Speicherverwaltung (4)

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.

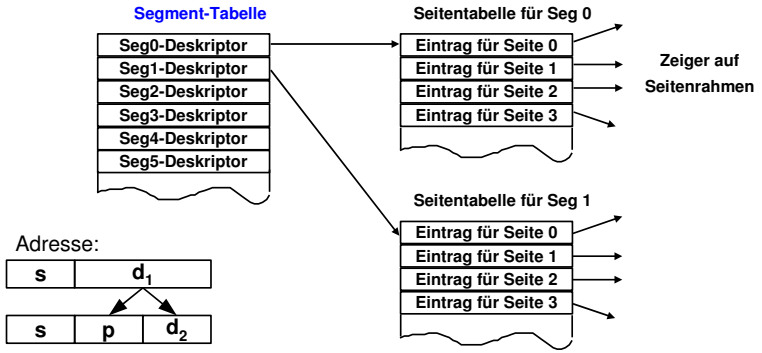
Segmentierung mit Paging (2)

Zwei Realisierungsmöglichkeiten:

- Eine Segmenttabelle und pro Segment (pro Prozess) 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 (pro Prozess) (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 (3)

- Eine eigene Seitentabelle pro Segment:



Segmentierung mit Paging (4)

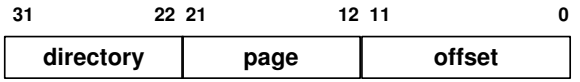
- Eine Adressangabe besteht aus Segmentnummer und linearer Adresse im Segment.
  Die Aufteilung der linearen Adresse in Seitennummer und Offset (für das Paging) ist transparent für das Programm.

Segmentierung mit Paging: Intel 80386

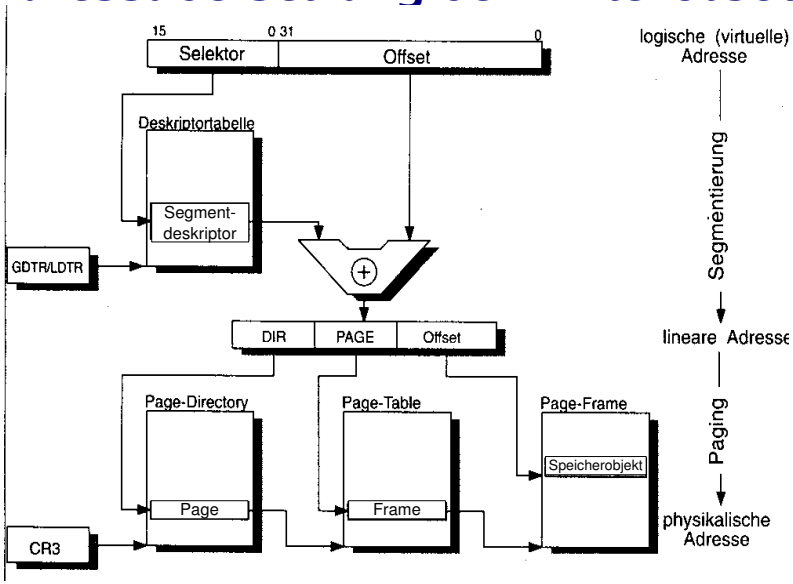
- Segmente:
  - Bis zu 8 K prozessprivate Segmente, beschrieben durch Einträge in der local descriptor table (LDT).
  - Bis zu 8 K von allen Prozessen gemeinsam benutzte (shared) Segmente, beschrieben durch Einträge in der global descriptor table (GDT).
  - Jedes Segment kann bis zu 4 GByte groß sein.
- Sechs Segmentregister, so dass zu jeder Zeit sechs Segmente gleichzeitig von einem Prozess benutzt werden können.

Segmentierung mit Paging: Intel 80386

- 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 so genannte lineare Adresse zu erhalten.
- Die lineare Adresse hat das Format (two-level paging):

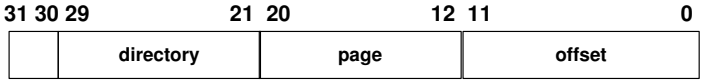


Adressübersetzung beim Intel 80386



Physical Address Extension (PAE) beim Pentium Pro (1)

- PAE-Modus: Prozessor nutzt dreistufiges 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 4 KByte, also eine Seite groß.)

Physical Address Extension (PAE) beim Pentium Pro (2)

- Seitenrahmennummern werden mit 24 Bits dargestellt. Physikalische Adressen haben somit 36 bit und der adressierbare physikalische Speicher ist 64 GByte.
- Der Intel 450NX-Chipsatz erlaubt x86-Prozessoren, PAE zu verwenden.

PAE und Linux 2.6 (1)

- Linux unterstützt (auf Intel-32-Bit-Plattformen) bis zu 64 GByte Hauptspeicher
- Auswahl bei der Kernel-Konfiguration:
  - bis 1 GByte,
  - >1 bis 4 GByte,
  - >4 bis 64 GByte

PAE und Linux 2.6 (2)

arch/i386/Kconfig:

```
choice
        prompt "High Memory Support"
        default NOHIGHMEM

config NOHIGHMEM
        bool "off"

config HIGHMEM4G
        bool "4GB"
        help
          Select this if you have a 32-bit processor and between 1
          and 4 gigabytes of physical RAM.

config HIGHMEM64G
        bool "64GB"
        depends on X86_CMPXCHG64
        help
          Select this if you have a 32-bit processor and more than 4
          gigabytes of physical RAM.
```

PAE und Linux 2.6 (3)

- Speicheraufteilung:
  ZONE_DMA: Erste 16 MByte
  für alte ISA-Geräte
  ZONE_NORMAL: 16 MByte – 896 MByte
  Standard (Kernel- und User-Mode)
  ZONE_HIGHMEM: 896 MByte – Rest
- Kernel hat 1 GByte virtuellen Speicher für sich, davon die ersten 896 MByte (ZONE_DMA und ZONE_NORMAL) fest abgebildet.
- Es bleiben 128 MByte, die dynamisch genutzt werden, um auf Speicher > 1 GByte zuzugreifen

PAE und Windows 2000 (1)

- 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 GByte Hauptspeicher hat, wird dieser Kernel geladen.

PAE und Windows 2000 (2)

- Mit dem PAE-Kernel können folgende Hauptspeichergrößen verwendet werden:
  - W2K Professional: 4 GByte
  - W2K Server: 4 GByte
  - W2K Advanced Server: 8 GByte
  - W2K Datacenter 64 GByte
- 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 (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.

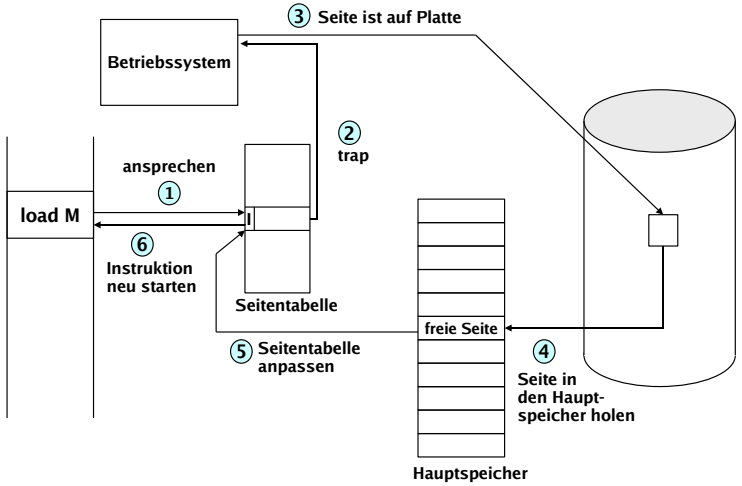
Voraussetzungen für Demand Paging (1)

- 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.

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)

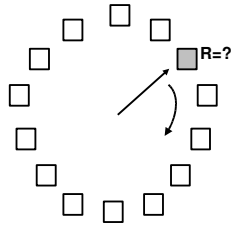
- Es wird eine der Seiten ersetzt, die den kleinsten Zählerwert enthalten.
  - Bei gleichem Zählerwert ist nicht bekannt, auf welche Seite zuletzt zugegriffen wurde.
  - Länger zurückliegende Zugriffe werden zunächst weniger stark gewichtet und schließlich „vergessen“ (aus dem Zähler hinausgeschoben).

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:

- 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.

Seitenersetzung, Beispiele