Virtuelle Speicherverwaltung

Konzepte von Betriebssystem-Komponenten

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

Sommersemester 2005

Olessia Usik
olessia@freenet.de

20. Juni 2005

Gliederung

1. Einleitung

Bei den modernen Computer gibt es oft nicht genug Hauptspeicher für alle aktiven Prozesse, also müssen einige von ihnen auf der Festplatte gespeichert und bei Bedarf dynamisch in den Speicher zurückgeholt werden. Für diese Art von Speicherverwaltung gibt es zwei grundlegende Ansätze. Die einfachere Strategie ist das so genannte **Swapping**, bei dem jeder Prozess komplett in den Speicher geladen wird, eine gewisse Zeit laufen darf und anschließend wieder auf die Festplatte ausgelagert wird. Bei der anderen Strategie, dem **virtuellen Speicher**, können Programme auch dann laufen, wenn sich nur ein Teil von ihnen im Hauptspeicher befindet.

2. Swapping

Der wichtigste Aspekt der realen Speicherverwaltung ist, dass nur wirklich vorhandener physikalischer Hauptspeicher benutzt wird. Programme können also nicht größer als der zur Verfügung stehende Hauptspeicher sein. Also können in der Regel nicht alle Prozesse gleichzeitig im Speicher gehalten werden. Das Ein- bzw. Auslagern von nicht aktiven Prozessen nennt man **Swapping**. Bei der Speicherverwaltung werden Bitmaps oder verketteten Listen (Freibereichslisten) benutzt.

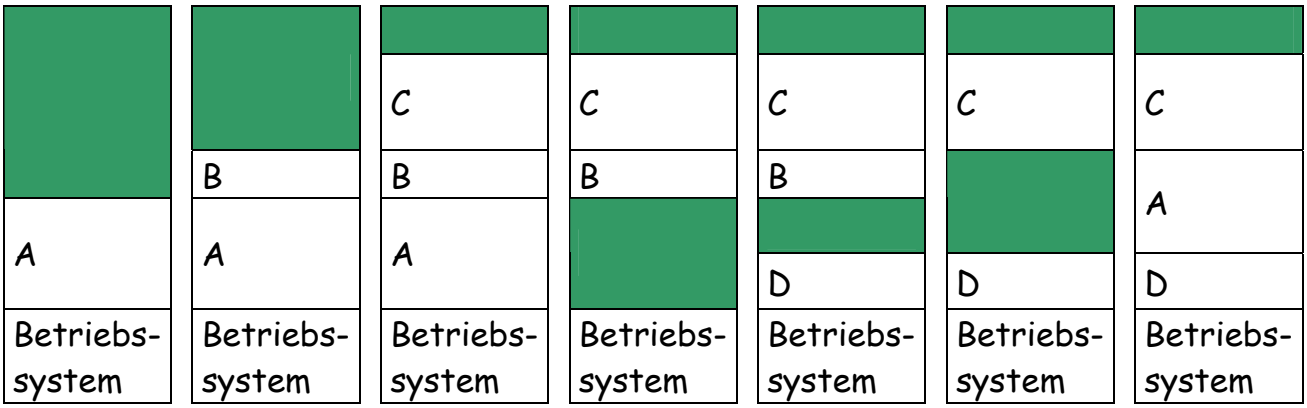

Abbildung 1: Mit der Ein- und Auslagerung von Prozessen ändert sich die Speicherzuteilung. Dunkle Bereiche sind ungenutzt. [1]

Algorithmen zum Auffinden des "optimalen" freien Speicherbereichs:

- **First Fit:** Wähle den ersten freien Bereich, der groß genug für das Programm ist.
- **Next Fit:** Wähle den nächsten (modulo Hauptspeichergröße) freien Bereich, der groß genug für das Programm ist. Starte an der Stelle, an der das letzte Programm eingefügt wurde. (Ringförmiges Durchsuchen)
- **Best Fit:** Wähle den kleinstmöglichen freien Bereich, der groß genug für das Programm ist.
- **Worst Fit**: Auswahl des jeweils größten Loches
- **Quick Fit:** Getrennte Listen für Löcher in einigen gebräuchlicheren Größen

Nachteile von Swapping:

- Der Platz für Programm und Daten ist durch die **Hauptspeicherkapazität beschränkt**.
- **Problematisch** bei den Prozessen, die **viele E/A-Operationen** ausführen
- Der **Zugriff auf den Speicherbereich** eines fremden Prozesses muss durch das Betriebssystem explizit **verhindert** werden
- Die zusammenhängende Belegung von Hauptspeicher für ein ganzes Programm **verschärft das Fragmentierungsproblem**.
- Der **komplette Adressraum** eines Prozesses wird beim Prozesswechsel von Hauptspeicher auf die Festplatte **ausgelagert** und ein anderer Adressraum von Festplatte in den Hauptspeicher **eingelagert**
- Extrem **aufwendiger Prozesswechsel**

Diese Nachteile werden durch das Konzept des virtuellen Speicher behoben.

3. Virtuelle Speicherverwaltung

Bei der realen Speicherverwaltung darf ein Prozess maximal so groß wie der Hauptspeicher sein. Aber die meisten Programme sind zu groß für den verfügbaren Speicher, da die meisten Prozesse einen großen Speicherbedarf haben. So wurde die virtuelle Speicherverwaltung erfunden (Fotheringham, 1961). Die Grundidee dahinter war, zu erlauben, dass der Programmcode, die Daten und der Stack größer sind als der verfügbarer Hauptspeicher.

Für die Umsetzung von virtuelle in reale Adressen sorgt die Speicherverwaltung des Betriebssystems im Zusammenspiel mit der MMU (memory managment unit), die bei heutigen Prozessoren zusammen mit der CPU auf einem Chip untergebracht ist. Befinden sich dabei die gewünschten Einträge nicht im Hauptspeicher, so liest sie das Betriebssystem von der Platte in den Hauptspeicher ein, um sie so zugreifbar zu machen. Um Platz zu schaffen, müssen eventuell andere Hauptspeichereinträge auf die Platte verdrängt werden. Zur Organisation des virtuellen Speichers gibt es einerseits den segmentorientierten Ansatz, bei dem der Speicher in Einheiten unterschiedlicher Größen aufgeteilt ist, und anderseits den seitenorientierten Ansatz, bei dem alle Speichereinheiten gleich lang sind.

3.1 Segmentorientierter Speicher

Beim segmentorientierten Ansatz besteht der virtuelle Speicher eines Prozesses aus einer Anzahl unterschiedlich großer Einheiten, den so genannten **Segmenten** (von einander unabhängige Adressräume). Die Aufteilung des Speichers in Segmente geschieht nach logischen Gesichtspunkten; z.B. kann ein Prozess ein Code-, ein Daten- und ein Stacksegment besitzen oder auch jeweils mehrere davon. Jedes Segment besteht aus einer linearen Folge von Adressen, von 0 bis zu einem bestimmten Maximum. Verschiedene Segmente können verschieden groß sein und sind es normalerweise auch. Außerdem kann sich die Größe eines Segments während der Ausführung ändern. Weil jedes Segment einen eigenen Adressraum darstellt, können Segmente unabhängig voneinander ihre Größe ändern. Natürlich kann einem Segment der Platz ausgehen, aber das kommt nur selten vor, weil Segmente normalerweise sehr groß sind.

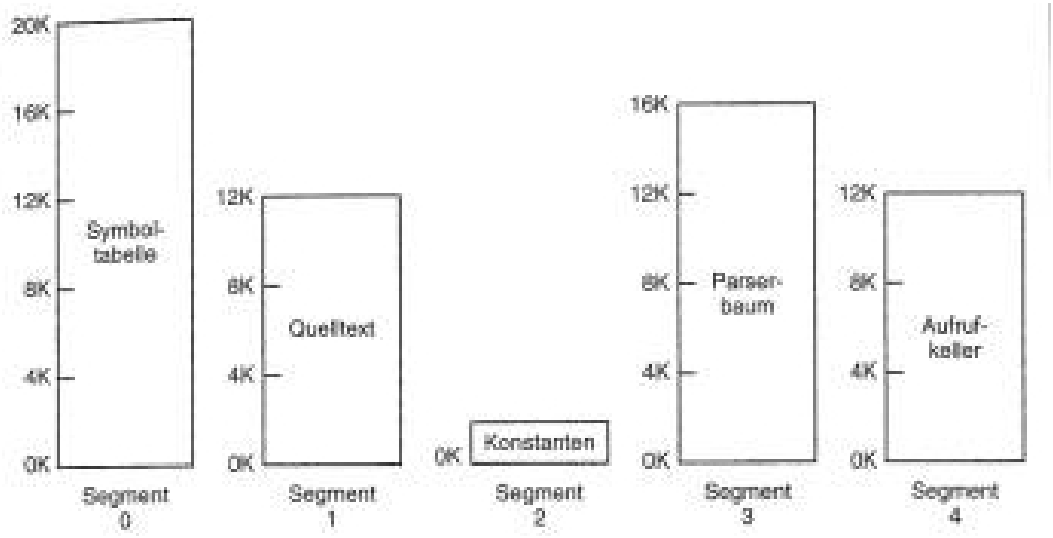

Abbildung 2: Tabellen von segmentiertem Speicher können unabhängig voneinander ihre Größe ändern. [1]

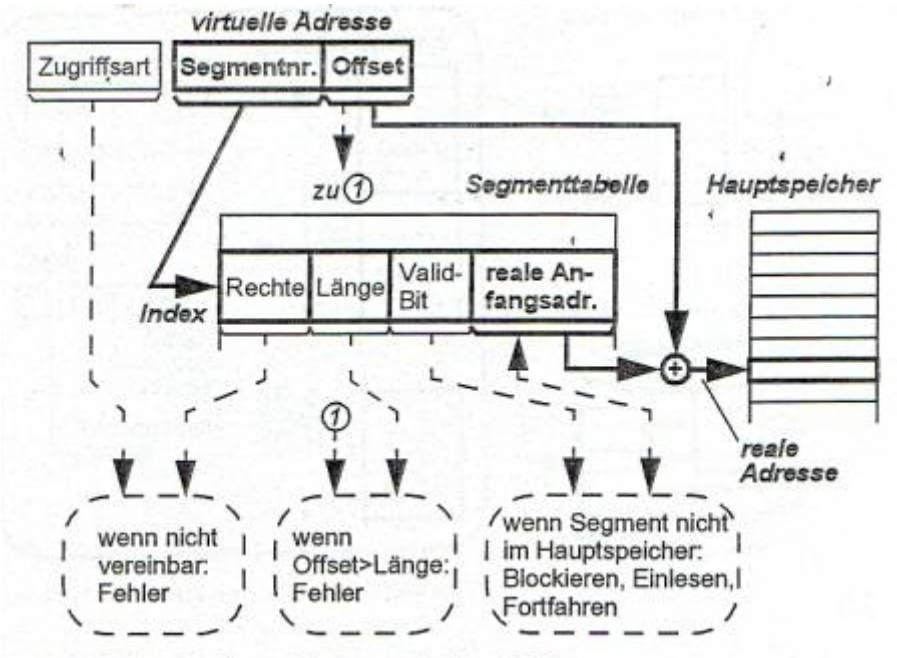

Abbildung 3: Berechnung der virtuellen Adresse bei segmentorientierten Speichern. [3]

Vorteile:

- Die Segmente entsprechen unmittelbar den **logischen Speichereinheiten** eines Prozesses
- Die Segmente können **individuell** durch verschiedene **Zugriffsrechte** geschützt werden
- Die Segmente können **einzeln** auf den Plattenspeicher **verdrängt werden**
- Durch **eigene Segmenttabellen** sind die Prozesse wirksam voneinander abgeschottet
- **leichte Zusammenarbeit** von Prozessen möglich
- **einfache Verwaltung** von Datenstrukturen, die ihre Größe ändern

Nachteile:

- für jedes Segment wird ein **zusammenhängender Speicherbereich** gebraucht, der möglicherweise recht groß ist
- **Verschnittprobleme**, da die Segmente unterschiedlich groß sind
- **Vollständiges Segment** muss bei einem Zugriff **im Hauptspeicher** stehen, obwohl vielleicht nur ein kleiner Teil davon benötigt wird

3.2 Seitenorientierter Speicher (Paging)

Bei seitenorientiertem Speicher wird der virtuelle Adressraum in Seiten (engl. *pages*) unterteilt, deren Größe variabel sein können. Der physikalische Speicher ist in Einheiten den sog. Seitenrahmen (*page frames)* derselben Größe wie die *Seiten* unterteilt. Da der virtueller Speicher mehr *pages* enthält als der physikalische Hauptspeicher, können nicht alle *Seiten* einem *Seitenrahmen* zugeordnet werden, das heißt aber, dass das Betriebssystem im Zusammenspiel mit der MMU einen Mechanismus haben muss, mit dem festgestellt werden kann, ob *Seiten* einem *Seitenrahmen* zugeordnet sind, und wenn ja, welchem?

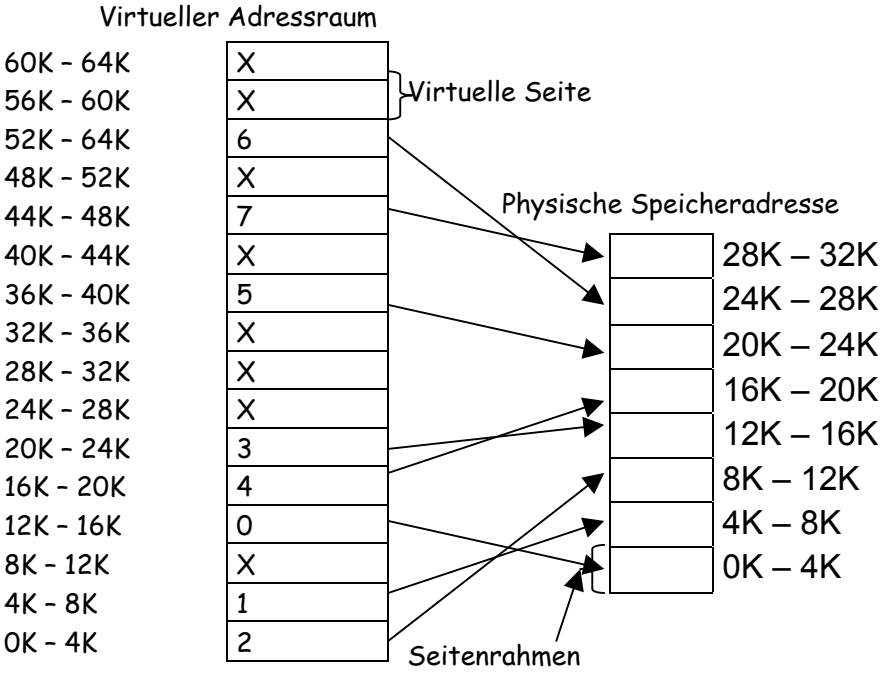

Abbildung 4:
Die Seitentabelle legt die Beziehung zwischen virtuellen und physischen Adressen fest. [1]

Zur **Speicherung der Seitentabelle** gibt es mehrere Möglichkeiten:

- Vollständig im Hauptspeicher
- Translation Lookaside Buffer (TLAB oder **TLB**)
- **Mehrstufige** Seitentabellen
- **Invertierten** Seitentabelle

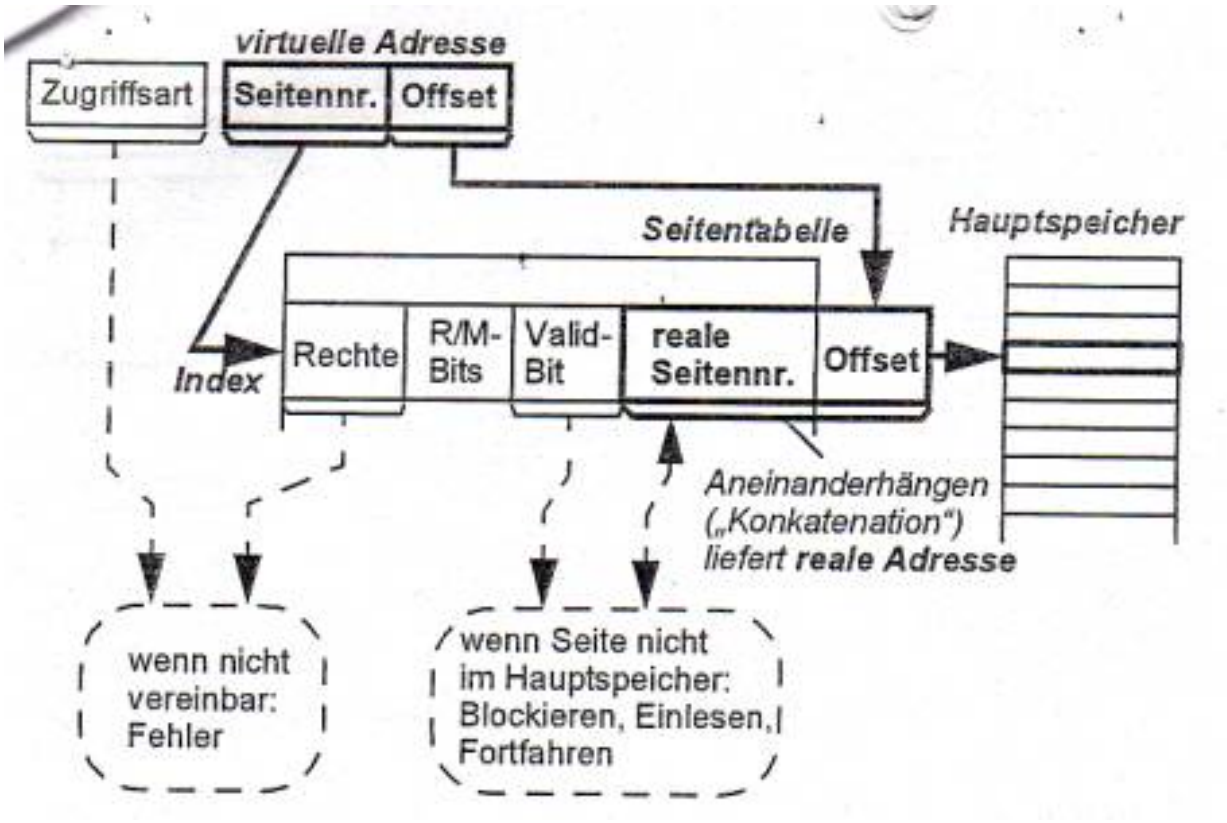

Abbildung 5: Berechnung der virtuellen Adresse bei seitenorientierten Speicher. [3]

Vorteile:

- **einfache Speicherverwaltung**
- **kein Verschnittproblem**
- **leichterer Zugriff** auf den Plattenspeicher
- es werden **nur die benötigten Daten** in den Hauptspeicher geladen
- Prozesse sind von einander abgeschottet (durch **eigene Seitentabellen**)

Nachteile:

- virtueller Speicher **spiegelt nicht den logischen Aufbau** der Prozesse wieder
- hoher Speicherbedarf für Seitentabellen
- **zeitaufwendige Adressberechnung**

3.3 Paging vs. Segmentierung

Überlegungen	Paging	Segmentierung
Muss der Programmierer wissen, dass diese Technik benutzt wird?	Nein	Ja
Wie viele lineare Adressräume gibt es?	1	Viele
Kann der gesamte Adressraum die Größe des physischen Speichers übersteigen?	Ja	Ja
Können Prozeduren und die Daten unterschieden und getrennt voneinander geschützt werden?	Nein	Ja
Können Tabellen mit wechselnder Größe verwaltet werden?	Nein	Ja
Wird das Benutzen der Prozeduren von mehreren Nutzern unterstützt?	Nein	Ja
Warum wurde diese Technik eingeführt?	Damit ein großer linearer Adressraum benutzt werden kann, der die Größe des physikalischen Hauptspeichers übersteigt	Damit Programm und Daten in unabhängige logische Adressräume gespalten werden und für gemeinsame Benutzung und den Schutz

3.4 Pagingstrategien

Das Betriebssystem verschiebt beim Paging ständig Seiten in der Speicherhierarchie: Es lädt Seiten in den Hauptspeicher, um sie zugreifbar zu machen (insbesondere zur Reaktion auf Seitenfehler), und verdrängt Seiten auf den Plattenspeicher, um im Hauptspeicher Platz zu schaffen. Dabei muss es nach drei Strategien vorgehen:

- **Ladestrategie**: Sie legt fest, zu welchem Zeitpunkt welche Seiten in den Hauptspeicher geladen wird

  ➔ *Demand Paging:* lädt eine Seite nur dann, wenn sie unmittelbar benötigt wird, wenn also bezüglich dieser Seite ein Seitenfehler aufgetreten ist.
  ➔ *Prepaging:* lädt Seiten vorzeitig, d.h. auch dann, wenn auf sie nicht unmittelbar zugegriffen werden soll.

- **Speicherzuteilungsstrategie**: Sie entscheidet darüber, wie viele Seitenbereiche des Hauptspeichers den einzelnen Prozessen zugeteilt werden.

  ➔ *Working-Set-Strategie*: versucht, für einen Prozess genau die Seiten im Hauptspeicher zu halten, auf denen er gerade arbeitet.
  ➔ *Page-Fault-Frequency-Strategie:* vergrößert den Bereich, wenn die Seitenfehlerrate des Prozesses einen oberen Schwellenwert übersteigt, und verkleinert ihn, wenn die Fehlerrate einen unteren Schwellwert unterschreitet.

- **Ersetzungsstrategie:** Sie bestimmt, welche Seiten verdrängt werden, wenn Hauptspeicher benötigt wird

Algorithmus	Kommentar
Optimal	Unrealisierbar, aber gut für Vergleiche
NRU(Not Recently Used)	Sehr primitiv
FIFO(First-In, First-Out)	Auch wichtige Seiten könnten entfernt werden
Second Chance	Enorme Verbesserung gegenüber FIFO
Clock	Realistisch
LRU(Least Recently Used)	Exzellent, aber schwierig zu Implementieren
NFU(Not Frequently Used)	Unoptimiert mit LRU vergleichbar
Aging	Effizienter Algorithmus, der fast LRU erreicht
Working set	Etwas aufwendig zu implementieren
WSClock	Guter und effizienter Algorithmus

♦ **lokal:** Prozess ersetzt nur immer seine eigene Seiten
- statische Zuteilung von Seiten pro Prozess
- Seitenfehlerverhalten liegt in der Verantwortung des Prozesses

♦ **global:** Prozess ersetzt auch Seiten anderer Prozesse
- dynamische Verhalten der Prozesse berücksichtbar
- bessere Effizienz, da ungenutzte Seiten von anderen Prozessen verwendet werden können

3.5 Behandlung von Seitenfehlern

1. Die Hardware **erkennt** den Seitenfehler und erzeugt eine **Unterbrechung**.
2. Das Betriebssystem **sichert die Register** des Prozesses.
3. Die Prozedur zur Behandlung von Seitenfehlern **ermittelt die Adresse** der fehlenden Seite.
4. Das Betriebssystem prüft, ob auf die Adresse überhaupt zugegriffen werden darf.
5. Das Betriebssystem wählt eine **freie Kachel** aus. Falls keine freie Kachel existiert, wird eine belegte Kachel ausgewählt.
6. Falls die ausgewählte Kachel belegt ist und modifiziert wurde, wird sie auf dem **Hintergrundspeicher gesichert**. Ein Prozesswechsel findet statt, sofern rechenbereite Prozesse existieren.
7. Sobald eine freie Kachel da ist, wird eine E/A-Operation gestartet, um die benötigte Kachel vom Hintergrundspeicher zu laden. Ein Prozesswechsel findet statt, sofern rechenbereite Prozesse existieren.
8. Sobald die Kachel geladen ist, wird die **Umsetzungstabelle aktualisiert**.
9. Der Befehlszähler wird auf den Befehl zurückgesetzt, der den Seitenfehler auslöste.
10. Der Prozess wird in den Zustand **ready** gesetzt und in die CPU-Warteschlange eingereiht

4. Virtuelle Speicherverwaltung bei verschiedenen Betriebssystemen

4.1 Unix

- Früher: Swapping -> Heute: **Demand Paging**
- Es wird kein Working-Set-Modell verwendet

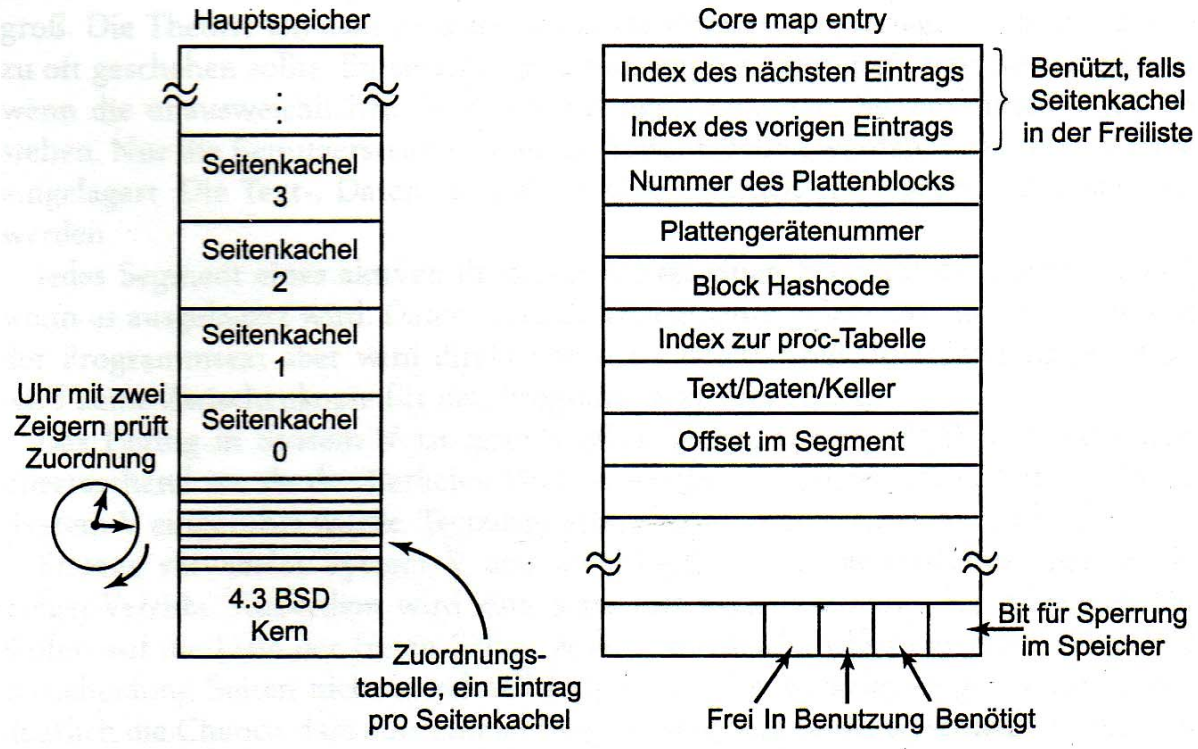

Abbildung 6: Das Kernabbild in 4BSD. [1]

• Paging wird vom **Page Daemon** durchgeführt
- wird periodisch gestartet
- Überprüfung, ob die Anzahl an freien Seiten zu klein geworden ist

► Falls ja -> Auslagerung von Seiten auf die Festplatte gemäß der eingesetzten Verdrängungsstrategie

►Falls nein -> Blockierung des Paging-Daemons bis zum nächsten Sollzeitpunkt

• globaler **Zwei-Zeiger-Uhr-Algorithmus** als Seitenersetzungsalgorithmus

• Um Thrashing zu vermeiden wird **Swapping** eingesetzt:
- Gibt es ein Prozess, der länger als 20 Sekunden untätig ist?

► Falls ja -> wird derjenige ausgelagert, der am längsten untätig war

► Falls nein -> die vier größten Prozesse werden untersucht und derjenige ausgelagert, der sich am längsten im Speicher befindet

- Soll ein bereiter Prozess wieder zurück eingebracht werden?
-> es wird nach Werten entschieden,
die von verschiedenen Größen abhängen

• Besonderheiten bei **System V**

- einfacher Uhralgorithmus
- statt *losfree*: *min* und *max*

4.2 Windows 2000

• Ein einziger, linearer 4-GB-Adressraum je Prozess
• Segmentierung wird nicht unterstützt -> **Seitenadressierung**
• Bei Zuordnung eines Bereiches des virtuellen Adressraums wird die Datenstruktur **VAD** (Virtual Address Descriptor) angelegt.

► VAD Inhalt:

- Bereich der abgebildeten Adressen
- zuständige Auslagerungsdatei auf dem Hintergrundspeicher
- Offset
- Schutzcode

► Erste Berührung einer Seiten führt zur Erzeugung des Seitentabellenverzeichnis -> Eintrag des Zeigers in VAD

• **Kein Prepaging** vorhanden

► Beim Prozessstart sind keine Seiten im Speicher
► Dynamische Einlagerung bei Seitenfehler

20	3	1	1	1	1	1	1	1	1	1
Seitenkachel	Nicht benutzt	G	L	D	A	C	Wt	U	W	V

G: Seite global für alle Prozesse
L: Große (4GB) Seite (large)
D: Seiteninhalt ungültig (dirty)
A: Auf Seite wurde zugegriffen (accessed)
C: Caching ein-/ausgeschaltet
Wt: Write through (kein Caching)
U: Seite sichtbar im Benutzermodus
W: Seite ist beschreibbar (write)
V: Gültiger Eintrag (valid)

Abbildung 7: Ein Seitentabelleneintrag für eine eingelagerte Seite auf einem Pentium. [1]

- Arten von **Seitenfehlern**:
  - Die referenzierte Seite ist nicht belegt
  - Eine Schutzverletzung ist aufgetreten
  - Auf eine geteilt genutzte Seite wurde geschrieben
  - Der Stack muss wachsen
  - Die referenzierte Seite ist belegt, aber momentan nicht eingelagert.

- Bei Seitenfehler werden **benachbarte Seiten** auch eingelagert

- Ziel bei Seitenersetzungsalgorithmus: Gewisser Prozentsatz aller Seiten soll freigehalten werden (Freibereichsliste)
  ► Keine Auslagerung bei Seitenfehlern erforderlich

- **Working-Set Verfahren**

- besteht aus eingelagerten Seiten eines Prozesses
- wird durch zwei Parameter beschrieben:
-> die minimale und die maximale Größe
- Zu Beginn ist das Minimum des Working-Sets auf einen Wert zwischen 20 und 50 und das Maximum zwischen 45 und 345 gesetzt
- Jeder Prozess startet mit demselben Minimum und Maximum
- Entscheidung bei Seitenfehler
  ► Größe des WS < MIN -> Seite wird hinzugefügt
  ► Größe des WS > MAX -> Seite wird aus WS entfernt (aber nicht aus dem Speicher)
- lokaler Algorithmus

- **Balance-Set-Manager**: Sind genügend freie Seiten vorhanden?

NEIN: Start des Working-Set-Managers zur Seitenfreigabe
- Prüfreihenfolge anhand von Leerlaufzeiten und Größe der Prozess wird festgelegt
- Prozesse mit Working-Set kleiner als Minimum oder mit einer großen Anzahl von Seitenfehlern werden ignoriert

5. Zusammenfassung

Swapping bedeutet, dass das Betriebssystem mehr Prozesse ausführen kann, als im Speicher Platz haben. Die Prozesse, für die kein Platz ist, werden auf die Festplatte ausgelagert. Der freie Platz im Speicher und auf der Platte kann mit Hilfe einer Bitmap oder einer Freibereichsliste verwaltet werden.

Moderne Computer unterstützen oft virtuellen Speicher. In der einfachsten Form wird der Adressraum eines Prozesses in gleich große Blöcke zerlegt, die Seiten genannt werden. Jede Seite kann in einen beliebigen Seitenrahmen im Speicher geladen werden. Es gibt viele verschiedene Seitenersetzungsalgorithmen, zum Beispiel Aging und WSClock.

Paging-Systeme können modelliert werden, in dem man ein Programm als eine Folge von Speicherzugriffen betrachtet und diese Referenzkette mit verschiedenen Algorithmen benutzt. Mit diesen Modellen lassen sich Vorhersagen über das Paging-Verhalten machen.

Segmentierung bietet Vorteile bei Datenstrukturen, die ihre Große ändern, und vereinfacht das Linken und die gemeinsame Benutzung von Code und Daten. Außerdem ermöglicht es, verschiedene Segmente vor verschiedenen Arten von Zugriffen zu schützen. Segmentierung und Paging werden manchmal zu einem zweidimensionalen virtuellen Speicher kombiniert.

Quellen:

[1] Andrew S. Tanenbaum: „Moderne Betriebssysteme“
2. Auflage, Pearson Studium, 2002
[2] Rüdiger Brause: „Betriebssysteme. Grundlagen und Konzepte“
3. Auflage, Springer Verlag, 2004
[3] Carsten Vogt: „Betriebssysteme“
1.Auflage, Spektrum Akademischer Verlag, 2001
[4] www.ibr.cs.tu-bs.de/lehre/ss04/ bsn/BSN-SoSe04-Kap03-Speicherverwaltung-3S.pdf
aufgerufen am 16.04.2005
[5] http://wwwcs.uni-paderborn.de/cs/ag-kao/de/teaching/ss05/bs1/script/BS_SS05_kap3_2seiten.pdf
aufgerufen am 05.06.2005