# Freispeicherverwaltung

Martin Wahl, 17.11.03

## Allgemeines zur Speicherverwaltung

Der physikalische Speicher wird in zwei Teile unterteilt: -Teil für den Kernel
-Dynamischer Speicher

Die Verwaltung des dynamischen Speichers entscheidet über die Geschwindigkeit des Gesamtsystems und wird vom Kernel gesteuert. Dazu wird der dynamische Speicher in Blöcke aufgeteilt.
Der Pentium 4 kann 4 MB- und 4 KB-Blöcke ansteuern, Linux verwendet die Verwaltung in 4 KB-Blöcken wegen der effizienteren Handhabung.

Es muss zusätzlich unterschieden werden zwischen virtuellem und physikalischem Speicher. Die Anzahl realer Hauptspeicheradressen ist durch die Größe des Hauptspeichers limitiert, während der virtuelle Speicher nur durch die Anzahl der Bits im virtuellen Adressraum eingeschränkt ist (z.B. 32-bit Adressen: $2^{32}$ = 4 GB Adressraum). Die virtuellen Adressen müssen dabei auf den physikalischen Adressraum abgebildet werden, bei überbelegtem Hauptspeicher müssen Speicherinhalte auf einen Sekundärspeicher ausgelagert werden.

## Methoden der Speicherverwaltung

### Suchstrategien

Um einen passenden Freibereich für eine Datenmenge im Speicher zu finden, werden mehrere Strategien angewandt:

-First-Fit: Suche nach dem ersten passenden Freibereich

-Next-Fit: Suche wie First-Fit, nur mit Zeiger, sodass die Suche an der Stelle fortgesetzt werden kann, wo die letzte Suche aufgehört hat

-Best-Fit: Suche nach dem kleinsten passenden Freibereich

-Worst-Fit: Suche nach dem größten passenden Freibereich

Es gibt noch weitere Stategien, die die hier aufgezeigten als Grundlage verwenden.

Verwaltungsstrategien

-Bit Maps:

Der Speicher wird in Einheiten gleicher Größe unterteilt, jeder Einheit einspricht ein Bit in der Bit Map; ist das betreffende Bit 0, so ist die Einheit frei, ist das Bit 1, ist die Einheit belegt. Je größer die Einheiten, desto kleiner die Bit Map und umgekehrt.
Vorteile:

-einfache Handhabung
-Bit Map nimmt immer eine feste Größe an

Nachteile:

-beim Suchen eines Freispeicherbereichs muss die Bit Map nach zusammenhängenden Nullen durchsucht werden
-im schlechtesten Fall muss dabei die gesamte Bit Map durchsucht werden
=> ineffizient, wird deshalb kaum verwendet

-Linked Lists:

Der Speicher wird in verketteten Listen verwaltet, wobei die einzelnen Segmente der Liste entweder von einem Prozess belegt sind oder eine Lücke zwischen zwei belegten Segmenten bildet bzw. die Information enthält, wie groß diese Lücke ist und wo sie beginnt. Jedes Segment verweist dabei auf das nächste. Die Liste ist meist nach der Reihenfolge der Adressen sortiert.
Enthalten zwei benachbarte Segmente die Information „Lücke“, werden sie zu einem Segment zusammengefasst. Ein neuer Prozess wird durch eines der Verfahren First/Next/Best/Worst-Fit in die Liste aufgenommen. Häufig wird eine „double linked List“ verwendet, wo die Segmente nicht nur auf das nachfolgende sondern auch auf das vorhergehende Segment verweisen, was die Freispeicher-Suche erleichtert.

-Buddy System Algorithmus (wird weiter hinten erklärt)

**Verschnitt / Fragmentierung**

-externer Verschnitt:

Je nach Speicherverwaltungsverfahren liegen die Prozesse als einzelne Stücke mehr oder weniger voneinander getrennt. Die freien Speicherblöcke zwischen den Prozessen liegen also nicht am Stück vor, sondern sind über den gesamten Speicher verteilt (externer Verschnitt). Soll nun ein neuer Prozess geladen werden, für den zwar genügen freier Speicher zur Verfügung steht, allerdings nicht am Stück, so wird er in Teilstücke zerlegt und über den Speicher verteilt; er ist fragmentiert.

-interner Verschnitt:

Teilt man den Speicher in Blöcke fester Größe auf und weist den Speicher nur blockweise zu, entsteht kein externer Verschnitt. Allerdings wird einem Prozess unter Umständen mehr Speicher zugewiesen, als er benötigt. Der ungenutzte Speicher ist der interne Verschnitt.

**Der Buddy System Algorithmus unter Linux**

Idee

Der Kernel muss dafür sorgen, dass die Daten im physikalischen Speicher nicht zu stark fragmentiert werden.

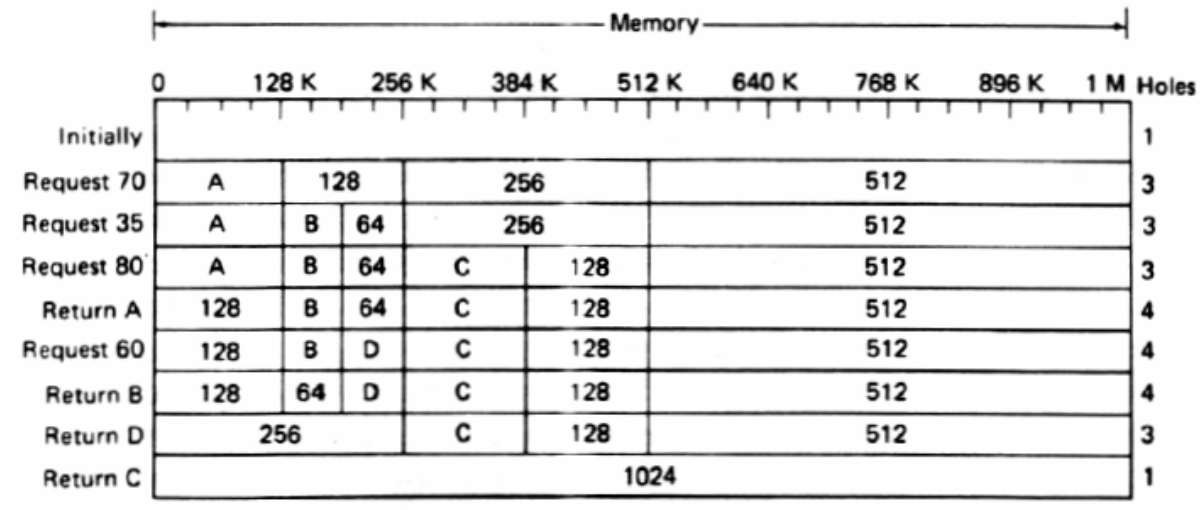
Zwei Möglichkeiten:

- -Gruppen von fragmentierten freien Seitenkacheln werden auf kontinuierliche lineare Adressen abgebildet
- -Entwickeln einer Technik, die das Fragmentieren schon beim Schreiben der Daten verhindert

Die zweite Möglichkeit findet Anwendung, weil es oft nicht ausreicht, dass nur die Adressen linear sind, sondern es müssen auch die Seitenkacheln am Stück vorliegen (z.B. beim DMA-Transfer: Seitenkacheln werden direkt adressiert). Außerdem bietet diese Möglichkeit einen Geschwindigkeitsvorteil, wenn z.B. Daten auf einen externen Speicher (Festplatte) ausgelagert sind.

Der Algorithmus

Der freie Speicher wird in Speicherbereiche aufgeteilt, die in Listen verwaltet werden. Die Größe der Bereiche in den Listen entspricht dabei einem Vielfachen von 2. Die Listen sind dabei größenmäßig von oben nach unten angeordnet: die erste zeigt auf den gesamten freien Speicher, die zweite auf die zwei Hälften, die dritte auf die vier Viertel usw. Ist der freie Speicher beispielsweise 1 MB groß, so wird dieser in 9 Listen verwaltet, wobei die unterste Liste die 4 KByte großen Seitenkacheln des freien Speichers verwaltet. Zu Beginn enthält die oberste Liste nur einen Eintrag auf den 1024 KBytes großen Speicherbereich, alle anderen Listen sind leer.




Wird nun ein 70 KB großer Prozess geladen, wird der Speicher zuerst in zwei 512 KB-Blöcke, dann der erste Block in zwei 256 KB-Blöcke und der erste davon wieder in zwei 128 KB-Blöcke unterteilt. Der 70 KB-Prozess wird dem ersten 128 KB-Block zugewiesen. Wird nun als nächstes ein 35 KB großer Prozess geladen, wird der noch freie 128 KB-Block in zwei

64 KB-Blöcke unterteilt und der Prozess in den ersten eingelagert. Für den nächsten 80 KB großen Prozess wird der zweite 256 KB-Block in zwei 128 KB-Blöcke unterteilt usw. Werden nun im Bild die Prozesse A, B und D wieder aus dem Speicher entfernt, können die freien Blöcke (=Buddies) teilweise wieder zusammengefasst werden.

Der Vorteil dieses Systems ist die schnelle Zusammenfassung von freien Speicherblöcken, da nur die entsprechenden Listen durchsucht werden müssen. Wird z.B. ein 32 KB-Block frei, muss nur die 32 KB-Liste nach freien Blöcken durchsucht werden, damit festgestellt werden kann, ob eine Zusammenfassung möglich ist.
Die Nachteile des Buddy-Algorithmus sind, dass er bei kleinen Datenmengen ineffizient ist und dass extremer interner Verschnitt entsteht, z.B. wenn ein 70 KB-Prozess zugewiesen wird: im 128 KB-Block bleiben 58 KB ungenutzt. Es gibt deshalb weitere Techniken, um die Verwendung des Buddy-Algorithmus effizienter zu machen bzw. ihn weniger einsetzen zu müssen.

**Slab Allocator**

Speicherbereiche werden bei diesem Verfahren als Objekte mit Methoden und Funktionen betrachtet. Die Objekte werden in Caches zusammengefasst, die ihrerseits in Slabs (= zusammenhängende Speicherbereiche) aufgeteilt werden. Der Slab Allocator speichert häufig benötigte Speicherkacheln, damit der Kernel sie nicht ständig neu initialisieren muss. Es kann nach Häufigkeit der Anfragen für eine bestimmte Speichergröße unterschieden werden; wenn erwartet wird, dass ein bestimmter Speicherbereich mehrfach angefordert wird, werden schon vorher Objekte der bestimmten Größe erzeugt. Dadurch werden Aufrufe des Buddy-Algorithmus verringert.

**Caches**

Caches werden für die Zwischenspeicherung von Prozessen benutzt, um einen schnelleren Zugriff darauf zu ermöglichen. Es muss dabei unterschieden werden zwischen

-generelle Caches: werden nur vom Slab Allocator benutzt. Der erste Cache enthält dabei die Beschreibung für die restlichen Caches
-spezifische Caches: werden vom Rest des Kernels benutzt

Wird ein Cache neu erzeugt, enthält er noch keine Slabs / Objekte, neue Slabs werden erst initialisiert, wenn eine Anfrage auf ein neues Objekt besteht und der Cache noch kein freies Objekt enthält.

**Slab Coloring**

Beim Cachen von Prozessen wird immer ein Cache-großer Speicherbereich in den Cache geladen. Ist der Prozess aber über zwei (oder mehr) Bereiche im physikalischen Speicher verteilt, kann es passieren, dass in den Cache immer wieder abwechselnd zwei verschiedene Speicherbereiche geladen werden müssen. Deswegen verwendet man „Farben“, um die Slabs zu kennzeichnen, zusammenhängende Prozesse in einen Slab einer Farbe zu speichern und damit zu vermeiden, dass z.B. ein Prozess zweimal zu Beginn eines neuen Slabs gespeichert wird. Das Einfärben geschieht durch unterschiedliches Verschieben von freiem Speicher vom Ende an den Anfang der Slabs und funktioniert nur bei genügend freiem Speicher innerhalb der Slabs.

**Quellen:**

http://www.sec.informatik.tu-darmstadt.de/de/lehre/WS01-02/bs1/folien/kapitel6.pdf (28.10.2003, Prof. C. Eckert)

http://www.iam.unibe.ch/~rvs/lectures/SS02/bs/arbeiten/Kai_Rolle-Hauptspeicherverwaltung.pdf (28.10.2003, Kai Rollé)

O'Reilly - Understanding the Linux Kernel (Daniel Bovet, Marco Cesati)