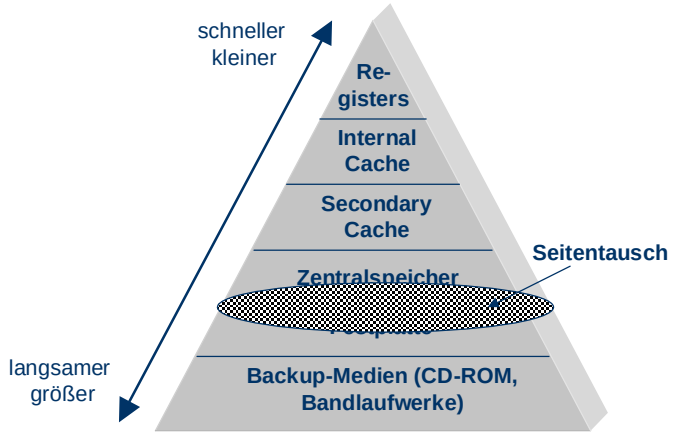
Seitentauschstrategien in Theorie und Praxis für den Zentralspeicher

Dr. Ulrich Margull

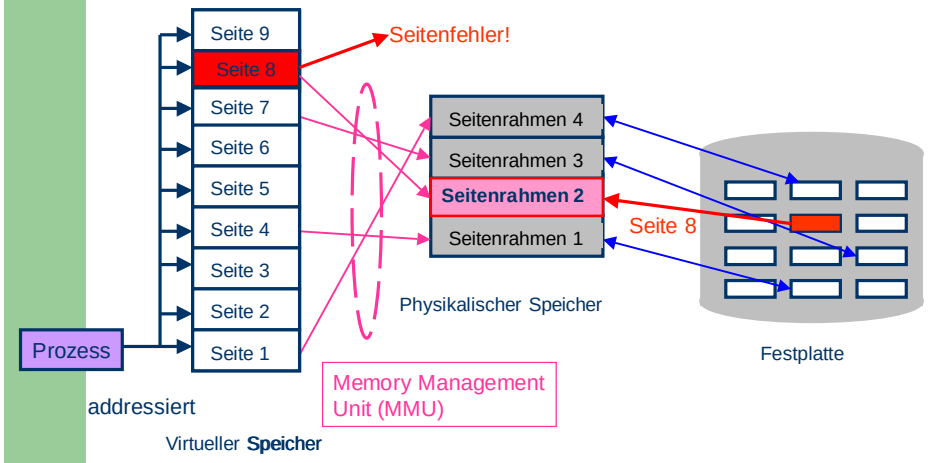
Die Speicherhierarchie



Übersicht

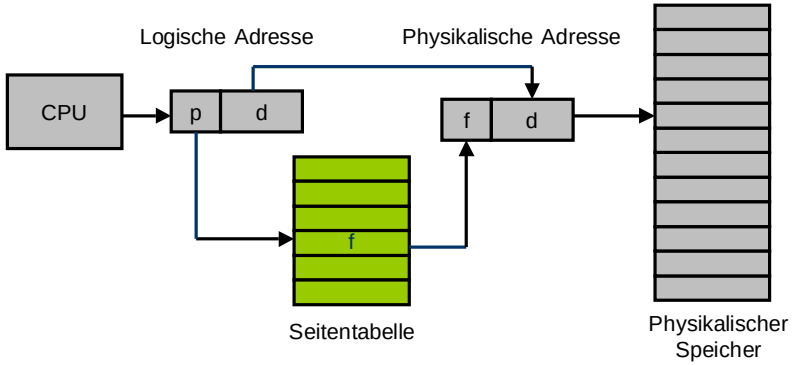
- Virtuelle Speicherverwaltung
- Seitentauschstrategien
- Praxisbeispiele
- Zusammenfassung
- Bibliographie

Prinzip des virtueller Speichers



Memory Management Unit (MMU)

- Umsetzung von logischer in physikalische Adresse (durch MMU)



Virtuelle Speicherverwaltung

- Prozess arbeitet mit logischen Adressen, werden vom BS auf physikalische umgesetzt
- Virtueller Speicher wird in *Seiten (pages)* unterteilt
- Hardware-Unterstützung durch MMU
  - Umsetzung von virtueller auf physikalische Adresse
  - Seitenfehler (Page Fault) bei Zugriff auf ausgelagerte Seite
  - R-Bit (Reference Bit): wird gesetzt, wenn Seite benutzt wurde
  - M-Bit (Modified Bit): wird gesetzt, wenn Seite geändert wurde
- Seiten werden auf Anforderung vom BS bereitgestellt (Demand Paging)

Virtueller Speicher

- Trennung von logischem und physikalischem Speicher
  - Ein Prozess adressiert logische (=virtuellen) Speicheradressen
  - Diese werden vom Betriebssystem in physikalische Adressen umgesetzt
- Vorteile:
  - Ein einzelner Prozess kann einen größeren Speicher adressieren als physikalisch vorhanden ist
  - Alle Prozesse erhalten den identischen (virtuellen), flachen Speicherbereich
  - Schutz der Programme voneinander
  - Gemeinsame Speicherbereiche und Dateien
  - Effektive Prozesserzeugung

Anforderungen an Prozessor

- Bei Seitenfehler: Unterbrechung der Prozessor-Anweisung -> muss später wiederholt werden!
  Problem: nicht immer möglich!
- Z.B. indiziertes Lesen: lade den Speicherinhalt, auf den R1 zeigt, an die Speicherstelle, auf die R2 zeigt; dabei wird R1 nach der Anweisung um eins erhöht, und R2 vor der Anweisung um eins erniedrigt:
  MOV (R1)+, -(R2)
  Die Anweisung enthält 3 Speicherzugriffe; nach einer Unterbrechung ist unklar, welche Register verändert wurden
- Mem-Copy Befehle, kopieren (oder verschieben) einen Speicherbereich, z.B. 256 Bytes auf einmal -> Page Fault kann jederzeit auftreten, wie kann die Anweisung wiederholt werden?
- Prozessor muss für virtuelles Speichermanagement geeignet sein!

Übersicht

- Virtuelle Speicherverwaltung
- Seitentauschstrategien
- Praxisbeispiele
- Zusammenfassung
- Bibliographie

Seitentauschstrategien

- Ziel: möglichst wenig Seitenfehler
- Implementierung muss mit vorgegebener Hardware möglich sein
- Test einer Seitentauschstrategie, indem eine spezielle Seitenanforderungsfolge (reference string) durchlaufen und die Anzahl der Seitenfehler gemessen wird
  - Beispiel: 0, 1, 2, 4, 0, 1, 3, 0, 1, 2, 4, 3
  - Seitenanforderungsfolgen können zufällig mit vorgegebenen Verteilungen erzeugt werden, oder von echten Programmläufen gewonnen werden

Seitentauschstrategien

- Problem: wenn kein freier Seitenrahmen vorhanden ist, wie kann eine neue Seite eingelagert werden?
  - Es muss zunächst eine andere entfernt werden!

Welche Seite kann zuerst entfernt werden?

- Ziel: möglichst wenig Seitenfehler
  - Verschiedene Seitentauschstrategien
  - Implementierung muss mit vorgegebener Hardware möglich sein

Optimale Strategie

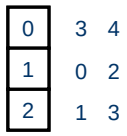
- Optimale Strategie: Suche die Seite, die am spätesten wieder benutzt wird
- Im Allgemeinen unmöglich zu implementieren, da keine Vorhersage in die Zukunft möglich ist
- Ist jedoch wichtig als Vergleichsmöglichkeit für andere Seitentauschstrategien

First-In First-Out (FIFO)

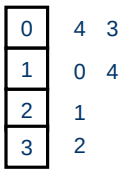
- Die Seite wird entfernt, die als erste geladen wurde
- Einfach zu implementieren
- Nur minimale Hardware-Unterstützung nötig
- Anfällig für Belady‘s Anomalie: in seltenen Fällen kommt es bei *mehr* Seitenrahmen zu einer *Erhöhung* der Seitenfehler (Belady et.al., 1969)
- *Wurde in den ersten VAX/VMS Rechnern verwendet[2]*

Belady‘s Anomalie

- Seitenanforderungsfolge:0,1,2,3,0,1,4,0,1,2,3,4



9 Seitenfehler
(Optimal: 7)



10 Seitenfehler
(Optimal: 6)

- Bei FIFO mit Tiefe 3 bzw. 4 und 5 Seitenrahmen sowie 12 Seitenanforderungen kommt die Anomalie genau zweimal vor (zweiter Fall: 012401301243)

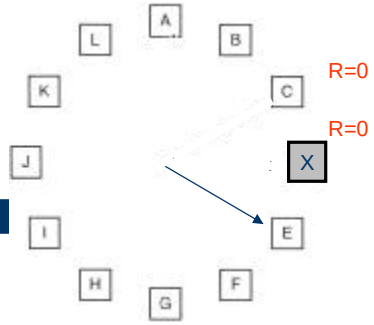
Least-Recently Used (LRU)

- Suche die Seite, die am längsten unbenutzt war
- Kommt der optimalen Strategie nahe, ist jedoch sehr aufwändig zu implementieren
- LRU Simulationen in Software:
  - Verwendung von Zähler
  - Stack Algorithmen
- Alle LRU-Algorithmen sind immun gegen Belady‘s Anomalie

LRU Näherung: Second Change (SC)

- Wie FIFO, mit folgender Änderung:
- Wurde die zu ersetzende Seite referenziert (R-Bit gesetzt), so wird sie nicht entfernt
  - Die Seite wird wie eine neue Seite an den Anfang der FIFO-Warteschlange gesetzt
  - Das R-Bit wird gelöscht.
  - SC sucht die älteste, nicht-benutzte Seite; falls alle Seiten benutzt werden, so ist SC identisch mit FIFO
- Wird auch *Clock* genannt

Second Chance: Beispiel



- Alle Seiten werden in einer kreisförmigen Liste angeordnet; Zeiger zeigt jeweils auf die älteste Seite
- Tritt ein Seitenfehler auf, so wird die älteste Seite ersetzt (auf die der Zeiger zeigt), falls sie nicht benutzt wurde (R-Bit gleich 0). Andernfalls wird das R-Bit gelöscht und der Zeiger solange weiter bewegt, bis eine Seite mit R=0 gefunden wird.
- Variationen mit mehreren Zeigern möglich (z.B. two-handed SC mit zwei Zeigern)
  - Wurde in Unix verwendet (4BSD, System V) [1]

LRU Näherung: Zählerbasierte Tauschstrategien

- LFU (Least Frequently Used): Für jede Seite wird ein Zähler mitgeführt; wurde die Seite benutzt (d.h. R-Bit ist 1), so wird der Zähler erhöht.
  - Die Seite mit dem niedrigsten Zähler wird entfernt
  - Problem: LFU vergisst nicht: unbenutzte Seiten, die vor langer Zeit mal stark genutzt wurden, verbleiben im Speicher
- Altern (Aging): Bei unbenutzten Seiten wird der Zähler wird vermindert:
  - Exponentielles Altern:

```
z = z >> 1;
if( r_bit > 0 ) z += 128;
r_bit = 0;
```

  - Lineares Altern:

```
if( r_bit > 0 ) z += 3;
else if( z>0 ) z--;
r_bit = 0;
```

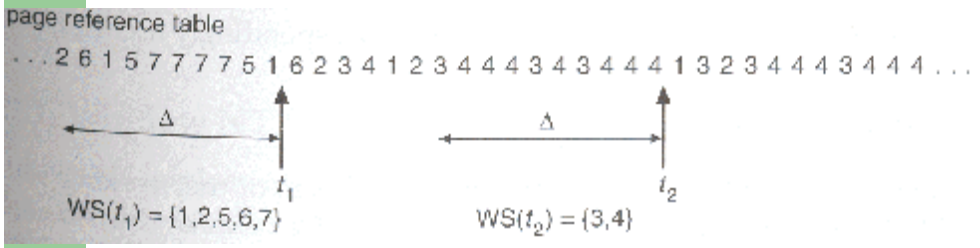
Verbesserter (Enhanced) Second Change

- Idee: unveränderte Seite ist schneller zu ersetzen als geänderte
- Die Seiten werden in 4 Klassen eingeteilt:
  - Klasse 0: nicht referenziert, nicht modifiziert (R=0, M=0)
  - Klasse 1: nicht referenziert, modifiziert (R=0, M=1)
  - Klasse 2: referenziert, nicht modifiziert (R=1, M=0)
  - Klasse 3: referenziert, modifiziert (R=1, M=1)
- Es wird eine beliebige Seite aus der niedrigsten, nicht-leeren Klasse genommen
- Wird im Macintosh VMM verwendet (nach [2])

Working Set Model

- Die meisten Prozesse nutzen nicht den gesamten virtuellen Speicher, sondern nur einen Teil davon. Die Menge der Seiten, die aktuell genutzt werden, heißt „Working Set“.
- Passt das Working Set nicht in den verfügbaren Speicher, so erzeugt der Prozess viele Seitenfehler: es kommt zum *Seitenflattern (Thrashing)*.
- Das Working Set ändert sich im Laufe der Zeit und muss entsprechend angepasst werden

Working Set Model: Beispiel



- Working Set wird dem aktuellen Bedarf des Prozesses angepasst

Übersicht

- Virtuelle Speicherverwaltung
- Seitentauschstrategien
- Praxisbeispiele
- Zusammenfassung
- Bibliographie

Weitere Aspekte Virtuellem Speichermanagements

- Pool freier Seiten
- Lokale versus globale Verteilungsstrategien
- Kontrolle der Auslastung (Vermeidung von Seitenflattern)
- Variable Seitengröße
- Von mehreren Prozessen genutzte Seiten (Memory Mapped Files, Shared Pages)
- Copy-on-write

Windows 2000/XP: VMM

- Pure Demand Paging, d.h. alle Seiten werden durch Seitenfehler geladen
- Berücksichtigung des Working Sets:
  - jeder Prozess bekommt Speicher zugeteilt (in Grenzen Min, Max)
  - Die Grenzen werden dem aktuellen Bedarf angepasst
  - Mischung aus lokaler und globaler Zuteilung
- Großer Vorrat an „freien“ Seiten
  - Mehrstufiger Seitenpuffer (geänderte, Standby, freie und gelöschte Seiten)

Windows 2000/XP: Algorithmus zum Entfernen von Seiten (Intel)

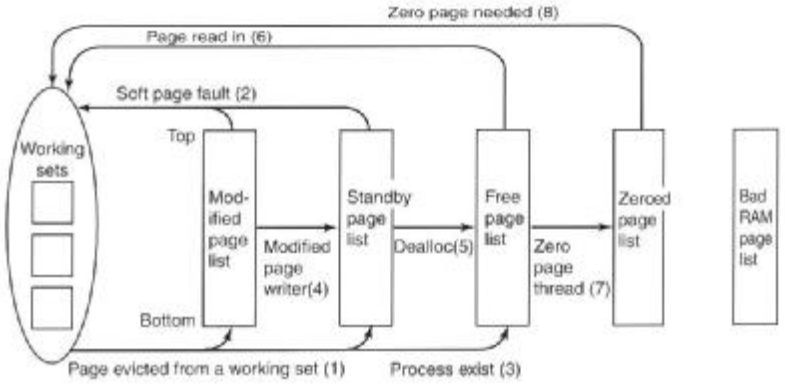
- Seitendieb (pagedaemon) startet zyklisch / bei Bedarf
- Alle Prozesse werden in bestimmter Reihenfolge untersucht
- Einprozessor-System: LRU Variante
  - Zähler für unbenutzte Seiten
    - Wird erhöht falls R-Bit=0
    - Wird auf null gesetzt, falls R-Bit=1
  - Seiten mit höchstem Zähler werden entfernt
  - Falls nötig, mehrere Durchläufe (zunehmend aggressiver)
- Multi-Prozessor: nur FIFO

Windows XP

- Pure Demand Paging, d.h. alle Seiten werden durch Seitenfehler geladen
- Vorrat an freien, gelöschten Seiten
- Working Set: jeder Prozess bekommt Speicher zugeteilt (in Grenzen, Min. und Max.) -> Lokale Zuteilung
- Das Working Set wird jedoch an den aktuellen Bedarf des Prozesses angepasst -> Globale Zuteilung
- Modifizierter Clock-Algorithmus
  - Alter der Seiten wird berücksichtigt
  - Falls nötig, jeweils mehrere zunehmend aggressivere Durchläufe

Windows 2000

- Mehrstufiger Seitenpuffer (nach [1]):



Unix allgemein

- Seitentausch durch Seitendieb (page daemon), z.B. alle 250 ms
- Großer Vorat an freien Seiten (typisch ¼ des Zentralspeichers), mit Seitenpufferung
- Globaler Algorithmus: keine Berücksichtigung des Working Sets
- Modifizierte Variante des Clock-Algorithmus (mit zwei Zeigern)
- 4BSD: Two-handed Clock
- System V: Seiten werden nur entfernt, wenn sie n-mal hintereinander nicht genutzt wurden

Linux

- Seitendieb *kswapd*, läuft einmal pro Sekunde (oder auf Anforderung, wenn Speicher voll ist)
- Großer Vorrat an freien Seiten
- Linux 2.2:
  - Kein Prepaging, keine Berücksichtigung des Working Sets
  - Seiten werden in mehreren Durchläufen entfernt (Variante von Second Chance)
- Linux 2.4:
  - Second Chance mit mehreren Durchläufen
  - LFU: Zähler mit Altern (jüngere Seiten haben hohen Wert)

Zusammenfassung Seitentauschstrategien

- Optimale Strategie nicht umsetzbar
- FIFO: einfach, nur minimale Hardware-Anforderungen, aber Belady Anomalie
- LRU: gut, schwer zu implementieren
- LRU Näherungen: Second Chance, LFU, Aging, u.a.
- Berücksichtigung des Working-Sets
  - Aufteilung des Speichers auf die verschiedenen Prozesse
- Seitentauschstrategie sind ein wichtige Teil der komplexen virtuellen Speicherverwaltung

Bibliographie

[1] „Modern Operating Systems“, 2nd Edition, Andrew Tanenbaum, Prentice Hall, 2001 (deutsch: „Moderne Betriebssysteme“ im Verlag Pearson Studium)

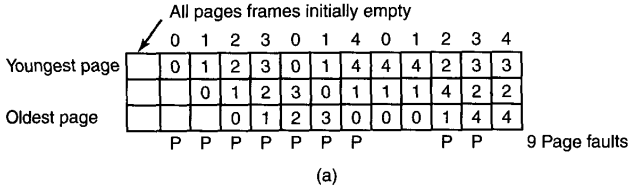
[2] „Operating System Concepts“, 6th Edition, A. Silberschatz et.al, Wiley, 2003

[3] „Vorlesung Betriebssysteme“, geschrieben von Dipl.Inf (FH) Eyke Langhans, überarbeitet von Prof. Dr. Jobst, SS1998 bis SS2001

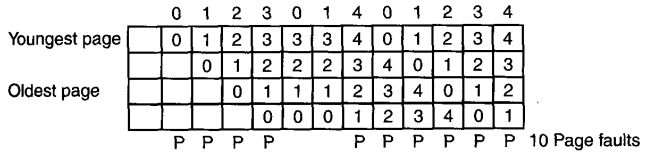
[4] Informationen zu Linux VM unter www.linux-mm.org, insbesondere „Too little, too slow“ von Rik van Riel

Appendix

Belady‘s Anomalie



(a)



(b)

- Bei FIFO mit Tiefe 3 bzw. 4 und 5 Seitenrahmen sowie 12 Seitenanforderungen kommt die Anomalie genau zweimal vor (zweiter Fall: 012401301243)

Nebeneffekte des Paging

- Paging kann zu Verzögerungen des Programms führen -> ungeeignet für Echtzeitsysteme
- Paging ist transparent für den Benutzer. Aber: Nebeneffekte sind möglich!
- Beispiel: Seitengröße 4kByte; Programm benötigt Feld mit 1024 mal 1024 Integer-Zahlen

```
int i, j, a[1024][1024];
// schnelle Schleife:
for(i=0; i<1024; i++)
  for(j=0; j<1024; j++)
        a[i][j] = 0;
```

```
// langsame Schleife:
for(i=0; i<1024; i++)
  for(j=0; j<1024; j++)
        a[j][i] = 0;
```

- Links 1024 Seitenanforderungen, rechts 1024*1024 !