

Linux Memory Management

Johannes Weiner hannes@cmpxchg.org

March 13, 2010

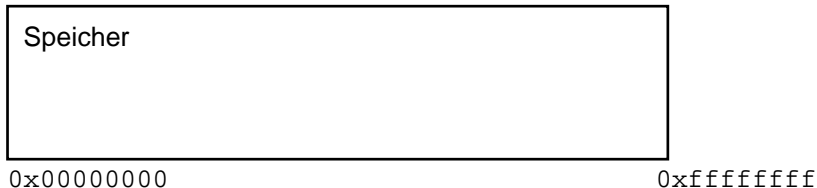
Virtueller Speicher

Linux VM

Page cache

Page reclaim

Speicher



- ▶ Sequenz von Speicherzellen
- ▶ byteweise adressierbar

Multitasking?

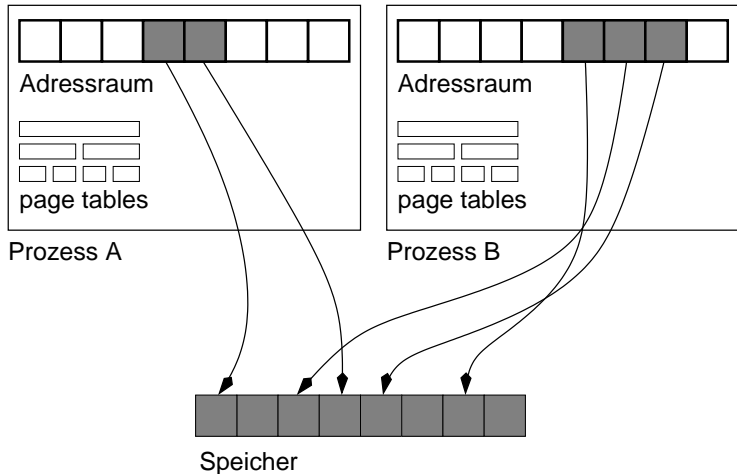
Segmentierung

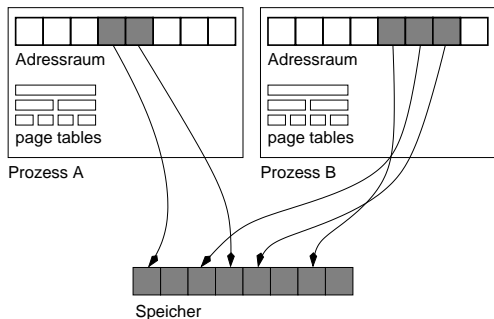


- ▶ Prozesse verwenden Segmente des Speichers
- ▶ Zugriffsprüfung durch hardware
- ▶ einfachste Form des Speicherschutzes

Skalierbarkeit?

Virtueller Speicher



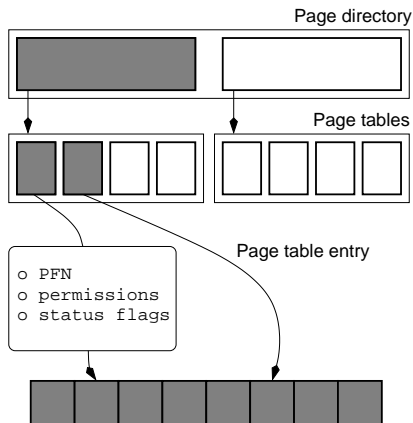


Indirektion zwischen Adressen und Speicher

- ▶ Einteilung von Adressen und Speicher in *pages*
- ▶ pro Prozess page tables zur beliebigen Zuweisung
- ▶ Adressen nun nichtmehr global sondern prozessweit

- ▶ Speicherschutz perfekt: keine Sichtbarkeit
- ▶ Speicherfragmentierung wird versteckt
- ▶ Adressraumfragmentierung
 - ▶ unabhängig von der Anzahl der Prozesse
 - ▶ maximale Adressraumgröße

Page tables



- ▶ mehrstufige Baumstruktur
- ▶ Partitionierung des Raumes auf jeder Ebene
- ▶ Auslassung von ungebrauchten Baumhierarchien

Page fault Mechanismus

Fehler beim page table lookup, CPU übergibt Kontrolle an den kernel

- ▶ page table entry nicht vorhanden
- ▶ page table entry verbietet Zugriff

Adressraum



- ▶ jeder Adressraum hat den Kernel gemappt
- ▶ prozesseigene mappings im userteil
 - ▶ `exec()`
 - ▶ `brk()`
 - ▶ `mmap()`
- ▶ Hint: `cat /proc/self/maps`

Arten von mappings

`MAP_PRIVATE` Dateisystem, private Kopie

- ▶ .data section einer Bibliothek

`MAP_SHARED` Dateisystem

- ▶ alternativ zu `read()`, `write()`

`MAP_ANONYMOUS` | `MAP_PRIVATE` kein filebacking, private Kopie

- ▶ stack oder heap Speicher

Datenstrukturen: struct page

```
struct page *page
```

```
include/linux/mm_types.h
```

- ▶ beschreibt einen physikalischen page frame (`mem_map`)
- ▶ zustandsflags `include/linux/page_flags.h`
- ▶ Felder je nach Verwendung und für reclaim

Datenstrukturen: struct vm_area_struct

```
struct vm_area_struct *vma  
include/linux/mm_types.h
```

- ▶ beschreibt ein konfiguriertes virtuelles mapping
- ▶ Art des mappings
- ▶ Position und Größe im Adressraum
- ▶ VM_ flags aus include/linux/mm.h

Datenstrukturen: struct mm_struct

```
struct mm_struct *mm
```

```
include/linux/mm_types.h
```

- ▶ beschreibt den Adressraum eines Prozesses
- ▶ Organisation von vmas
- ▶ page tables

Page fault: Applikationsfehler

- ▶ Schreiben trotz Schreibschutz
- ▶ Zugriff außerhalb eines gemappten Bereichs

Prozess wird kontrolliert mit einem SIGSEGV Signal getötet:
„Segmentation fault“

z.B. `arch/x86/mm/fault.c::do_page_fault()`

Copy on write (COW) Optimierung

- ▶ MAP_PRIVATE file mapping
- ▶ MAP_PRIVATE mapping + fork()

statt echter Kopie

- ▶ vma erlaubt Schreibzugriff
- ▶ page table entry nicht
- ▶ page fault handler erstellt Kopie

`mm/memory.c::do_wp_page()`

Demand paging

- ▶ `mmap()` konfiguriert nur eine vma
- ▶ page fault handler beschafft einen page frame und setzt das page table entry
- ▶ meist werden nicht alle pages in einem mapping verwendet
- ▶ pages können im reclaim geklaut werden

Demand anonymous paging

- ▶ Reservierung eines page frames
- ▶ Beschreiben mit Nullen
- ▶ Setzen des page table entries

```
mm/memory.c::do_anonymous_page()
```

Page cache

- ▶ caches bei Übergängen in der Speicherhierarchie
- ▶ page cache speichert das Dateisystem im Hauptspeicher zwischen

Page cache

- ▶ Aufteilung einer Datei in pages
 - ▶ Organisation der pages in Baumstruktur wie page tables
1. virtuelle Adresse → vma
 2. vma->vm_file->f_mapping → page cache Deskriptor der gemappten Datei
 3. Ermittlung des page cache Index anhand von Adresse und vma

`mm/memory.c::do_linear_fault()`

Page cache

minor fault page existiert im cache

1. Setzen des page table entries

major fault page existiert nicht im cache

1. Reservierung eines neuen page frames
2. Anweisung zum Einlesen an Dateisystem
3. Hinzufügen zum cache
4. Setzen des page table entries

```
mm/filemap.c::filemap_fault()
```

- ▶ Hint: `grep fault /proc/vmstat`

Page reclaim

- ▶ aggressive page caching
- ▶ Schrumpfen des page caches bei Speicherknappheit
- ▶ Klauen von gemappten pages bei erhöhtem Druck

LRU Listen

- ▶ Listen aller im userspace verwendeten pages
- ▶ inaktive und aktive Liste
- ▶ Suchen von reclaim Kandidaten auf inaktiver Liste
- ▶ `read()`, `write()` verschieben pages synchron auf den Listen
- ▶ gemappte pages?

rmap

- ▶ CPU (oder kreative Verwendung des page fault handlers) notiert Zugriffe in den page table entries (ACCESSED status bit)
- ▶ rmap: reverse mapping, findet effizient alle page tables, die einen page frame referenzieren

file rmap

- ▶ `page->mapping` zeigt auf den page cache Deskriptor der gemappten Datei (= `vma->vm_file->f_mapping`)
- ▶ page cache Deskriptor verwaltet einen Baum mit allen `vmas`, die die Datei gemappt haben
- ▶ `page->index` ist Index der page im page cache

```
mm/rmap.c::page_referenced_file()
```

Anon rmap

- ▶ COW mappings können sich anonyme pages teilen
- ▶ `fork()` merkt sich 'verwandte' vmas in einer Liste
- ▶ `vma->anon_vma`
- ▶ `page->mapping`
- ▶ `page->index` ist page index im Adressraum

`mm/rmap.c::page_referenced_anon()`

Reclaim gemappter pages

- ▶ verschieben ans Ende der inaktiven Liste
- ▶ verschieben auf aktive Liste
- ▶ page frame wiederverwenden

vm_end

Vielen Dank! Btw,

- ▶ „Understanding the Linux Kernel“
- ▶ the source
- ▶ #mm auf irc.kernelnewbies.org