Universität Bremen

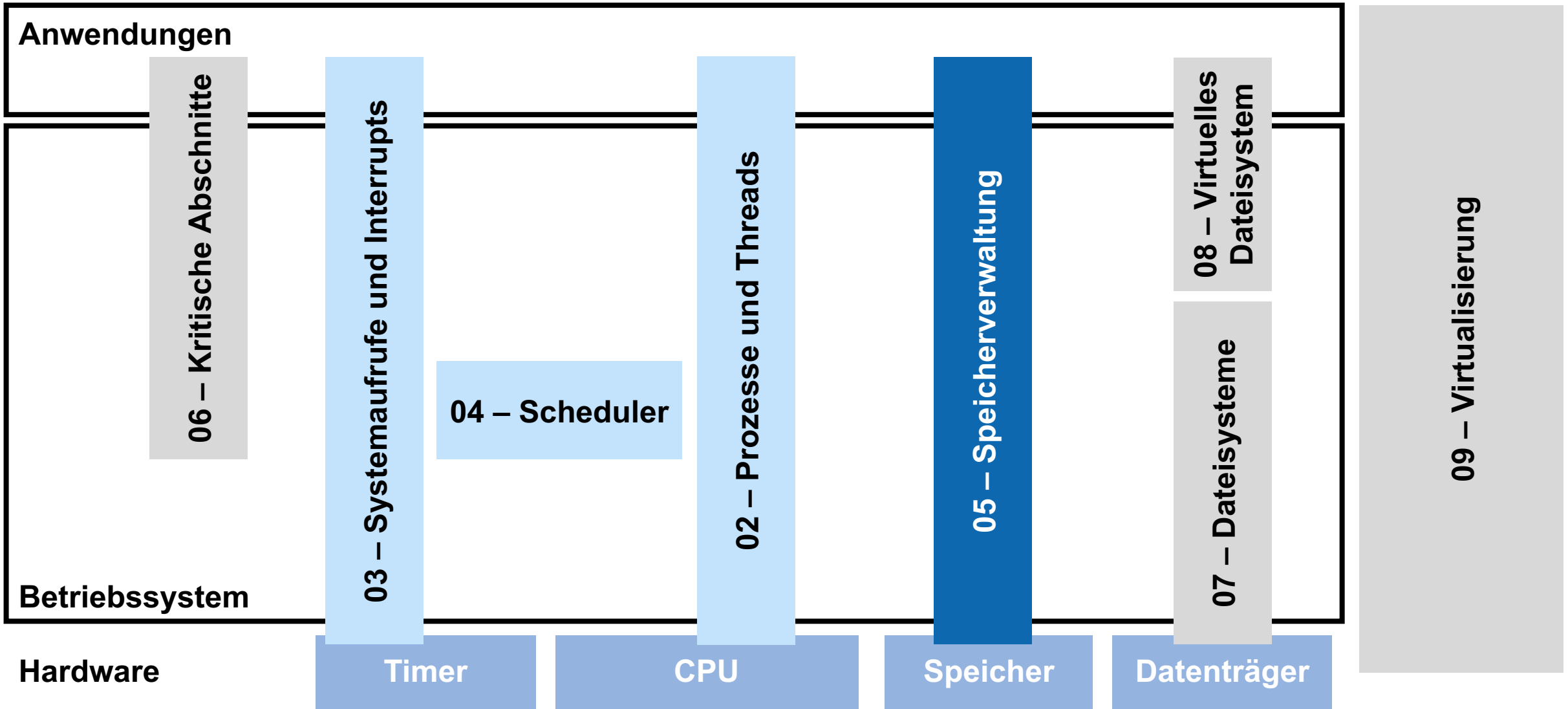
Betriebssysteme

Speicherverwaltung

Dr. Bernhard J. Berger
AG Sichere Systeme

Universität Bremen

Themenübersicht



Universität Bremen

Themenübersicht

- Motivation
- Virtuell. vs physisch
- Paging
- Swapping

Grundlagen

- Datenstrukturen
- Mechanismen
  - Demand Paging
  - Copy-on-Write
  - Memory Mapping

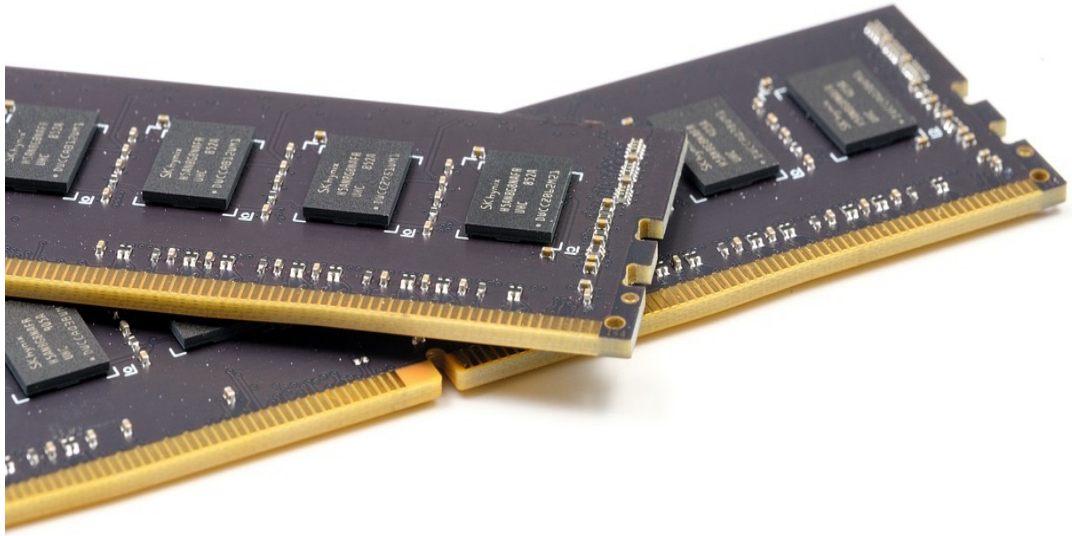
Implementierung

- Huge Tables
- ASLR
- OOM-Killer

Weiterführendes

Motivation

- Arbeitsspeicher ist Ressource des Rechners
- Anwendungen konkurrieren um Ressource
- Daten entscheiden in Von-Neumann-Rechner
- Aufgaben des Betriebssystems
  - Zuteilung der Ressource Arbeitsspeicher
  - Schutz des Speichers des Betriebssystems
  - Schutz der Anwendungen voreinander
  - Effiziente Verwaltung
- Potenzielle Probleme
  - Speicher ist vollständig belegt
  - Unerlaubter Speicherzugriff
  - Schnelles Wachsen und Schrumpfen des Bedarfs



Physischer und virtueller Speicher

Physischer Speicher

- Adressraum des Arbeitsspeichers
- Endliche Ressource
  - Adressraum (32 Bit, 64 Bit)
  - Verbauter RAM
- CPU hat direkten Zugriff
  - Direkte Adressierung
  - Keine Abstraktion
- Zugriff auf gesamten Speicherbereich
- Erweiterung
  - Mehr RAM einbauen

Virtueller Speicher

- Abstrakter Adressraum
- Endliche Ressource
  - Größer als physischer Speicher
  - Hardware wird eingeblendet
- CPU hat indirekten Zugriff
  - Memory Management Unit
  - Übersetzung der Adressen notwendig
- Prozess sieht nur eigenen virtuellen Speicher
- Erweiterung
  - Mehr RAM einbauen
  - Swap-Datei/Partition

Paging

- Abb. des virtuellen auf den phys. Speicher
- Virtueller Speicher ist in Blöcke (Pages) aufgeteilt
- Physischer Speicher ist in Blöcke (Frames) unterteilt
- Referenzzähler für Frames (`struct page`)
- Page-Table übernimmt Zuordnung
  - Page → Frame
  - Größe ist identisch
- Betriebssystem unterscheidet zwischen
  - Reservierter Speicher (VSS – Virtual Set Size)
  - Geladenen Speicher (RSS – Reserved Set Size)
  - $VSS \geq RSS$
- Prozesse können sich Pages teilen
- RAM enthält Frames aller Prozesse

Swapping

- RAM reicht nicht immer aus
- Arbeitsspeicher teuer (und beschränkt)
- Massenspeicher billiger
- Frames können aus RAM verdrängt werden
- Unterschiedliche Verdrängungsstrategien
  - LRU häufig genutzt
  - Alternativen möglich
- Ablauf
  - Speicherknappheit tritt auf
  - Frame wird ausgewählt
  - Frame wird verdrängt (Swap-Out)
- Swap-In notwendig, wenn Frame nicht im RAM
  - Zugriff erzeugt Page-Fault
  - Notwendige Page wird geladen (eventl. Swap-Out)
- Swapping hat Nachteile für Massenspeicher

Universität Bremen

Themenübersicht

- Motivation
- Virtuell. vs physisch
- Paging
- Swapping

Grundlagen

- Datenstrukturen
- Mechanismen
  - Demand Paging
  - Copy-on-Write
  - Memory Mapping

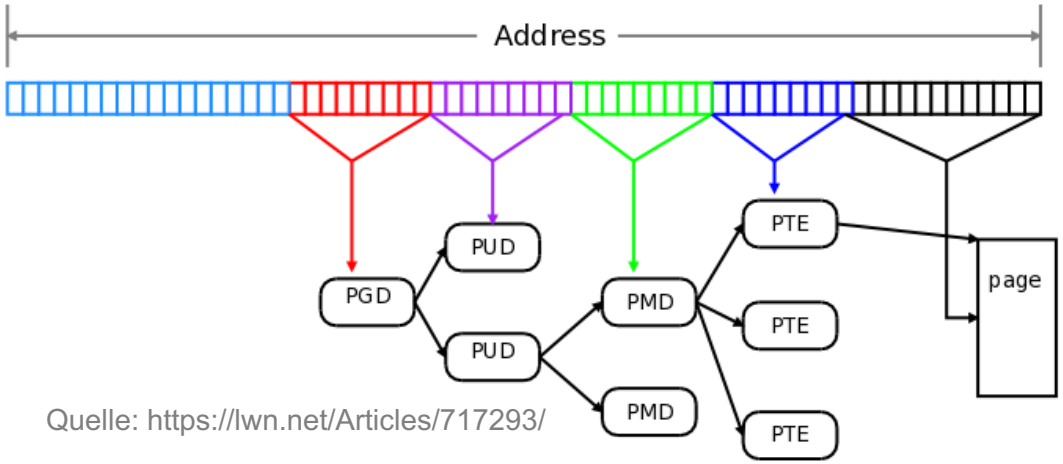
Implementierung

- Huge Tables
- ASLR
- OOM-Killer

Weiterführendes

Datenstrukturen – Page-Tabelle (x86_64)

- Page-Tabelle ist zentrale Datenstruktur (asm/pgtable_types.h)
- Hardware-Architektur spezifisch (MMU)
- 4-Level-Design (48-Bit)
  - Level 1) Page Global Directory
  - Level 2) Page Upper Directory (optional)
  - Level 3) Page Middle Directory
  - Level 4) Page Table Entry
  - Physical Offset
- CR3 Register

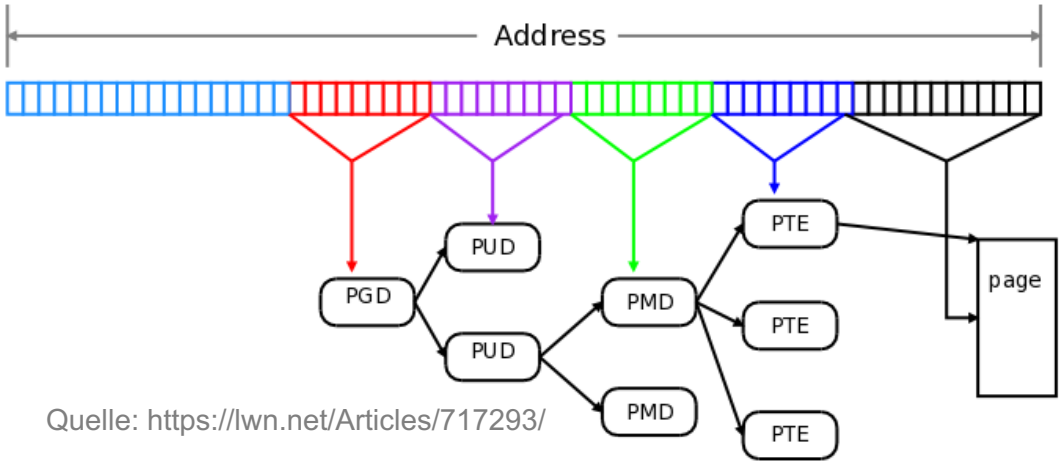


Quelle: https://lwn.net/Articles/717293/

Level	# Bits	# Einträge	Inhalt
PGD	9	512	Verweist auf P4D (oder PMD)
PUD	9	512	Verweist auf PMD
PMD	9	512	Zeigt auf PTE
PTE	9	512	Physische Adresse + Flags
Offset	12	4096 (Bytes)	

Datenstrukturen – Page-Tabelle (x86_64)

- PTE Flags für Pages (Obersten 12 Bits)
  - P / Present
  - RW / Read/Write-Berechtigung
  - US / User-Space / Kernel-Space
  - A / Accessed (wurde kürzlich verwendet)
  - D / Dirty (Seite wurde verändert)
  - NX / No-Execute
- Rest entspricht Adresse (48 Bit)
- Tabelleneinträge können leer sein (Speicherplatz sparen)



Quelle: https://lwn.net/Articles/717293/

Level	# Bits	# Einträge	Inhalt
PGD	9	512	Verweist auf P4D (oder PMD)
PUD	9	512	Verweist auf PMD
PMD	9	512	Zeigt auf PTE
PTE	9	512	Physische Adresse + Flags
Offset	12	4096 (Bytes)	

Demand Paging

- Programm ist nach Start zunächst nicht im Speicher
- Flag P der PTE ist auf 0 gesetzt
- Restlichen Bits des Eintrags verweisen auf Swap-Tabelle
- Swap-Tabellen Eintrag enthält Information über
  - Swap-Datei oder Swap-Partition
  - Offset innerhalb des Swaps
- Zugriff auf den PTE sorgt für Swap-In, wenn $PTE = 0$
- Speicher wird erst bei Bedarf belegt

Copy-on-Write (CoW)

- Optimierung für Prozesskopien
- Prozesse werden über `fork` erzeugt
- Linux kopiert **nicht** den gesamten virtuellen Speicher
- Page-Tabelle wird kopiert
- Einträge beider Prozesse werden als RO markiert
- Schreibender Zugriff erzeugt Page-Fault
- Frame wird kopiert und PTE angepasst
- Referenzzähler der Frames ist hier wichtig
  - Original-Frame wird erst freigegeben, wenn dieser auf 0

Memory Mapping

- Einblenden in den Arbeitsspeicher
  - Dateien
  - Speicher
  - Geräte
- Programme können wie mit normalen Speicher arbeiten
- Mapping wird vom Kernel verwaltet
- Bei Page-Fault werden Daten eingeblendet

Universität Bremen

Themenübersicht

- Motivation
- Virtuell. vs physisch
- Paging
- Swapping

Grundlagen

- Datenstrukturen
- Mechanismen
  - Demand Paging
  - Copy-on-Write
  - Memory Mapping

Implementierung

- Huge Tables
- ASLR
- OOM-Killer

Weiterführendes

Huge Tables – Huge Pages

- Performance nimmt bei vielen Pages ab
  - Viele Einträge in Page-Tabelle
  - Häufige Page-Misses möglich
  - Einsparung wünschenswert
- Huge Page
  - PUD 1GB
  - PMD 2MB
- Optionen
  - Transparent Huge Pages (Kernel entscheidet)
  - Explicit Huge Pages
    - `mmap`
    - `shmget`
- Flag für Huge Page in der Page-Tabelle

Universität Bremen

ASLR – Address Space Layout Randomisation

- Verhinderung von Angriffen
  - Return-Oriented Programming
  - Code Injection
- Betriebssystem unterstützt ASLR
  - cat /proc/sys/kernel/randomize_va_space
  - 0 → ASLR deaktiviert
  - 1 → ASLR für Stack und Heap aktiviert
  - 2 → ASLR für gesamten Speicher aktiviert
- Anordnung des Virtueller Adressraums zufällig
  - Bestimmung eines Zufallswert
  - Reihenfolge der Segmente variiert
- Virtueller Adressraum wird mit MMU auf physischen Speicher abgebildet

OOM Killer

- Physischer Speicher und Swap laufen voll
- Betrieb soll aufrecht erhalten werden
- OOM-Killer wird ausgeführt
  - Laufende Prozesse werden bewertet
  - Schlechtester Prozess wird beendet

OOM Killer

```
/**
* oom_badness - heuristic function to determine which candidate task to kill
* @p: task struct of which task we should calculate
* @totalpages: total present RAM allowed for page allocation
*
* The heuristic for determining which task to kill is made to be as simple and
* predictable as possible. The goal is to return the highest value for the
* task consuming the most memory to avoid subsequent oom failures.
*/
long oom_badness(struct task_struct *p, unsigned long totalpages) {
  long points;
  long adj;

  if (oom_unkillable_task(p))
    return LONG_MIN;

  p = find_lock_task_mm(p);
  if (!p)
    return LONG_MIN;
```

OOM Killer

```
  /* Do not even consider tasks which are explicitly marked oom
   * unkillable or have been already oom reaped or the are in the middle of vfork
   */
  adj = (long)p->signal->oom_score_adj;
  if (adj == OOM_SCORE_ADJ_MIN ||
    test_bit(MMF_OOM_SKIP, &p->mm->flags) ||
    in_vfork(p)) {
    task_unlock(p);
    return LONG_MIN;
  }

  /* The baseline for the badness score is the proportion of RAM that each
   * task's rss, pagetable and swap space use.
   */
  points = get_mm_rss(p->mm) + get_mm_counter(p->mm, MM_SWAPENTS) +
    mm_pgtables_bytes(p->mm) / PAGE_SIZE;
  task_unlock(p);

  /* Normalize to oom_score_adj units */
  adj *= totalpages / 1000;
  points += adj;

  return points;
}
```

Universität Bremen

Themenübersicht

- Motivation
- Virtuell. vs physisch
- Paging
- Swapping

Grundlagen

- Datenstrukturen
- Mechanismen
  - Demand Paging
  - Copy-on-Write
  - Memory Mapping

Implementierung

- Huge Tables
- ASLR
- OOM-Killer

Weiterführendes