

Speicherverwaltung

Linux Kernel 2.6

Marcus Obst

Technische Universität Chemnitz

29.05.2006 / Proseminar Linux Internals

Gliederung

1 Einleitung

2 Speicherverwaltung

- Verwaltung von zusammenhängendem Speicher
- Verwalten von Speicherbereichen
- Nichtzusammenhängender Speicher

3 Zusammenfassung

Dynamischer Speicher - ein Überblick

Wer braucht den?

- Kernel (`kmalloc()`, `vmalloc()`), `alloc_pages()`
- Prozesse (`malloc()`, `new()`)

Anforderungen:

- schnell, sicher & zuverlässig
- wenig Verschnitt (Fragmentierung!)
- Hardwareanforderungen berücksichtigen

Ziel

Die Bereitstellung (und Vernichtung) von dynamischem Speicher bestimmt die Performance eines Systems.

Der Page Descriptor

Seitengröße (`PAGE_SIZE`): 4 kb oder 4 MB

- jede Seite wird mit Hilfe des `page descriptor` beschrieben
 - ▶ Wer benutzt die Seite (Kernel/Prozess)?
 - ▶ Ist die Seite frei?
 - ▶ 32 Byte groß
- `mem_map[]` Array hält alle Page Descriptoren.

Speicherbereiche (Memory Zones)

Annahme: 80x86 mit 32-Bit

- `ZONE_DMA` → unterhalb von 16 MB
- `ZONE_NORMAL` → von 16 MB bis 896 MB
- `ZONE_HIGHMEM` → ab 896 MB

Verwendung

Wofür brauche ich den Speicher?



Reservierte Seiten (Reserved Page Frame Pool)

Problem

- Speicherbeschaffung kann prinzipiell schief gehen!
- Normalfall: blockiert solange, bis Speicher verfügbar
- für Interrupts nicht akzeptabel

Lösung

Atomische Speicherbeschaffung aus einem extra Pool. Gibt entweder sofort den `Page Descriptor` oder `NULL` zurück

Zoned Page Frame Allocator

- `alloc_pages(gpf_mask, order)`
- `__get_free_pages(gpf_mask, order)`
- `free_pages(addr, order)`

ausgewählte `gpf_mask`-Flags

<code>_GPF_DMA</code>	soll in <code>ZONE_DMA</code> liegen
<code>_GPF_HIGHMEM</code>	soll in <code>ZONE_HIGHMEM</code> liegen
<code>_GPF_HIGH</code>	reservierte Seiten können genutzt werden
<code>_GPF_REPEAT</code>	versuche unendlich
<code>_GPF_ZERO</code>	ausnullen
<code>_GPF_WAIT</code>	schlafen legen

`_GPF_DMA` und `_GPF_HIGHMEM` sind Zone Modifier.

Speicherseiten in der Zone `ZONE_HIGHMEM`

Problem

- Speicher jenseits von 896 MB hat keine logische Adresse (Adressraum zu klein).
- `__get_free_pages(__GPF_HIGHMEM, ...)` kann keine gültige Adresse liefern und gibt deswegen `NULL` zurück.

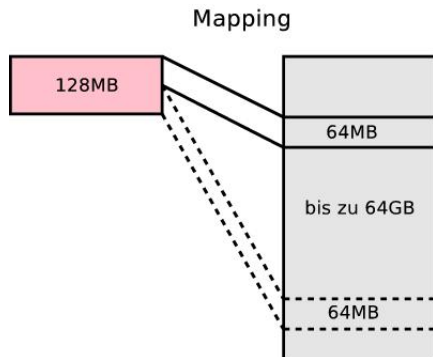
Lösung

Abhilfe auf 80x86 durch `alloc_pages()`.

Es steht ein „Fenster“ von 128 MB zur Verfügung um in den restlichen Speicher zu blicken (page mapping).

Arten von Page Mapping

- permanent
 - ▶ aufwändige Implementierung
 - ▶ kann Blocken (nicht für Interrupts geeignet!)
- temporär
 - ▶ pro CPU stehen 13 zur Verfügung (für Komponenten)
 - ▶ nur exklusiver Zugriff
- für nichtzusammenhängenden Speicher



HIGHMEM ist teuer

Das Einblenden von HIGHMEM in den 128 MB großen Adressraum des Kernels kostet Zeit.

Linux Kernel v2.6.16.14 Configuration

Memory split

Use the arrow keys to navigate this window or press the hotkey of the item you wish to select followed by the <SPACE BAR>. Press <?> for additional information about this option.

- 1G/1G user/kernel split
- 3G/1G user/kernel split (for full 1G low memory)
- 1G/2G user/kernel split
- 1G/3G user/kernel split

<Select>

< Help >

Das Buddy-System

Was wir bisher haben

Basis: eine Seite (Page)

Voraussetzung: zusammenhängender Speicher

Problem: externe Fragmentierung

Wie uns das Buddy-System weiterhilft

Alle freien Seiten (eigentlich Bereiche!) werden einer von 11 Gruppen (1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024) zugeordnet.

Die Zahl gibt die Anzahl von zusammenhängenden Seiten an (→ $1024 * \text{PAGE_SIZE (4 kb)} = 4 \text{ MB}$).

Das Buddy-System am Beispiel

Aufgabe: Beschaffung von 256 zusammenhängenden Seiten

- 1 System schaut in der 256-Gruppe nach freien Seiten → nein
- 2 System schaut in der 512-Gruppe nach freien Seiten → ja
 - 1 Aufteilen des einen 512-Bereiches in 2 256-Bereiche
 - 2 die ersten 256 werden als markiert gekennzeichnet, die Anforderung ist geglückt
 - 3 die verbleibenden 256 Seiten werden in die 256-Gruppe eingetragen

Das Buddy-System am Beispiel (Fortsetzung)

- ❶ System schaut in der 512-Gruppe nach freien Seiten → nein
 - ❶ System schaut in die 1024-Gruppe → ja
 - ❷ die ersten 256 Seiten erfüllen die Speicheranforderung
 - ❸ von den restlichen 768 Seiten werden 512 in die 512-Gruppe und 256 in die 256-Gruppe eingefügt

Geht auch das schief, wird `NULL` zurückgegeben.

Per-CPU Page Frame Cache

- in der Praxis werden oft einzelne Seiten angefordert
- Zwischenspeichern/Vorhalten sinnvoll
- zwei Caches pro Zone und CPU
 - ▶ hot cache (CPU Operationen)
 - ▶ cold cache (DMA)

Eigenschaften des Zone Allocator

Ist die direkte Schnittstelle im Kernel zu Speicheranforderung.

- reservierte Speicherseiten beachten
- Bereitstellung auch bei knappem Speicher ermöglichen (Swap!)
- `ZONE_DMA` möglichst unberührt lassen

Schwächen des Buddy-Allocators

- Buddy-Allocator hat `PAGE_SIZE` als kleinste Einheit
- Was ist wenn wir nur ein paar Byte benötigen? → interne Fragmentierung

Klassische Lösung: Speicher Seite mit Hilfe einer Geometrischen Reihe aufteilen → Fragmentierung < 50%.

Der Slab-Allocator

- Speicherbereiche werden als Objekte gesehen (Datenstrukturen + Funktionen)
- freigegebene Objekte werden nicht sofort verworfen (Cache)
- Anforderungen von Speicherbereichen werden klassifiziert
- für oft angeforderte Speicherbereiche werden zur Laufzeit spezielle Objekte angelegt
- Gerätetreiber können ihre eigenen Slab Caches anfordern
- `# cat /proc/slabinfo`

Beispiel: Öffnen einer Datei

- `open-file`-Objekt wird benötigt (immer feste Größe!)
- Slab Cache: `filp`

Memory Pools

- neu in Kernel 2.6
- erlaubt Komponenten (Gerätetreiber) eigene Reserven zu beantragen
- werden im Falle von knappem Speicher genutzt

Nichtzusammenhängender Speicher (vmalloc())

Bisher nur zusammenhängenden Speicher:

- von Hardware gefordert (DMA)
- Geschwindigkeit (Cache)
- relativ unkomplizierter Zugriff (nur Offset)

jetzt auch nichtzusammenhängender Speicher:

- vermeiden von externe Fragmentierung
- für Software geeignet (I/O-Buffer)
- Mehraufwand durch Umsetzen der `Kernel Page Table`

Zusammenfassung

Haben wir die Anforderungen erfüllt?

- schnell (per-CPU page frame cache, slab caches)
- sicher (Reserved Page Frame Pool)
- wenig Verschnitt (Buddy- und Slab-Allocator, Nichtzusammenhängender Speicher)
- Hardwareanforderungen (Memory Zones)

- *Linux Kernel Source 2.6.16*
- *Understanding the LINUX Kernel*, BOVET & CESATI, O'Reilly 2005
- *Linux Gerätetreiber*, RUBINI & COBERT, O'Reilly 2002
- *Understanding The Linux Virtual Memory Manager*, MEL GORMAN, 2004, <http://www.csn.ul.ie/~mel/projects/vm/guide/pdf/understand.pdf>