Speicherverwaltung

Die Speicherverwaltung oder Hauptspeicherverwaltung ist ein Teil des Betriebssystems und erledigt alle erforderlichen Arbeiten zur Verwaltung des physikalischen und des virtuellen Speichers eines Computers.

- Virtuelle Speicherverwaltung
- Swapping und Paging
- Shared Memory

Virtuelle Speicherverwaltung

- Ein **Prozess** sollte auch dann noch *ablaufen* können, wenn er nur *teilweise im Hauptspeicher* ist.
- Wichtig ist, dass die Teile des Prozesses (Daten und Code) im **physikalischen Speicher** sind, die gerade *benötigt* werden.
- Der **Speicherbedarf** eines Prozesses sollte *größer* als der *physikalisch vorhandene Hauptspeicher* sein können.

> Ein **Seitenrahmen** versteht man einen zusammenhängenden Block von Speicherzellen des *physikalischen* Speichers.

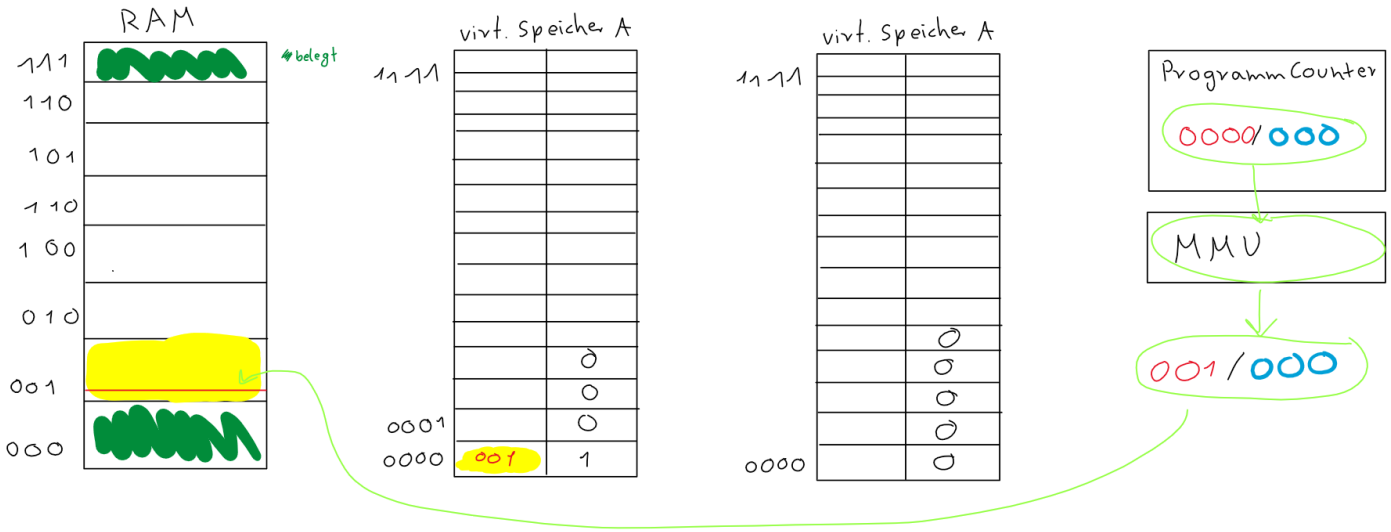
> Eine **Seite** ist ein Block von Speicherzellen des *virtuellen* Speichers. Blockgröße einer Seite ist Größe eines Seitenrahmens.

- **alle Seiten eines Systems haben stets die gleiche Größe**

> die Aufgabe der MMU besteht in der **Umrechnung von virtuellen in physikalische Adressen**

Seitentabellen

- werden benötigt, damit **MMU die Umrechnung einer virtuellen Adresse** in eine physikalische Adresse vornehmen kann
- Für jeden vom Betriebssystem zu verwaltenden **virtuellen Adressraum** gibt es jeweils **eine zugehörige Seitentabelle**
- **Jeder Prozess besitzt seine eigene Seitentabelle!**



1. **MMU** prüft in der *Seitentabelle des aktuellen Prozesses* den Eintrag an entsprechenden Stelle aus der virtuellen Adresse.
2. Ist `P/A-Bit = false (0)`, kommt es zu **Page-Fault (Seitenfehler)**
   - Liegt daran, dass noch keine physische Seite eingelagert wurde
   - Die Seite muss vom **DMA-Controller** in den RAM-Speicher geladen werden
   - Ist der RAM bereits voll, kommen **Seitenersetzungsverfahren** ins Spiel.
3. Ist das dann `P/A-Bit = true (1)`, kann die Rahmennummer aus der Tabellenzeile entnommen werden.
4. **MMU** guckt sich sich bei 0000 das *present absent bit* an, weil es eine 1 ist, kann es umgerechnet werden
5. wird im **RAM** bei 001 abgespeichert
6. wenn die fetch Phase beendet ist, wird der **PC** um 1 hochgesetzt und auf 0000001 gesetzt

- so viel wie in Seitenrahmen reinpasst muss der DMA-Controlller kopieren
- bei Pagefault wird der Prozess auf ***blocket*** gesetzt, da er so nicht weiterarbeiten kann
- wenn alle Daten im Ram sind wird ein ***Interrrupt*** eingestellt und es geht auf ***Ready***
- Prozess kann mit den Daten im Ram arbeiten und wenn Sceduler den Prozess wieder nimmt geht es wieder in ***Running***

KONTEXTWECHSEL

1. Wert des PC wird überschrieben mit *Wert vom virtuellen Speicher B*
2. In Seitentabelle wird gesehen das absent 0 ist ->**Seitenfehler** -> muss in *Ram* eingelagert werden (egal welche Stelle)
3. Present wird zu Absend

- in 111 wird der Wert des PC gespeichert

DLL Datein sorgen dafür, dass weniger Speicher gebraucht wird

Swapping und Paging

Unter Swapping versteht man das Aus- bzw. Einlagern eines kompletten Prozesses.

> **Swapping** kommt nur bei Betriebssystemen zum Einsatz, die **keine virtuelle Speicherverwaltung** unterstützen!

eine Alternative zur virtuellen Speicherverwaltung mit Hilfe der MMU, ist das **Paging**, welches flexibler als ***Swapping*** agiert.

> Unter **Paging** versteht man das Ein- bzw. Auslagern von **Teilen eines Prozesses**.

- Eine *Seite* enthält einen **bestimmten Teil eines Prozesses**.
- Die *Gesamtheit aller Seiten* eines Prozesses repräsentiert somit den *kompletten Prozess*.
- Eine (virtuelle) Seite kann einerseits in einem (physikalischen) Seitenrahmen eingelagert sein.
  - befindet sich die Seite im RAM und der zugehörige Prozess kann auf die Befehle Daten innerhalb der Seite zugreifen.
  - Zum Einsatz kommt hier die *Adressumrechnung mit Hilfe von MMU und Seitentabelle*.
- Andererseits kann eine (virtuelle) Seite auf einen Hintergrundspeicher (wie beispielsweise die *Festplatte)* ausgelagert sein.
  - so steht die Seite nicht im RAM zur Verfügung, Zugriff des Prozesses auf Befehle schlägt fehl ->**Page fault**
  - ausgelagerte Seite muss vom *Hintergrundspeicher in Seitenrahmen* eingelagert werden.
  - Anschließend kann die *Adressumrechnung mit Hilfe von MMU und Seitentabelle*.

Beim Paging werden virtuelle Seiten in (physikalische) Seitenrahmen eingelagert, oder andersrum

---

Page Fault

> **Page fault** tritt auf, falls die MMU bei der Umrechnung einer virtuellen in eine physische Adresse feststellt, dass die benötigte (virtuelle) Seite nicht in einem (physischen) Seitenrahmen eingelagert ist.

Seitenersetzung

Tritt **Seitenfehler** auf, lagert *BS* ***virtuelle Seite*** aus **Hintergrundspeicher** in *freien Seitenrahmen* des ***physikalischen Speichers***

- Entscheidend ist bei der Einlagerung, ob es einen *freien Seitenrahmen* gibt
- Falls ja: dieser für die *Einlagerung der benötigten Seite* genutzt werden
- Falls nein: **Seitenersetzung** entscheidet, welche *momentan eingelagerte virtuelle Seite* in *HS* verschoben wird

---

Was bei der Seitenersetzung passiert

1. Die MMU stellt fest, dass ***virtuelle Seite B*** nicht in ***Seitenrahmen*** eingelagert ist und löst einen Seitenfehler aus.
2. Es wird festgestellt, dass *kein freier* ***Seitenrahmen*** verfügbar ist, das **Seitenersetzungsverfahren** wird deshalb *gestartet*.
3. Die zu ***ersetzende Seite E*** wird bestimmt, sie ist derzeit in ***Seitenrahmen X*** eingelagert.
4. Die zu ***ersetzende Seite E*** wird in den Hintergrundspeicher geschrieben,. damit ist ihr Inhalt gesichert
5. Die **benötigte Seite B** wird eingelagert, ihr Inhalt wird in den ***Seitenrahmen X*** geschrieben

> Schreiben der zu ersetzenden Seite E in den Hintergrundspeicher kostet viel Zeit! Das ist schlecht für das Gesamtsystem.

→ sinnvoll, eine zu *ersetzende Seite* nur **dann** in den Hintergrundspeicher zu **kopieren**, wenn dies auch **tatsächlich notwendig** ist.

Das Modifiziert-Bit

→ BS kann mit dem **M-Bit** feststellen, ob die Inhalte einer eingelagerten Seite modifiziert und zum auslagern bereit sind

Seitenrahmen-Nr.	Present-/ Absent-Bit	M-Bit
0 1111 0000 0011	1	0
0 1100 0111 0101	0	0
1 0101 0011 1110	0	0
0 1001 1111 1001	1	1
...	...	...

- Das M-Bit ist *gesetzt*, also **1**:

Der Inhalt der zugehörigen Seite wurde **modifiziert**.

- Das M-Bit ist *nicht gesetzt*, also **0**:
  Der Inhalt der zugehörigen Seite wurde **nicht modifiziert**.

Seitenersetzungsverfahren → Paging

> Die optimale Seitenersetzung kann nicht programmiert werden.

No Recently Used→ Welche Seite wurde in der Vergangenheit am *längsten nicht genutzt*?

Für jede einzelne Seite wird mit Hilfe eines einzelnen Bits festgehalten, ob die *betreffende Seite referenziert* wurde.

Seitenrahmen-Nr.	Present-/ Absent-Bit	R-Bit	M-Bit
0 1111 0000 0011	1	1	0
0 1100 0111 0101	0	0	0
1 0101 0011 1110	0	0	0
0 1001 1111 1001	1	1	1
...	...	...	...

- Das R-Bit ist gesetzt, also **1**:
  Inhalt der Seite wurde *referenziert*, es wird **verwendet**.
- Das R-Bit ist nicht gesetzt, also **0**:
  Inhalt der Seite wurde *nicht referenziert*, es wird **nicht verwendet**.

First-In-First-Out

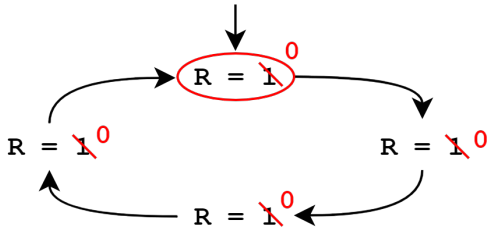
→ Die Seiten, die **zuerst** im RAM gespeichert wurden, werden als erstes überschrieben.
In der Praxis hat dieses Verfahren keine Bedeutung, da die ausgelagerte Seite oft benötigt wird → hohen Zahl an **Seitenfehlern**

Working Set

bei vielen nacheinander auf der CPU ausgeführten Befehlen werden nur wenig verschiedene virtuelle Seiten angesprochen
→ versucht, alle zum Working Set eines *Prozesses* gehörenden Seiten **ständig im Hauptspeicher** zu halten

---

Second Chance



er prüft, ob diese *Seite* in der **letzten Zeit auch angesprochen** wurde.

- Falls **nein:** sie wird ersetzt
- Falls **ja:** wird mit der am **zweitlängsten eingelagerten Seite** fortgefahren.
- Auch hier wird zunächst geprüft, ob Seite in der letzten Zeit angesprochen wurde.
- Usw.

Shared Memory

Mehrere Prozesse sollen die Möglichkeit haben, dass bestimmte Speicherbereiche gemeinsam genutzt werden können.
Dies ist eigentlich *nicht* erwünscht! **Grund** ist: *RAM kann man nie genug haben*

Beispiel:
Was passiert nun, wenn das gleiche Programm mehrmals gestartet wird? Dann resultieren daraus auch mehrere Prozesse.
Wenn für jeden Prozess der gleiche Programmtext in unterschiedliche Bereiche des Hauptspeichers eingelagert: **Redundanz**

gemeinsam genutzte Speicher entspricht einem bestimmten Seitenrahmen des *physikalischen Speichers* (z.B: DLL-Dateien)
Dieser **eine** Seitenrahmen kann nun sehr einfach in **alle** virtuellen Adressräume der beteiligten Prozesse eingebunden werden
-> Nötig ist dazu allein ein entsprechender Eintrag in **allen** betreffenden Seitentabellen.

> **Bedenke:** Jeder Prozess besitzt seinen eigenen virtuellen Speicher und damit auch seine eigene Seitentabelle!