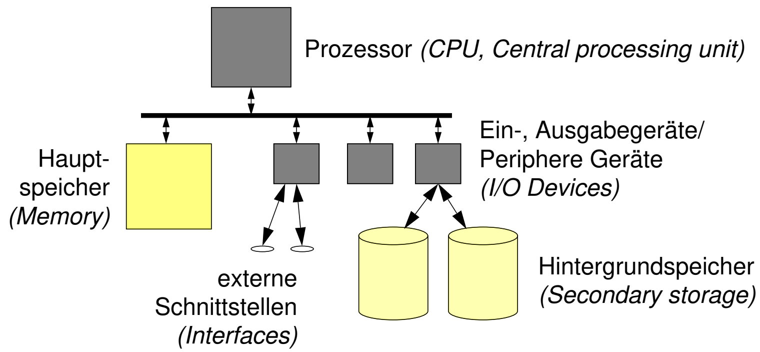
H Speicherverwaltung

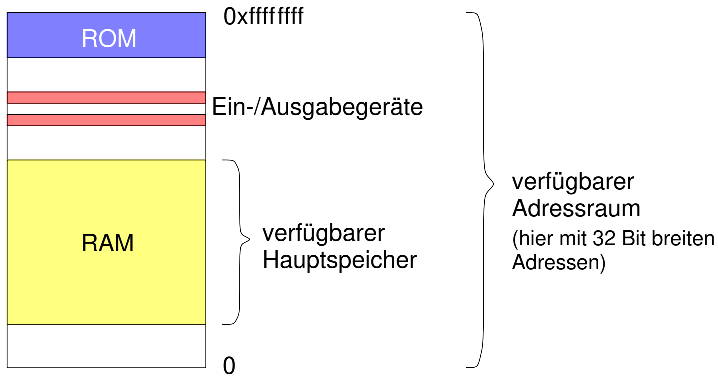
- Betriebsmittel Speicher



H.1 Speichervergabe

1 Problemstellung

- Verfügbarer Speicher



1 Problemstellung (2)

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

★ Zuteilung des Speichers nötig

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

3 Dynamische Speicherzuteilung

- Segmente
  - ◆ zusammenhängender Speicherbereich (Bereich mit aufeinanderfolgenden Adressen)
- Allokation (Anforderung) und Freigabe von Segmenten
- Ein Anwendungsprogramm besitzt üblicherweise folgende Segmente:
  - ◆ Codesegment
  - ◆ Datensegment
  - ◆ Stacksegment (für Verwaltungsinformationen, z.B. bei Funktionsaufrufen)

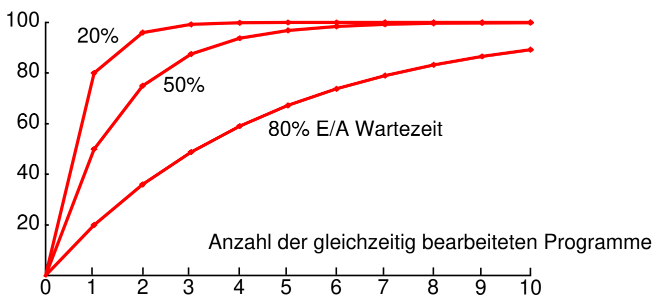
▲ Suche nach geeigneten Speicherbereichen zur Zuteilung

★ Speicherzuteilungsstrategien nötig

H.2 Mehrprogrammbetrieb

1 Problemstellung

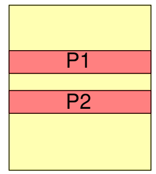
- Mehrere Prozesse laufen gleichzeitig
  - ◆ Wartezeiten von Ein-/Ausgabeoperationen ausnutzen
  - ◆ CPU Auslastung verbessern
- CPU-Nutzung in Prozent, abhängig von der Anzahl der Prozesse



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

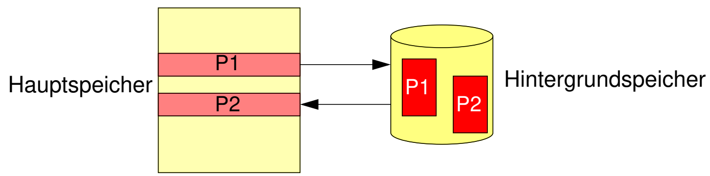
★ Relokation von Programmbefehlen (Binaries)

★ Ein- und Auslagern von Prozessen

★ Hardwareunterstützung

2 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

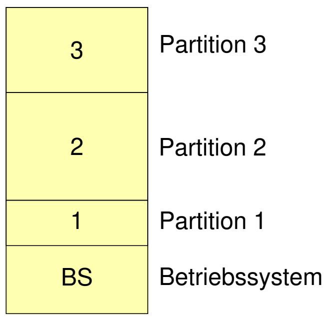


▲ Aus-, Einlagerzeit ist hoch

- ◆ Latenzzeit der Festplatte
- ◆ Übertragungszeit

2 Ein-, Auslagerung (2)

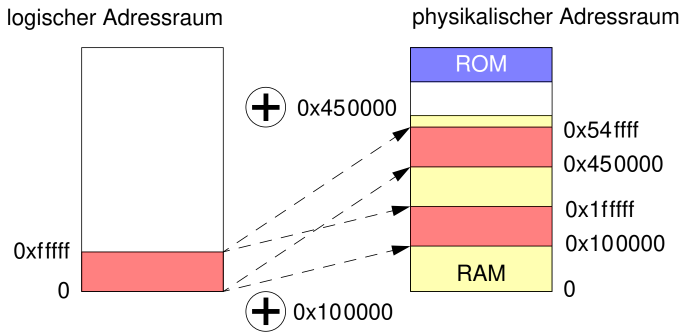
- Prozess ist statisch gebunden
  - ◆ kann nur an gleiche Stelle im Hauptspeicher wieder eingelagert werden
  - ◆ Kollisionen mit eventuell neu im Hauptspeicher befindlichen Segmenten
- ★ Mögliche Lösung: Partitionierung des Hauptspeichers



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

3 Segmentierung

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

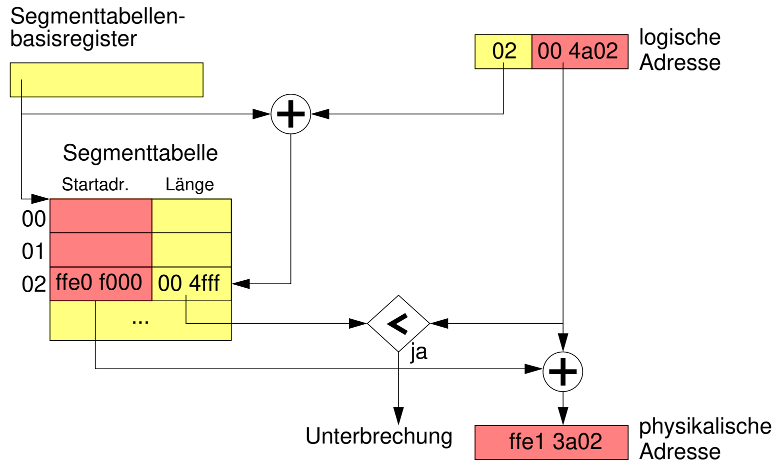


Das Segment im logischen Adressraum kann an jeder beliebige Stelle im physikalischen Adressraum liegen.

- Hardware wird MMU *(Memory management unit)* genannt

3 Segmentierung (2)

- Realisierung mit Übersetzungstabelle

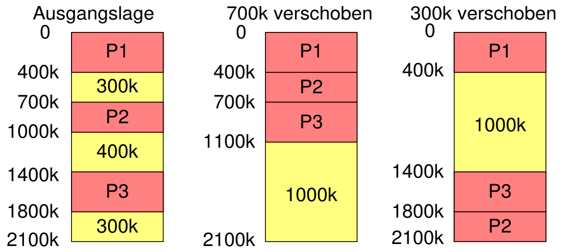


3 Segmentierung (3)

- Schutz vor Segmentübertretung
  - ◆ Unterbrechung zeigt Speicherverletzung an
  - ◆ Programme und Betriebssystem voreinander geschützt
- Zugriffsschutz einfach integrierbar
  - ◆ z.B. Rechte zum Lesen, Schreiben und Ausführen von Befehlen, die von der MMU geprüft werden
- 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*)

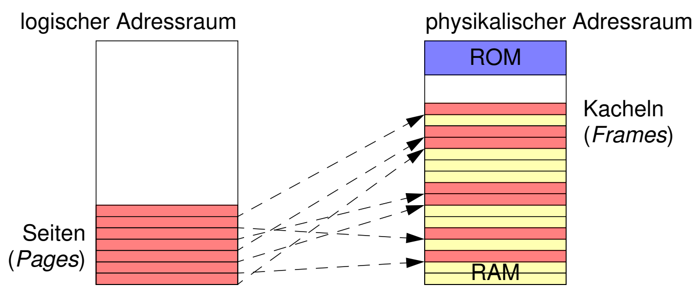
3 Segmentierung (4)

- ▲ 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
  - Erzeugen von weniger aber größeren Lücken ➜ Verringern des Verschnitts
  - aufwändige Operation, abhängig von der Größe der verschobenen Segmente



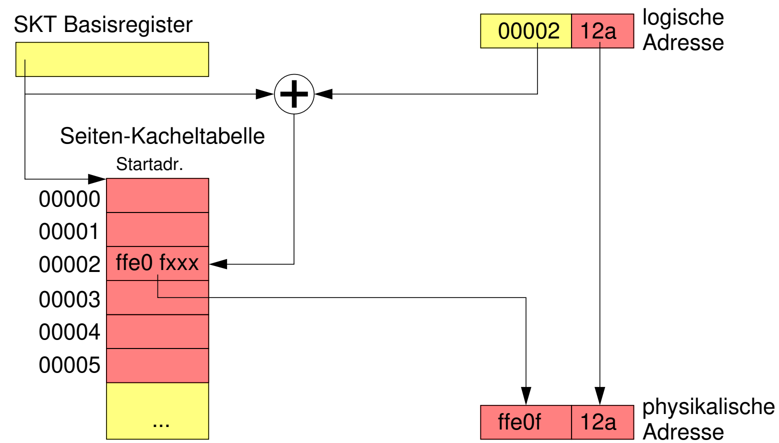
H.3 Seitenadressierung (*Paging*)

- ■ Einteilung des logischen Adressraums in gleich große 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

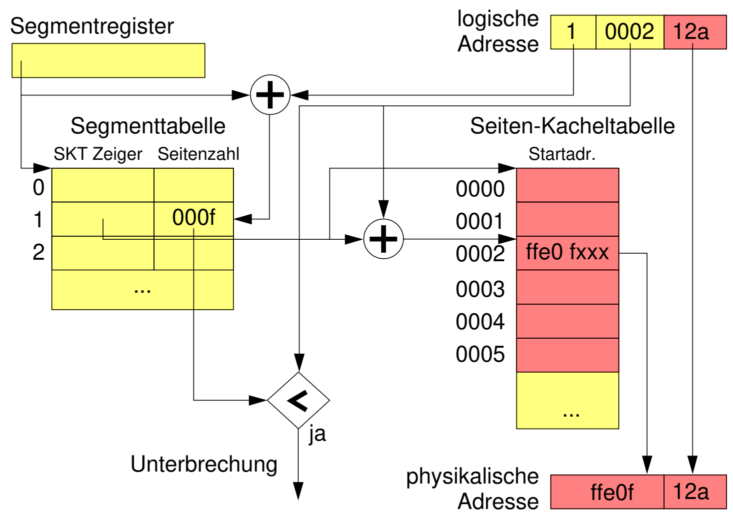
- ■ Tabelle setzt Seiten in Kacheln um



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

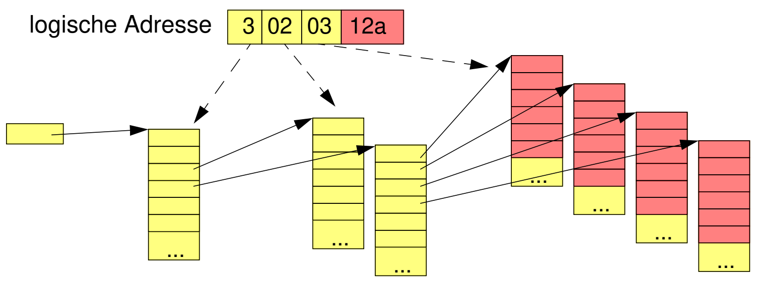
2 Segmentierung und Seitenadressierung



3 Segmentierung und Seitenadressierung (2)

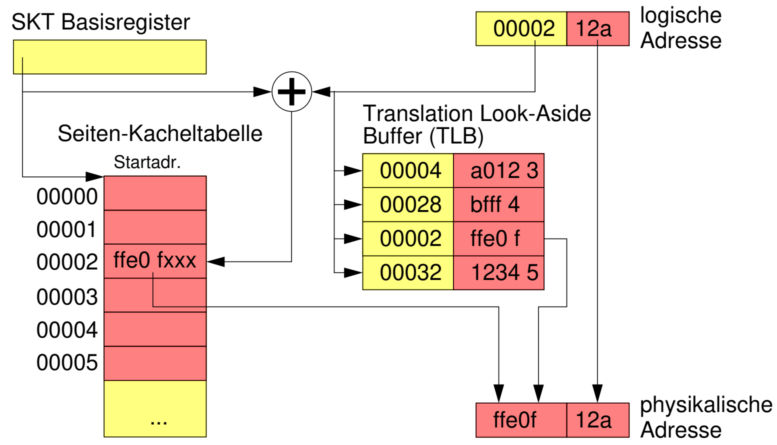
- ▲ noch mehr implizite Speicherzugriffe
- ▲ große Tabellen im Speicher
- ★ Mehrstufige Seitenadressierung mit Ein- und Auslagerung

4 Mehrstufige Seitenadressierung



5 Translation Look-Aside Buffer

- Schneller Registersatz wird konsultiert bevor auf die SKT zugegriffen wird



5 Translation Look-Aside Buffer (2)

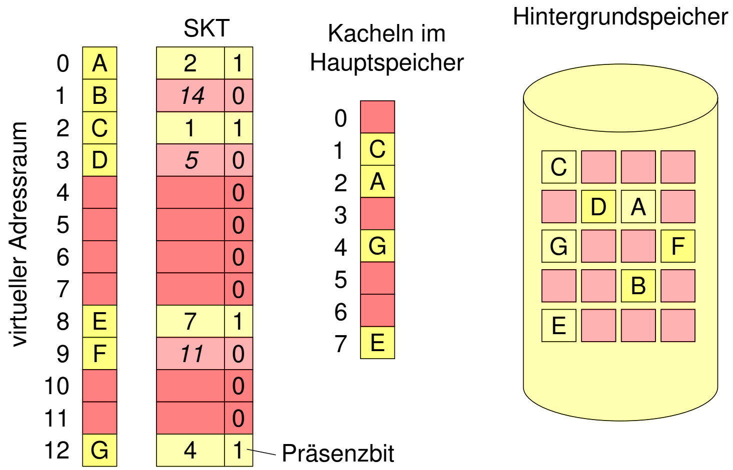
- Schneller Zugriff auf Seitenabbildung, falls Information im voll-assoziativen Speicher des TLB
  - ◆ keine impliziten Speicherzugriffe nötig
- Bei Kontextwechseln muss TLB gelöscht werden (*Flush*)
- Bei Zugriffen auf eine nicht im TLB enthaltene Seite wird die entsprechende Zugriffsinformation in 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

H.4 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 benötiger Speicherbereiche
  - ◆ Abfangen von Zugriffen auf nicht-eingeblendete Bereiche
  - ◆ Bereitstellen der benötigen Bereiche auf Anforderung
  - ◆ Auslagern nicht-benötigter Bereiche

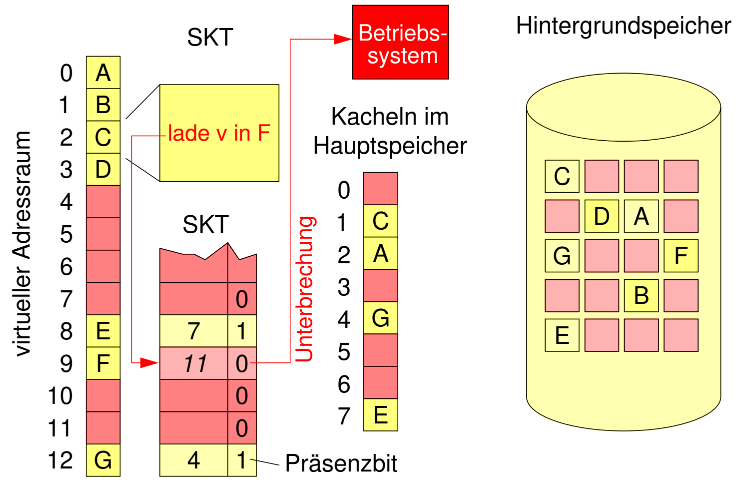
1 Demand Paging

- Bereitstellen von Seiten auf Anforderung



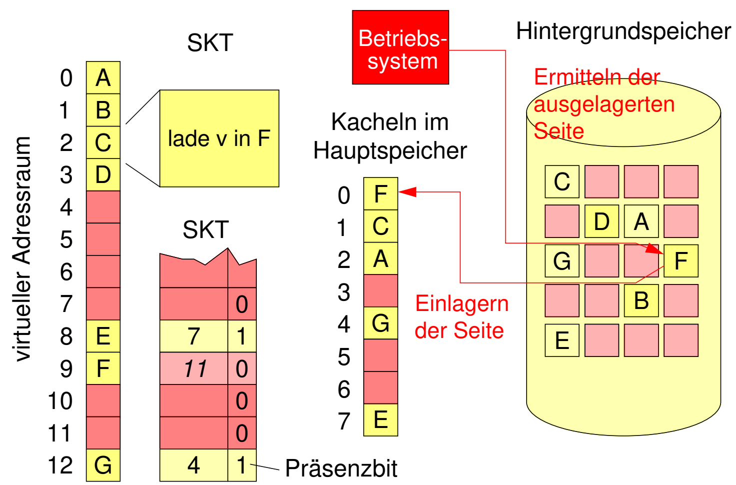
1 Demand Paging (2)

- Reaktion auf Seitenfehler



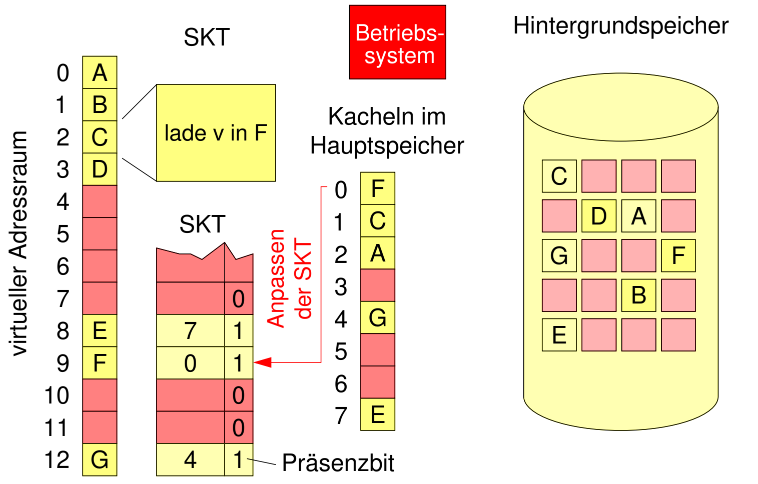
1 Demand Paging (3)

- Reaktion auf Seitenfehler



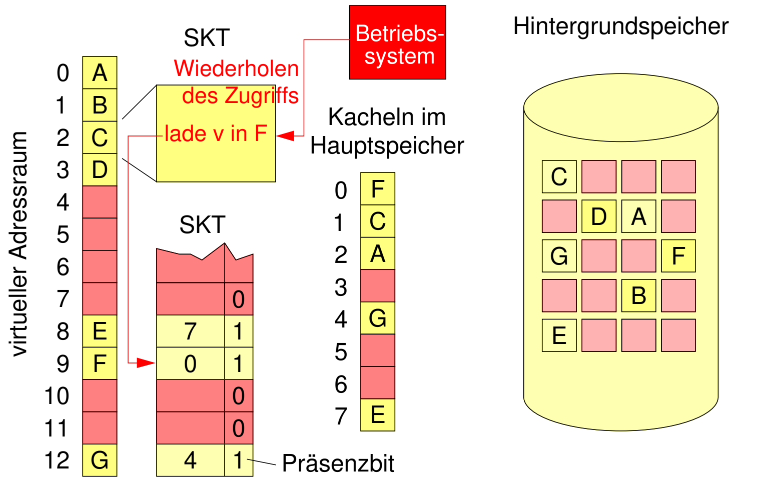
1 Demand Paging (4)

- Reaktion auf Seitenfehler



1 Demand Paging (5)

- Reaktion auf Seitenfehler



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?

H.5 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

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

2 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)

3 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	>

3 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

3 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)

4 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

4 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

4 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

4 Least Recently Used (4)

- Implementierung von LRU nicht ohne Hardwareunterstützung möglich
  - 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

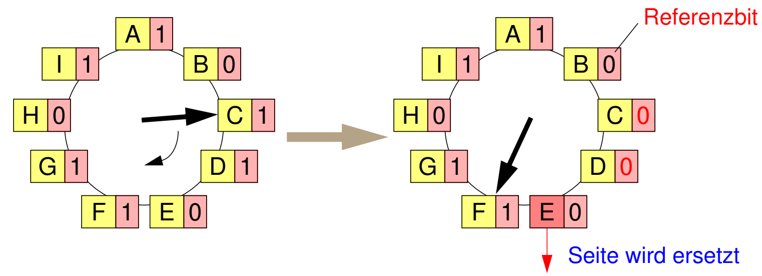
- Es muss jeder Speicherzugriff berücksichtigt werden
  ➜ viele zusätzliche Speicherzugriffe

5 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

5 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

5 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

5 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 (Refernzbits)	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

5 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)*
  - drei Klassen: (0,0), (1,0) und (1,1) mit (Referenzbit, Modifikationsbit)
  - Suche nach der niedrigsten Klasse (Einsatz im MacOS)

6 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.

H.6 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 benötigt zu viele Seiten
  - zu viele Prozesse gleichzeitig im System
  - schlechte Ersetzungsstrategie
- ★ Prozess-lokale Seitenanforderung behebt Thrashing zwischen Prozessen
- ★ Zuteilung einer genügend großen Zahl von Kacheln behebt Thrashing innerhalb der Prozessseiten
  - Begrenzung der Prozessanzahl