Betriebssysteme I – Sommersemester 2009
Kapitel 6: Speicherverwaltung und Dateisysteme

Hans-Georg Eßer
Hochschule München

Teil 6: Nicht-zusammenhängende Speicherzuordnung (3/3)

07/2009

Demand Paging (1)

- Der Adressbereich eines Prozesses muss nicht vollständig im Hauptspeicher sein.
  - Das Lokalitätsprinzip besagt, dass ein Prozess in einer Zeitspanne nur relativ wenige, nahe beieinanderliegende Adressen anspricht.
  - Teile des Programms werden bei einem bestimmten Ablauf möglicherweise gar nicht benötigt (Spezialfälle, Fehlerbehandlungsroutinen etc.).

Demand Paging (2)

- Demand Paging bedeutet:
  - dass eine Seite nur dann in den Speicher geladen wird, wenn der Prozess sie anspricht,
  - dass eine Seite auch wieder aus dem Speicher entfernt werden kann.
- Vorteile von Demand Paging:
  - Der Adressbereich eines Prozesses kann größer sein als der physikalische Hauptspeicher.
  - Prozesse belegen weniger Platz im Hauptspeicher, somit können mehr Prozesse gleichzeitig aktiv sein.

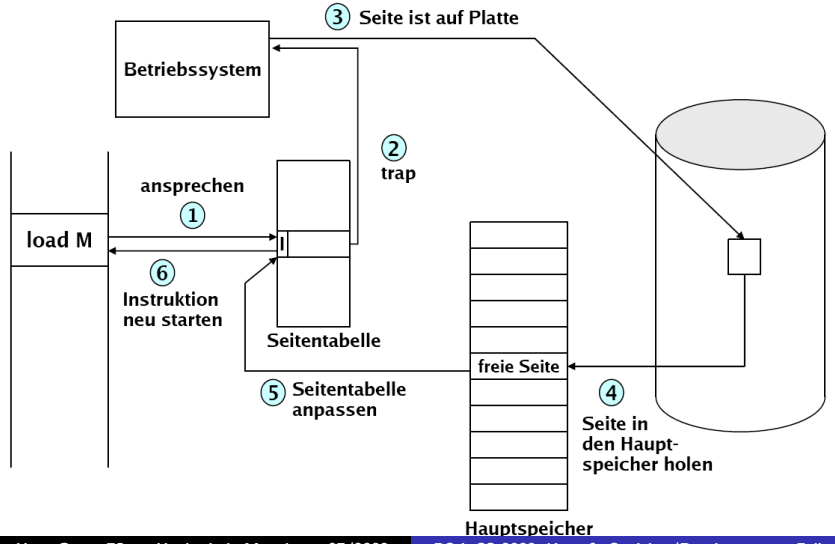
Voraussetzungen für Demand Paging (1)

- Jeder Eintrag in der Seitentabelle enthält ein valid bit, das angibt, ob die Seite im Speicher ist oder nicht.
- Wenn ein Prozess eine Seite anspricht, die nicht im Speicher ist, wird eine spezielle Exception ausgelöst, ein sog. page fault.
- Eine Betriebssystem-Routine, der page fault handler, lädt bei einem page fault die benötigte Seite in den Speicher.

Voraussetzungen für Demand Paging (2)

- Falls kein freier Seitenrahmen im Speicher vorhanden ist, muss eine andere Seite ersetzt werden. Für die Auswahl der zu ersetzenden Seite muss eine Strategie implementiert werden.
- Die durch den page fault unterbrochene Instruktion muss erneut ausgeführt werden (können).

Page-Fault-Behandlung



Seitenersetzung (1)

- Wenn bei einem Page Fault kein freier Seitenrahmen zur Verfügung steht, muss das Betriebssystem einen frei machen.
- Ein Algorithmus wählt nach einer bestimmten Strategie diesen Seitenrahmen aus.

Seitenersetzung (2)

- Falls die zu ersetzende Seite, seit sie zuletzt in den Speicher geholt wurde, verändert wurde, muss ihr aktueller Inhalt gesichert werden:
  - Ein modify bit (oder dirty bit) im Seitentabelleneintrag vermerkt, ob die Seite verändert wurde.
  - Eine veränderte Seite wird auf Platte gesichert (im sog. Page- oder Swap-Bereich).

Seitenersetzung (3)

- Eine unveränderte Seite kann später – bei Bedarf – wieder von der alten Stelle auf der Platte geladen werden.
- Im Seitentabelleneintrag für die ersetzte Seite wird
  - das valid bit gelöscht,
  - vermerkt, von wo die Seite wieder geladen werden kann.

Seitenersetzungsstrategien (1)

- Ziel: Es sollen so wenig Page Faults wie möglich auftreten.
- Zwei prinzipielle Arten von Seitenersetzungsstrategien:
  - lokale Ersetzung
  - globale Ersetzung

Seitenersetzungsstrategien (2)

Lokale Ersetzung: Es wird immer eine Seite desjenigen Prozesses ersetzt, der eine neue Seite anfordert.

- Die Zahl der Seiten, die ein Prozess im Speicher belegen kann, ist nach oben beschränkt. Die maximale Anzahl wird pro Prozess festgelegt (z. B. vom Systemverwalter) und kann die Laufzeit eines Prozesses stark beeinflussen.
- Ein Prozess, der viele Page Faults verursacht, z. B. weil er sich nicht an das Lokalitätsprinzip hält, beeinträchtigt nur sich selbst, nicht aber das Gesamtsystem.

Seitenersetzungsstrategien (3)

Globale Ersetzung: Es wird eine beliebige Seite im Speicher ersetzt.

- Prozesse nehmen sich gegenseitig Seiten weg.
- Ein Prozess, der viele Page Faults verursacht, erhält automatisch mehr Speicher. (Dies kann sowohl ein Vorteil als auch ein Nachteil sein.)

Optimale Strategie

Diejenige Seite ersetzen, auf die in Zukunft am längsten nicht zugegriffen wird.

- Vorteil: Diese Strategie verursacht die kleinste Zahl an Page Faults.
- Nachteil: Diese Strategie ist nicht implementierbar. (vgl. idealer Scheduler)

Die optimale Strategie kann modellhaft zur Bewertung anderer Strategien benutzt werden.

First In First Out (FIFO)

Die Seite ersetzen, die schon am längsten im Speicher ist.

- Vorteil: Sehr einfach zu implementieren:
  - Es wird eine verkettete Liste der Seiten im Speicher (globale Strategie) bzw. der Seiten eines Prozesses (lokale Strategie) unterhalten.
  - Bei einem Page Fault wird die erste Seite der Liste ersetzt und die neue Seite ans Ende der Liste angefügt.
- Nachteil: Die ersetzte Seite kann in dauernder Benutzung sein und gleich wieder angefordert werden.

Least Recently Used (LRU) (1)

Die Seite ersetzen, die am längsten nicht benutzt worden ist.

- Vorteil: In der Regel weniger Page Faults als FIFO.
- Nachteil: Aufwändige Implementierung.

Zwei mögliche Implementierungen:

- mit Zähler
- mit verketteter Liste

Least Recently Used (LRU) (2)

Implementierung mit Zähler:

- Systemweiten Zähler bei jedem Speicherzugriff inkrementieren.
- Aktuellen Wert des Zählers in einem Feld in der Datenstruktur vermerken, welche die angesprochene Seite beschreibt.
- Seite mit dem kleinsten Zählerwert ersetzen.

Least Recently Used (LRU) (3)

Implementierung mit verketteter Liste:

- Eine verkettete Liste enthält alle Seiten.
- Bei jedem Speicherzugriff wird die angesprochene Seite an den Anfang der Liste gebracht. (Liste durchsuchen und Reihenfolge ändern, also Zeiger umsetzen!)
- Die Seite am Ende der Liste wird ersetzt.

Benutzen eines Referenz-Bits (1)

- Jeder Seitentabelleneintrag kann ein Referenz-Bit enthalten
  - das bei einem Zugriff auf die Seite gesetzt wird (Hardware),
  - das nach bestimmten Kriterien wieder gelöscht wird (Software).
- Ein Referenz-Bit
  - liefert die Information, ob auf eine Seite seit dem letzten Löschen des Bits zugegriffen wurde,
  - sagt nichts über den Zeitpunkt des Zugriffs auf eine Seite aus,
  - sagt nichts über die Reihenfolge der Zugriffe auf mehrere Seiten aus.

Benutzen eines Referenz-Bits (2)

- Mit Referenz-Bits kann man weitere Seitenersetzungsstrategien implementieren, z. B.
  - Modifikationen von LRU, die weniger aufwändig sind.
  - Second-Chance-Algorithmus, eine Verbesserung der FIFO-Strategie.

Modifikation von LRU (1)

- Es wird ein binärer Zähler für jede Seite unterhalten.
- In regelmäßigen Abständen wird
  - jeder Zähler eine Position nach rechts geschoben („Aging“),
  - das Referenzbit in das höchste Bit des Zählers kopiert,
  - das Referenzbit gelöscht.

Modifikation von LRU (2)

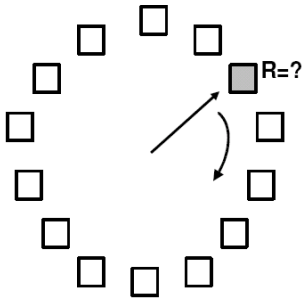
- Es wird eine der Seiten ersetzt, die den kleinsten Zählerwert enthalten.
  - Bei gleichem Zählerwert ist nicht bekannt, auf welche Seite zuletzt zugegriffen wurde.
  - Länger zurückliegende Zugriffe werden zunächst weniger stark gewichtet und schließlich „vergessen“ (aus dem Zähler hinausgeschoben).

Second-Chance-Algorithmus (1)

- Modifikation des FIFO-Algorithmus: Ist bei der Seitenersetzung das Referenz-Bit der ältesten Seite gesetzt, so wird
  - das Referenz-Bit gelöscht und die Seite am Ende der Liste eingereiht,
  - die gleiche Prüfung für die nächstälteste Seite durchgeführt.
- Es wird also
  - die älteste Seite ersetzt, deren Referenz-Bit gelöscht ist,
  - einer kürzlich benutzten Seite zunächst eine „zweite Chance“ gegeben.

Second-Chance-Algorithmus (2)

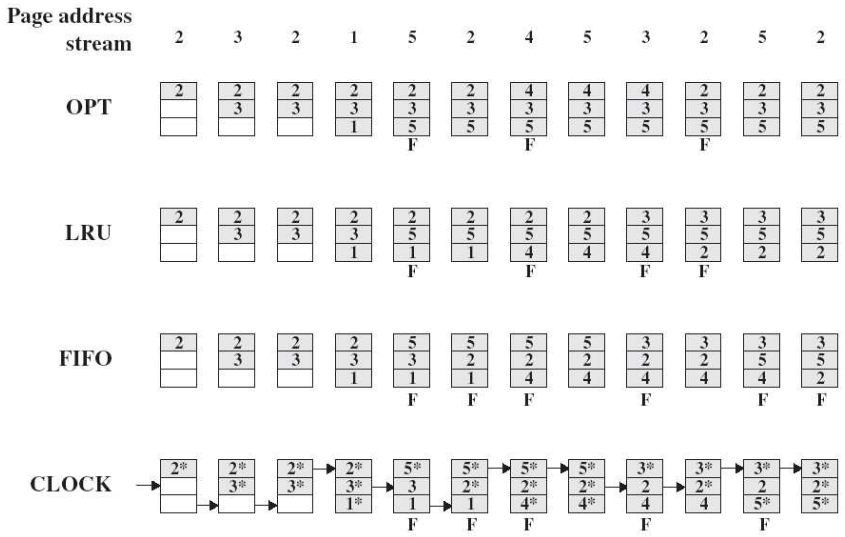
- Einfachere Implementierung: „Uhrzeiger“
  - Anordnung der Seiten in einer Ringliste, und Verschieben eines Zeigers statt Umpositionieren eines Listenelements.



Überprüfen der Seite, auf die der Zeiger zeigt:

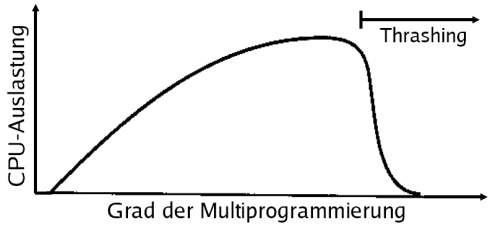
- Wenn $R = 0$: Seite ersetzen, Zeiger weiter bewegen
- Wenn $R = 1$: R löschen, Zeiger weiter bewegen, nächste Seite prüfen

Seitenersetzung, Beispiele



Thrashing (1)

- Thrashing bedeutet, dass ein Prozess exzessiv viele Page Faults macht (alle paar tausend Instruktionen).
- Thrashing entsteht, wenn ein Prozess mehr Seiten aktiv benutzt, als ihm Seitenrahmen zur Verfügung stehen.
- Thrashing mehrerer Prozesse führt zu niedriger CPU-Auslastung:



Thrashing (2)

Lösungen

- Falls noch freier Speicher vorhanden: Zuteilung weiterer Seitenrahmen an den betreffenden Prozess (z. B. durch globale Ersetzungsstrategie).
- Falls kein freier Speicher mehr: Auslagern (Swapping) von Prozessen.

Swapping (1)

Wichtig: Es gibt zwei Bedeutungen von Swapping:

- Eigentlich: komplettes Auslagern eines Prozesses (oder bei Segmentierung: eines ganzen Segments) auf die Festplatte (to swap: vertauschen; ein Prozess geht aus dem Hauptspeicher, damit ein anderer herein kommen kann)
- Allgemeiner: Auslagern von Teilen eines Prozesses (Paging: einzelne Seiten) auf die Platte

Swapping (2)

- Swapping ist die zeitweise Auslagerung **aller** von einem Prozess benutzten Speicherseiten (oder zumindest kompletter Segmente) auf einen Hintergrundspeicher (Platte), um z. B. bei zu wenig freiem Hauptspeicher Platz zu schaffen.
- Bei zusammenhängender Speicherzuteilung
  - ist Swapping die einzige Möglichkeit, mehr Programme gleichzeitig auszuführen, als im Hauptspeicher Platz haben,
  - muss der Speicher eines Prozesses, der dynamisch wächst, u. U. ausgelagert werden.

Swapping (3)

- Kriterien für die Auslagerung können z. B. sein:
  - Prozesszustand
  - Prozesspriorität
  - Prozessgröße (im Hauptspeicher)
  - Zeit, die der Prozess im Hauptspeicher war
- Die Zuteilung und Verwaltung des Platzes im Swapbereich auf Platte geschieht mit einem der Verfahren der zusammenhängenden Speicherverwaltung.

Swapping (4)

- Swapping ist ein Vorläufer von Paging
- Da Prozesse immer „ganz oder gar nicht“ im Speicher sind, lassen Swapping-basierte BS keine Prozesse zu, deren Speicherbedarf größer als der Hauptspeicher ist
- Linux und Windows: kein Swapping, sondern Paging (nur aus Traditionsgründen heißt es oft „Swap-Partition“, „Swap-Datei“ etc.)
- sehr alte Unix-Versionen: Swapping