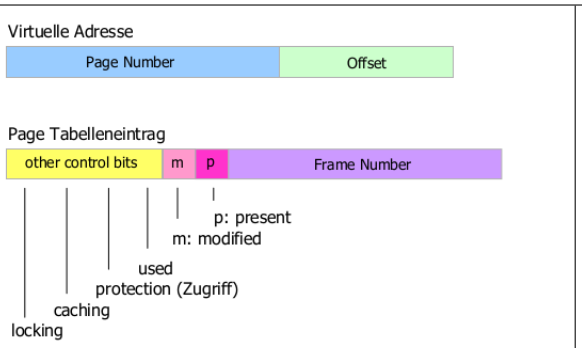
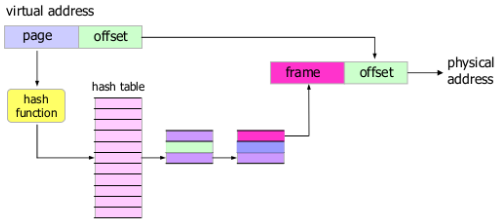
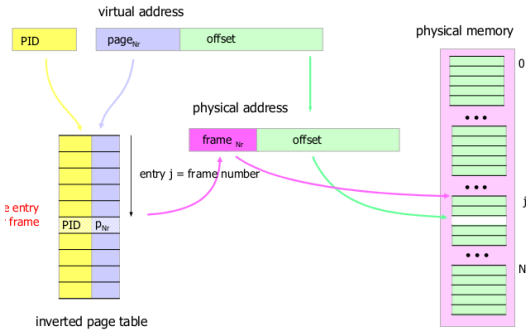
Virtual Memory

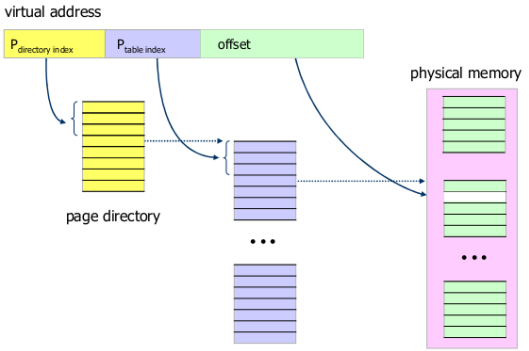


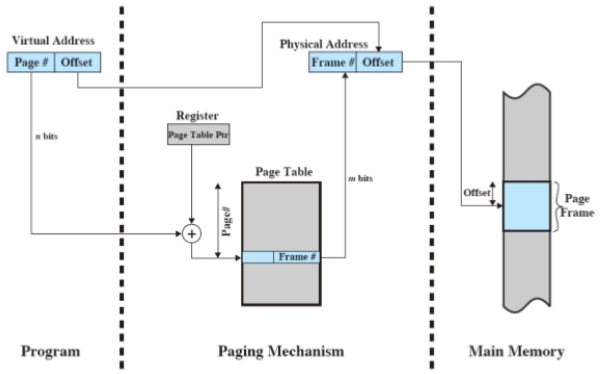
- geeignet für Invertierte Page Tabelle
- Adressräume > 32 Bit
- verschiedene Page Nummern → gleicher Hash Wert

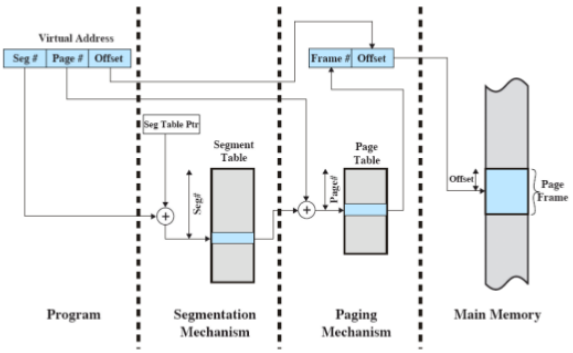


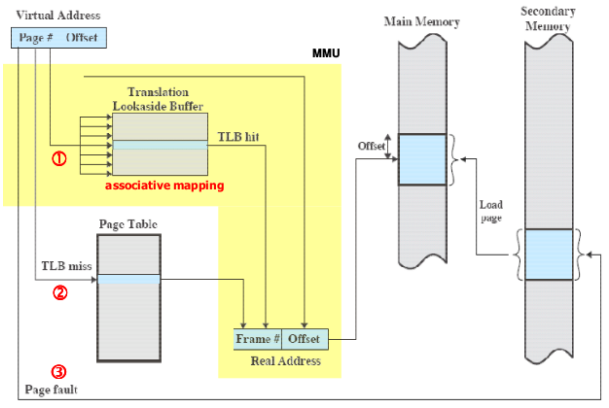


Multilevel Organisation









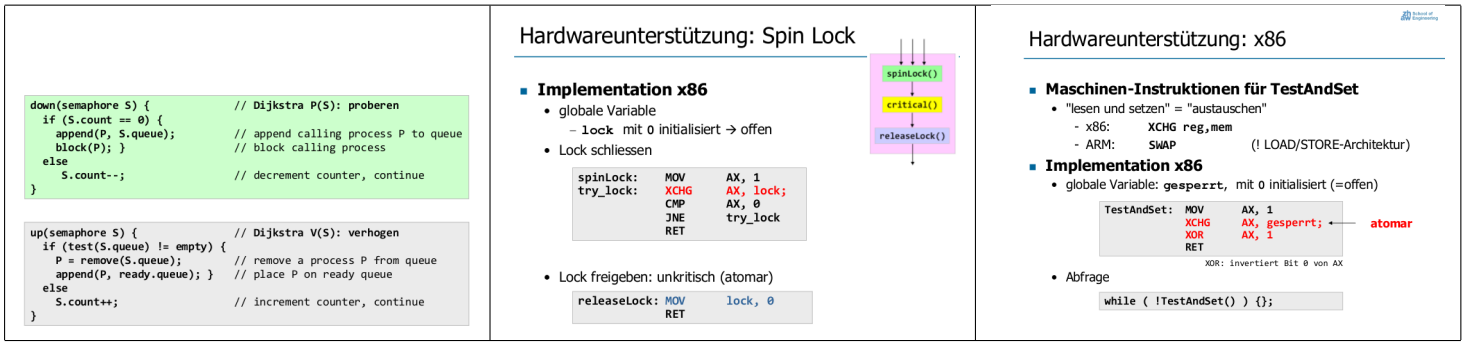
Page Replacement

- optimal
  - ersetzt Page die am spätesten referenziert wird, nicht implementierbar, beweisbar minimale Pagefaults
- least recently used
  - Lokalitätsprinzip: wird warscheinlich nicht mehr referenziert (fastz so gut wie optimal), aufwändige implementation mit TimeStamp und UsageCOunter
- fifo
  - älteste wird ersetzt: aber eben auch oft referenzierte, sehr einfach, schlechte resultate
- clock
  - used bit 1 bei laden und referenzieren, bei u=0 ersetzten und angetroffene u -> 1 setzen

Load Control

(IO Bound)(optimal)(Thrashing)
Thrashing: zu viele Prozesse im Speicher, CPU Auslastung sinkt.
Ein Prozess läuft nur, wenn minimaler WorkinSet im Speicher verfügbar ist.
Process-Suspension: Welcher soll zugunsten des Speichers suspendiert werden: tiefste Prio, meissten Pagefaults, Inaktivität, grössere Prozesse, Verarbeitungsfenster (SJF)

Deadlocks



Ressourcen-Klassen

- Preemtable (ohne Nebenwirkungen entziehbar) -> CPU, Hauptspeicher
- Nonpreemptable (Nebenwirkungen) -> Drucker, CD-Brenner

Grundsätzliche Probleme

- Starvation (Verhungern)
  - Prozess erhält keinen Zutritt zu Ressource
  - Ursache, z.B. unfaire Zuweisungspolicy: FILO (Stack)
  - Abhilfe: nur faire Policies verwenden, z.B. FIFO
- Deadlock (Verklemmung)
  - Prozesse warten gegenseitig auf Freigabe von Ressourcen
  - Die Prozesse und eventuell das gesamte System bleiben hängen

Voraussetzungen

- Mutual Exclusion
  - mindestens eine Ressource ist exklusiv reserviert
- Hold and wait
  - mindestens ein Task hat eine Ressource exklusiv reserviert und wartet auf weitere Ressourcen
- No preemption
  - reservierte Ressourcen können dem Task nicht entzogen werden (freiwillige Rückgabe nur, wenn Aufgabe gelöst)
- Circular wait
  - geschlossene "Kettenot be delayed infinitly" von Tasks existiert, in der jeder Prozess mindestens eine Ressource reserviert hat, die auch von einem Nachfolger in der Kette benötigt wird

1-3 sind Vorbedingungen, mit 4 ist es ein Deadlock.
die Circular Wait Bedingung kann nicht gelöst werden, wenn Bedingungen 1, 2 und 3 gegeben sind

Umgang mit Deadloks

- Prevention: eines dieser verhindern: Mut. excl., hold & wait, no preemption, circular wait
  - ineffiziente "Serialisierung"
- Avoidance: neuer Zustand sicher
- Detection: zulassen, beim Auftreten Massnahmen treffen
  - OS muss Auftreten bemerken
  - DL auflösen und lauffähigen Zustand widerherstellen
  - alle 40min oder bei tiefer CPU Auslastung überprüfen

Deadlock lösen

- alle beteligten Prozesse stoppen
- checkpoint restore
- der reihe nach stoppen (bis gelöst)ź
- Ressourcen entziehen (bis gelöst)ź

źStrategie: wenigsten CPU / wenigsten Output / wenigsten Ressourcen / kleinste Prio / längste gesch. Rechenzeit
=> Am besten werden die Ressourcen in Klassen eingetelit und je nach Klasse eine Strategie festgelegt.

Race Condition

Def: Gemeinsame Daten lesen und schreiben -> das Resultat hängt von der Ausführungsreihenfolge ab.
Forderungen:

- only one in the critical section
- for a finite time only
- makes no assumptions on speed/cores
- not be delayed infinitly
- requests entry and granted without delay
- halt in non critical section doesnt interfere with other processes

Software: Algorithmen: busy-wait / spinlock
Hardware: Maschineninstruktionen: atom. Inst: TestAndSet, CompareAndSwap, (Intterups off)
OS: Mutex, Semaphore
Kombiniert mit Programmiersprache: Monitore (Java)

Problem: Priority Inversion -> Solution: Priority Inheritance

-> high prio tast wants lock on resource already locked by lower prio task

-> low prio task inherits highter prio so he can finish

Monitor: Klasse mit synchronized Methoden
Mutex: Zugriff auf kritische Ressource
Synchronisation: Reihenfolge der Verarbeitung (Barriere)

Paging

Buddy-Algo

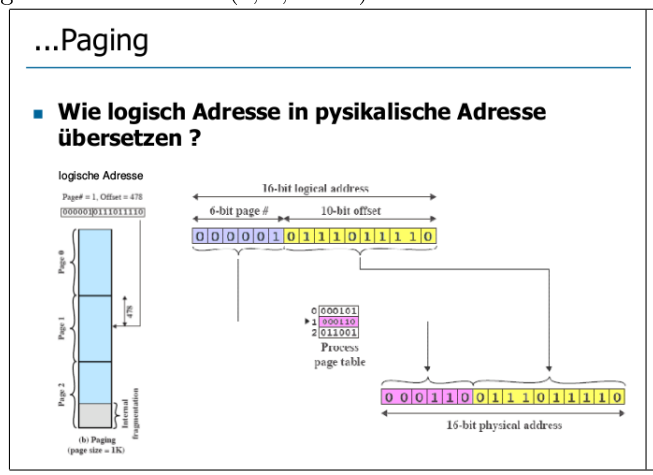
solange Blockgrösse halbieren bis Block minimaler Grösse zur verfügung steht

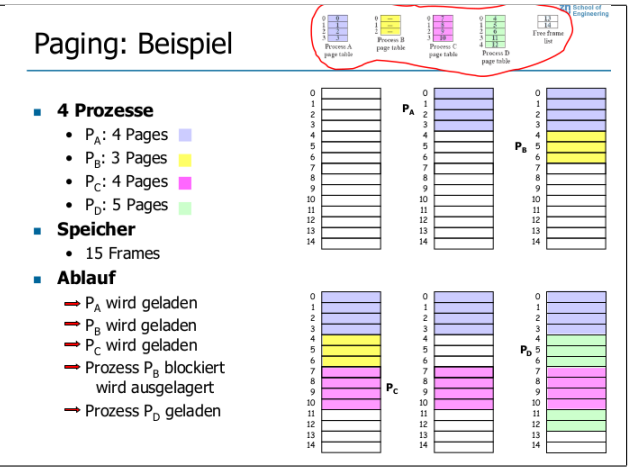
Paging

Aktuelle Betriebssysteme nur Paging, Segmentation wird auf logischer Ebene realisiert (Zugriffsrechte auf Pages Code, Stack, Data).

- logische Adressen -> Pages
- physikalische Adressen -> Frames
- pro Prozess eine Pagetable (lookup)

Pagesize = Framesize (1, 4, 8 KB)





Segmente

- Program, Data, Stack
- keine interne Fragmentierung, dafür externe
- pro Prozess eine Segment-Tabelle (OS) (lookup + Add)
  - + Speicherschutz +Programme unabh. verändern

...Segmentation

- **Wie wird eine logische Adresse in eine pysikalische Adress übersetzt ?**