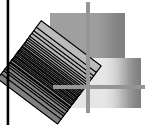


Adreßräume

Seitenersetzung (Paging)

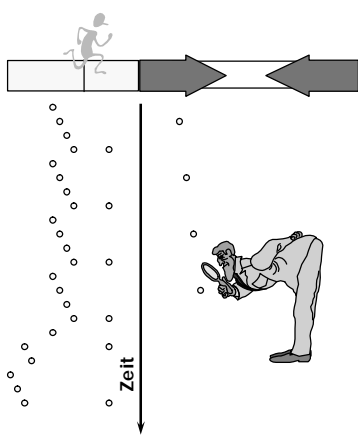
1



Motivation

- Instruktionen werden wiederholt ausgeführt
- Variablen werden wiederholt referenziert
- Gründe:
 - Sequentielle Ausführung überwiegt
 - Tatsächliche Prozedurverschachtelung gering
 - Kurze Schleifen
 - Häufige Verarbeitung von Feldern und Records

= Referenzlokalität



2

Erfahrungen mit Pascal und C

- Relative dynamische Frequenz von Hochsprachenkonstrukten im Quellcode
- Zuweisung und Bedingung ergeben über 80%
- Referenzen:
 - T. Huck
Comparative Analysis of Computer Architectures
Technical Report 83-243, Stanford University, Mai 1983
 - D. Patterson, C. Sequin
A VLSI RISC
IEEE Computer, Sept. 1982

Konstrukt	Pascal [Huck]	C [Patterson, Sequin]
Zuweisung	~75%	~40%
Bedingung	~20%	~45%
Prozedur	~5%	~15%
Schleife	~5%	~5%
Sprung	~5%	~5%

3

Caching: Ausnutzung der Referenzlokalität

```

    graph LR
      CPU[CPU] --> S1[Speicher S1  
1. Ebene  
(Cache)]
      S1 --> S2[Speicher S2  
2. Ebene]
      S1 -- Hit --> CPU
      S2 -- Miss --> S1
    
```

- 2-Ebenen-Speicher
- Kenngrößen eines Speichers S_k :
 - T_k = Mittlere Zugriffszeit
 - P_k = Preis pro Byte
 - G_k = Größe in Byte
- Trefferrate H
 - Wahrscheinlichkeit, daß referenzierter Wert im ersten Speicher ist

4

Mittlere Zugriffszeit und Kosten

- Mittlere Zugriffszeit:

$$T = HT_1 + (1 - H)(T_1 + T_2) = T_1 + (1 - H)T_2$$
- Kosten:

$$P = \frac{P_1G_1 + P_2G_2}{G_1 + G_2}$$
- Zugriffseffizienz:

$$\frac{T_1}{T} = \frac{1}{H + (1 - H)\frac{T_2}{T_1}}$$

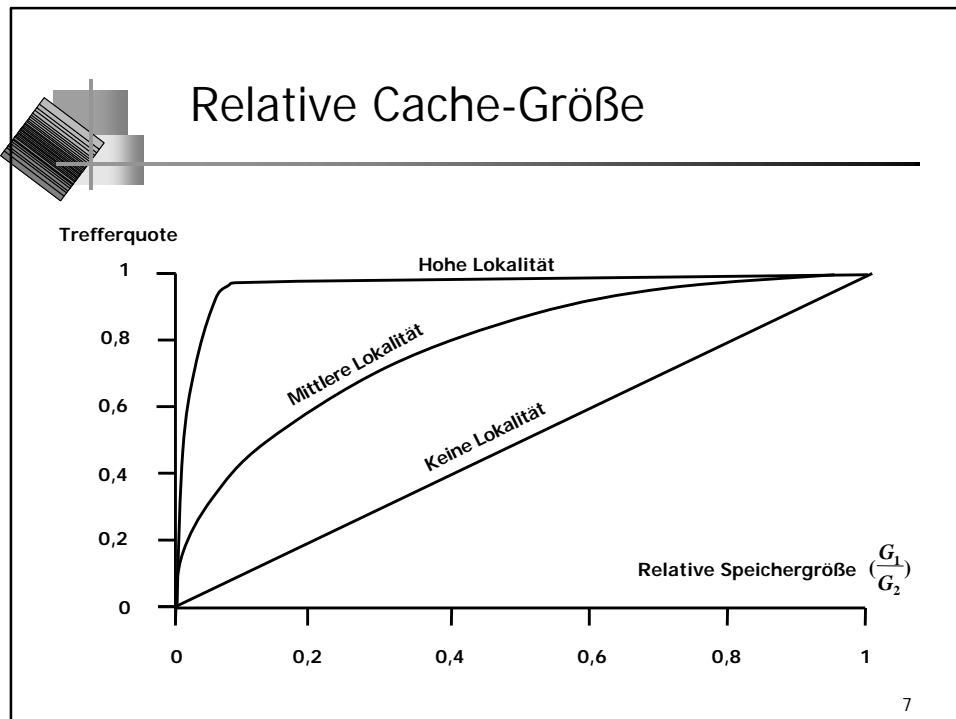
5

Zugriffseffizienz

The graph shows the access efficiency $\frac{T_1}{T}$ on the y-axis (logarithmic scale from 0,001 to 1) against the hit rate on the x-axis (linear scale from 0 to 1). Three curves are plotted for different ratios of $\frac{T_2}{T_1}$: 10, 100, and 1000. The curve for $\frac{T_2}{T_1} = 10$ starts at approximately 0,1 for a hit rate of 0 and reaches 1 at a hit rate of 1. The curve for $\frac{T_2}{T_1} = 100$ starts at 0,01 and reaches 1 at a hit rate of 1. The curve for $\frac{T_2}{T_1} = 1000$ starts at 0,001 and reaches 1 at a hit rate of 1. All curves show that efficiency increases with hit rate and decreases as the $\frac{T_2}{T_1}$ ratio increases.

- Typisch bei L2-Cache zu Hauptspeicher: $5 \leq \frac{T_2}{T_1} \leq 10$

6



- ### Optimierungsziel
- Anstrebenswert ist:
 - $T_1 \ll T_2: T \approx T_1$
 - $C_2 \ll C_1: C \approx C_2$
 - Erfordert Trefferquote H zwischen 0.8 und 0.9
 - Bereits sehr kleine Caches ($G_1 \ll G_2$) liefern hohe Trefferquote
- 8

Caching über mehrere Ebenen

- Ausnutzung der Speicherhierarchie
 - L1-Cache und MMU-TLB zu L2-Cache
 - L2-Cache zu Hauptspeicher
 - Hauptspeicher
 - Externer Speicher
- Caching zwischen internem und externem Speicher besonders vielversprechend:
 - Verhältnis bei Zugriffszeit: $1:10^5$
 - Größenverhältnis: $1:10^2-10^3$
 - Kostenverhältnis: $1:2000$

9

Paging

- Vorteile:
 - Vergrößerung des virtuellen Adreßraums
 - Adreßraum nur durch Größe des externen Speichers beschränkt
 - Gleichzeitige Unterstützung mehrerer Adreßräume
- Referenzlokalität verspricht minimale Zeitverzögerung?
- Ein- und Auslagern einzelner Seiten
 - P-Bit moderner MMU's
- Seitenfehler beim Zugriff auf ausgelagerte Seiten

10

Seitenfehler

The diagram illustrates the hardware mechanism of a page fault. The CPU provides a virtual address (split into 'p' for page and 'd' for displacement) to the MMU. The MMU consults the page table. If the P-bit (Present bit) is 0, it indicates that the page is not in memory, resulting in a 'Seitenfehler' (page fault) that is sent back to the CPU. The page table entry is shown with 'P-Bit = 0' and a question mark, indicating the state before the fault is resolved.

- P-Bit = 0:
 - Seite ausgelagert
 - Adreßraum nicht angelegt
- Unterscheidung durch:
 - speziellen Adreßwert im Seitendeskriptor
 - Nutzung der freien Bits in den Deskriptoren

11

Behandlung von Seitenfehlern

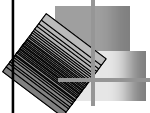
1. Adreßraum nicht angelegt (ungültige Referenz)?
 - Programm terminieren

Seite ausgelagert!
2. Freie Kachel im Arbeitsspeicher suchen
 - Belegte Kachel verdrängen (Auslagern), wenn keine freie Kachel vorhanden
3. Seitentransfer von Platte einplanen

Kontextwechsel: Ausführung anderer Kontrollflüsse

4. Seitentabelle aktualisieren
 - P-Bit setzen
 - Adresse der gewählten Kachel eintragen
5. Instruktion wiederholen
 - Instruktionwiederholung muß von CPU unterstützt werden

12



Effizienz von Seitenersetzungsverfahren

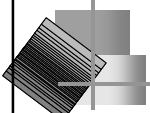
- Effektive Zugriffszeit abhängig von der Wahrscheinlichkeit eines Seitenfehlers p :

$$t_{\text{effektiv}} = (1 - p) * t_{HS} + p * t_{SF}$$
 - t_{HS} = Zugriffszeit Hauptspeicher (ca. 70ns)
 - t_{SF} = Zeit für die Behandlung eines Seitenfehlers
- Seitenfehlerbehandlung:

$$t_{SF} = t_{\text{Interrupt}} + t_{\text{Suchen}} + t_{\text{Auslagern}} + t_{\text{Einlagern}} + t_{\text{Wdh}}$$

$$\approx 2 * t_{\text{Ein-/Auslagern}}$$

13



Effektive Zugriffszeit

- Annahme:
 - Speicherzugriffszeit: 100 nsec
 - Plattenzugriff: 10 msec
$$t_{\text{effektiv}} = (1 - p) * 10^{-7} + 2p * 10^{-2}$$

$$\approx 2p * 10^{-2}$$
- Weniger als 10% Zeitverzögerung:

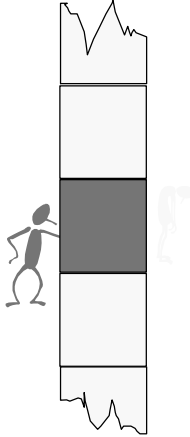
$$1.1 * 10^{-7} > 10^{-7} + 2p * 10^{-2}$$

$$p < 5 * 10^{-7}$$
 - höchstens alle 2 Millionen Referenzen ein Seitenfehler

14

Seitenverdrängungsstrategien

- Seitenfehler: keine freie Kachel für einzulagernde Seite
- Konsequenzen:
 - Anwendung wird terminiert
 - ein anderer Adreßraum wird vollständig ausgelagert (Swapping)
 - eine bereits belegte Kachel wird frei gemacht
- Fall 3: Seitenverdrängung
- Welche Seite soll verdrängt werden?
= Verdrängungsstrategie
 - minimale Seitenfehlerrate



15

Referenzstring

- Bewertung von Verdrängungsstrategien
- Berechnet sich aus der Zugriffsfolge
 - bei welchen Zugriffen entstehen Seitenfehler?
 - nur Seitenanteil wichtig
 - Seite für eine bestimmte Zeit (bis zur Verdrängung) im Speicher (Seite nicht zweimal hintereinander)
 - abhängig von der Anzahl an verfügbaren Kacheln
- Beispiel:
 - Zugriffe: 102, 103, 601, 702, 200, 104, 602, 603, 104, 101, 701, ...
 - Seitengröße 100
 - Referenzstring (1 Kachel): 1, 6, 7, 2, 1, 6, 1, 7
 - Referenzstring (2 Kacheln): 1, 6, 7, 2, 1, 6, 7
- Referenz-Referenzstring: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 1, 2, 0, 1, 7, 0, 1

16

Optimale Verdrängung (Belady)

- Die Seite verdrängen, die in der Zukunft am längsten nicht gebraucht wird
 - praktisch nicht implementierbar
 - dient der Bewertung anderer Verdrängungsstrategien
- Warum ist diese Strategie optimal?
- Beispiel:

7	0	1	2	0	3	0	4	2	3	0	3	1	2	0	7	0	1
7	7	7	2		2		2		2		2		2		7		7
	0	0	0		0		4		0		0		0		0		0
		1	1		3		3		3		3		1		1		1

17

FIFO

- Die Seite verdrängen, die sich am längsten im Speicher befindet
- Beispiel:

7	0	1	2	0	3	0	4	2	3	0	3	1	2	0	7	0	1
7	7	7	2		2	2	4	4	4	0		0	0		7	7	7
	0	0	0		3	3	3	2	2	2		1	1		1	0	0
		1	1		1	0	0	0	3	3		3	2		2	2	1

- Berücksichtigt Referenzlokalität absolut nicht
 - häufig verwendete Seite (Hot Spot) kann ausgelagert werden
 - selten referenzierte Seiten bleiben lange im Hauptspeicher
- Realisierung über Liste

18

Belady's Anomalie bei FIFO

- Referenzstring:
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- Anomalie:
 - Höhere Seitenfehlerrate bei mehr verfügbaren Kacheln

Anzahl Seitenfehler

Kacheln

19

Least Recently Used (LRU)

- Die Seite verdrängen, die in der Vergangenheit am längsten nicht referenziert wurde
 - Referenzlokalität: Was in der Vergangenheit lange nicht referenziert wurde, wird auch in der Zukunft lange nicht referenziert
 - Projektion der Vergangenheit in die Zukunft
- LRU nähert optimale Verdrängung am besten an
- Schwierig zu implementieren

7	0	1	2	0	3	0	4	2	3	0	3	1	2	0	7	0	1
7	7	7	2		2		4	4	4	0		0	2	2	2		1
	0	0	0		0		0	0	3	3		3	3	0	0		0
		1	1		3		3	2	2	2		1	1	1	7		7

20

Implementierung von LRU

- Probleme
 - Seiten nach dem letzten Zugriff ordnen
 - **JEDE** einzelne Referenz (Lesend oder Schreibend) muß berücksichtigt werden
- Realisierung mit Hilfe eines Zählers:
 - Seitendeskriptor um ein Zählerfeld erweitern
 - CPU um eine logische Uhr erweitern (bei jedem Zugriff inkrementieren)
 - MMU überträgt bei jedem Zugriff CPU-Uhr in Seitendeskriptor (Schreiboperation)
 - Seite mit kleinstem Stempel wird verdrängt
- Realisierung mit Hilfe eines Kellers:
 - Keller referenzierter Seiten
 - Referenzierte Seite wird aus Keller entfernt und kommt an die Spitze (Keller = doppelt verkettete Liste)
 - Letzte Seite wird verdrängt

21

LRU-Näherung I

The diagram illustrates the LRU approximation I mechanism. On the left, a 'Seitentabelle' (page table) has five rows, each with an 'R' bit. An arrow points from these 'R' bits to a vertical counter containing five '0's. Another arrow points from the counter to a 'Zusatztable' (reference bit table) on the right. The 'Zusatztable' has five rows, each with an 'R' bit and a binary value: 00100011, 01000000, 01010110, 00111001, and 10000100. An arrow labeled 'Shift um 1 nach links' points to the right above the 'Zusatztable', indicating a left shift operation.

- Zusätzliche Tabelle:
 - Nutzung des R-Bits in der Seitentabelle
 - Periodisch (z.B. 100 msec) Übernahme der Referenzbits in die Zusatztable
 - Tabelleneintrag spiegelt die Referenzen in den letzten n Zyklen wider
- Seiten mit dem kleinsten Wert am längsten nicht referenziert

22

LRU-Näherung II: 2nd Chance

- Basiert auf FIFO-Algorithmus
- FIFO wählt Seite, die am längsten im Speicher ist
 - R-Bit = 0: Seite wird verdrängt
 - R-Bit = 1: Seite bekommt eine zweite Chance
 - Seite wird wieder an den Kopf der FIFO-Liste eingereiht
 - R-Bit wird zurückgesetzt
- 2nd Chance degeneriert zu FIFO wenn alle R-Bits gesetzt sind

23

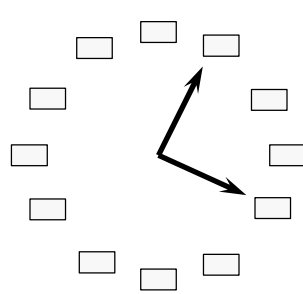
Clock-Algorithmus

- Implementierungsvariante des 2nd Chance
 - Aktualisierung der FIFO-Liste zu aufwendig
 - Verkettung der Seiten in einer zirkulären Liste
- "Uhrzeiger" zeigt auf die älteste Seite:
 - R-Bit = 0: Seite verdrängen
 - R-Bit = 1:
 - Zurücksetzen
 - 1 "Stunde" weiter

24

Variante des Clock-Algorithmus

- Clock-Algorithmus zu langwierig bei großem Hauptspeicher
- Zwei Zeiger:
 - Vordere Zeiger setzt R-Bit zurück
 - Hintere Zeiger prüft R-Bit
 - Beide Zeiger springen 1 Stunde vor
- Geringer Abstand zwischen Zeiger: Nur sehr häufig referenzierte Seiten bleiben
- Maximaler Zeigerabstand: Original Clock-Algorithmus
- BSD UNIX



Das Diagramm zeigt eine kreisförmige Anordnung von 12 rechteckigen Boxen, die Speicherseiten darstellen. Zwei schwarze Pfeile, die als 'Zeiger' bezeichnet werden, bewegen sich im Uhrzeigersinn durch den Kreis. Der vordere Zeiger (oben rechts) zeigt auf eine Box, die als 'R' markiert ist, was den Rücksetzpunkt des R-Bits darstellt. Der hintere Zeiger (unten rechts) zeigt auf eine Box, die ebenfalls als 'R' markiert ist, was den Prüfpunkt darstellt. Die Boxen sind in einem Abstand von einer 'Stunde' (einer bestimmten Anzahl von Boxen) voneinander angeordnet.

25

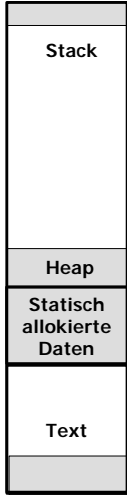
Weitere Verdrängungsstrategien

- Zufallswahl
 - Eine zufällig ausgewählte Seite wird verdrängt
- Least Frequently Use (LFU)
 - Zähler, der bei jedem Seitenzugriff inkrementiert wird
 - Seite mit dem kleinsten Zählerstand wird verdrängt
 - Problem: Kurzzeitig häufig und dann nicht mehr benötigte Seiten (z.B. Initialisierung)
 - Periodisch Links-Shift des Zählers (Exponentielle Dämpfung)
- ...

26

Kachelzuteilung

- **Initiale Seitentabelle für neu eingerichteten Adreßraum:**
 - Leer
 - Partiiell besetzt
 - Vollständig initialisiert
- **Demand Paging**
 - Eine Seite wird erst beim ersten Zugriff (Seitenfehler) geladen
- **Prepaging**
 - Seiten werden vor dem ersten Zugriff geladen
 - Ausnutzung der Zugriffscharakteristik auf Platten
 - Zugriffszeit auf erste Seite lang
 - weitere Seiten (Sektoren) schnell
- **Mischformen**



Das Diagramm zeigt die vertikale Anordnung von Speicherbereichen. Von oben nach unten sind dies: Stack, Heap, Statisch allokierte Daten und Text. Die Bereiche sind als vertikale Balken dargestellt, die durch horizontale Linien getrennt sind.

27

Kachelanzahl

- **Wechselwirkungen:**
 - Möglichst wenig Kachel zuordnen
 - mehr Adreßräume
 - freie Kacheln für Notfälle (?)
 - Nachteil: u.U. Leistungseinbußen trotz freier Kacheln
 - Möglichst viele Kacheln zuordnen
 - geringe Seitenfehlerrate
 - Nachteil: u.U. Ressourcenverschwendung
- **Jedem Adreßraum gleich viele Kacheln?**
- **Minimum:**
 - Nach Seitenfehler müssen alle Seiten geladen werden, die für die Instruktionwiederholung notwendig sind
 - 8 Seiten und mehr (z.B. Blockkopierbefehle)

28

Seitenflattern (Thrashing)

- Schwellwert K
- Mehr als K Kacheln allokiert:
 - Seitenfehlerrate klein genug
 - viele Seiten selten referenziert
 - u.U. für andere Adreßräume besser einsetzbar
- Genau K Kacheln
- Weniger als K Kacheln allokiert:
 - Kontrollfluß löst ständig Seitenfehler aus
 - Betriebssystem lädt nur Seiten nach
 - Kontrollfluß kommt praktisch nicht mehr an die Reihe
- Wie groß ist K ?

Relative CPU-Auslastung

K

Kachelanzahl

29

Seitenflattern ist ein globaler Effekt

- Seitenflattern beginnt in ein oder mehreren Adreßräumen
- Viele Ein- und Auslagerungs-aufträge für Platte
- Mittlere Zugriffszeit bei E/A wächst
- Ohne Vorkehrungen kommt praktisch das ganze System zum Stillstand

Hauptspeicher

Platte

30

Working Sets

- Magische Schwellwert K
- P. Denning 1968
The Working Set Model for Program Behavior
CACM, Mai 1968
- Working-Set (WS) eines Kontrollflusses:

WS(t, Δ) = letzten Δ referenzierten Seiten vor t

- Wahl von Δ kritisch:
 - zu klein: Lokalmengemenge nicht enthalten
 - zu groß: mehrere Lokalmengemengen enthalten
 - günstige Werte liegen bei ca. 10⁴

31

Beispiel

.... 2 4 5 6 3 4 3 3 3 2 3 5 6 7 4 2 3 4 5 4 4 3 3 3 3 2 3 2 3 4 5 1 2 3 4 5 6 5 6 5 4 3 3 3 4 5

← D ↑

WS(t1,D) = {2,3,4} t1

← D ↑

WS(t2,D) = {1,2,3,4,5} t2

- Größe des Working-Set für Kachelzuteilung wichtig:

D = ∑ |WS_i(t, Δ)|

- D > Verfügbare Kachelanzahl: Thrashing-Gefahr
 - Adreßräume auslagern (Swapping)
 - Kontrollflüsse suspendieren
- D < Verfügbare Kachelanzahl: Weitere Kontrollflüsse aktivieren

32

Bestimmung des Working Set

- Muß für jeden Prozeß berechnet werden
- Aufwendige Realisierung:
 - Referenzierte Seite kommt an den Anfang einer Δ großen Schlange
 - Letzte in der Schlange gespeicherte Seite fällt raus
 - Jede Seite in der Schlange ist im Working Set
- Änderungen rechtzeitig mitbekommen
- Approximation: siehe LRU-Näherung I

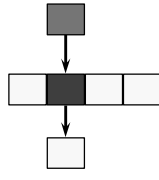
33

Vermeidung von Thrashing

- Beobachtung der Seitenfehlerrate
- Überschreitung einer Maximalfrequenz: Thrashing
 - Zusätzliche Kacheln zuordnen
 - AdreBräume auslagern (Swapping)
- Unterschreitung einer Minimalfrequenz:
 - Kacheln entziehen

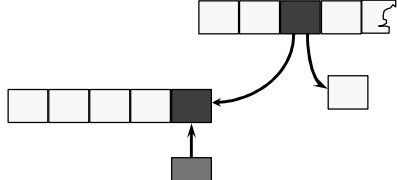
34

Globale und lokale Verdrängung



Lokale Verdrängung

- Eine Kachel des eigenen Adreßraums wird verdrängt
 - Anzahl zugewiesener Kacheln bleibt gleich
 - Kachelanzahl selbst-bestimmt
 - Working-Set sinnvoll
 - Geringerer Durchsatz (u.U. schwer, weitere Kacheln zu bekommen)



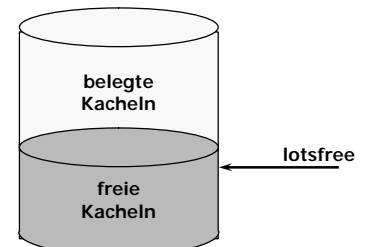
Globale Verdrängung

- Eine Kachel eines beliebigen Adreßraums wird verdrängt
 - Höherpriore Kontrollflüsse können ihren Bedarf bei anderen Adreß-räumen decken
 - Kachelanzahl hängt auch von anderen Kontrollflüssen ab
 - Höherer Durchsatz

35

UNIX: Dämon-Paging

- Seitenfehlerbehandlung und Verdrängung trennen
- Spezieller, nebenläufiger Prozeß sorgt für freie Kacheln
 - alle 250 msec aktiv
 - Schwellwert (lotsfree) an freien Kacheln unterschritten?
 - Auslagern von Seiten gemäß Clock-Algorithmus (2-Zeiger-Variante)
 - Globale Verdrängung
- lotsfree ca. 1/4 des verfügbaren Speichers



36