5 Speicherverwaltung

- Speicherhierarchie (CPU-Register, L2-Cache, L3-Cache, RAM, Platte)
- Verwaltung, wann welche Speicherbereiche sich wo in der Speicherhierarchie befinden
- Auf oberster Ebene (Level1/2/3-Cache meist per Hardware)
- Wir konzentrieren uns auf die Verwaltung des Hauptspeichers (RAM) und des Hintergrundspeichers (Festplatte)

Aufgabe der Speicherverwaltung.:

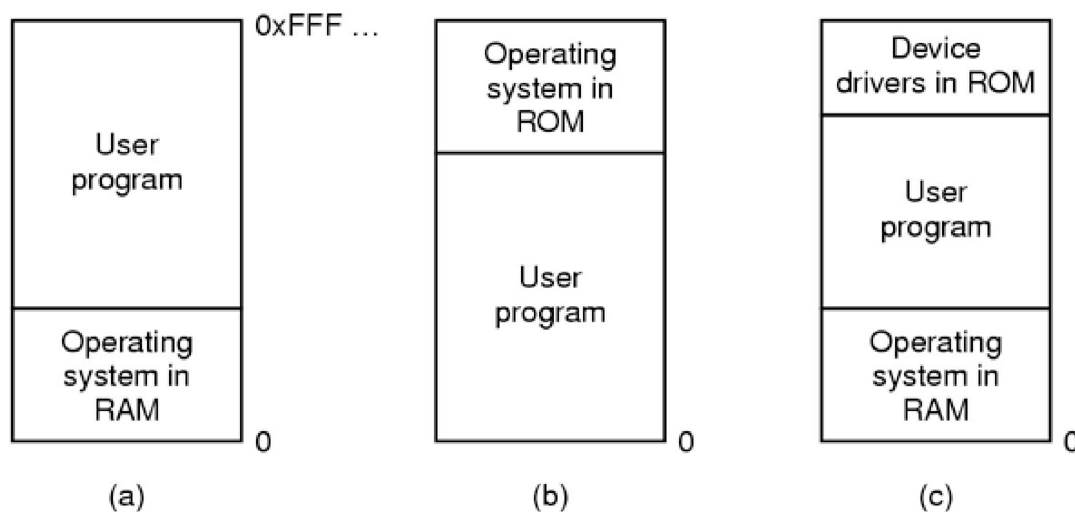
- Welche Bereiche werden gerade von welchen P. benutzt?
- Zuteilung von S. an P. bei Bedarf
- Freigeben von S. nach Beendigung von P.
- Auslagerung von S. auf Platte, wenn nicht ausreichend RAM vorhanden („swapping“ und „paging“). Heute immer noch notwendig (!)

Historische Entwicklung auch heute noch Relevant für eingebettete Systeme, Smartcards, etc.

Zwei Klassen: mit swapping/paging und ohne.

Monoprogrammierung

Ein Programm wird in den Speicher geladen und ausgeführt. Nach Beendigung wird das nächste Programm geladen und überschreibt das alte.
Anzutreffen bei z.B. Palmtops und früher bei MS-DOS:

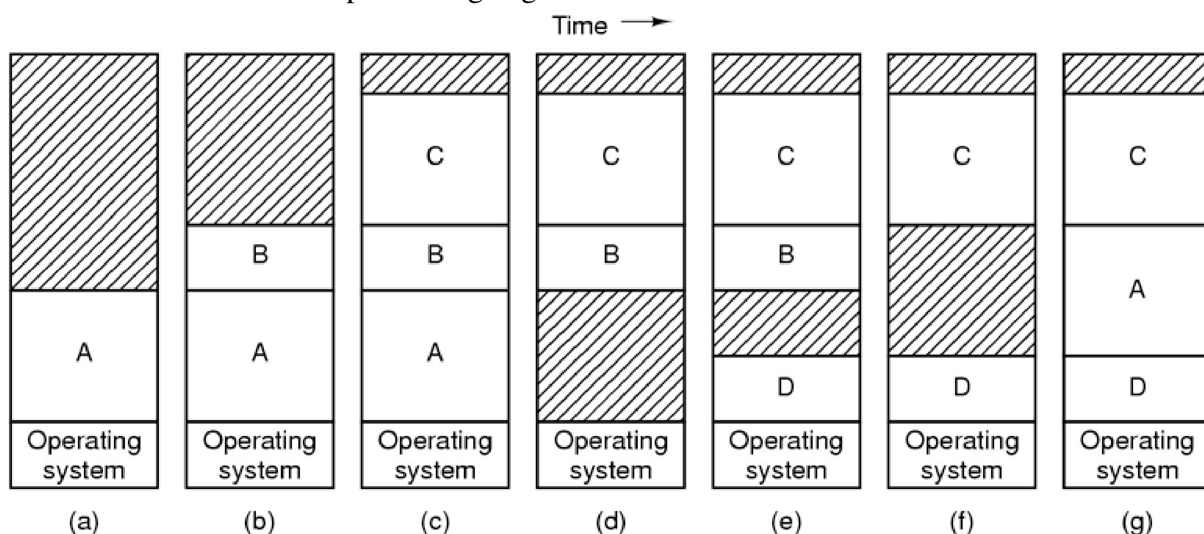


Durch Multitasking entstehen mehrere Probleme:

- Relokation: Ein Programm „weiss“ vorher nicht, an welcher Adresse es läuft. Entweder alle Adressen vor Programmstart ändern oder Basisadresse angeben.
- Speicherschutz zwischen Programmen (und BS)
- Verfügbarer Speicher reicht evtl. nicht aus (zusätzliches Starten von P. oder Programme brauchen plötzlich mehr Speicher)

Swapping

- Jeder Prozess wird komplett in den Speicher geladen, ausgeführt und dann ggfls. wieder auf Festplatte ausgelagert.



- Problem: Fragmentierung
- Problem: Auslagerung des kompletten P.
- Problem: P. können schlecht/nicht wachsen (deshalb Puffer vorsehen)

Virtueller Speicher (Paging)

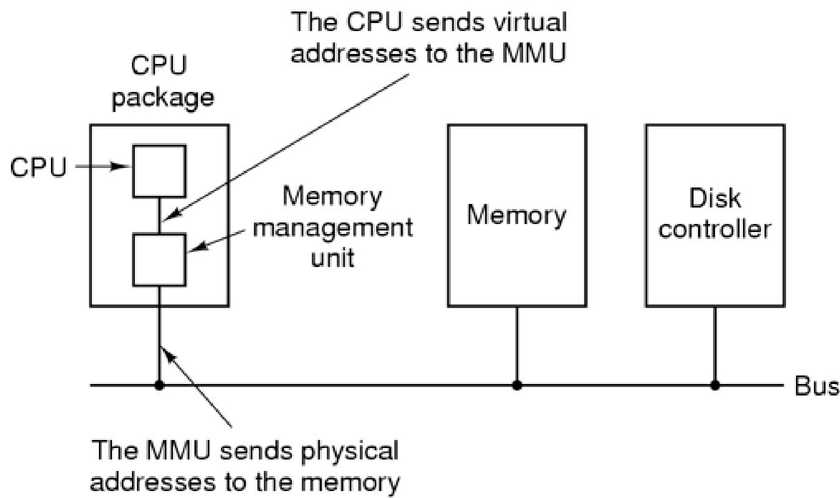
Idee: Zwischenschicht zwischen Speicher und CPU

Maschinenbefehl adressiert Speicherzelle:

MOV REG, 21334

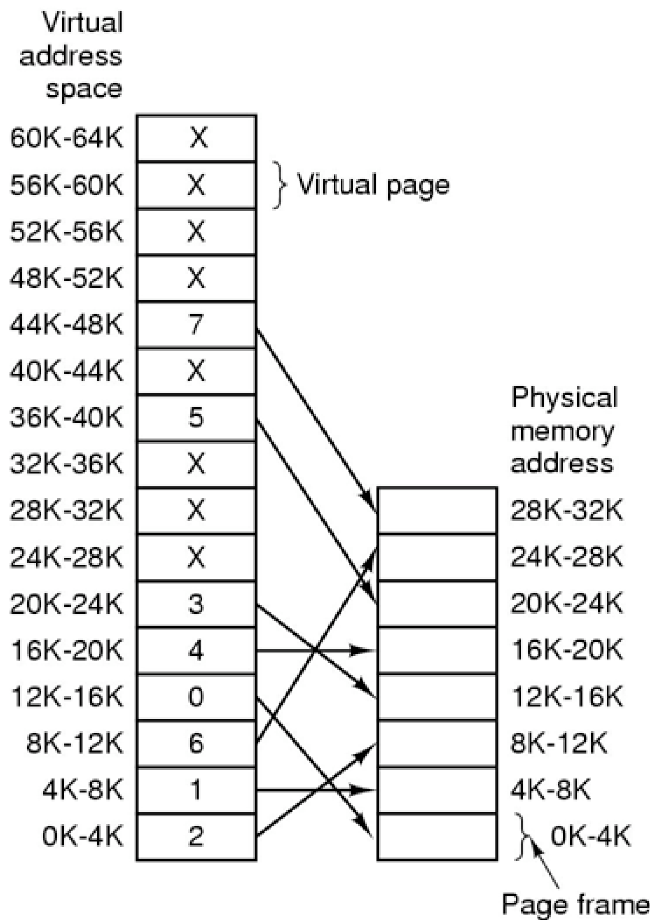
Bedeutung: Bewege den Inhalt eines CPU-Registers „REG“ an die Speicherzelle 21334. (Hex: 0x5356) Normalerweise: Adresse wird auf Speicherbus gelegt und die physische Speicheradresse überschrieben.

Mit virtuellem Speicher geht die Adresse an eine MMU (Memory Management Unit), welche die virtuelle Adresse auf die physische abbildet.



Der virtuelle Adressraum ist in gleichgroße Seiten (page frames, pages) unterteilt (Größen meist zwischen 512Byte und 64kByte, typisch ist 4kByte). Typischerweise wird der virtuelle Adressraum größer gewählt als der physische.

Beispiel: Seitengröße 4 kByte (4096 bytes = 0x1000). Real: 32k, virtuell: 64k



MOV REG, 0 → MOV REG, 13142 (0x3356) (durch die MMU (!))
Zugriff auf Seite, welche nicht existiert (markiert mit „X“ = present/absent-bit:

MOV REG, 32780

- Seitenfehler (page fault) wird generiert (Systemaufruf)
- BS sucht freie Seite und ordnet sie zu.
- Falls keine Seite frei, wird eine vorhandene Seite ausgewählt, auf Festplatte geschrieben und die dann freigewordene Seite zugeordnet.
- Falls nötig, wird angeforderte Seite von Platte geladen (Prozess ist in dieser Zeit blocked!)
- Anpassen der Abbildungstabelle
- Ein weiterer Grund dafür, blockierte Prozesse hoch zu priorisieren

Manipulation der Zuordnung nur im Kernel-Modus möglich.
BS hält eine Liste freier Seiten in einer Freelist.

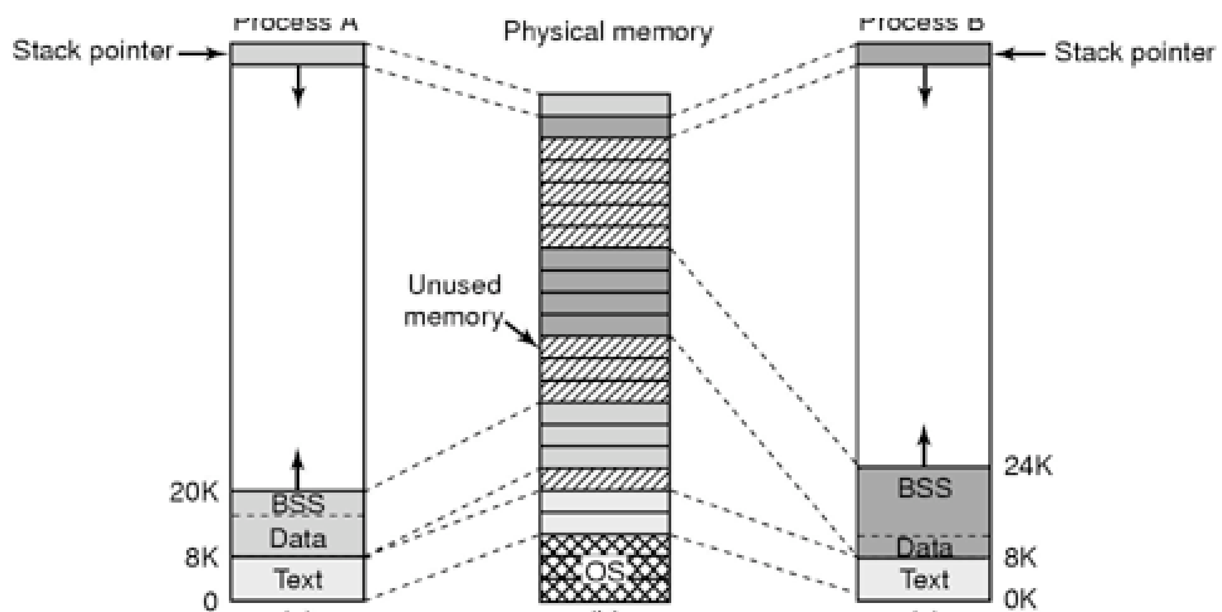
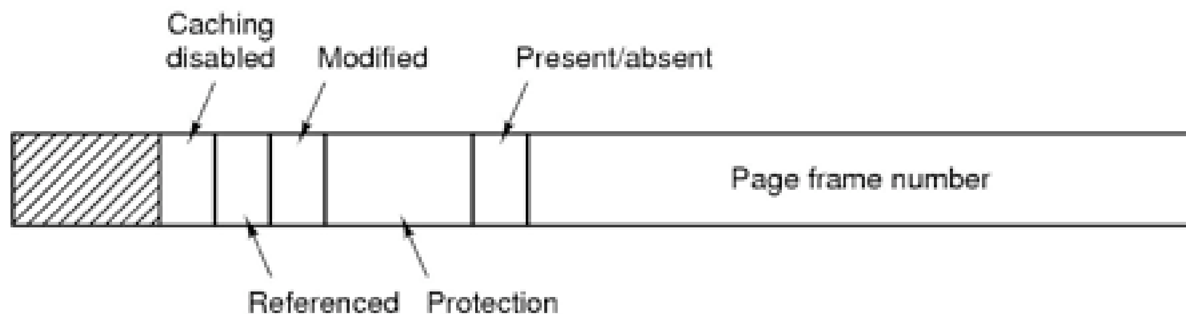


Abb.: Beispielhafte Speicherzuordnung zweier Prozesse.

MMU unterhält Seitentabelle:

- Zuordnung auf physische Seite
- Present/absent-bit
- Protection-bit (read-only). Manchmal 2 bits: execute ja/nein.
- Copy-on-write-bit
- M(odified)-Bit (dirty bit) um festzustellen, ob geschrieben wurde
- R(eferenced)-Bit um festzustellen, ob gelesen wurde.



Weitere Vorteile:

- Jeder Prozess hat eigene Seitentabelle. Speicherschutz damit durch die MMU garantiert. Beim fork() wird eine Seitentabelle kopiert.
- Jeder Prozess kann eigenen Speicherraum haben (0 ...)
- Der „Raum“ zwischen Stack und Heap muss nicht real vorhanden sein.
- Prozess kann größer sein als Physischer Speicher.
- Mehrere virtuelle Seiten können auf die selbe physische Seite zeigen (textsegment, copy-on-write)
- Der Speicher kann physisch bewegt werden, ohne dass der Prozess es merkt.
- Ein-/Auslagerung für den Prozess unmerklich.
- Seite 0 wird in der Regel nicht vergeben (Zugriff zeugt von kaputtem Pointer!)
- Shared memory einfach möglich.
- Heuristiken zur präventiven Einlagerung „naher“ Seiten.

Reale Implementierung des Plattenspeichers:

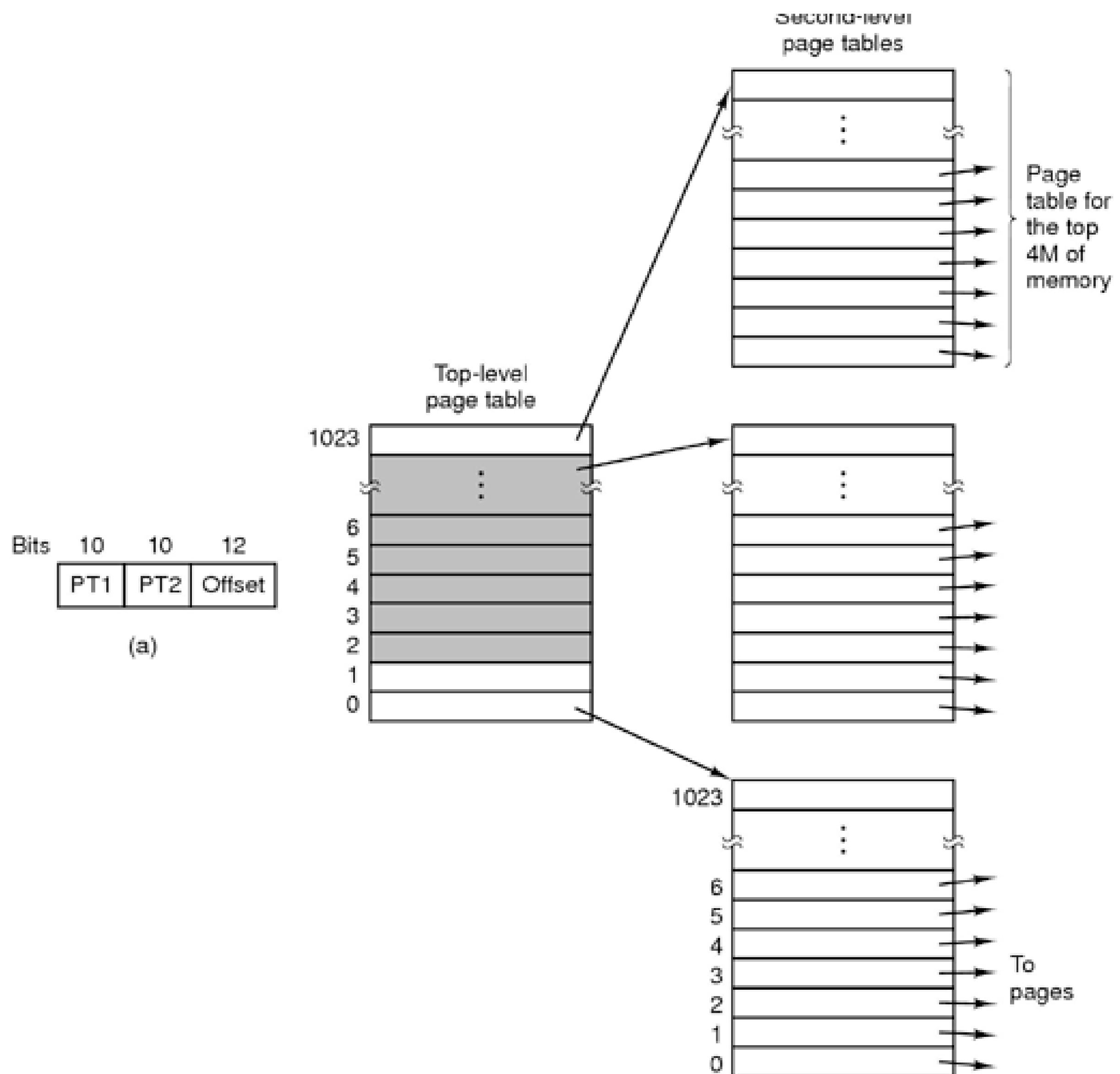
- Zuordnung von Seiten zu Plattenblöcken ist eine eigene Problematik(!)
- In Unix/Linux meist eigene Partition auf der Platte (fälschlicherweise als Swapspace bezeichnet) ohne Dateisystem. Größe ist fest. Einige System legen Swapdateien an (so z.B. MacOSX), die auch wieder gelöscht werden können (dynamisch)
- Windows: Auslagerungsdatei

Verwaltung der Seitentabelle

- Seitentabelle kann *groß* werden (virtuelle Adressen meist 32/64bit lang)
- Umrechnung muss *schnell* sein.

Naive Lösung: Eine Tabelle in der MMU. Macht Taskwechsel sehr aufwändig.

Populäre Lösung: Mehrstufige Seitentabelle im Hauptspeicher (siehe Abb.)



Jede Adresse wird aufgeteilt in 10 Bit für PT1, 10 Bit für PT2 und der Rest als Adresse innerhalb der Seite. Fragen: Adressraum? Seitengröße? Wie groß ist ein Eintrag in PT1? Grauer Bereich wird nicht genutzt (!)

Beispiel: P. benutzt 12MB. 4MB für Textsegment, 4MB Stack, 4MB Heap. Dazwischen ein Loch.

PT1 hat ebenfalls alle Informations bits (insbesondere present/absent-Bit)

Obwohl Der P. 4GB virtuell zur Verfügung hat, brauchen wir nur 4 Seitentabellen für den virtuellen Speicher (mit jeweils 1024*16 Bit) = 8kByte.

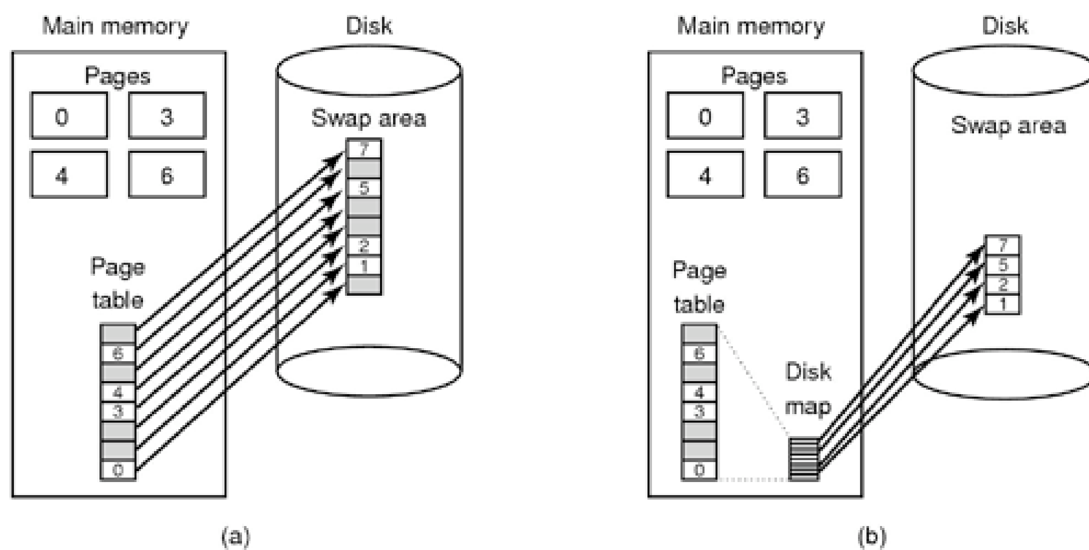
Aber: Jeder Speicherzugriff dauert jetzt dreimal so lang!

Lösung: TLB (Translation Lookaside Buffer), Assoziativspeicher

- Teil der MMU
- Beobachtung: Die meisten Programme haben eine sehr kleines Working Set
- In der Regel 8-64 Einträge, welche als Caches der Seitentabelle dienen.
- MMU kann bei einkommender Adresse alle Einträge gleichzeitig prüfen, ob Adresse vorkommt.
- Liegt Adresse nicht vor, so wird normal geholt und im TLB gespeichert.

Mit Einführung von 64bit-CPUs werden aufwändigere Algorithmen notwendig, TLB wichtiger. (z.B. Hashtabellen, welche (ProzessID, Seite) auf phys. Seite abbilden.
Linux: 3-Stufige Seitentabelle.

Verwaltung des Spwapspace



a) feste Zuordnung aller virtuellen Seiten zu Platte
b) Weitere Zuordnungstabelle

Seitenersetzungsalgorithmen

Welche Seite herauswerfen, wenn eine neue benötigt wird (wenn dirty-bit gesetzt: erst auf Platte schreiben)?

Generelles Problem beim vielen Systemen, die Caches halten: CPU-Cache, Web-Proxy mit Seitencache, Datenbank, etc.

Optimale Strategie

Seiten erhalten eine Zahl, bei welchem Befehl das nächste Mal zugegriffen wird. Seite mit der höchsten Zahl wird ausgelagert.
Problem: Diese Zahl ist nicht bekannt, daher Strategie nicht realisierbar

FIFO (First in First out)

Reihenfolge durch Alter der Seiten bestimmt. Älteste Seite wird ausgelagert.

LRU (Least Recently Used)

Seiten nach Datum des letzten Zugriffs sortiert halten.
Problem: aufwändige Listenoperationen sind notwendig.
Lösung per Hardware-Zähler, der hochläuft und bei jedem Seitenzugriff in der Seitentabelle gespeichert wird. Älteste Seite ist die mit dem kleinsten Zählerstand.

NRU-Algorithmus (Not Recently Used)

R-bits werden von der MMU beim Zugriff (lesend/schreibend) gesetzt, bis das BS sie auf 0 setzt. Die geschieht regelmäßig durch einen Timer.
Beim Seitenfehler können 4 Klassen von Seiten auftreten:

- Klasse 0: nicht R, nicht M
- Klasse 1: nicht R, M
- Klasse 2: R, nicht M
- Klasse 3: R, M

(Frage: Wie entsteht eine Seite der Klasse 1?)
NRU entfernt eine zufällige Seite aus der niedrigsten Klasse.

Weitere Wichtige Begriffe:

Demand Paging: Programmstart via paging: Ein Programm muss nicht in den Speicher geladen zu werden, es kann sich durch page-faults selber laden.

Working Set: Der Bereich eines Programmes, in dem die meiste Zeit verbracht wird und deshalb im Speicher verbleibt (oder verbleiben sollte)

Trashing: Der Working Set aller aktiven Programme übersteigt die physische Speichergröße, so dass permanent Seiten ein/ausgelagert werden und der Rechner „zusammenbricht“

Prepaging: Seiten werden geladen, noch bevor sie gebraucht werden. Ziel: Plattenkopfbewegungen minimieren.

Memory-Mapped-Files: Unix kann Dateien in den Speicher einblenden. Anstatt mit read()/write() drauf zuzugreifen, greift das Programm auf die Datei wie auf ein Array zu. Einblenden geschieht über MMU/CPU. Häufig ist der Zugriff auf dynamische Bibliotheken (Windows: .dll-Dateien) so geregelt.

Teile des Speichers können nicht ausgelagert werden (z.B. der Teil des BS, der für das Paging zuständig ist!) Häufig muss das gesamte BS im Speicher bleiben.

Teile von Textsegmenten müssen beim Auslagern nicht geschrieben werden. Sie sind ja bereits auf der Platte vorhanden (nämlich als Programmcode).

Fordert ein P. eine neue Speicherseite an, so muss diese i.Allg. voher gelöscht werden, da sie (vertrauliche!) Daten anderer P. enthalten kann.

Einige Rechner unterstützen Hardwareseitig getrennte Adressierung von Daten und Programmcode: Verdopplung des verfügbaren Adressraumes.

Page Daemon (Unix) / Balance-Set-manager (Windows) überprüft ständig, ob noch genügend freie Seiten vorhanden sind. Wenn nicht, werden Seiten vorsorglich ausgelagert (aber noch nicht freigegeben!).

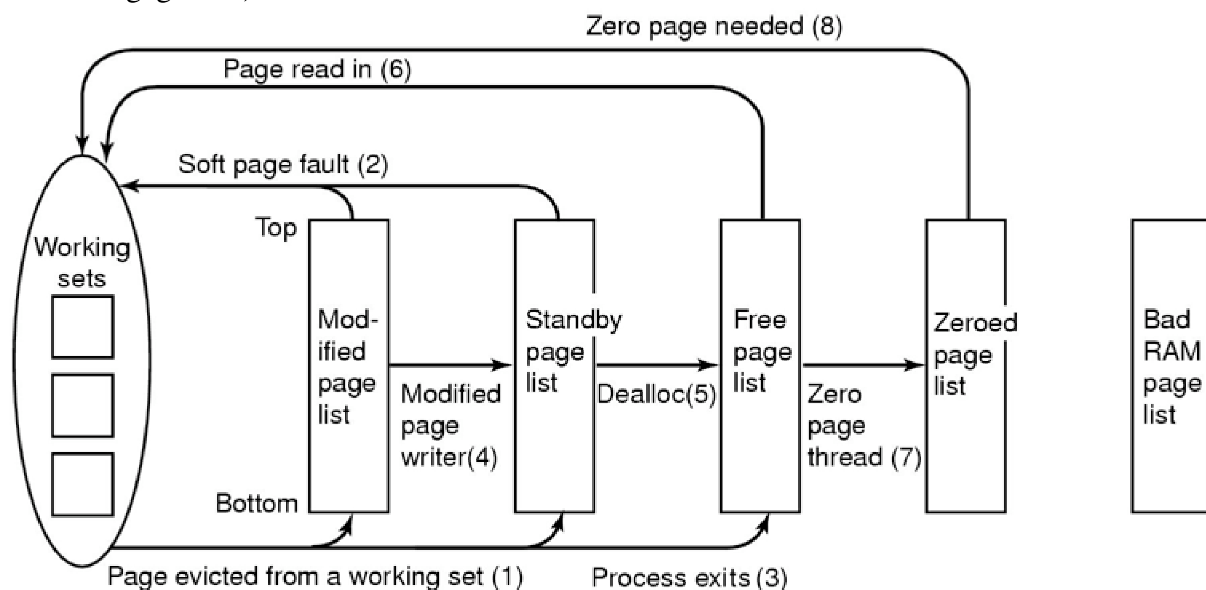


Abb.: Speicherverwaltung in Windows 2000

Verschiedene Threads verschieben Seiten bei Bedarf Seiten von einer Liste in die nächste.