Kapitel 11: Speicherverwaltung

- Motivation
- Speicherverwaltungsmodul
- Entwurfsparameter
- Konkrete Speicherverwaltungsverfahren
  - Ringpuffer
  - Stapel
  - Randkennzeichnungsverfahren
  - Halbierungsverfahren
  - Linux Slab Allocator

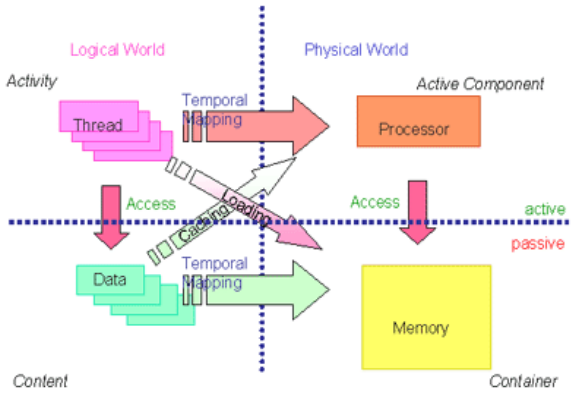
11.1 Motivation und Einführung

Jede Anwendung (*application*), aber auch jede als Dienstgeber (*server*) realisierte Systemaufgabe benötigt zu ihrer Abarbeitung Teile des Hauptspeichers zur Ablage des Codes und der Daten. Außerdem muss auch der Systemkern (u.U. in von der Rechnerarchitektur erzwungenen) Hauptspeicherabschnitten realisiert werden. Somit enthält jedes System zumindest eine Speicherverwaltung; diese kann je nach Ausstattung sehr einfach gehalten werden, sie kann aber auch sehr vielschichtig und sehr komplex werden. Manchmal wird argumentiert, dass bei den heute am Markt angebotenen Speicherbausteinen und dem virtuellen Speicherprinzip das Thema Speicherverwaltung nur noch bedingt interessant sei; dem ist natürlich nicht so. Egal wie groß ein Hauptspeicher ausgebaut werden kann, zumindest eine Systeminstanz muss darüber informiert sein, welche Speicherbereiche aktuell frei und welche belegt sind. Beim realen Speicherprinzip ist dies ganz offenkundig dann von Interesse, wenn mal wieder ein neuer Prozess oder eine neue Task in das System geladen werden soll. Sofern dies nicht ausdrücklich mittels des Konzepts “Shared Memory” gewünscht wird, sollen zwei verschiedene Tasks nicht in die gleichen Speicherbereiche geladen werden. Beim virtuellen Speicherprinzip müssen der oder die Seitentauscher (*pager*) darauf achten, dass sie nur auf ausdrücklichen Wunsch Seiten (pages) verschiedener logischer Adressräume auf die gleichen physischen Speicherkacheln (*frames*) abbilden. In dynamischen Systemen muss der für die Kerndatenstrukturen wie (PCB, TCB, Adressraumtabelle, Seitentabelle, etc.) benötigte Platz verwaltet und kontrolliert werden

Frage: Wer oder was ist im Rahmen einer Anwendung (application) für deren internen Speicherbedarf zuständig?

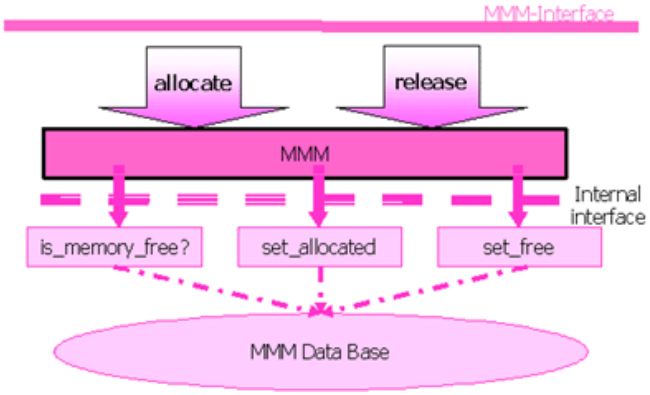
Ein magisches Zauberwort moderner Speicherverwaltungen heißt "Speicherhierarchie"; hierunter versteht man den Aufbau eines Rechners mit verschiedenen Speichertechnologien, wobei die prozessornahen Caches weniger Kapazität, dafür aber schnellere Zugriffszeiten haben, während der in Relation zur CPU relativ langsame Hauptspeicher quasi die letzte Stufe eines zwar noch flüchtigen aber zumindest bei den SMP-Systemen gemeinsam noch direkt zugreifbaren Speichermediums darstellt. Die nächsten Speichermedienschichten dienen dann der persistenten Speicherung (heute meist auf Platten), der tragbaren nicht flüchtigen Speichermedien (heute meist CDs) sowie der Backupmedien für Archivzwecke.

Die Rolle einer Speicherverwaltung kann im Zusammenspiel mit den bisher behandelten diversen Aktivitätsformen wie folgt dargestellt werden:



11.2 Speicherverwaltungsmodul

Hauptaufgabe einer Speicherverwaltung ist somit das Bereitstellen von freien Speicherplätzen, wenn eine Anwendung dies fordert, ferner das Zurücknehmen von nicht mehr benötigtem Speicherplatz und somit auch die Integration freier Stücke in eine Freispeicherliste. U.U. ist es auch eine weitere Aufgabe der Speicherverwaltung dafür zu sorgen, dass bei einer Speicherzerstückelung eine Art Speicherbereinigung durchgeführt wird, d.h. zum einen werden dabei benachbarte kleinere freie Stücke zu einem größeren freien Stück wiedervereinigt (*garbage collection*), und zum anderen könnte man bei Speicherknappheit, alle belegten Stücke zusammenschieben, um so doch noch zu einem möglichst großen freien Speicherstück zu gelangen (*storage compaction*).



Frage: Welche Vorteile kann in der obigen Architektur einer abstrakten Speicherverwaltung entdeckt werden?

11.3 Orthogonale Entwurfsparameter der Speicherverwaltung

Man kann eine Reihe von orthogonalen Entwurfsparametern erkennen, die bei geschickter Auswahl unter gegebenen Randbedingungen, die Effektivität und die Effizienz eines Systems erheblich steigern können. Zu diesen gehören u.a.:

- Reihenfolge von Belege/Freigabeoperationen (allocate/release)
  - FIFO, LIFO, beliebig
- Größe der Speicherblöcke
  - Gleichgroße Blöcke, Blöcke fixer Größe, Speicherblöcke als Zweierpotenzen, beliebige Speicherblockgrößen
- Speicherverwaltungsdatenstrukturen
  - Integrierte bzw. Abgesetzte Datenstruktur
- Verschnitt
  - Mit internen bzw. externem Verschnitt
- Belegungsstrategie
  - First-, Next-, Best-, Worst-, ..., Nearest-Fit
- Wiedervereinigung frei werdender Speicherblöcke
  - Unmittelbare und faule Wiedervereinigung

11.3.1 Reihenfolge von Anforderungs- und Freigabeoperationen

Wenn keine Besonderheiten zu verzeichnen sind, wird jede allgemeine Speicherverwaltung i.d.R. komplexer ausfallen als Spezialvarianten, die sich in ihrer Implementierung auf Besonderheiten bei der Operationenreihenfolge verlassen können. Typisches Beispiel für eine einfache Spezialspeicherverwaltung ist der Stapel (*stack*), bei welchem stets das zuletzt belegte Stück auch zuerst wieder freigegeben wird (LIFO).

Überlegen Sie sich Anwendungsfälle für den Spezialfall: FIFO.

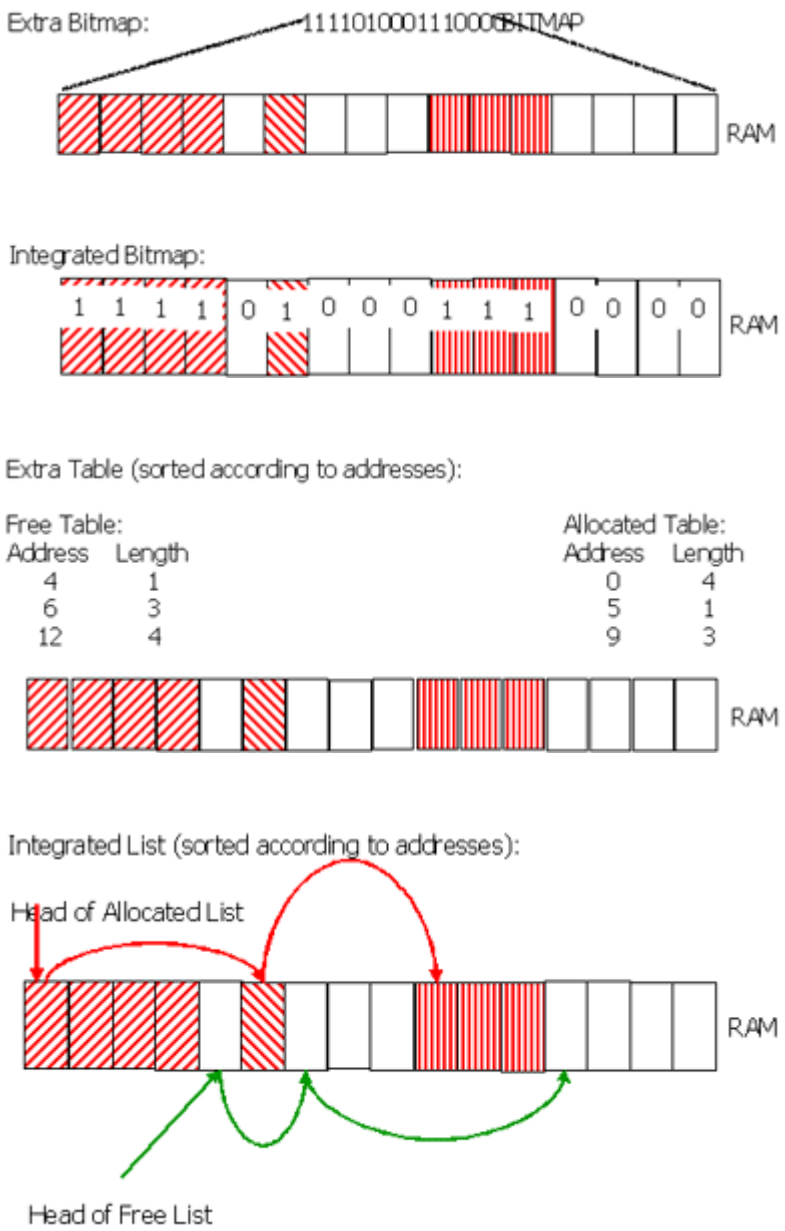
11.3.2 Speicheranforderungsgrößen

Erarbeiten Sie sich selbst entsprechende Verfahren und deren typische Anwendungen.

11.3.3 Speicherverwaltungsdatenstrukturen

Prinzipiell unterscheidet man zwei verschiedene Datenstrukturvarianten:

- Bitmap oder
- Tabelle/Liste



11.3.4 Externer und Interner Verschnitt (fragmentation)

Von externem Verschnitt spricht man dann, wenn zwar in der Summe genügend freier Speicherplatz vorhanden wäre, aber leider nicht in zusammenhängender Form, so dass die aktuelle Anforderung nicht erfüllt werden kann. Von internem Verschnitt spricht man dann, wenn bei der Vergabe von Speicherplatz auf einen passenden größeren Block aufgerundet wird, da diese Blockgröße leichter zu verwalten ist. Man

beachte, dass der zuviel belegte Speicher nicht verwendet wird, aus der Sicht der Speicherverwaltung ist er belegt, aus der Sicht der Anwendung ist es schlichtweg Verschwendung.

11.3.5 Belegungsstrategien

- First-fit
- Next-fit
- Best-fit
- Nearest-fit

Analysieren Sie deren Vor- und Nachteile und studieren Sie kritisch die in Lehrbüchern getroffenen Aussagen.

11.3.6 Wiedervereinigungsstrategien

Zum einen kann man bei jeder Zurückgabe von Speicher überprüfen, ob das neue frei werdende Speicherstück auch ein oder sogar zwei freie benachbarte Stücke besitzt, die dann in einem Aufwasch zu einem größeren freien Stück verschmolzen werden können.

Alternativ hierzu könnte man zunächst jedes freiwerdende Stück in die Freiliste aufnehmen und erst dann, wenn Not am Mann ist (sprich kein genügend großes Speicherstück mehr vorhanden ist) im Nachhinein versuchen, benachbarte freie Stücke zu größeren freien Stücken zu verschmelzen.

Insbesondere bei der obigen "faulen" Wiedervereinigung kann es notwendig werden, den Speicher zu bereinigen (*garbage collection*), womit nur der zerstückelte freie Speicher gemeint ist. Sofern diese Maßnahme nicht ausreichend ist, kann man auch dazu übergehen, alle belegten Stücke an den Anfang (oder ans Ende) zu verschieben, um so eine bislang nicht erfüllbare Speicheranforderung doch noch befriedigen zu können.

11.4 Konkrete Speicherverwaltungsverfahren

Es wird nützlich sein, verschiedene spezielle Speicherverwaltungsverfahren zu studieren, um so deren Besonderheiten und insbesondere deren Stärken einschätzen zu können.

11.4.1 Ringpuffer und Stapel

Sie sollten die Besonderheiten beider Verfahren schon hinlänglich kennen.

11.4.2 Randkennzeichnungsverfahren (Boundary Tag)

Das Randkennzeichnungsverfahren zeichnet sich dadurch aus, dass es beliebig große Speicherstücke verwalten kann. In den verwalteten Stücken selbst sind die Verwaltungsdaten untergebracht.

Die Entwurfsparameter sind wie folgt ausgelegt:

- Beliebige Reihenfolge von Anforderungs- und Freigabeoperationen
- Beliebige Anforderungsgrößen
- Integrierte Speicherverwaltungsdatenstrukturen
- Belegungsstrategien nach xyz-Fit (Best-fit ist möglich)
- Externer Verschnitt
- I.d.R. sofortige Wiedervereinigung

11.4.3 Halbierungsverfahren (Buddy-System)

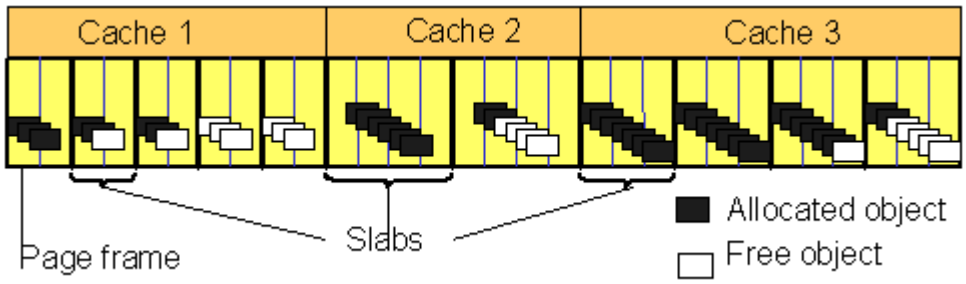
Das Charakteristische am Buddysystem ist es, dass nur Blockgrößen in Zweierpotenzen angefordert werden können. Die Wiedervereinigung von freigewordenen Stücken mit ihren Buddies kann sofort oder auch verzögert stattfinden. Die Anfangsadresse eines Buddy kann leicht errechnet werden.

Die Entwurfsparameter sind wie folgt ausgelegt:

- Beliebige Reihenfolge der Anforderungs- und Freigabeoperationen
- Anforderungsgrößen nur 2^0, 2^1, ...
- Explizite Speicherverwaltungsdatenstrukturen
- Belegungsstrategien nach Best-Fit
- Interner und externer Verschnitt
- I.d.R. sofortige Wiedervereinigung

11.4.4 Slab-Allocation

Das in Linux verwendete Speicherverwaltungsverfahren Slab-Allocator geht davon aus, dass es in einem System (insbesondere im Kern) dynamische Objekte mit wohl definierten Größen gibt. Wenn somit eines dieser dynamischen Objekte (z.B. ein TCB) nicht mehr benötigt wird, dann kann man mit hoher Wahrscheinlichkeit davon ausgehen, dass demnächst wieder ein solches Objekt benötigt wird. Statt den TCB an eine allgemeine Speicherverwaltung als neues freies Stück zurückzugeben, wird das freiwerdende Stück neu initialisiert der speziellen TCB-Verwaltung zurück gegeben. Pro solchem Objekttyp richtet man demzufolge einen Pufferspeicher (*cache*) ein, der aus mehreren Slabs bestehen kann.



Zur Einrichtung eines Caches benötigt man selbstverständlich Kernadressraumspeicher, den man sich über das Buddysystem beschafft. Slab-Allocating wurde von Jeff Bonwick im Rahmen der Weiterentwicklung von Solaris erfunden (siehe: J. Bonwick: "The Slab Allocator: An Object-Caching Kernel Memmory Allocator", USENIX Summer 1994)

Zur Vertiefung wird empfohlen, die folgende Literaturstelle zu studieren, in der Vorschläge für eine bessere Implementierung des Slab-Allocators auf SMP-Systemen vorgestellt wird. Jeff Bonwick, Jonathan Adams: "Magazines and Vmem: Extending the Slab Allocator to Many CPUs and Arbitrary Resources", USENIX Annual Technical Conference, General Track 2001