

E Speicherverwaltung

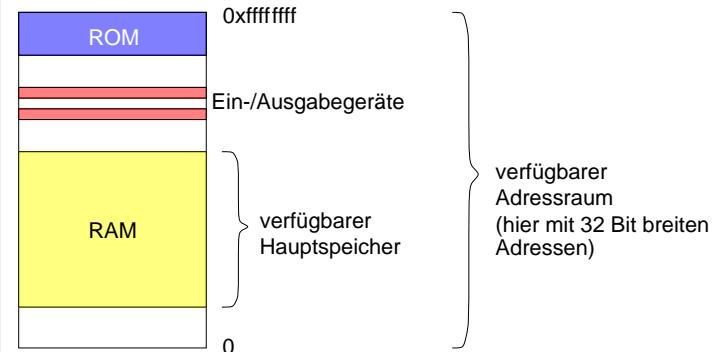
Systemprogrammierung I
© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

E - 1

1 Speichervergabe

1.1 Problemstellung

- Verfügbarer Speicher

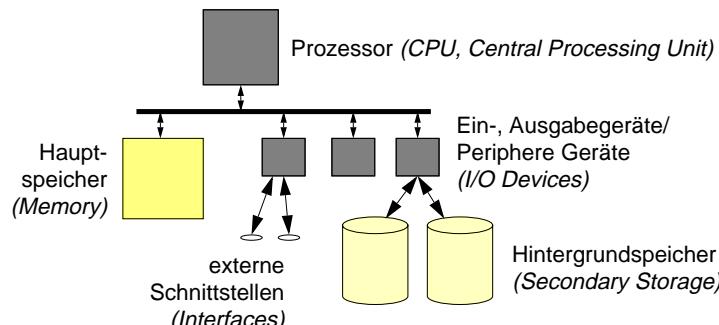


Systemprogrammierung I
© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

E - 3

E Speicherverwaltung

- Betriebsmittel



Systemprogrammierung I
© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

E - 2

1.1 Problemstellung (2)

- Belegung des verfügbaren Hauptspeichers durch
 - ◆ Benutzerprogramme
 - Programmbefehle (Code, Binary)
 - Programmdaten
 - ◆ Betriebssystem
 - Betriebssystemcode
 - Puffer
 - Systemvariablen
- ★ Zuteilung des Speichers nötig

Systemprogrammierung I
© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

E - 4

1.2 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

E - 5

1.3 Dynamische Speicherzuteilung

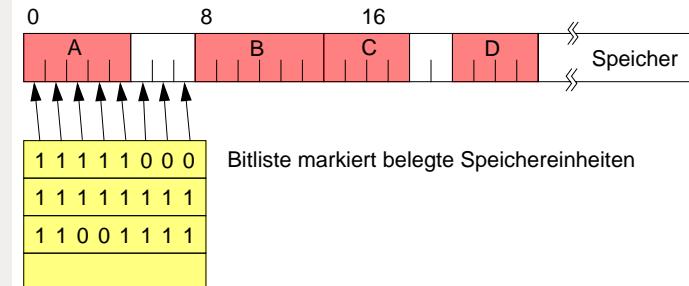
- Segmente
 - ◆ zusammenhängender Speicherbereich
(Bereich mit aufeinanderfolgenden Adressen)
- Allokation (Anforderung) und Freigabe von Segmenten
- Ein Anwendungsprogramm besitzt üblicherweise folgende Segmente
(siehe auch D.2.4):
 - ◆ Codesegment
 - ◆ Datensegment
 - ◆ Stacksegment (für Verwaltungsinformationen, z.B. bei Funktionsaufrufen)
- ▲ Suche nach geeigneten Speicherbereichen zur Zuteilung
- ★ Speicherzuteilungsstrategien nötig

E - 6

1.4 Freispeicherverwaltung

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

■ Bitlisten



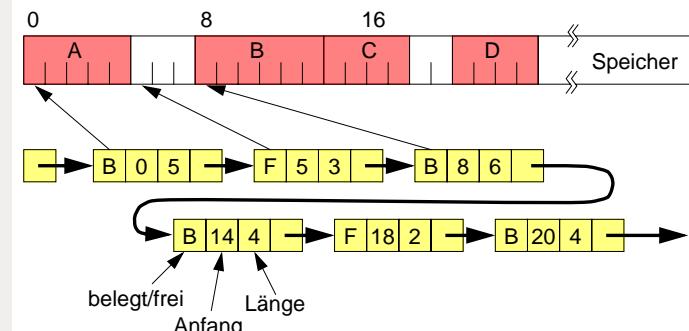
Bitliste markiert belegte Speichereinheiten

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

E - 7

1.4 Freispeicherverwaltung (2)

■ Verkettete Liste

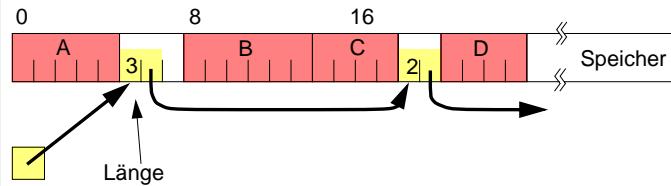


Repräsentation auch von freien Segmenten

E - 8

1.4 Freispeicherverwaltung (3)

- Verkettete Liste in dem freien Speicher



Mindestlückengröße muss garantiert werden

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

1.6 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

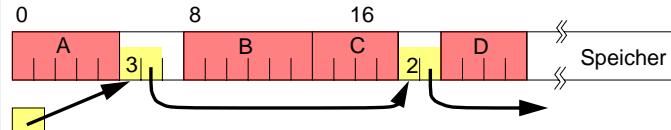
- ▲ Probleme:

◆ Speicherverschnitt

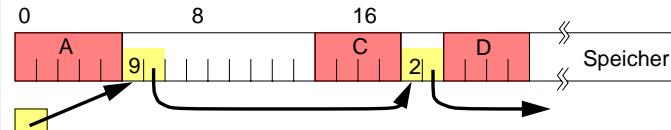
◆ zu kleine Lücken

1.5 Speicherfreigabe

- Verschmelzung von Lücken



nach Freigabe von B:



1.7 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			

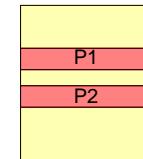
Effiziente Repräsentation der Lücken und effiziente Algorithmen

1.8 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*)

2.1 Problemstellung (2)

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



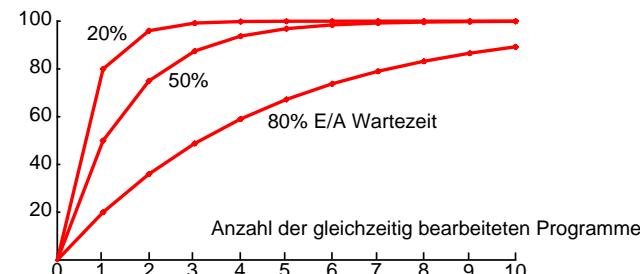
zwei Prozesse und deren Codesegmente im Speicher

- ★ Relokation von Programmbefehlen (Binaries)
- ★ Ein- und Auslagern von Prozessen
- ★ Hardwareunterstützung

2 Mehrprogrammbetrieb

2.1 Problemstellung

- Mehrere Prozesse laufen gleichzeitig
 - ◆ Wartezeiten von Ein-/Ausgabeoperationen ausnutzen
 - ◆ CPU Auslastung verbessern
 - ◆ CPU-Nutzung in Prozent, abhängig von der Anzahl der Prozesse

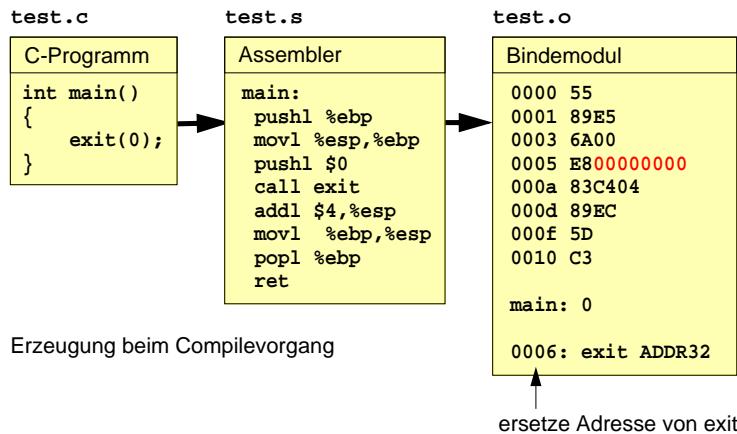


2.2 Relokation

- Festlegung absoluter Speicheradressen in den Programmbefehlten
 - ◆ 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)
 - ◆ Relocationsinformation nötig, die vom Compiler oder Assembler geliefert wird

2.2 Relokation (2)

- Compilevorgang (Erzeugung der Relocationsinformation)



Erzeugung beim Compilevorgang

Systemprogrammierung I

© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]

E - 17

2.2 Relokation (4)

- Relocationsinformation im Bindemodul

◆ erlaubt das Binden von Modulen in beliebige Programme

- Relocationsinformation im Lademodul

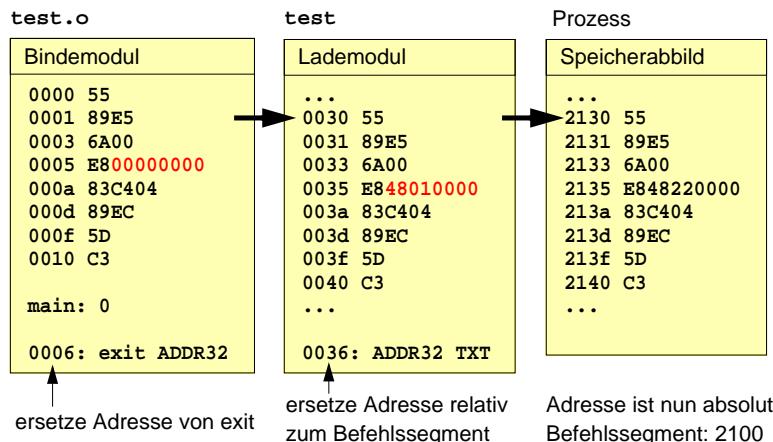
◆ erlaubt das Laden des Programms an beliebige Speicherstellen
◆ absolute Adressen werden erst beim Laden generiert

- ★ Alternative

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

2.2 Relokation (3)

- Binde- und Ladevorgang



Systemprogrammierung I

© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]

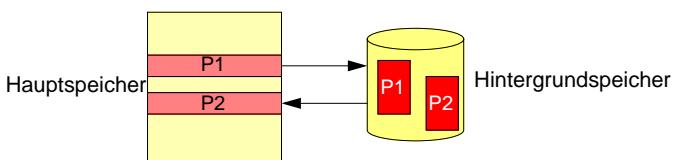
E - 18

2.3 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

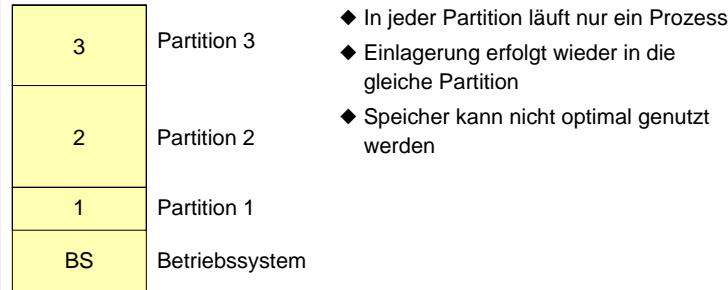
Systemprogrammierung I

© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]

E - 20

2.3 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



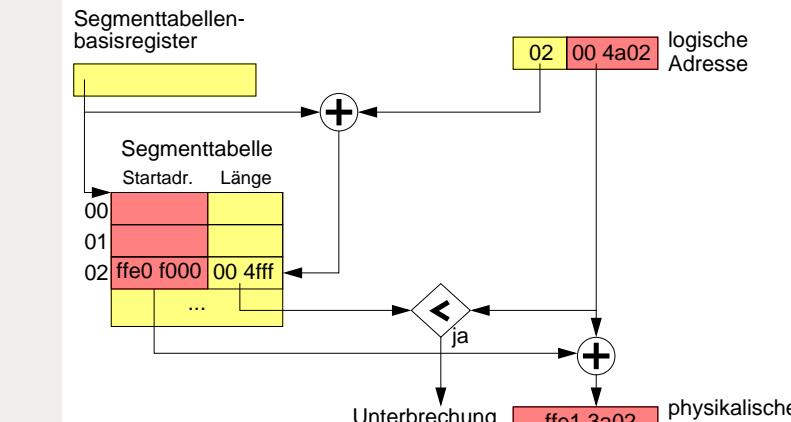
E - 21

Systemprogrammierung I

© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2.4 Segmentierung (2)

- Realisierung mit Übersetzungstabelle



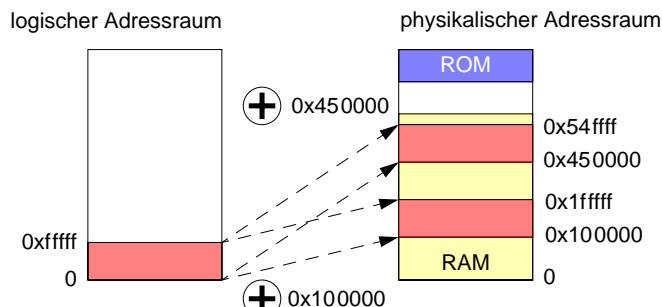
E - 23

Systemprogrammierung I

© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2.4 Segmentierung

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



E - 22

Systemprogrammierung I

© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2.4 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*)

E - 24

Systemprogrammierung I

© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2.4 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

Systemprogrammierung I

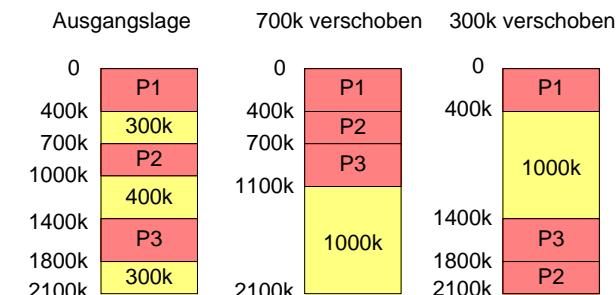
© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]

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

E - 25

2.5 Kompaktifizieren

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



Systemprogrammierung I

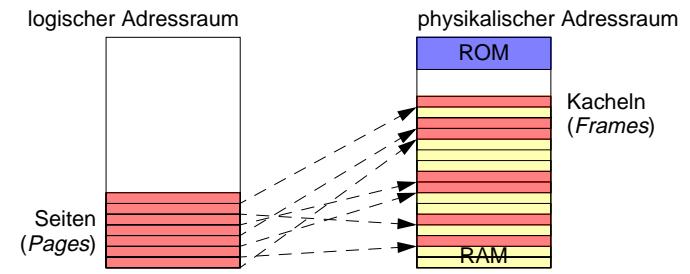
© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]

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

E - 26

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



Systemprogrammierung I

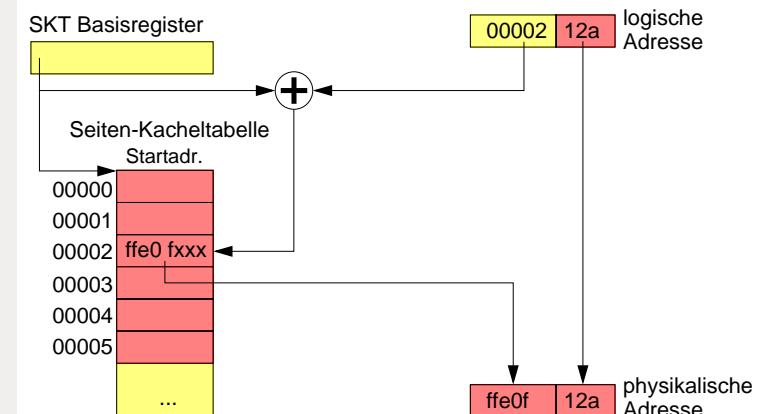
© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]

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

E - 27

3.1 MMU mit Seiten-Kacheltabelle

- Tabelle setzt Seiten in Kacheln um



Systemprogrammierung I

© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]

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

E - 28

3.1 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

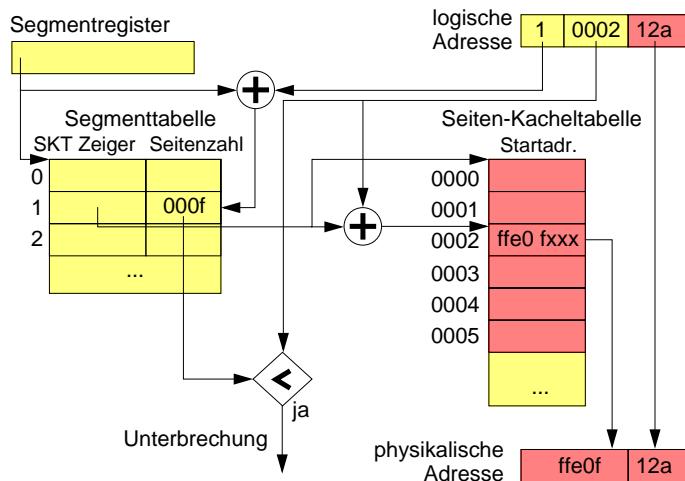
E – 29

3.2 Segmentierung und Seitenadressierung (2)

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

E – 31

3.2 Segmentierung und Seitenadressierung



E – 30

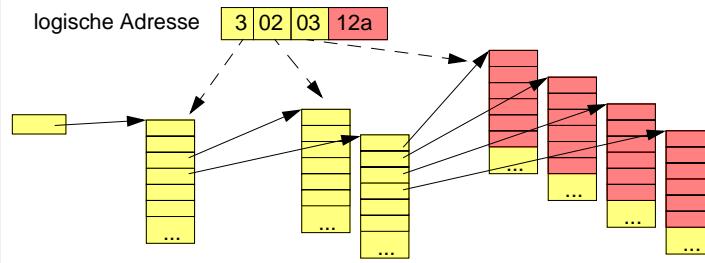
3.3 Ein- und Auslagerung von Seiten

- Es ist nicht nötig ein gesamtes Segment aus- bzw. einzulagern
 - ◆ Seiten können einzeln ein- und ausgelagert werden
- Hardware-Unterstützung
 - ◆ 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).

E – 32

3.4 Mehrstufige Seitenadressierung

- Beispiel: zweifach indirekte Seitenadressierung



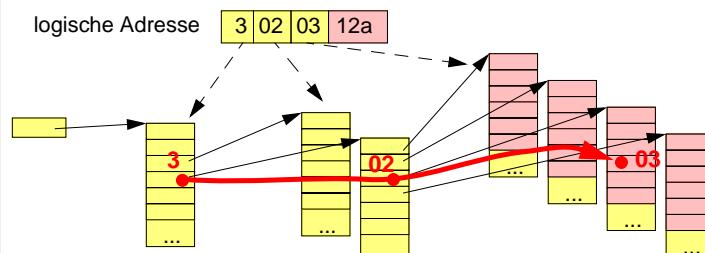
E - 33

Systemprogrammierung I

© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3.4 Mehrstufige Seitenadressierung

- Beispiel: zweifach indirekte Seitenadressierung



- Präsenzbit auch für jeden Eintrag in den höheren Stufen
 - Tabellen werden aus- und einlagerbar
- Noch mehr implizite Speicherzugriffe

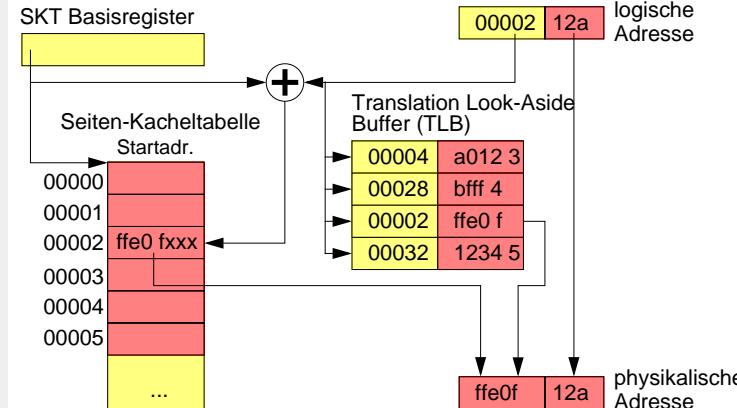
E - 33

Systemprogrammierung I

© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3.5 Translation Look-Aside Buffer

- Schneller Registersatz wird konsultiert bevor auf die SKT zugegriffen wird:



E - 34

Systemprogrammierung I

© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3.5 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

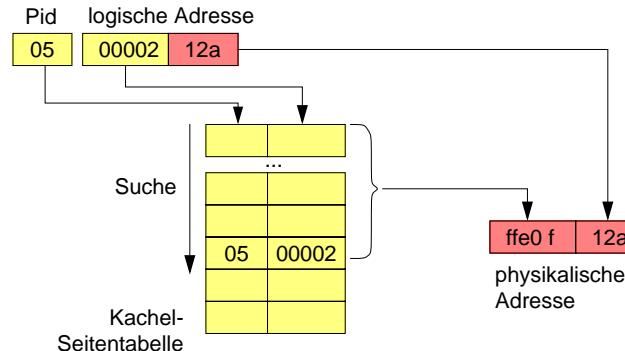
E - 35

Systemprogrammierung I

© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3.6 Invertierte Seiten-Kacheltabelle

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



E - 36

3.6 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

3.7 Systemaufruf

- Ermitteln der Seitengröße des Betriebssystems

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

Systemprogrammierung I

© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]

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

E - 37

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

Systemprogrammierung I

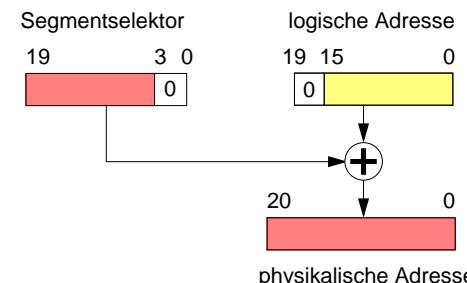
© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]

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

E - 38

4.1 Real Mode Adressierung

- Adressgenerierung im Real Mode
 - ◆ 16 bit breiter Segmentselktor wird als 20 bit breite Adresse interpretiert und auf die logische Adresse addiert



Systemprogrammierung I

© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]

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

E - 39

4.2 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
- Segmentselektoren enthalten Privilegierungsstufe
 - ◆ Stufe des Codesgments entscheidet über Zugriffserlaubnis
- Segmentselektoren werden als Indizes interpretiert
 - ◆ Tabellen von Segmentdeskriptoren
 - Globale Deskriptor Tabelle
 - Lokale Deskriptor Tabelle

Systemprogrammierung I

© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]

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

E - 40

4.2 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)

Systemprogrammierung I

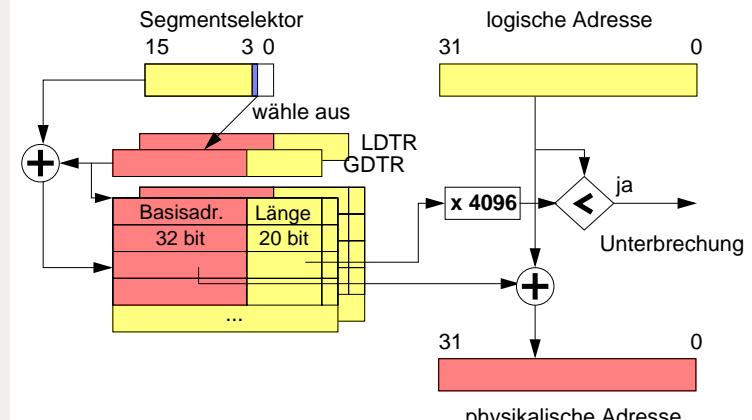
© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]

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

E - 41

4.3 Adressberechnung bei Segmentierung

- Verwendung der Protected mode Adressierung



Systemprogrammierung I

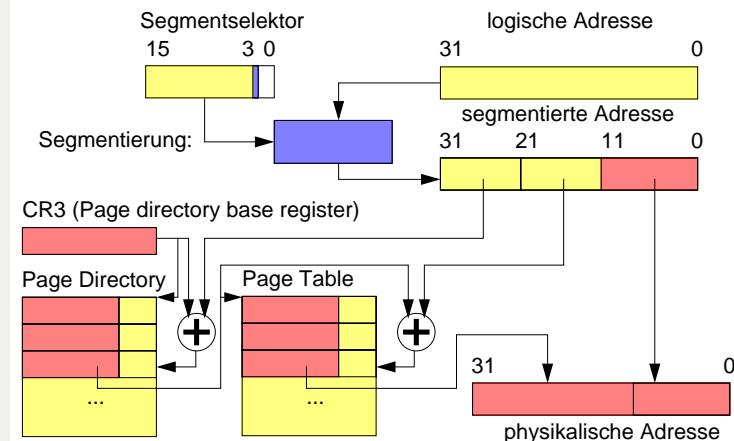
© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]

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

E - 42

4.4 Adressberechnung bei Paging

- Seitenadressierung wird der Segmentierung nachgeschaltet



Systemprogrammierung I

© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]

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

E - 43

4.4 Adressberechnung bei Paging (2)

- 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

E - 44

Systemprogrammierung I

© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4.5 Physical-Address-Extension (PAE) (2)

- Ab Pentium Pro
 - ◆ vierelementige Tabelle von Page-Directory-Pointers
 - ◆ 24 statt 20 Bit breite physikalische Adressumsetzung für den Seitenanfang
 - ◆ 64 Bit statt 32 Bit breite Tabelleneinträge
 - ◆ spezieller Chipsatz erforderlich
- Adressierbarer Hauptspeicher von 64 GByte

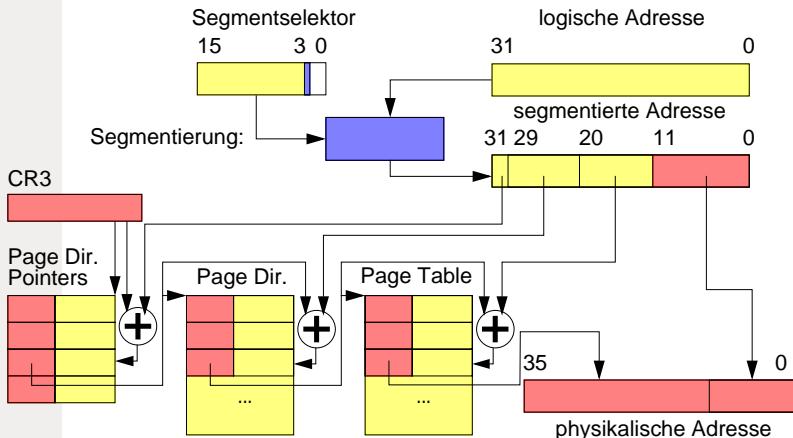
E - 46

Systemprogrammierung I

© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4.5 Physical-Address-Extension (PAE)

- Nachgeschaltete dreistufige Seitenadressierung



E - 45

Systemprogrammierung I

© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

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 - 47

Systemprogrammierung I

© 1997-2001, Franz J. Hauck; 2002, F. Hofmann, Inf 4, Univ. Erlangen-Nürnberg[E-Memory.fm, 2002-11-28 15.28]
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

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

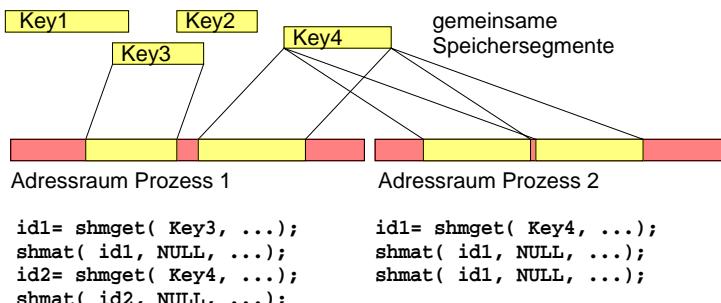
```
void *shmat( int shmid, void *shmaddr, int shmflg );
int shmdt( void *shmaddr );
```
 - ◆ Kontrolloperation

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

E - 48

5 Gemeinsamer Speicher (3)

- Prinzip der **shm*** Operationen



E - 49

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 - 50

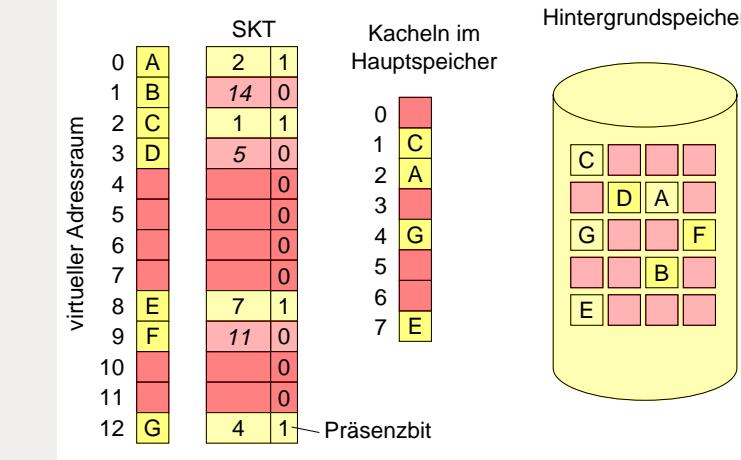
6 Virtueller Speicher

- 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

E - 51

6.1 Demand Paging

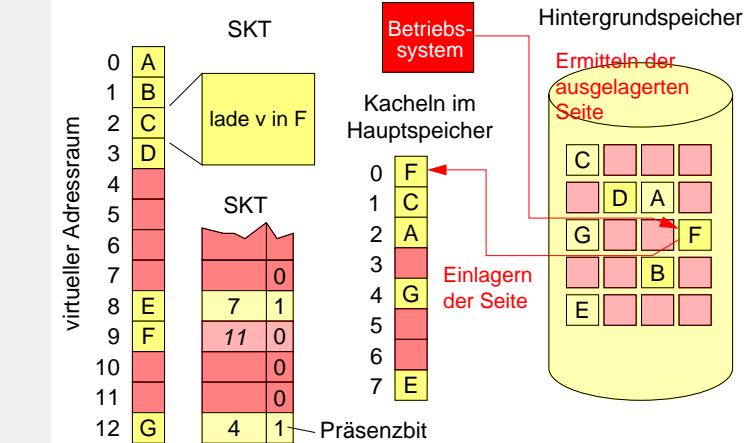
- Bereitstellen von Seiten auf Anforderung



E - 52

6.1 Demand Paging (3)

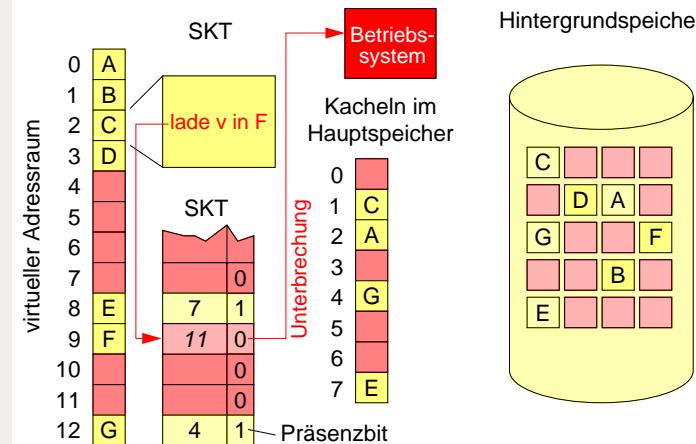
- Reaktion auf Seitenfehler



E - 54

6.1 Demand Paging (2)

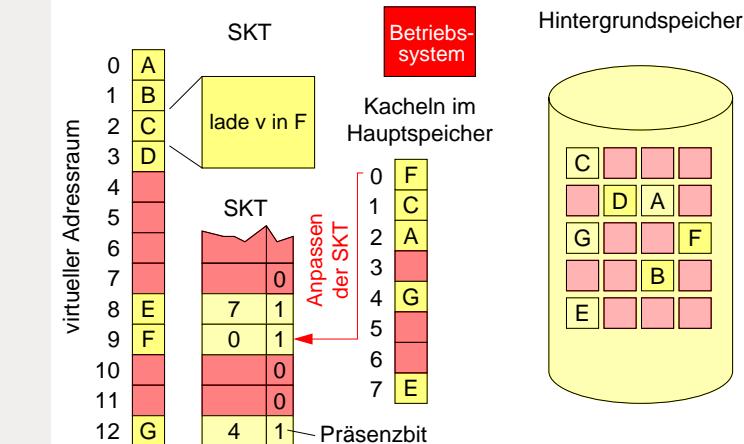
- Reaktion auf Seitenfehler



E - 53

6.1 Demand Paging (4)

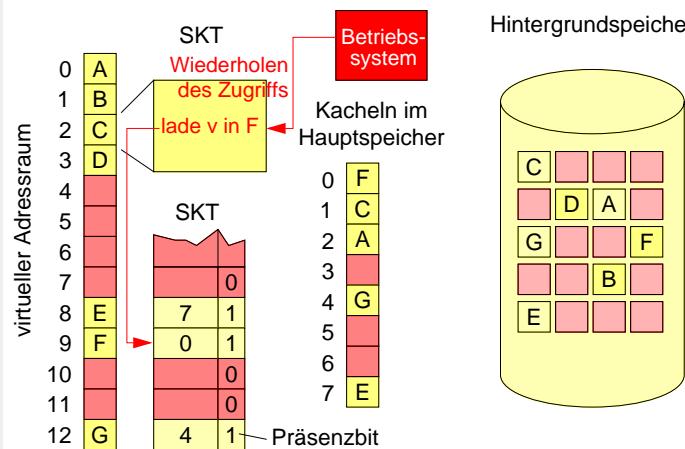
- Reaktion auf Seitenfehler



E - 55

6.1 Demand Paging (5)

■ Reaktion auf Seitenfehler



E - 56

6.2 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 - 58

6.1 Demand Paging (6)

▲ Performanz von Demand paging

- ◆ Keine Seitenfehler
 - effektive Zugriffszeit zw. 10 und 200 Nanosekunden

- ◆ Mit Seitenfehlern
 - 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

E - 57

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 zugegriffenen 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

E - 59