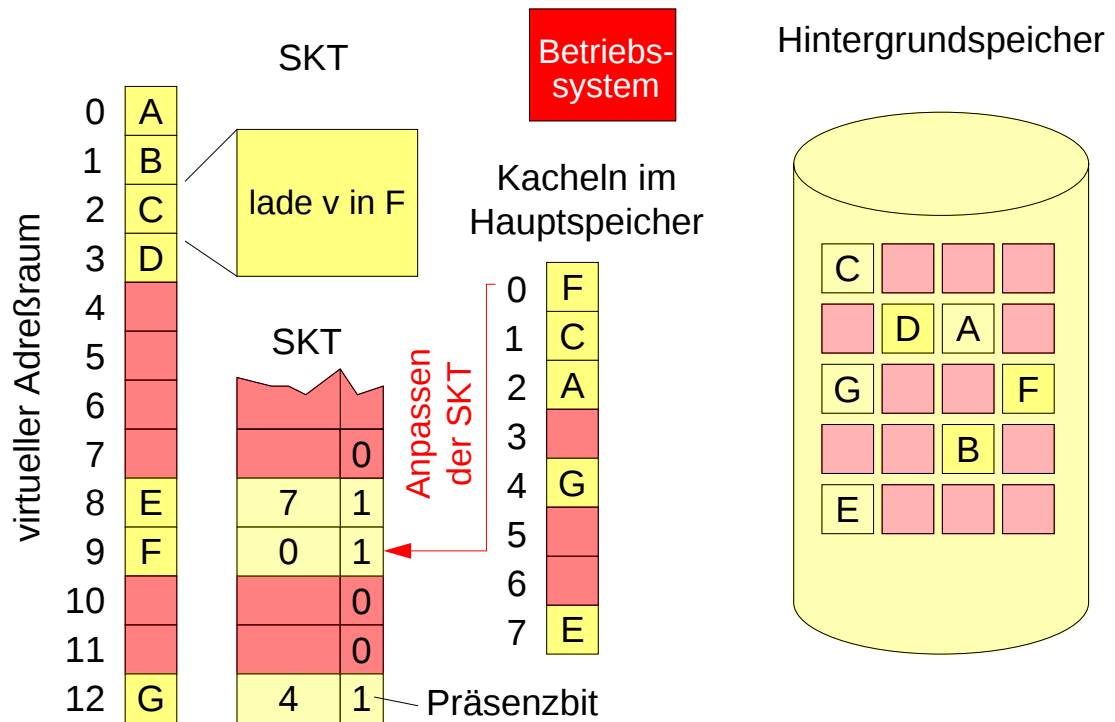
1 Demand Paging (3)

■ Reaktion auf Seitenfehler



1 Demand Paging (4)

■ Reaktion auf Seitenfehler



SPI

Systemprogrammierung I

© Franz J. Hauck, Univ. Erlangen-Nürnberg, IMMD IV, 1998/1999

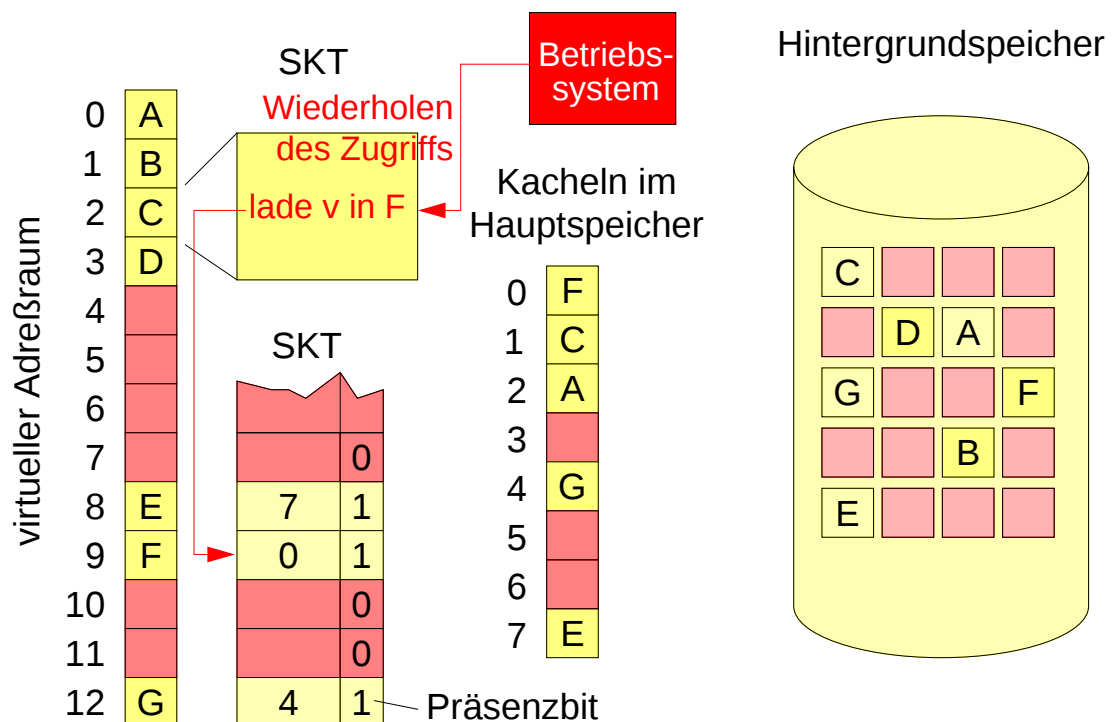
E-Memory.fm 1998-12-07 10.46

E.51

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Demand Paging (5)

■ Reaktion auf Seitenfehler



SPI

Systemprogrammierung I

© Franz J. Hauck, Univ. Erlangen-Nürnberg, IMMD IV, 1998/1999

E-Memory.fm 1998-12-07 10.46

E.52

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Demand Paging (6)

- ▲ Performanz von Demand paging
 - ◆ Keine Seitenfehler
 - effektive Zugriffszeit zw. 10 und 200 Nanosekunden
 - ◆ Mit Seitenfehler
 - p sei Wahrscheinlichkeit für Seitenfehler; p nahe Null
 - Annahme: Zeit zum Einlagern einer Seite vom Hintergrundspeicher gleich 25 Millisekunden (8 ms Latenz, 15 ms Positionierzeit, 1 ms Übertragungszeit)
 - Annahme: normale Zugriffszeit 100 ns
 - Effektive Zugriffszeit:
 $(1 - p) \times 100 + p \times 25000000 = 100 + 24999900 \times p$
- ▲ Seitenfehler müssen so niedrig wie möglich gehalten werden
- Abwandlung: *Demand zero* für nicht initialisierte Daten

2 Seitenersetzung

- Was tun, wenn keine freie Kachel vorhanden?
 - ◆ Eine Seite muß verdrängt werden, um Platz für neue Seite zu schaffen!
 - ◆ Auswahl von Seiten, die nicht geändert wurden (*Dirty bit* in der SKT)
 - ◆ Verdrängung erfordert Auslagerung, falls Seite geändert wurde
- Vorgang:
 - ◆ Seitenfehler (*Page fault*): Unterbrechung
 - ◆ Auslagern einer Seite, falls keine freie Kachel verfügbar
 - ◆ Einlagern der benötigten Seite
 - ◆ Wiederholung des Zugriffs
- ▲ Problem
 - ◆ Welche Seite soll ausgewählt werden?

E.7 Ersetzungsstrategien

- Betrachtung von Ersetzungsstrategien und deren Wirkung auf Referenzfolgen
- Referenzfolge
 - ◆ Folge von Seitennummern, die das Speicherzugriffsverhalten eines Prozesses abbildet
 - ◆ Ermittlung von Referenzfolgen z.B. durch Aufzeichnung der zugriffenen Adressen
 - Reduktion der aufgezeichneten Sequenz auf Seitennummern
 - Zusammenfassung von unmittelbar hintereinanderstehenden Zugriffen auf die gleiche Seite
 - ◆ Beispiel für eine Referenzfolge: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1 First-In, First-Out

- Älteste Seite wird ersetzt
- Notwendige Zustände:
 - ◆ Alter bzw. Einlagerungszeitpunkt für jede Kachel
- Ablauf der Ersetzungen (9 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	4	4	4	5	5	5	5	5	5
	Kachel 2		2	2	2	1	1	1	1	1	3	3	3
	Kachel 3			3	3	3	2	2	2	2	2	4	4
Kontrollzustände (Alter pro Kachel)	Kachel 1	0	1	2	0	1	2	0	1	2	3	4	5
	Kachel 2	>	0	1	2	0	1	2	3	4	0	1	2
	Kachel 3	>	>	0	1	2	0	1	2	3	4	0	1

1 First-In, First-Out

■ Größerer Hauptspeicher mit 4 Kacheln (10 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	1	1	1	5	5	5	5	4	4
	Kachel 2		2	2	2	2	2	2	1	1	1	1	5
	Kachel 3			3	3	3	3	3	3	2	2	2	2
	Kachel 4				4	4	4	4	4	4	3	3	3
Kontrollzustände (Alter pro Kachel)	Kachel 1	0	1	2	3	4	5	0	1	2	3	0	1
	Kachel 2	>	0	1	2	3	4	5	0	1	2	3	0
	Kachel 3	>	>	0	1	2	3	4	5	0	1	2	3
	Kachel 4	>	>	>	0	1	2	3	4	5	0	1	2

■ FIFO Anomalie (Belady's Anomalie, 1969)

SPI

Systemprogrammierung I

© Franz J. Hauck, Univ. Erlangen-Nürnberg, IMMD IV, 1998/1999

E-Memory.fm 1998-12-07 10.46

E.57

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Optimale Ersetzungsstrategie

■ Vorwärtsabstand

◆ Zeitdauer bis zum nächsten Zugriff auf die entsprechende Seite

■ Strategie B_0 (OPT oder MIN) ist optimal (bei fester Kachelmenge): minimale Anzahl von Einlagerungen/Ersetzungen (hier 7)

◆ „Ersetze immer die Seite mit dem größten Vorwärtsabstand!“

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	1	1	1	1	1	1	3	4	4
	Kachel 2		2	2	2	2	2	2	2	2	2	2	2
	Kachel 3			3	4	4	4	5	5	5	5	5	5
Kontrollzustände (Vorwärts- abstand)	Kachel 1	4	3	2	1	3	2	1	>	>	>	>	>
	Kachel 2	>	4	3	2	1	3	2	1	>	>	>	>
	Kachel 3	>	>	7	7	6	5	5	4	3	2	1	>

SPI

Systemprogrammierung I

© Franz J. Hauck, Univ. Erlangen-Nürnberg, IMMD IV, 1998/1999

E-Memory.fm 1998-12-07 10.46

E.58

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Optimale Ersetzungsstrategie (2)

- Vergrößerung des Hauptspeichers (4 Kacheln): 6 Einlagerungen

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	1	1	1	1	1	1	1	4	4
	Kachel 2		2	2	2	2	2	2	2	2	2	2	2
	Kachel 3			3	3	3	3	3	3	3	3	3	3
	Kachel 4				4	4	4	5	5	5	5	5	5
Kontrollzustände (Vorwärts- abstand)	Kachel 1	4	3	2	1	3	2	1	>	>	>	>	>
	Kachel 2	>	4	3	2	1	3	2	1	>	>	>	>
	Kachel 3	>	>	7	6	5	4	3	2	1	>	>	>
	Kachel 4	>	>	>	7	6	5	5	4	3	2	1	>

◆ keine Anomalie

2 Optimale Ersetzungsstrategie (3)

- Implementierung von B_0 nahezu unmöglich
 - ◆ Referenzfolge müsste vorher bekannt sein
 - ◆ B_0 meist nur zum Vergleich von Strategien brauchbar
- Suche nach Strategien, die möglichst nahe an B_0 kommen
 - ◆ z.B. *Least recently used* (LRU)

3 Least Recently Used (LRU)

■ Rückwärtsabstand

◆ Zeitdauer, seit dem letzten Zugriff auf die Seite

■ LRU Strategie (10 Einlagerungen)

◆ „Ersetze die Seite mit dem größten Rückwärtsabstand !“

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	4	4	4	5	5	5	3	3	3
	Kachel 2		2	2	2	1	1	1	1	1	1	4	4
	Kachel 3			3	3	3	2	2	2	2	2	2	5
Kontrollzustände (Rückwärts- abstand)	Kachel 1	0	1	2	0	1	2	0	1	2	0	1	2
	Kachel 2	>	0	1	2	0	1	2	0	1	2	0	1
	Kachel 3	>	>	0	1	2	0	1	2	0	1	2	0

3 Least Recently Used (2)

■ Vergrößerung des Hauptspeichers (4 Kacheln): 8 Einlagerungen

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	1	1	1	1	1	1	1	1	5
	Kachel 2		2	2	2	2	2	2	2	2	2	2	2
	Kachel 3			3	3	3	3	5	5	5	5	4	4
	Kachel 4				4	4	4	4	4	4	3	3	3
Kontrollzustände (Rückwärts- abstand)	Kachel 1	0	1	2	3	0	1	2	0	1	2	3	0
	Kachel 2	>	0	1	2	3	0	1	2	0	1	2	3
	Kachel 3	>	>	0	1	2	3	0	1	2	3	0	1
	Kachel 4	>	>	>	0	1	2	3	4	5	0	1	2

3 Least Recently Used (3)

■ Keine Anomalie

- ◆ Allgemein gilt: Es gibt eine Klasse von Algorithmen (Stack-Algorithmen), bei denen keine Anomalie auftritt:
 - Bei Stack-Algorithmen ist bei n Kacheln zu jedem Zeitpunkt eine Untermenge der Seiten eingelagert, die bei $n+1$ Kacheln zum gleichen Zeitpunkt eingelagert wären!
 - LRU: Es sind immer die letzten n benutzten Seiten eingelagert
 - B_0 : Es sind die n bereits benutzten Seiten eingelagert, die als nächstes zugegriffen werden

▲ Problem

- ◆ Implementierung von LRU nicht ohne Hardwareunterstützung möglich
- ◆ Es muß jeder Speicherzugriff berücksichtigt werden

3 Least Recently Used (4)

■ Hardwareunterstützung durch Zähler

- ◆ CPU besitzt einen Zähler, der bei jedem Speicherzugriff erhöht wird (inkrementiert wird)
- ◆ bei jedem Zugriff wird der aktuelle Zählerwert in den jeweiligen Seitendeskriptor geschrieben
- ◆ Auswahl der Seite mit dem kleinsten Zählerstand

▲ Aufwendige Implementierung

- ◆ viele zusätzliche Speicherzugriffe

4 Second Chance (Clock)

■ Einsatz von Referenzbits

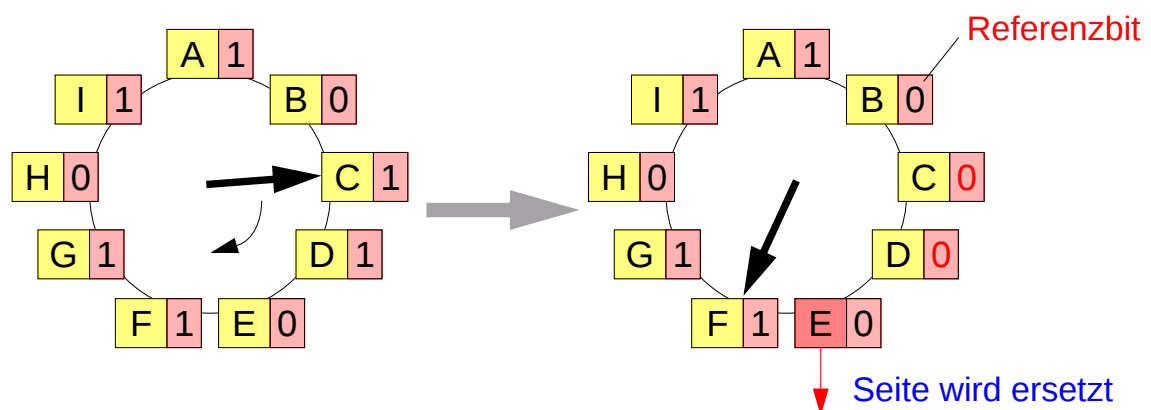
- ◆ Referenzbit im Seitendeskriptor wird automatisch durch Hardware gesetzt, wenn die Seite zugegriffen wird
 - einfacher zu implementieren
 - weniger zusätzliche Speicherzugriffe
 - moderne Prozessoren bzw. MMUs unterstützen Referenzbits (z.B. Pentium: *Access bit*)

■ Ziel: Annäherung von LRU

- ◆ das Referenzbit wird zunächst auf 0 gesetzt
- ◆ wird eine Opferseite gesucht, so werden die Kacheln reihum inspiziert
- ◆ ist das Referenzbit 1, so wird es auf 0 gesetzt (zweite Chance)
- ◆ ist das Referenzbit 0, so wird die Seite ersetzt

4 Second Chance (2)

■ Implementierung mit umlaufendem Zeiger (*Clock*)



- ◆ an der Zeigerposition wird Referenzbit getestet
 - falls Referenzbit eins, wird Bit gelöscht und Zeiger weitergestellt
 - falls Referenzbit gleich Null, wurde ersetzbare Seite gefunden
- ◆ falls alle Referenzbits auf 1 stehen, wird Second chance zu FIFO

4 Second Chance (3)

■ Ablauf bei drei Kacheln (9 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	4	4	4	5	5	5	5	5	5
	Kachel 2		2	2	2	1	1	1	1	1	3	3	3
	Kachel 3			3	3	3	2	2	2	2	2	4	4
Kontroll- zustände (Referenzbits)	Kachel 1	1	1	1	1	1	1	1	1	1	0	0	1
	Kachel 2	0	1	1	0	1	1	0	1	1	1	1	1
	Kachel 3	0	0	1	0	0	1	0	0	1	0	1	1
	Umlaufzeiger	2	3	1	2	3	1	2	2	2	3	1	1

4 Second Chance (4)

■ Vergrößerung des Hauptspeichers (4 Kacheln): 10 Einlagerungen

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	1	1	1	5	5	5	5	4	4
	Kachel 2		2	2	2	2	2	2	1	1	1	1	5
	Kachel 3			3	3	3	3	3	3	2	2	2	2
	Kachel 4				4	4	4	4	4	4	3	3	3
Kontroll- zustände (Referenzbits)	Kachel 1	1	1	1	1	1	1	1	1	1	1	1	1
	Kachel 2	0	1	1	1	1	1	0	1	1	1	0	1
	Kachel 3	0	0	1	1	1	1	0	0	1	1	0	0
	Kachel 4	0	0	0	1	1	1	0	0	0	1	0	0
	Umlaufzeiger	2	3	4	1	1	1	2	3	4	1	2	3

4 Second Chance (5)

- Second chance zeigt FIFO Anomalie
 - ◆ Wenn alle Referenzbits gleich 1, wird nach FIFO entschieden
- Erweiterung
 - ◆ Modifikationsbit kann zusätzlich berücksichtigt werden (*Dirty bit*)
 - ◆ vier Klassen: (0,0), (0,1), (1,0) und (1,1) mit (Referenzbit, Modifikationsbit)
 - ◆ Suche nach der niedrigsten Klasse (Einsatz im MacOS)

5 Freiseitenpuffer

- Statt eine Seite zu ersetzen wird permanent eine Menge freier Seiten gehalten
 - ◆ Auslagerung geschieht im „voraus“
 - ◆ Effizienter: Ersetzungszeit besteht im Wesentlichen nur aus Einlagerungszeit
- Behalten der Seitenzuordnung auch nach der Auslagerung
 - ◆ Wird die Seite doch noch benutzt bevor sie durch eine andere ersetzt wird, kann sie mit hoher Effizienz wiederverwendet werden.
 - ◆ Seite wird aus Freiseitenpuffer ausgetragen und wieder dem entsprechenden Prozeß zugeordnet.

6 Seitenanforderung

▲ Problem: Zuordnung der Kacheln zu mehreren Prozessen

■ Begrenzungen

- ◆ Maximale Seitenmenge: begrenzt durch Anzahl der Kacheln
- ◆ Minimale Seitenmenge: abhängig von der Prozessorarchitektur
 - Mindestens die Anzahl von Seiten nötig, die theoretisch bei einem Maschinenbefehl benötigt werden
(z.B. zwei Seiten für den Befehl, vier Seiten für die adressierten Daten)

■ Gleiche Zuordnung

- ◆ Anzahl der Prozesse bestimmt die Kachelmenge, die ein Prozeß bekommt

■ Größenabhängige Zuordnung

- ◆ Größe des Programms fließt in die zugeteilte Kachelmenge ein

6 Seitenanforderung

■ Globale und lokale Anforderung von Seiten

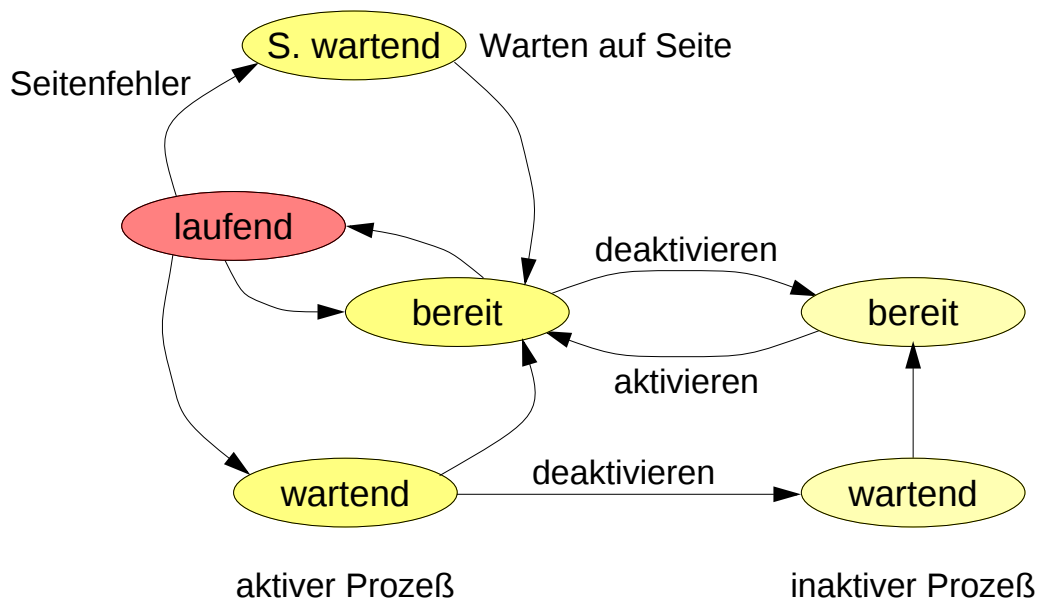
- ◆ lokal: Prozeß ersetzt nur immer seine eigenen Seiten
 - Seitenfehler-Verhalten liegt nur in der Verantwortung des Prozesses
- ◆ global: Prozeß ersetzt auch Seiten anderer Prozesse
 - bessere Effizienz, da ungenutzte Seiten von anderen Prozessen verwendet werden können

E.8 Seitenflattern (*Thrashing*)

- Ausgelagerte Seite wird gleich wieder angesprochen
 - ◆ Prozeß verbringt mehr Zeit mit dem Warten auf das Beheben von Seitenfehler als mit der eigentlichen Ausführung
- Ursachen
 - ◆ Prozeß ist nahe am Seitenminimum
 - ◆ zu viele Prozesse gleichzeitig im System
 - ◆ schlechte Ersetzungsstrategie
- ★ Lokale Seitenanforderung behebt Thrashing zwischen Prozessen
- ★ Zuteilung einer genügend großen Zahl von Kacheln behebt Thrashing innerhalb der Prozeßseiten
 - ◆ Begrenzung der Prozeßanzahl

1 Deaktivieren von Prozessen

- Einführung von „Superzuständen“



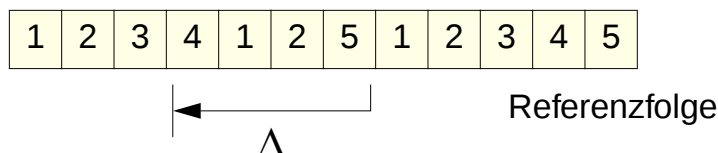
- ◆ inaktiver Prozeß benötigt keine Kacheln; Prozeß ist vollständig ausgelagert (swapped out)

1 Deaktivieren von Prozessen (2)

- Sind zuviele Prozesse aktiv, werden welche deaktiviert
 - ◆ Kacheln teilen sich auf weniger Prozesse auf
 - ◆ Verbindung mit dem Scheduling nötig
 - Verhindern von Aushungerung
 - Erzielen kurzer Reaktionszeiten
 - ◆ guter Kandidat: Prozeß mit wenigen Seiten im Hauptspeicher
 - geringe Latenz bei Wiedereinlagerung bzw. wenige Seitenfehler bei Aktivierung und Demand paging

2 Arbeitsmengenmodell

- Menge der Seiten, die ein Prozeß wirklich braucht (*Working set*)
 - ◆ kann nur angenähert werden, da üblicherweise nicht vorhersehbar
- Annäherung durch Betrachten der letzten Δ Seiten, die angesprochen wurden
 - ◆ geeignete Wahl von Δ
 - zu groß: Überlappung von lokalen Zugriffsmustern
 - zu klein: Arbeitsmenge enthält nicht alle nötigen Seiten



2 Arbeitsmengenmodell (2)

■ Beispiel: Arbeitsmengen bei verschiedenen Δ

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
$\Delta = 3$	Seite 1	x	x	x		x	x	x	x	x	x		
	Seite 2		x	x	x		x	x	x	x	x	x	
	Seite 3			x	x	x					x	x	x
	Seite 4				x	x	x					x	x
	Seite 5							x	x	x			x
$\Delta = 4$	Seite 1	x	x	x	x	x	x	x	x	x	x	x	
	Seite 2		x	x	x	x	x	x	x	x	x	x	x
	Seite 3			x	x	x	x				x	x	x
	Seite 4				x	x	x	x				x	x
	Seite 5							x	x	x	x		x

3 Bestimmung der Arbeitsmenge

■ Annäherung der Zugriffe durch die Zeit

- ◆ bestimmtes Zeitintervall ist ungefähr proportional zu Anzahl von Speicherzugriffen

▲ Virtuelle Zeit des Prozesses muß gemessen werden

- ◆ nur die Zeit relevant, in der der Prozeß im Zustand laufend ist
- ◆ Verwalten virtueller Uhren pro Prozeß

■ Referenzbit, Altersangabe und Timer-Interrupt

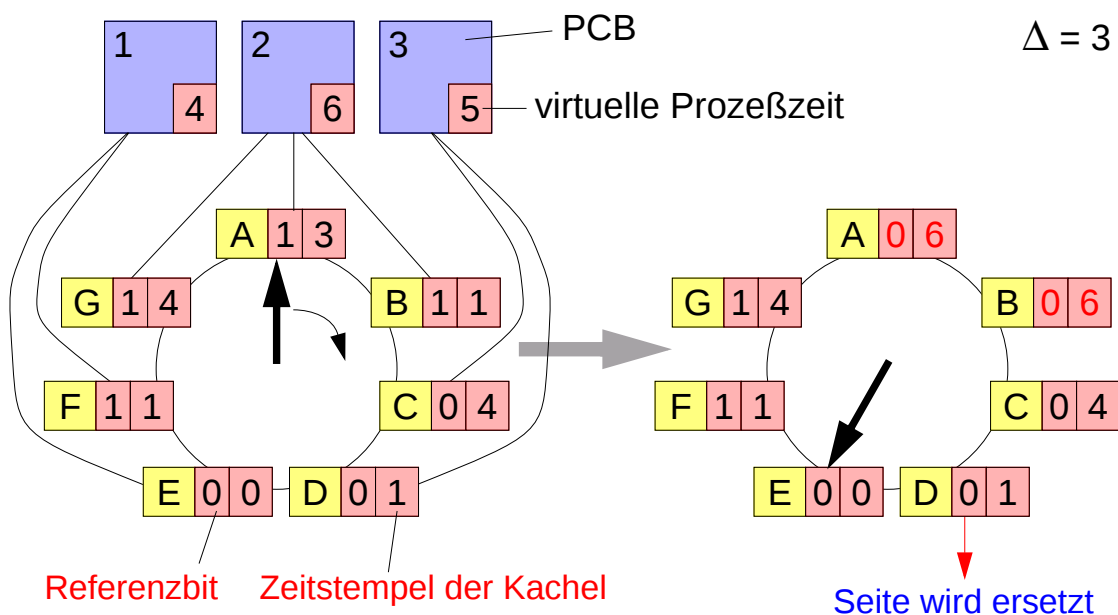
- ◆ jede Seite enthält eine Altersangabe (Zeitintervall ohne Benutzung)
- ◆ durch regelmäßigen Interrupt wird mittels Referenzbit die Altersangabe fortgeschrieben (bei Benutzung auf Null gesetzt, ansonsten erhöht); dabei werden nur die Seiten des gerade laufenden Prozesses „gealtert“
- ◆ Seiten mit Alter $> \Delta$ sind nicht mehr in der Arbeitsmenge des jeweiligen Prozesses

3 Bestimmung der Arbeitsmenge (2)

- ▲ Ungenau: System ist aber nicht empfindlich auf diese Ungenauigkeit
 - ◆ Verringerung der Zeitintervalle: höherer Aufwand, genauere Messung
- Algorithmus WSClock (Working set clock)
 - ◆ arbeitet wie Clock
 - ◆ Seite wird nur dann ersetzt, wenn sie nicht zur Arbeitsmenge ihres Prozesses gehört oder der Prozeß deaktiviert ist
 - ◆ Bei Zurücksetzen des Referenzbits wird die virtuelle Zeit des jeweiligen Prozesses eingetragen, die z.B. im PCB gehalten und fortgeschrieben wird
 - ◆ Bestimmung der Arbeitsmenge erfolgt durch Differenzbildung von virtueller Zeit des Prozesses und Zeitstempel in der Kachel

3 Bestimmung der Arbeitsmenge (3)

■ WSClock Algorithmus

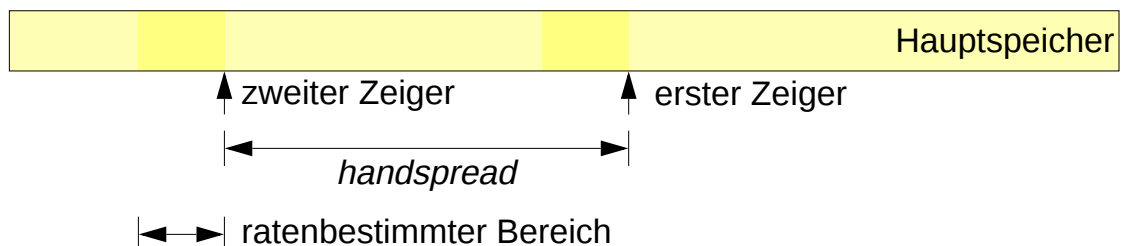


3 Bestimmung der Arbeitsmenge (4)

- ▲ Zuordnung zu einem Prozeß nicht immer möglich
 - ◆ gemeinsam genutzte Seiten in modernen Betriebssystemen eher die Regel als die Ausnahme
 - Seiten des Codesegments
 - Shared libraries
 - Gemeinsame Seiten im Datensegment (*Shared memory*)
 - ◆ moderne System bestimmen meist eine globale Arbeitsmenge von Seiten

4 Ersetzungsstrategie bei Solaris

- Prozeß *pageout* arbeitet Clock-Strategie ab
 - ◆ Prozeß läuft mehrmals die Sekunde (4x)
 - ◆ adaptierbare Rate: untersuchte Seiten pro Sekunde
 - ◆ statt ein Zeiger: zwei Zeiger
 - am ersten Zeiger werden Referenzbits zurückgesetzt
 - am zweiten Zeiger werden Seiten mit gelöschtem Referenzbit ausgewählt
 - nötig, weil sonst Zeitspanne zwischen Löschen und Auswählen zu lang wird (großer Hauptspeicher; 64 MByte entsprechen 8.192 Seiten)
 - Zeigerabstand einstellbar (*handspread*)



4

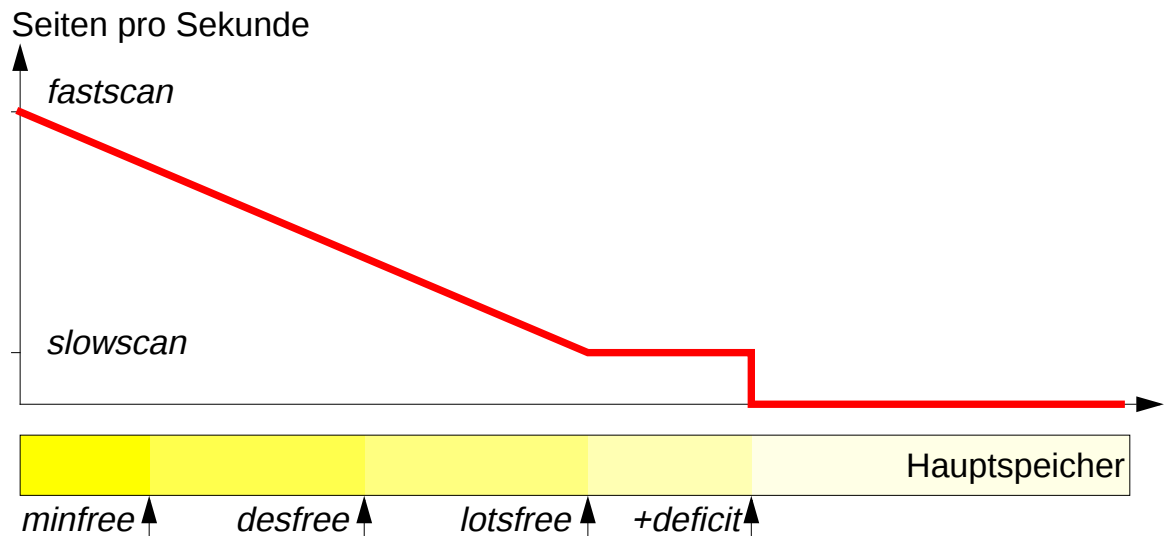
- SP I**

4

- SP I**

4 Ersetzungsstrategie bei Solaris (4)

■ Seitenuntersuchungsrate des *pageout* Prozesses



- ◆ je weniger freier Speicher verfügbar ist, desto höher wird die Untersuchungsrate
- ◆ *slowscan* und *fastscan* sind einstellbar

4 Ersetzungsstrategie bei Solaris (5)

■ Weitere Parameter

- ◆ *maxpgio*: maximale Transferrate bei Auslagerungen (vermeidet Plattensaturierung)
- ◆ *autoup*: Zeitdauer des regelmäßigen Auslagerns alter Seiten durch den Prozeß *flushd* (Default: alle 30 sec)

■ Aktivieren und Deaktivieren (*Swap in*, *Swap out*)

- ◆ Auswahl wird dem Scheduler überlassen
- ◆ Deaktivierung wird lediglich von Speicherverwaltung angestoßen