Speicherverwaltung Teil I

Hard- und Software-Komponenten zur Speicherverwaltung

fachhochschule münchen
munich university of applied sciences

Inhaltsübersicht

- Zusammenhängende Speicherzuteilung
  - Partitionen fester Größe
  - Partitionen variabler Größe
  - Methoden zur Verwaltung des freien Speichers
  - Segmentierung
- Nicht zusammenhängende Speicherzuteilung
  - Virtuelle Speicherverwaltung (Paging)
  - Mehrstufiges Paging
  - Segmentierung mit Paging
- Demand Paging
  - Page Faults und deren Behandlung
  - Strategien für die Seitenersetzung
  - Weitere Design-Möglichkeiten
- Swapping

Motivation

Motivierende Fragen:

- Wie setzt sich eine Adresse zusammen ?
- Was geschieht bei einem Adreßzugriff eigentlich ?
- Wie kann man als Administrator oder Softwareentwickler Nutzen aus Kentnissen der Speicherverwaltung ziehen ?

Prinzipielle Arten der Speicherverwaltung

Zwei prinzipielle Arten der Speicherverwaltung:

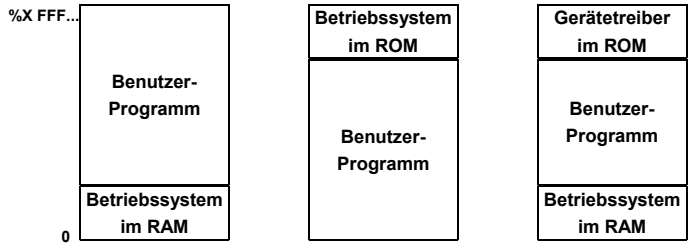
- Zusammenhängende Speicherzuteilung
  - Jede Anforderung eines Prozesses nach einer bestimmten Menge Speicher muß durch zusammenhängenden (contiguous) Speicher erfüllt werden.
- Nicht zusammenhängende Speicherzuteilung
  - Eine Speicheranforderung wird durch Zuweisung mehrerer kleiner Speicherbereiche erfüllt, die zusammen die geforderte Menge Speicher darstellen.
  - Das Wiederauffinden der verstreuten Speicherbereiche muß von der Speicherverwaltung übernommen werden.

Heutzutage wird der Hauptspeicher fast immer nicht-zusammenhängend verwaltet (virtuelle Speicherverwaltung). Die zusammenhängende Speicherverwaltung wird verwendet für

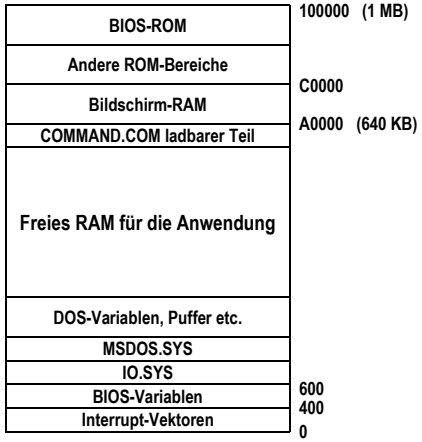
- die Verwaltung der sog. „Pools“ innerhalb des Hauptspeichers (siehe weiter unten)
- die Verwaltung von Plattenplatz,
- die Verwaltung des Platzes in Page- und Swap-Dateien.

Monoprogramming ohne Swapping und Paging

- Aufteilung des Speichers (RAM und ROM) in Bereiche für
  - das Betriebssystem
  - ein Benutzerprogramm
- Beim Beenden eines Programmes: Überlagern des Programmbereichs durch ein neues Programm.

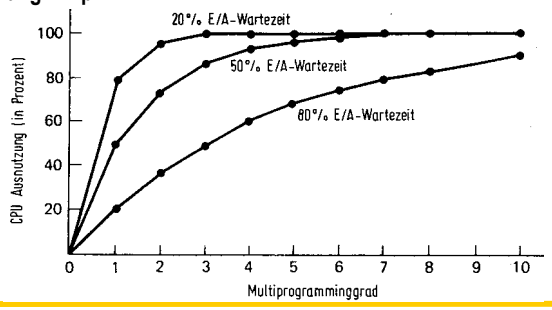


Speicherbelegung unter MS-DOS



Multiprogramming

- Programme verbringen einen großen Teil ihrer Ausführungszeit mit Warten (auf I/O etc.).
- Auslagern (Swapping) auf Platte bei jedem Warten ist ineffizient.
- Lösung: mehrere Programme gleichzeitig im Speicher.
- Voraussetzung: verschiebbarer Code und Speicherschutz.
- Wenn n Programme den Bruchteil p ihrer Zeit mit Warten verbringen ist die CPU-Ausnutzung = $1-p^n$.



Skalierung

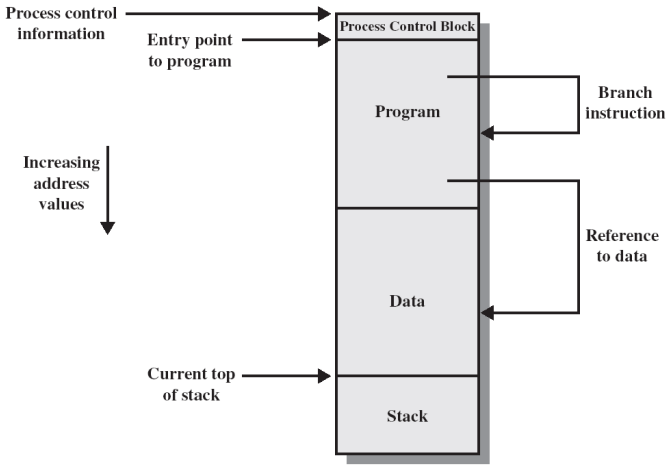
Code-Verschiebung und Speicherschutz

- Ein Programm muß an verschiedenen Stellen im Speicher laufen können.
  Zwei Möglichkeiten, dies zu erreichen:
  - Der Linker vermerkt, welche Code-Stellen absolute Adressen sind. Beim Laden des Programms werden diese Stellen entsprechend abgeändert.
  - Der Rechner hat ein spezielles Hardware-Register, ein sog.
    - Basisregister

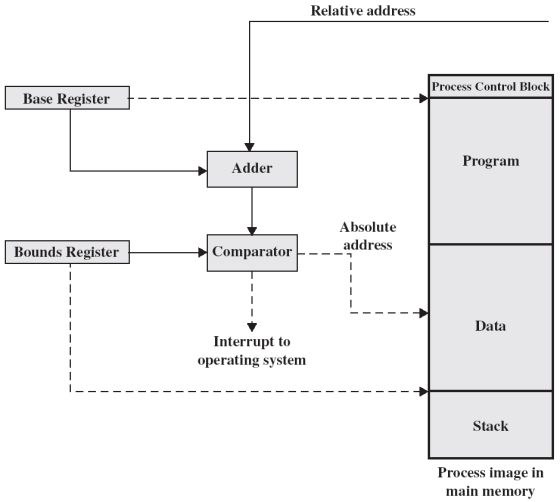
    Bei jeder Adreßberechnung (zur Laufzeit) wird die Adresse im Basisregister zu der Adresse im Programm addiert.
- Ein Programm darf nicht auf den Speicherbereich eines anderen Programms zugreifen.
  Zwei Möglichkeiten dies zu erreichen:
  - Schutzcodes
  - Der Rechner hat ein spezielles Hardware-Register, ein sog.
    - Längenregister

    Durch Überprüfen des Längenregisters wird festgestellt, ob die zugegriffene Adresse in der dem Programm zugewiesenen Partition liegt.

Code-Verschiebung: Prozess - Adressierungsanforderungen

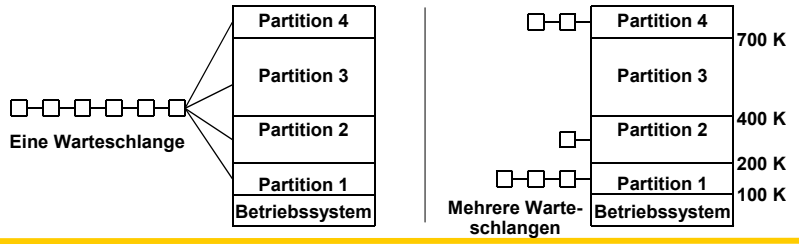


Unterstützung für Code-Verschiebung und Speicherschutz



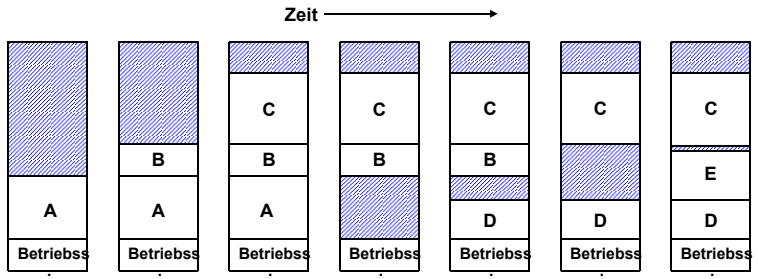
Multiprogramming mit festen Partitionen

- Aufteilung des Speichers in Partitionen fester (ungleicher) Größe.
- Zuweisung eines Programms zu einer freien Partition. Alternativen:
  - Erstes Programm, das in die freie Partition paßt (eine Warteschlange)
  - FIFO für jede einzelne Partition (mehrere Warteschlangen)
  - Größtes Programm, das in die freie Partition paßt
    - Nachteil: kleine Programme werden zurückgestellt
    - Lösung: kleines Programm nur k-mal überspringen
- Evtl. große freie Bereiche in der Partition: interne Fragmentierung.



Multiprogramming mit variablen Partitionen (1)

- Anzahl und Größe der Partitionen werden dynamisch festgelegt.



- Es bleiben u.U. viele kleine freie Bereiche im Hauptspeicher (Löcher). Dies wird als externe Fragmentierung bezeichnet.
- Evtl. müssen diese Löcher durch Verschieben der Partitionen entfernt werden (memory compaction).

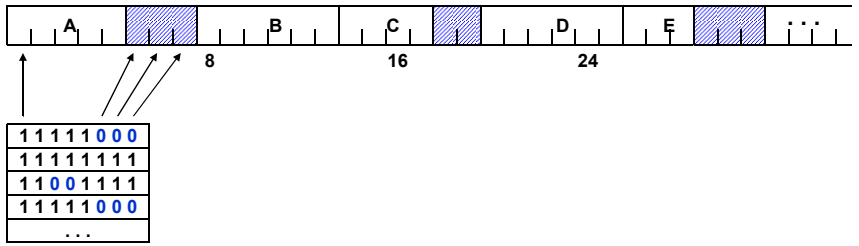
Multiprogramming mit variablen Partitionen (2)

Was ist, wenn der Speicherbedarf eines Prozesses wächst?

- Wenn neben der Partition ein freier Speicherbereich ist, kann die Partition vergrößert werden.
- Es kann eine neue, ausreichend große Partition reserviert und der Inhalt dorthin verschoben werden.
- Wenn keine ausreichend große Partition zur Verfügung steht, müssen ein oder mehrere Prozesse auf Platte ausgelagert werden (Swapping).
- Es kann auch dem Prozeß von vornherein ein größerer Speicherbereich zugewiesen werden, als angefordert.
  - Dies führt zu interner Fragmentierung.
  - Wenn auch dieser größere Bereich nicht ausreicht, muß dennoch eines der vorhergehenden Verfahren angewandt werden.

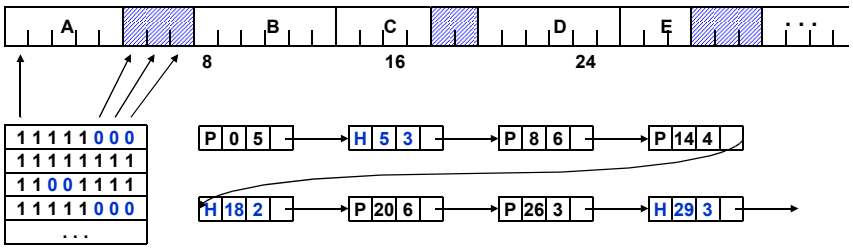
Speicherverwaltung mit Bit Maps

- Aufteilung des Speichers in Einheiten (einige Byte bis einige Kbyte)
- Ein Bit pro Einheit: 0 wenn die Einheit frei ist
  1 wenn die Einheit belegt ist
- Vorteil: feste Größe der Bit Map
- Nachteil: Suche nach freiem Platz bestimmter Größe aufwendig.



Speicherverwaltung mit Linked Lists

- Verkettete Liste von Beschreibungen der Speicherbereiche:
  - Von Prozeß belegt (P) oder frei (H=hole)
  - Anfangsadresse des Bereichs
  - Länge des Bereichs
  - Zeiger auf nächsten Eintrag



Zuteilung freien Speichers

Ein angeforderter Speicherbereich einer bestimmten Größe kann wie folgt zugeteilt werden:

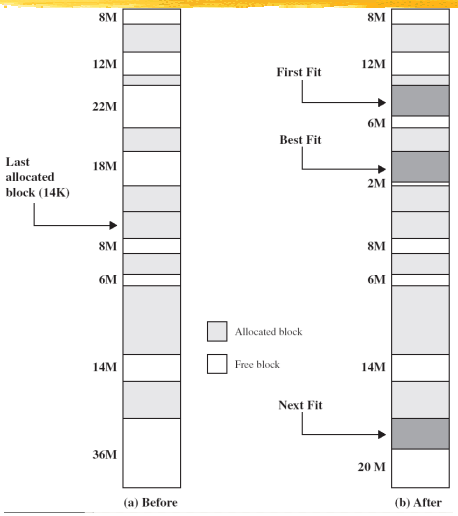
- First-Fit-Methode
  - Der erste genügend große freie Speicherbereich wird zugeordnet.
- Best-Fit-Methode
  - Der kleinste ausreichende freie Speicherbereich wird zugeordnet.
  - Aufwendiger, da die gesamte Liste bzw. Bitmap durchsucht werden muß.
  - Starke Fragmentierung des freien Speichers in viele kleine Bereiche.
- Worst-Fit-Methode
  - Der größte freie Speicherbereich wird zugeteilt.
  - Es bleiben verhältnismäßig große freie Bereiche übrig.
- Quick-Fit-Methode
  - Unterhalten von mehreren Listen freier Speicherbereiche für bestimmte Standardgrößen (z.B. 4 KB, 8 KB, 12 KB etc.) führt zu einer schnellen Zuteilung.
  - Bei Freigabe eines Bereiches muß dieser mit evtl. benachbarten freien Bereichen zusammengelegt werden. Dies bedeutet bei mehreren Listen einen erhöhten Aufwand.
  - Spezielle Variante: Buddy System

? Bewertung ?

Zuteilung freien Speichers: Beispiel

Beispiel für eine Speicherbelegung
- vor und
- nach

der Alloziierung eines 16-MByte Blockes



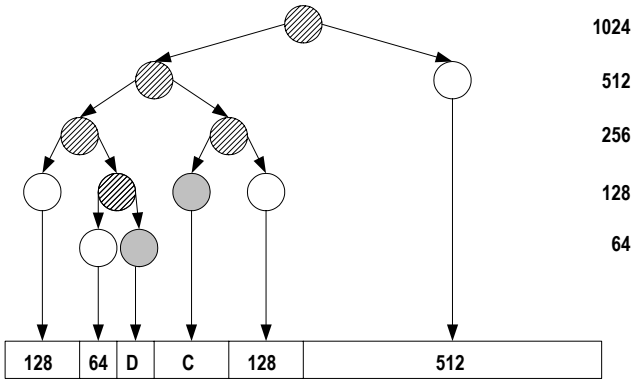
Speicherverwaltung mit dem Buddy System

- Separate Listen freier Bereiche der Größen 1, 2, 4, 8, 16 etc. Bytes bis zur Speichergröße.
- Bei Freigabe eines Speicherbereichs muß nur eine der Listen durchsucht werden, um festzustellen, ob der Bereich mit einem anderen freien Bereich zusammengefaßt werden kann.
- Blockgröße immer Zweierpotenz ist nicht sehr speichereffizient.

	0	128 K	256K	384 K	512K	640K	768K	896K	1M
Anfangs	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								

Buddy-System: Baum-Repräsentierung

Zustand vor der Freigabe von Prozess D



Segmentierung (1)

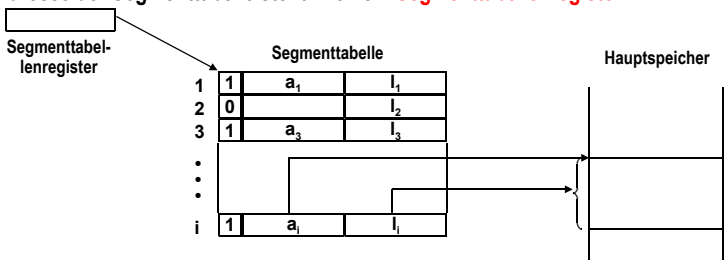
- Aufteilung von Programmen in mehrere Segmente, z.B. Codesegment, Datensegment(e), Stacksegment(e) etc.
- Jedes Segment entspricht einem linearen Adreßraum von 0 bis zu einem Maximalwert.
- Adreßangaben bestehen aus:
  - einer Segmentangabe
  - einer relativen Adresse im Segment

  (Zweidimensionaler Adreßraum)
- Vorteile der Segmentierung:
  - Für die kleineren Segmente ist leichter Hauptspeicher zuzuteilen.
  - Segmente können unabhängig voneinander wachsen.
  - Es müssen nicht alle Segmente gleichzeitig im Hauptspeicher sein (Swapping von Segmenten).
  - Segmente können unterschiedlich geschützt werden.
  - Unterscheidung von prozeß-privaten Segmenten und solchen Segmenten, die von mehreren Prozessen gemeinsam benutzt werden können.

<- Unterschiede Paging
<-> Overlays

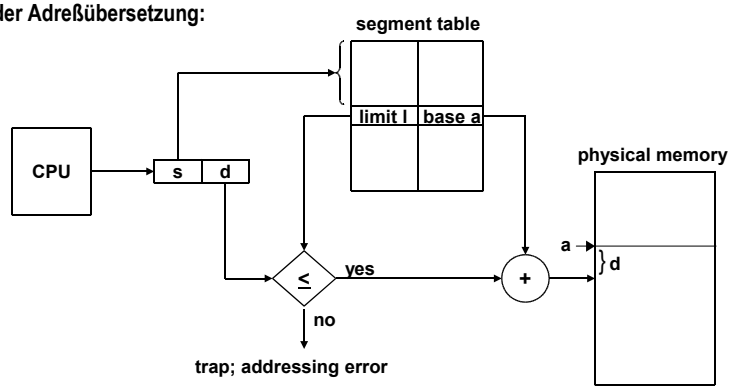
Segmentierung (2)

- Segmentierung muß von der Hardware unterstützt werden.
- Für jeden Prozeß gibt es eine Segmenttabelle im Speicher, die für jedes Segment drei Angaben enthält:
  - Ist das Segment im Hauptspeicher?
  - Anfangsadresse des Segments im Hauptspeicher
  - Länge des Segments
- Die Adresse der Segmenttabelle steht in einem Segmenttabellenregister.



Segmentierung (3)

- Ablauf der Adreßübersetzung:



- Für jeden Hauptspeicherzugriff wird ein zusätzlicher Hauptspeicherzugriff auf die Segmenttabelle benötigt. Dies muß durch Register in der Hardware beschleunigt werden!

Segmentierung (4)

- Bei einem Zugriff auf ein Segment, das nicht im Hauptspeicher ist, erfolgt ein Segment Fault: das Betriebssystem muß das Segment erst in den Hauptspeicher laden.

- Schutz eines Segmentes:
  Die Einträge in der Segmenttabelle enthalten zusätzlich einen Schutzcode (z.B. für die Zugriffe Lesen, Schreiben, Ausführen), der vom Programmierer (oder Compiler/Linker) festgelegt wird.

- Sharing eines Segmentes im Hauptspeicher ist einfach:
  Man läßt die Einträge in den Segmenttabellen mehrerer Prozesse auf die gleiche Hauptspeicheradresse zeigen.

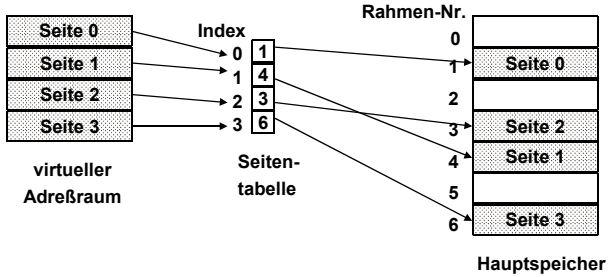
- Fragmentierung:
  Da jedes Segment im Hauptspeicher zusammenhängend abgelegt wird, tritt externe Fragmentierung auf.

Virtuelle Speicherverwaltung (Paging)

- Aufteilung des Adreßraums in Seiten (pages) fester Größe, und des Hauptspeichers in Seitenrahmen (page frames) gleicher Größe.
  - Typische Seitengrößen: 512 byte bis 8192 byte (immer Zweierpotenz).
- Der lineare, zusammenhängende Adreßraum eines Prozesses („virtueller“ Adreßraum) wird auf beliebige, nicht zusammenhängende Seitenrahmen abgebildet.
- Eine einzige Liste freier Seitenrahmen wird vom Betriebssystem verwaltet.
- Die Berechnung der physikalischen Speicheradresse aus der vom Programm angegebenen virtuellen Adresse
  - geschieht zur Laufzeit des Programms,
  - ist transparent für das Programm,
  - muß von der Hardware unterstützt werden.
- Vorteile der virtuellen Speicherverwaltung:
  - Einfache Zuteilung von Hauptspeicher.
  - Keine externe Fragmentierung, geringe interne Fragmentierung.
  - Kein Aufwand für den Programmierer.

Virtueller Adreßraum

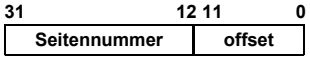
- Beim Paging wird der Zusammenhang zwischen Programmadresse und physikalischer Hauptspeicheradresse erst zur Laufzeit mit Hilfe der Seitentabellen hergestellt.
- Die vom Programm verwendeten Adressen werden deshalb auch virtuelle Adressen genannt.
- Der virtuelle Adreßraum eines Programms ist der lineare, zusammenhängende Adreßraum, der dem Programm zur Verfügung steht.



Adreßübersetzung beim Paging (1)

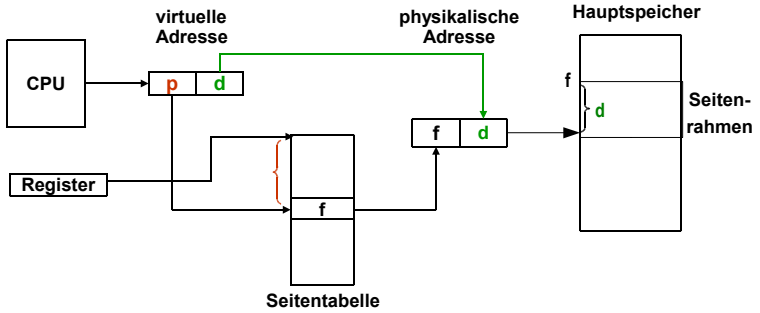
- Die Programmadresse wird in zwei Teile aufgeteilt:
  - eine Seitennummer
  - eine relative Adresse (offset) in der Seite

  Beispiel: 32-bit-Adresse bei einer Seitengröße von 4096 byte:



- Für jeden Prozeß gibt es eine Seitentabelle (page table). Diese enthält für jede Prozeßseite
  - eine Angabe, ob die Seite im Speicher ist,
  - die Nummer des Seitenrahmens im Hauptspeicher, der die Seite enthält.
- Ein spezielles Register enthält die Anfangsadresse der Seitentabelle für den aktuellen Prozeß.
- Die Seitennummer wird als Index in die Seitentabelle verwendet.

Adreßübersetzung beim Paging (2)



- Für jeden Hauptspeicherzugriff wird ein zusätzlicher Hauptspeicherzugriff auf die Seitentabelle benötigt. Dies muß durch Caches in der Hardware beschleunigt werden!
- Ist die Seite nicht im Speicher, wird eine spezielle Exception, ein sog. page fault ausgelöst.

Virtueller Speicher allgemein (1)

- Mehr Prozesse können effektiv im Speicher gehalten werden --> bessere Systemauslastung
- Ein Prozeß kann viel mehr Speicher anfordern als physikalisch verfügbar
- allgemeiner Vorgang:
  - Nur Teile des Prozesses befinden sich im physikalischen Speicher
  - falls Zugriff auf eine Adresse, die ausgelagert ist:
    - das BS setzt den Prozeß auf blockiert
    - das BS setzt eine Disk-IO-Leseanfrage ab
    - Nach laden des fehlenden Stückes (Seite oder Segment) wird ein I/O-Interrupt abgesetzt
    - das BS setzt Prozeß zuletzt wieder in den Bereit- (Ready-) Zustand
- Frage:
  - ist es effizient, daß ein Prozeß nur deswegen unterbrochen werden darf (Interrupt), weil nicht alle seine Teilstücke (Seiten oder Segmente) im physikalischen Speicher geladen sind ?

Virtueller Speicher allgemein (2)

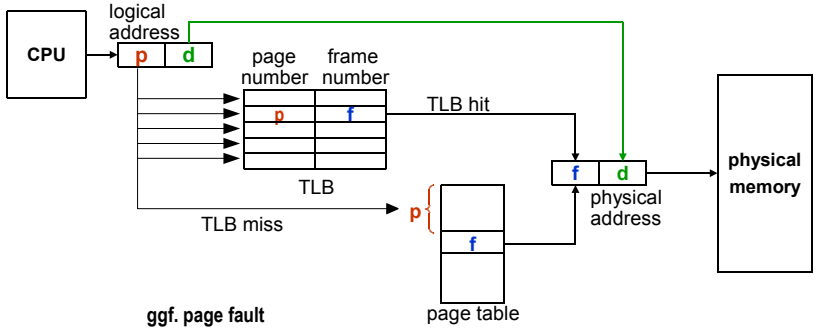
- „*thrashing*“ (siehe später): Der Prozessor verbringt die meiste Zeit mit dem Ein- und Auslagern von Prozeßteilen anstatt der Ausführung von Prozeßanweisungen

- Lokalitätsprinzip:
  - Zugriffe auf Daten und Programmcode sind häufig lokal gruppiert;
  - --> Annahme gerechtfertigt, daß nur wenige Prozeßstücke während einer kurzen zeitlichen Periode gleichzeitig vorgehalten werden müssen

Iterieren über Bild

Translation Look-Aside Buffer (1)

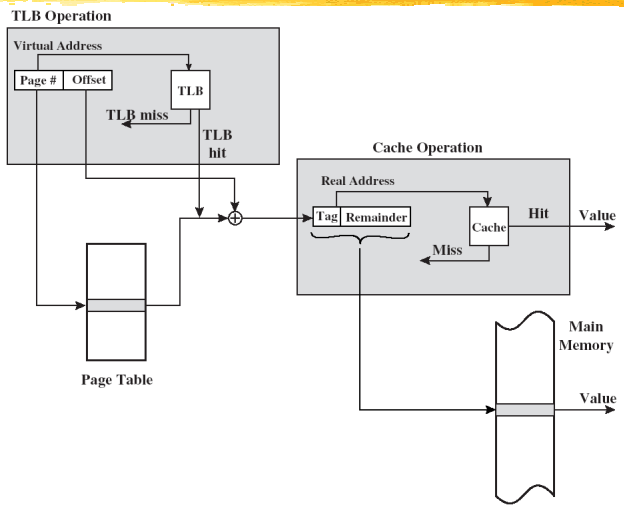
- Ein Translation Lookaside Buffer (TLB) ist ein schneller Hardware-Cache, der die zuletzt benutzten Seitentabelleneinträge enthält.
- Aufgebaut als Assoziativ-Speicher: bei der Übersetzung einer Adresse wird deren Seitennummer gleichzeitig mit allen Einträgen des TLB verglichen.



Translation Look-Aside Buffer (2)

- Bei einem Treffer im TLB erübrigt sich der Speicherzugriff auf die Seitentabelle.
- Bei einem Fehltreffer wird auf die Seitentabelle zugegriffen. Ein alter Eintrag im TLB wird durch den neuen ersetzt.
- Die Trefferquote (hit ratio) beeinflußt die durchschnittliche Zeit einer Adreßübersetzung.
- Das Lokalitätsprinzip (Programme greifen meist auf benachbarte Adressen zu) sorgt auch bei kleinen TLBs für hohe Trefferquoten (typisch: 80-98%).
- Der Inhalt des TLB ist prozeßspezifisch! Zwei Möglichkeiten:
  - Jeder Eintrag im TLB enthält ein "valid bit". Bei einem Prozeßwechsel (Context Switch) wird der gesamte Inhalt des TLB invalidiert.
  - Jeder Eintrag im TLB enthält eine Prozeßidentifikation (PID), die mit der PID des zugreifenden Prozesses verglichen wird.
- Beispiele für TLB-Größen:
  - Motorola 68030: 22 Einträge
  - Intel 80486: 32 Einträge. Beim Pentium: getrennt für Code und Daten.

Translation Look-Aside Buffer und Speicher-Cache

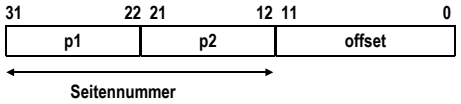


Mehrstufiges Paging (1)

- Die Seitentabelle kann sehr groß werden.

  Beispiel: 32-bit-Adressen, 4KB Seitengröße, 4 Byte pro Eintrag:
  Seitentabelle: 1M Einträge, 4MB Größe (pro Prozeß!)

- **Zweistufiges Paging:**
  - Die Seitennummer wird noch einmal unterteilt, z.B.:

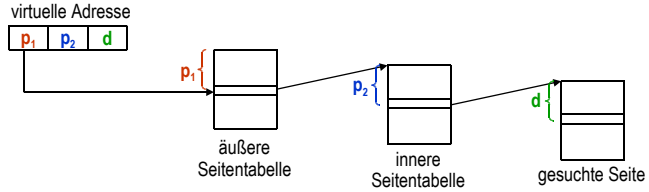


  - p1 ist Index in eine äußere Seitentabelle, deren Einträge jeweils auf eine innere Seitentabelle zeigen.
  - p2 ist Index in eine der inneren Seitentabellen, deren Einträge auf Seitenrahmen im Speicher zeigen.
  - Die inneren Seitentabellen müssen nicht alle speicherresident sein!

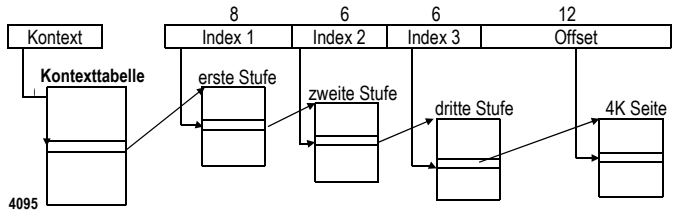
- Analog kann dreistufiges Paging etc. implementiert werden.

Adreßübersetzung bei mehrstufigem Paging

- **Zweistufiges Paging:**



- **Dreistufiges Paging bei SPARC-Prozessoren:**



Mehrstufiges Paging (2)

- Größe der Seitentabellen:

  Beispiel:

p1	p2	offset
10	10	12

  - Die äußere Seitentabelle hat 1024 Einträge, die auf (potentiell) 1024 innere Seitentabellen zeigen, die wiederum je 1024 Einträge enthalten.
  - Bei einer Länge von 4 Byte pro Seitentabelleneintrag ist also jede Seitentabelle genau eine 4KB-Seite groß.
  - Es werden nur soviele innere Seitentabellen verwendet, wie nötig.

- Jede Adreßübersetzung benötigt noch mehr Speicherzugriffe, deshalb ist der Einsatz von TLBs noch wichtiger.

- Als Schlüssel für den TLB werden alle Teile der Seitennummer zusammen verwendet.

Speicherschutz beim Paging

- **Schutz vor dem Zugriff durch andere Prozesse:**
  - Da jeder Prozeß eine eigene Seitentabelle hat, ist Zugriff auf Speicherbereiche anderer Prozesse nicht möglich.
    (Dies macht andererseits die Implementierung von gemeinsam benutzten Speicherbereichen aufwendiger.)

- **Schutz vor (z.B.) unberechtigtem Schreiben:**
  - Die Einträge der Seitentabellen enthalten zusätzlich einen Schutzcode, der z.B. angibt, ob die Seite gelesen und/oder geschrieben werden darf
    (evtl. auch noch abhängig davon, ob der Zugriff im User- oder im Kernel-Mode erfolgt).

- Die Seiteneinteilung ist transparent für Programmierer !

- Festlegen des Schutzcodes durch Compiler und/oder Linker:
  - Das Programm wird in Abschnitte eingeteilt, deren Größe ein Vielfaches der Seitengröße ist.
  - Pro Abschnitt wird ein Schutzcode für alle Seiten dieses Abschnitts festgelegt und im Kopf der Programmdatei vermerkt.
  - Der Lader setzt die Schutzcodes in den Seitentabelleneinträgen.

Seiten-Sharing beim Paging

- *Theoretisch* könnten Einträge verschiedener Seitentabellen auf den gleichen Seitenrahmen zeigen.

  Probleme:
  - Wie stellt man fest, ob eine Seite bereits von einem anderen Prozeß benutzt wird, und in welchem Seitenrahmen sich diese befindet?
  - Bei Änderungen (z.B. des verwendeten Seitenrahmens) müßten viele Seitentabellen angepaßt werden.

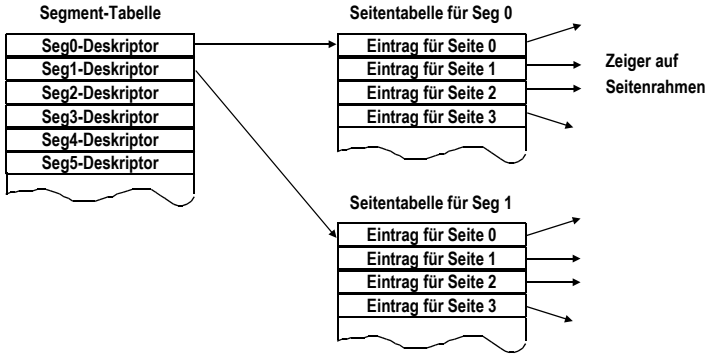
- *Praktisch* wird der gemeinsam zu benutzende Teil des Adressraums
  - entweder als gemeinsam benutzbares Segment mit eigener Seitentabelle implementiert (Kombination von Segmentierung und Paging, z.B. bei Unix) oder
  - es werden die gemeinsam zu nutzenden Teile als eine Art Pseudo-Prozess-Adressbereich implementiert, für den es eine eigene (globale) Seitentabelle gibt (z.B. bei Windows).

Segmentierung mit Paging (1)

- Programme werden in Segmente unterteilt, die aber nicht zusammenhängend, sondern mit Hilfe von Paging im Speicher abgelegt werden.
- Der Programmierer gibt die Segment-Nummer und eine relative Adresse in diesem Segment an. Das Paging wird - transparent für das Programm - von der Hardware durchgeführt.
- Zwei Realisierungsmöglichkeiten:
  - eine Segmenttabelle und pro Segment eine eigene Seitentabelle.
    Die Einträge in der Segmenttabelle zeigen auf die Seitentabelle des jeweiligen Segments (z.B. Unix).
  - Es gibt eine Segmenttabelle und eine einzige Seitentabelle (z.B. Intel-CPUs).
    - Die Einträge in der Segmenttabelle enthalten die Basisadresse des Segments in einem linearen (virtuellen) Adreßraum.
    - Die relative Adresse im Segment wird zu dieser Basisadresse addiert. Die resultierende lineare (virtuelle) Adresse wird mittels der Seitentabelle in eine physikalische Adresse übersetzt.

Segmentierung mit Paging (2)

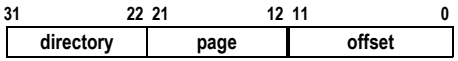
- Eine eigene Seitentabelle pro Segment:



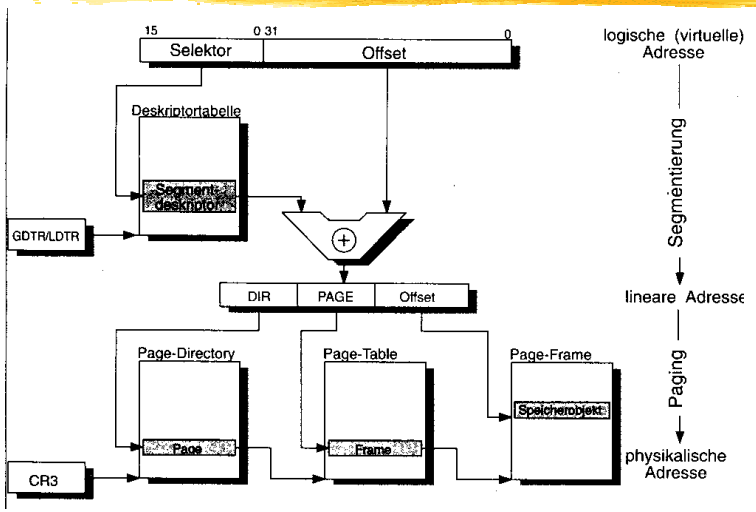
- Eine Adreßangabe besteht aus Segmentnummer und linearer Adresse im Segment. Die Aufteilung der linearen Adresse in Seitennummer und Offset (für das Paging) ist transparent für das Programm.

Segmentierung mit Paging beim INTEL 80386

- Segmente:
  - Bis zu 8K prozeßprivate Segmente, beschrieben durch Einträge in der local descriptor table (LDT).
  - Bis zu 8K von allen Prozessen gemeinsam benutzte (shared) Segmente, beschrieben durch Einträge in der global descriptor table (GDT).
  - Jedes Segment kann bis zu 4 GB groß sein.
- Sechs Segmentregister, so daß zu jeder Zeit sechs Segmente gleichzeitig von einem Prozeß benutzt werden können.
- Sechs interne Register, die die entsprechenden Segment-Deskriptoren enthalten.
- Eine Adressangabe ist ein Paar (Segment-Selektor, Offset).
- Der Segment-Deskriptor ist die Basisadresse des Segments in einem 32-bit-Adreßraum, zu der der Offset addiert wird, um die sogenannte lineare Adresse zu erhalten.
- Die lineare Adresse hat das Format (two-level paging):

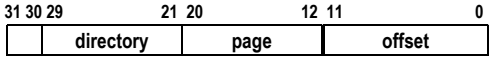


Adreßübersetzung beim INTEL 80386



Physical Address Extension (PAE) beim Pentium Pro

- Im PAE-Modus arbeitet der Prozessor mit dreistufigem Paging.



- Die höchsten beiden Bits enthalten den „Page Directory Pointer Index“, der eines von 4 Page Directories auswählt.
- Die Einträge in den Page Directories und in den Page Tables sind 8 Byte lang. (Da es nur noch je 512 Einträge gibt, sind diese Tabellen jeweils wieder genau eine Seite groß.)
- Seitenrahmennummern werden mit 24 Bits dargestellt. Physikalische Adressen haben somit 36 bit und der adressierbare physikalische Speicher ist 64 Gigabyte.
- Der Intel 450NX-Chipsatz erlaubt x86-Prozessoren, PAE zu verwenden.

PAE und Windows 2000

- Um PAE zu verwenden, muß das Betriebssystem modifiziert werden.
- Für Windows 2000 hat Microsoft einen eigenen PAE-Kernel geschrieben (ntkrnlpa.exe). Wenn der Prozessor PAE-fähig ist und das System mehr als 4 GB Hauptspeicher hat, wird dieser Kernel geladen.
- Mit dem PAE-Kernel können folgende Hauptspeichergrößen verwendet werden:
  - W2K Professional: 4 Gigabyte
  - W2K Server: 4 Gigabyte
  - W2K Advanced Server: 8 Gigabyte
  - W2K Datacenter 64 Gigabyte
- Microsoft bietet die „Address Windowing Extensions (AWE) API“ an, mit der Applikationen physikalischen Speicher für ihren exklusiven Gebrauch allokieren und Teile ihres Adreßraums in diesen Speicher abbilden können.

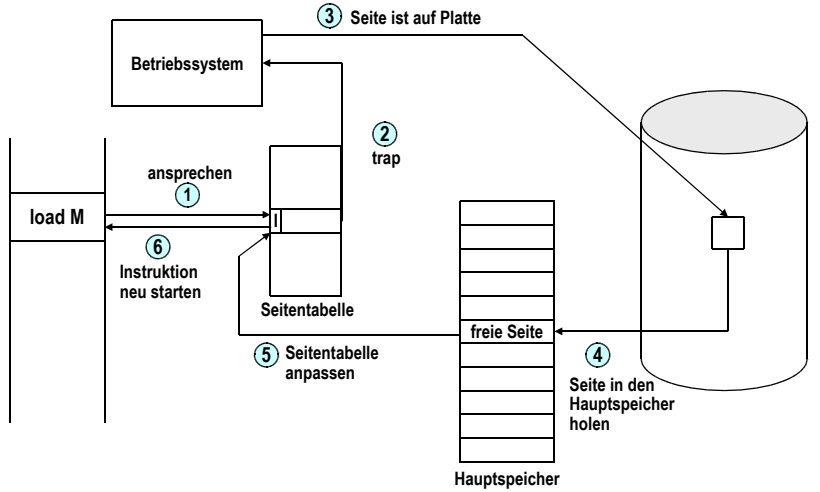
Demand Paging

- 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 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

- 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.
- 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

- Wenn bei einem Page Fault kein freier Seitenrahmen zur Verfügung steht, muß einer frei gemacht werden.
- Ein Algorithmus legt nach einer bestimmten Strategie fest, welcher Seitenrahmen frei gemacht wird.
- 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 Swapbereich).
- 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.

Strategien für die Seitenersetzung

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

  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 System Manager), 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.

  Globale Ersetzung: Es wird eine beliebige Seite im Speicher ersetzt.
  - Prozesse nehmen sich gegenseitig Seiten weg.
  - Ein Prozeß, der viele Page Faults macht, erhält automatisch mehr Speicher. (Dies kann sowohl ein Vorteil als auch ein Nachteil sein.)

Optimale Strategie

- Es wird diejenige Seite ersetzt, 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.

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

First In First Out (FIFO)

- Es wird diejenige Seite ersetzt, 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)

Es wird diejenige Seite ersetzt, die am längsten nicht benutzt worden ist.

Vorteil: In der Regel weniger Page Faults als FIFO.

Nachteil: Aufwändige Implementierung.

- Implementierung mit einem Zähler:
  - Ein systemweiter Zähler wird bei jedem Speicherzugriff inkrementiert.
  - Der aktuelle Wert des Zählers wird in einem Feld in der Datenstruktur vermerkt, die die angesprochene Seite beschreibt.
  - Es wird die Seite mit dem kleinsten Zählerwert ersetzt.
- 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!).
  - Die Seite am Ende der Liste wird ersetzt.

Benutzen eines Referenz-Bits

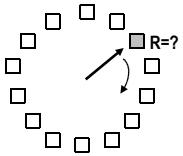
- Jeder Seitentabelleneintrag kann ein Referenz-Bit enthalten
  - das bei einem Zugriff auf die Seite gesetzt wird (Hardware!),
  - das nach bestimmten Kriterien wieder gelöscht wird (Software).
- Ein Referenz-Bit
  - liefert die Information, ob eine Seite seit dem letzten Löschen des Bits zugegriffen wurde,
  - sagt nichts über den Zeitpunkt des Zugriffs auf eine Seite aus,
  - sagt nichts über die Reihenfolge der Zugriffe auf mehrere Seiten aus.
- Mit Hilfe eines Referenz-Bits können weitere Seitenersetzungsstrategien implementiert werden, z.B.
  - Modifikationen von LRU, die weniger aufwendig zu implementieren sind.
  - Second-Chance-Algorithmus, eine Verbesserung der FIFO-Strategie.

Modifikation von LRU

- 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.
- 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

- Ist eine 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.
- 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:
- Ist R=0 wird die Seite ersetzt, und der Zeiger weiterbewegt.
- Ist R=1 wird R gelöscht, der Zeiger weiterbewegt, und die nächste Seite überprüft.

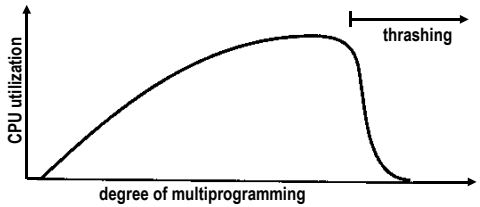
Seitenersetzungsalgorithmen: Beispiele

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2	2 3	2 3	2 3 1	2 3 5 F	2 3 5	4 3 5 F	4 3 5	4 3 5	2 3 5 F	2 3 5	2 3 5
LRU	2	2 3	2 3	2 3 1	2 5 1 F	2 5 1	2 5 4 F	2 5 4	3 5 4 F	3 5 2 F	3 5 2	3 5 2
FIFO	2	2 3	2 3	2 3 1	5 3 1 F	5 2 1 F	5 2 4 F	5 2 4	3 2 4 F	3 2 4	3 5 4 F	3 5 2 F
CLOCK	2*	2* 3*	2* 3*	2* 3* 1*	5* 3 1 F	5* 2* 1 F	5* 2* 4* F	5* 2* 4*	3* 2 4 F	3* 2* 4	3* 2 5* F	3* 2* 5*

Thrashing

- 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:



- Lösungen:
  - Falls noch freier Speicher vorhanden: Zuteilung weiterer Seitenrahmen an den betreffenden Prozeß (z.B. durch dynamische Working-Set-Anpassung oder globale Ersetzungsstrategie).
  - Falls kein freier Speicher mehr: Auslagern (Swapping) von Prozessen.

Weitere Design-Möglichkeiten (1)

- Asynchrones Auslagern modifizierter Seiten:
  - Modifizierte, ersetzte Seiten werden z.B. erst dann ausgelagert, wenn die Platte mit dem Swapbereich wenig benutzt ist, oder wenn der Vorrat freier Seiten zu klein wird.
  - Modifizierte, nicht ersetzte Seiten werden vorab ausgelagert und ihr Modify-Bit gelöscht.

- Vorrat an freien Seitenrahmen:
  - Durch frühzeitige (proaktive) Seitenersetzung wird dafür gesorgt, daß bei der Anforderung einer Seite ein freier Seitenrahmen verfügbar ist.

- Page Buffering (Aufbewahren des Inhalts einer ersetzten Seite):
  - Eine ersetzte Seite behält zunächst ihren Inhalt, wird jedoch dem Vorrat an freien Seitenrahmen zugeordnet bzw. zum Auslagern vorgemerkt.
  - Wenn der Prozeß diese Seite kurz danach wieder anspricht, ist sie noch im Speicher und kann ohne Plattenzugriff wieder dem Prozeß zugeordnet werden (soft page fault).
  - Es muß bekannt sein, ob die Seite zwischenzeitlich "wiederverwendet", d.h. einem anderen Prozeß zugeteilt wurde.

Weitere Design-Möglichkeiten (2)

- Prepaging:
  - Eine gewisse Anzahl von Seiten wird vorab (z.B. beim Aufruf eines Programms) in den Speicher gebracht, noch bevor sie von dem Prozeß angesprochen werden.

- Clustering:
  - Bei einem Page Fault wird nicht eine einzelne Seite, sondern ein Cluster aus mehreren Seiten in den Speicher gebracht (spezielles Prepaging).
  - Modifizierte Seiten werden nicht einzeln, sondern in Clustern von mehreren Seiten auf Platte geschrieben.

- Locking von Seiten im Speicher:
  - Einzelne Seiten eines Prozesses werden vom Paging ausgenommen.
  - Implementierung über einen (privilegierten) System Call.
  - Beim Swapping des ganzen Prozesses werden auch diese Seiten mit ausgelagert.

- Auch das Design der Programme hat Auswirkungen auf die Zahl der Page Faults und somit auf die Laufzeit des Programms selbst wie auch auf die Leistung des gesamten Systems!

Swapping

- Swapping ist die zeitweise Auslagerung aller von einem Prozeß benutzten Speicherseiten (oder zumindest kompletter Segmente) auf einen Hintergrundspeicher (Platte), um z.B. bei zuwenig freiem Hauptspeicher Platz zu schaffen.

- Bei zusammenhängender Speicherzuteilung
  - ist Swapping die einzige Möglichkeit, mehr Programme gleichzeitig auszuführen, als Platz im Hauptspeicher haben,
  - müssen Speicherbereiche eines Prozesses, die dynamisch wachsen, u.U. ausgelagert werden.

- 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.