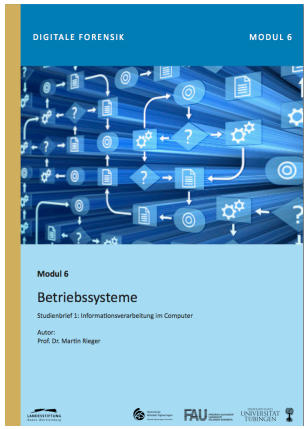
Betriebssysteme

SS 2012

Hans-Georg Eßer
Dipl.-Math., Dipl.-Inform.

SB 3 (15.04.2012)
Speicherverwaltung
(Ein-/Ausgabegeräte und Schnittstellen)



Studienbrief 3

- Speicherverwaltung
- Speicher-Forensik (Linux)

3.5 Virtueller Speicher

Motivation

- Speicher zu knapp für große Programme
  → Overlay-Programmierung
- Programmteile dynamisch nachladen, wenn sie benötigt werden
- Programmierer muss sich um Aufteilung in Overlays kümmern

Overlay-Programmierung

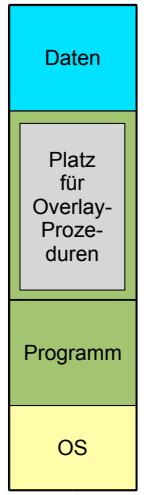
Turbo Pascal, um 1985-90:

```
program grossesprojekt;

overlay procedure kundendaten;
...

overlay procedure lagerbestand;
...

{ Hauptprogramm }
begin
  while input <> "exit" do begin
    case input of
      "kunden": kundendaten;
      "lager":  lagerbestand;
    end;
  end;
end.
```



Lösung des Problems

- Virtueller Speicher, der das gesamte Programm aufnehmen kann
- Programm sieht Speicherbereich, der ihm zur Verfügung gestellt wurde – wie viel wirklich vorhanden ist, spielt (für das Programm) keine Rolle

Speicherschutz

- Speicherbereiche der Prozesse sollen durch das Betriebssystem von anderen Prozessen abgeschirmt werden
  - Frühe CPUs: Segmentierung des Speichers
  - Moderne CPUs: Virtueller Speicher (Paging); normale Prozesse dürfen Paging nicht deaktivieren
  - Ganz alte CPUs: Kein Schutz, darum auch keine stabilen / sicheren BS möglich

Virtuelle Speicherverwaltung (Paging)

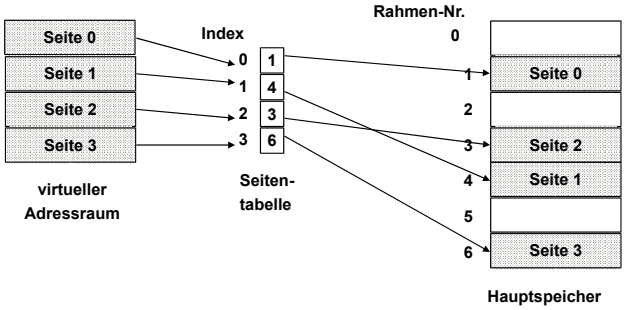
- Aufteilung des Adressraums in **Seiten** (**pages**) fester Größe und des Hauptspeichers in **Seitenrahmen** (**page frames**) gleicher Größe.
  - Typische Seitengrößen: 512 Byte bis 8192 Byte (immer Zweierpotenz).
- Der lineare, zusammenhängende Adressraum eines Prozesses (**„virtueller" Adressraum**) wird auf beliebige, nicht zusammenhängende Seitenrahmen abgebildet.
- BS verwaltet eine einzige Liste freier Seitenrahmen

Virtuelle Speicherverwaltung (Paging)

- Die Berechnung der **physikalischen Speicheradresse** aus der vom Programm angegebenen **virtuellen Adresse**
  - geschieht zur Laufzeit des Programms,
  - ist transparent für das Programm,
  - muss von der Hardware unterstützt werden.
- Vorteile der virtuellen Speicherverwaltung:
  - Einfache Zuteilung von Hauptspeicher.
  - Kein Aufwand für den Programmierer.

Virtueller Adressraum (1)

- Paging stellt den Zusammenhang zwischen Programmadresse und physikalischer Hauptspeicheradresse erst zur Laufzeit mit Hilfe der Seitentabellen her.



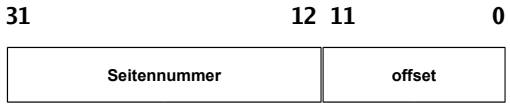
Virtueller Adressraum (2)

- Die vom Programm verwendeten Adressen werden deshalb auch **virtuelle Adressen** genannt.
- Der **virtuelle Adressraum** eines Programms ist der lineare, zusammenhängende Adressraum, der dem Programm zur Verfügung steht.

Adressübersetzung beim Paging (1)

- Die Programmadresse wird in zwei Teile aufgeteilt:
  - eine Seitennummer
  - eine relative Adresse (offset) in der Seite

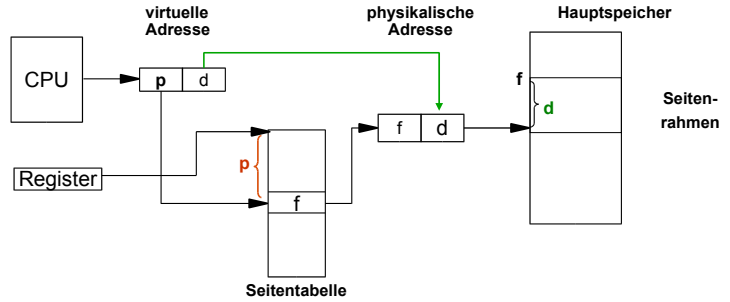
Beispiel: 32-bit-Adresse bei einer Seitengröße von 4096 ($=2^{12}$) Byte:



Adressübersetzung beim Paging (2)

- Für jeden Prozess gibt es eine **Seitentabelle** (**page table**). Diese enthält für jede Prozessseite
  - eine Angabe, ob die Seite im Speicher ist,
  - die Nummer des Seitenrahmens im Hauptspeicher, der die Seite enthält.
- Das Page Table Register (PTR) enthält die Anfangsadresse der Seitentabelle für den aktuellen Prozess.
- Die Seitennummer wird als Index in die Seitentabelle verwendet.

Adressübersetzung beim Paging (3)



- Für jeden Hauptspeicherzugriff wird ein zusätzlicher Hauptspeicherzugriff auf die Seitentabelle benötigt. Dies muss durch Caches in der Hardware beschleunigt werden.
- Seite nicht im Speicher → spezielle Exception, einen sog. **page fault (Seitenfehler)** auslösen.

Virtueller Speicher allgemein (1)

- Mehr Prozesse können effektiv im Speicher gehalten werden
  → bessere Systemauslastung
- Ein Prozess kann viel mehr Speicher anfordern als physikalisch verfügbar

Virtueller Speicher allgemein (2)

- allgemeiner Vorgang:
  - Nur Teile des Prozesses befinden sich im physikalischen Speicher
  - falls Zugriff auf eine Adresse, die ausgelagert ist:
    - BS setzt den Prozess auf blockiert
    - BS setzt eine Disk-I/O-Leseanfrage ab
    - Nach Laden der fehlenden Seite wird ein I/O-Interrupt erzeugt
    - das BS setzt Prozess zuletzt wieder in den Bereit- (Ready-) Zustand

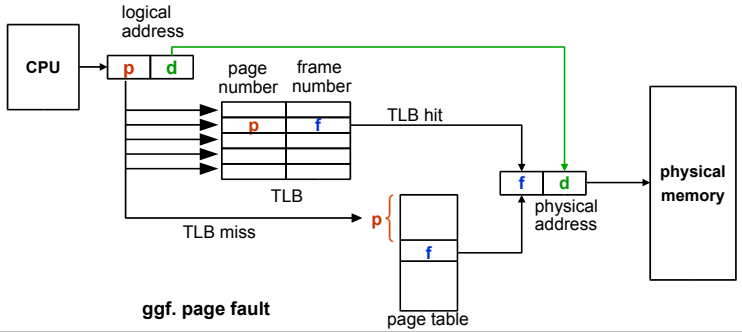
Virtueller Speicher allgemein (3)

- **Thrashing:** Prozessor verbringt die meiste Zeit mit Ein- und Auslagern von Prozessteilen statt mit der Ausführung von Prozessanweisungen
- **Lokalitätsprinzip:**
  - Zugriffe auf Daten und Programmcode häufig lokal gruppiert;

    → Annahme gerechtfertigt, dass nur wenige Prozessstücke während einer kurzen zeitlichen Periode gleichzeitig vorgehalten werden müssen

Translation Look-Aside Buffer (1)

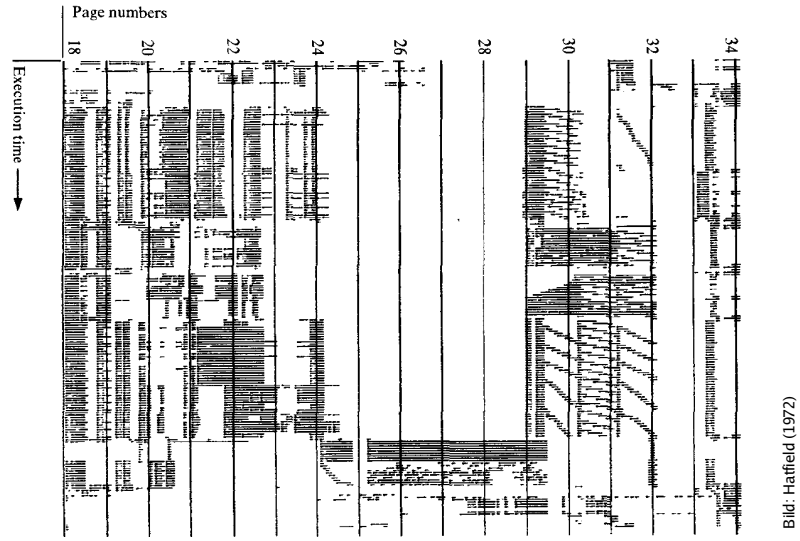
- **Translation Look-Aside Buffer** (**TLB**): schneller Hardware-Cache für zuletzt benutzte Seitentabelleneinträge
- Assoziativ-Speicher: bei Übersetzung einer Adresse wird deren Seitennummer gleichzeitig mit allen Einträgen des TLB verglichen.



Translation Look-Aside Buffer (2)

- Treffer im TLB → Speicherzugriff auf Seitentabelle unnötig
- Fehltreffer → Zugriff auf die Seitentabelle; alten Eintrag im TLB durch neuen ersetzen
- Trefferquote (hit ratio) beeinflusst die durchschnittliche Zeit einer Adressübersetzung.
- Lokalitätsprinzip: Programme greifen meist auf benachbarte Adressen zu → auch bei kleinen TLBs hohe Trefferquoten (typisch: 80-98%).

Lokalitätsprinzip



Translation Look-Aside Buffer (3)

- Inhalt des TLB ist prozess-spezifisch! Zwei Möglichkeiten:
  - Jeder Eintrag enthält ein „valid bit“. Bei Prozesswechsel (Context Switch) ganzen TLB invalidieren.
  - Jeder Eintrag im TLB enthält Prozessidentifikation (PID), die mit der PID des zugreifenden Prozesses verglichen wird.
- Beispiele für TLB-Größen:
  - Intel 80486: 32 Einträge.
  - Pentium-4, PowerPC-604: 128 Einträge für jeweils Code und Daten.

Translation Look-Aside Buffer (4)

Was macht hier eigentlich das Betriebssystem?

- Page-Table-Register laden
- Im Falle eines Page Fault: Fehlende Seite aus dem Swap holen und Seitentabelle aktualisieren
- Evtl. vorher: Seitenverdrängung – welche Seite aus dem Hauptspeicher entfernen? (→ später)

Alles andere: Hardware

- Zugriff auf TLB und ggf. auf Seitentabelle
- Wenn Seite im Speicher: Berechnung der phys. Adresse
- Inhalt aus Cache oder ggf. aus Hauptspeicher holen

Invertierte Seitentabellen (1)

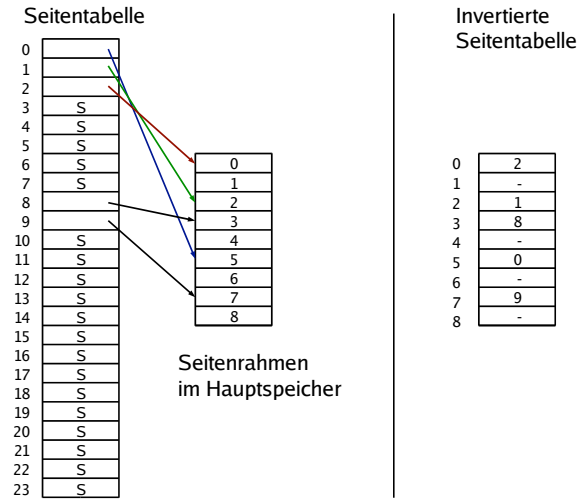
- Bei großem virtuellen Speicher sehr viele Einträge in der Seitentabelle nötig,
  z. B. 2^{32} Byte Adressraum, 4 KByte/Seite
  → über 1 Millionen Seiteneinträge,
  also Seitentabelle > 4 MByte (pro Prozess!)
- Platz sparen durch invertierte Seitentabellen:
  - normal: ein Eintrag pro (virtueller) Seite mit Verweis auf den Seitenrahmen (im Hauptspeicher)
  - invertiert: ein Eintrag pro Seitenrahmen mit Verweis auf Tupel (Prozess-ID, virtuelle Seite)

Invertierte Seitentabellen (2)

- Problem: Suche zu Prozess *p* und seiner Seite *n* nach dem Eintrag (*p*,*n*) in der invertierten Tabelle → langwierig
- Auch hier TLB einsetzen, um auf „meist genutzte“ Seiten schnell zugreifen zu können
- Bei TLB-Miss hilft aber nichts: Suchen...

- Andere Lösung für Problem der großen Seitentabellen: Mehrstufiges Paging (→ gleich)

Invertierte Seitentabellen (3)



Hochschule Albstadt-Sigmaringen
Albstadt-Sigmaringen University

Mehrstufiges Paging (1)

Die Seitentabelle kann sehr groß werden.

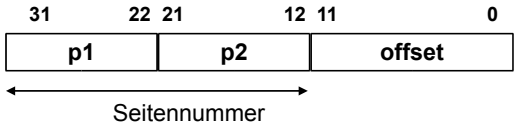
Beispiel: - 32-Bit-Adressen,
- 4 KByte Seitengröße,
- 4 Byte pro Eintrag

Seitentabelle:
>1 Million Einträge,
4 MByte Größe (pro Prozess!)

Hochschule Albstadt-Sigmaringen
Albstadt-Sigmaringen University

Mehrstufiges Paging (2)

- Zweistufiges Paging:
  - Seitennummer noch einmal unterteilen, z. B.:

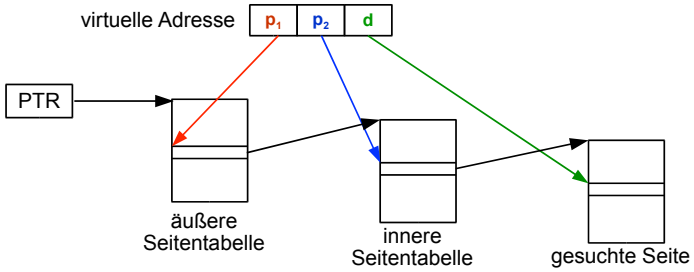


  - p_1: Index in **äußere Seitentabelle**, deren Einträge jeweils auf eine **innere Seitentabelle** zeigen
  - p_2: Index in eine der inneren Seitentabellen, deren Einträge auf Seitenrahmen im Speicher zeigen
  - Die inneren Seitentabellen müssen nicht alle speicherresident sein
- Analog dreistufiges Paging etc. implementieren

Hochschule Albstadt-Sigmaringen
Albstadt-Sigmaringen University

Mehrstufiges Paging (3)

Adressübersetzung bei zweistufigem Paging:

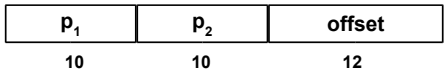


Hochschule Albstadt-Sigmaringen
Albstadt-Sigmaringen University

Mehrstufiges Paging (4)

- Größe der Seitentabellen:

Beispiel:



  - Die äußere Seitentabelle hat 1024 Einträge, die auf (potentiell) 1024 innere Seitentabellen zeigen, die wiederum je 1024 Einträge enthalten.
  - Bei einer Länge von 4 Byte pro Seitentabelleneintrag ist also jede Seitentabelle genau eine 4-KByte-Seite groß.
  - Es werden nur so viele innere Seitentabellen verwendet, wie nötig.

Mehrstufiges Paging (5)

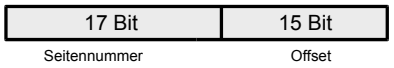
- Jede Adressübersetzung benötigt noch mehr Speicherzugriffe, deshalb ist der Einsatz von TLBs noch wichtiger.
- Als Schlüssel für den TLB werden alle Teile der Seitennummer zusammen verwendet (p_1,p_2,...).

Aufgabenbeispiel (1)

Paging mit folgenden Parametern:
- 32-Bit-Adressbus
- 32 KB Seitengröße
- 64 MB RAM
- 1-stufiges Paging

Zu berechnen:
a) maximale Anzahl der adressierbaren virtuellen Seiten
b) Größe der erforderlichen Seitentabelle (in KB)

a) 32 KB (Seitengröße) = $2^5 \times 2^{10}$ Byte = 2^{15} Byte
d.h.: Offset ist 15 Bit lang



Also gibt es 2^{17} virtuelle Seiten

b) Zur Seitentabelle:
In 64 MB RAM passen 64 M / 32 K = 2 K = 2048 (2^{11}) Seitenrahmen
Ein Eintrag in der Seitentabelle benötigt darum 11 Bit, in der Praxis 2 Byte.

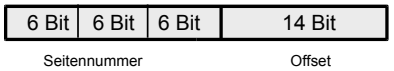
→ Platzbedarf:
#(virt. Seiten) x Größe(Eintrag)
= 2^{17} x 2 Byte = 2^{18} Byte = 256 KB

Aufgabenbeispiel (2)

Paging mit folgenden Parametern:
- 32-Bit-Adressbus
- 16 KB Seitengröße
- 2 GB RAM
- 3-stufiges Paging

Zu berechnen:
a) maximale Anzahl der adressierbaren virtuellen Seiten
b) Größe der Seitentabelle(n)
c) Anzahl der Tabellen

a) 16 KB (Seitengröße) = $2^4 \times 2^{10}$ Byte = 2^{14} Byte,
d.h.: Offset ist 14 Bit lang



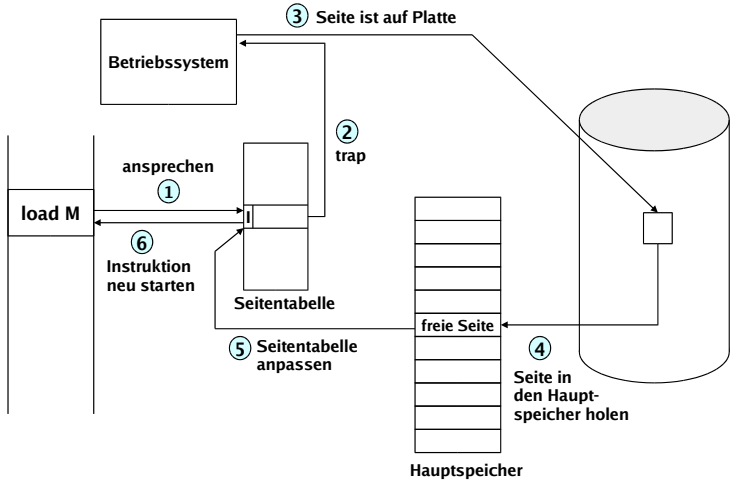
Also gibt es 2^{18} virtuelle Seiten

b) Zur Seitentabelle:
In 2 GB RAM passen 2 G / 16 K = 128 K = 2^{17} Seitenrahmen
Ein Eintrag in der Seitentabelle benötigt darum 17 Bit, in der Praxis 4 Byte.

→ Platzbedarf **einer** Tabelle:
#(Einträge) x Größe(Eintrag)
= 2^6 x 4 Byte = 2^8 Byte = 256 Byte

Es gibt 1 äußere, 2^6 mittlere und 2^{12} innere Seitentabellen

Page-Fault-Behandlung



Hochschule Albstadt-Sigmaringen
Albstadt-Sigmaringen University

Speicher-Forensik (Linux)

- Speicherabbild (unter Linux) z. B. über `/dev/fmem` erstellen
  - benötigt Kernel-Modul `fmem`
  - Die Datei `/dev/kmem` (→ Skript) gibt es in aktuellen Linux-Kernel-Versionen nicht mehr; `/dev/mem` greift nur auf I/O-Speicher zu.
- Problem: Speicherabbild analysieren
  - physischer Speicher! BS verwendet virt. Adressen
  - ausgelagerte Seiten fehlen (waren auf Platte)
- Aktuelle Forschung: RamParser (2010)

Hochschule Albstadt-Sigmaringen
Albstadt-Sigmaringen University

Speicher-Forensik (Linux)

- Umsetzen virtueller in physische Adressen kein großes Problem: Muss Seitentabelle finden und die MMU-Funktion simulieren.
- Aber: Wo stehen die diversen Kernel-Datenstrukturen? Bei jeder Kernel-Version und für jede CPU-Architektur verschieden.
- RamParser [1] analysiert Kernel-Code und leitet daraus die Positionen im Speicher ab.

[1] Andrew Case, Lodovico Marziale, Golden G. Richard: *Dynamic Recreation of kernel data structures for live forensics*, http://www.dfrws.org/2010/proceedings/2010-304.pdf

Hochschule Albstadt-Sigmaringen
Albstadt-Sigmaringen University

Speicher-Forensik (Linux)

- RamParser kann u. a. die Funktionen von `ps` und `netstat` nachbilden
- unterstützt diverse Linux-2.6.x-Kernel für x86, x86_64 und PPC64
- verwendet Datei `System.map` des zu untersuchenden Systems (Namen und Adressen aller im Kernel definierten Symbole)
- Es gab schon vor RamParser ähnliche Tools für Linux, aber immer nur für eine bestimmte Linux-Version und Architektur