Speicherverwaltung (2)

Zuteilung freien Speichers

Angeforderter Speicherbereich einer bestimmten Größe kann wie folgt zugeteilt werden:

- First-Fit-Methode
  - Der erste genügend große freie Speicherbereich wird zugeordnet.
- Best-Fit-Methode
  - Der kleinste ausreichende freie Speicherbereich wird zugeordnet.
  - Aufwendiger, da die gesamte Liste bzw. Bitmap durchsucht werden muss.
  - Starke Fragmentierung des freien Speichers in viele kleine Bereiche.

Zuteilung freien Speichers

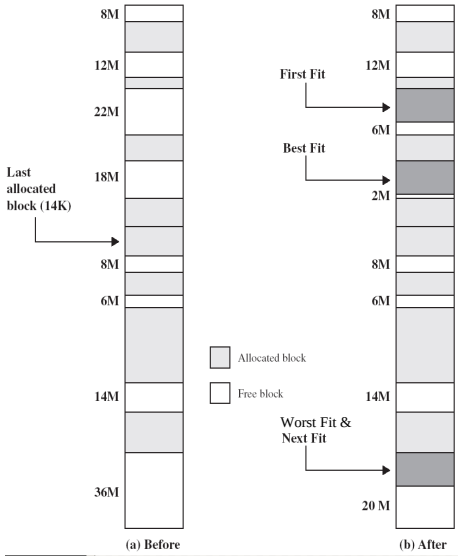
- Worst-Fit-Methode
  - Der größte freie Speicherbereich wird zugeteilt.
  - Es bleiben verhältnismäßig große freie Bereiche übrig.
- Quick-Fit-Methode
  - Unterhalten von mehreren Listen freier Speicherbereiche für bestimmte Standardgrößen (z. B. 4 KB, 8 KB, 12 KB etc.) führt zu einer schnellen Zuteilung.
  - Bei Freigabe eines Bereiches muss dieser mit evtl. benachbarten freien Bereichen zusammengelegt werden. Dies bedeutet bei mehreren Listen einen erhöhten Aufwand.
  - Spezielle Variante: Buddy System

Zuteilung freien Speichers: Beispiel

Beispiel für eine Speicherbelegung

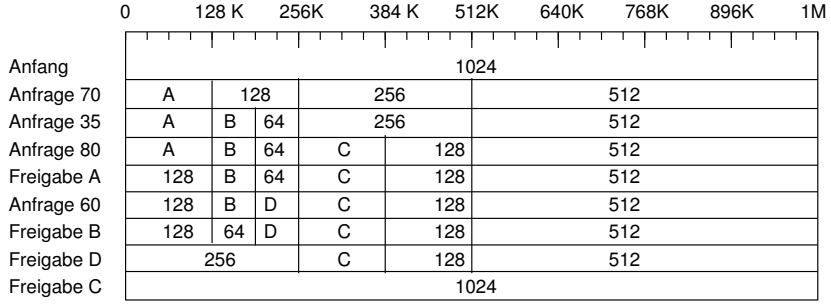
- vor und
- nach

der Allozierung eines 16-MByte-Blocks



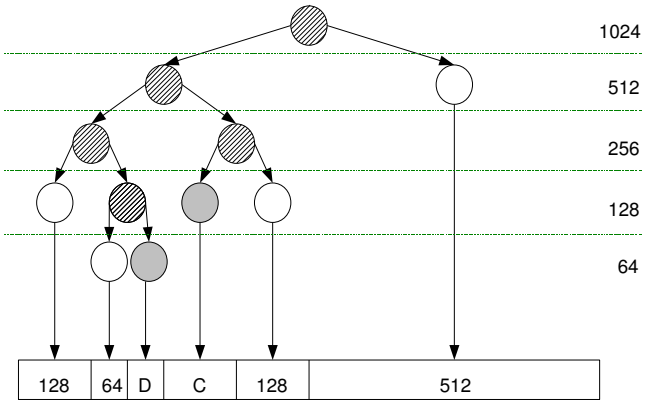
Speicherverwaltung: Buddy System

- Separate Listen freier Bereiche der Größen 1, 2, 4, 8, 16 etc. KByte bis zur Speichergröße.
- Bei Freigabe eines Speicherbereichs muss nur eine der Listen durchsucht werden, um festzustellen, ob der Bereich mit einem anderen freien Bereich zusammengefasst werden kann.
- Blockgröße immer Zweierpotenz ist nicht sehr speichereffizient.



Buddy-System: Baum-Repräsentierung

Zustand vor der Freigabe von Prozess D



Segmentierung (1)

- Aufteilung von Programmen in mehrere Segmente, z. B. Codesegment, Datensegment(e), Stacksegment(e) etc.
- Jedes Segment entspricht einem linearen Adressraum von 0 bis zu einem Maximalwert.
- Adressangaben bestehen aus:
  - einer Segmentangabe
  - einer relativen Adresse im Segment

  (Zweidimensionaler Adressraum)

Segmentierung (2)

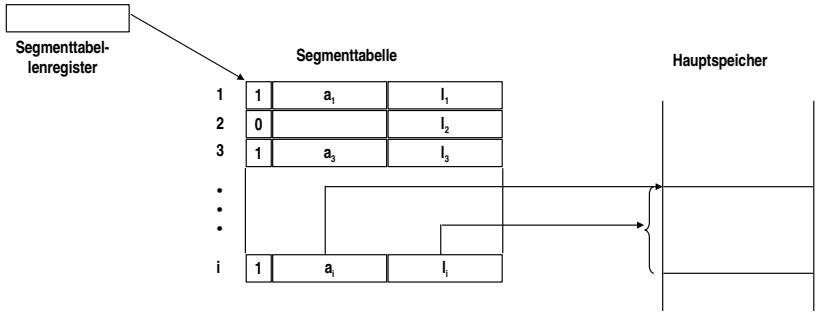
- Vorteile der Segmentierung:
  - Für die kleineren Segmente findet man leichter freien Hauptspeicher.
  - Segmente können unabhängig voneinander wachsen.
  - Es müssen nicht alle Segmente gleichzeitig im Hauptspeicher sein (Swapping von Segmenten).
  - Segmente können unterschiedlich geschützt werden.
  - Unterscheidung von prozess-privaten Segmenten und solchen Segmenten, die von mehreren Prozessen gemeinsam benutzt werden können.

Segmentierung (3)

- Segmentierung muss von der Hardware unterstützt werden.
- Für jeden Prozess gibt es eine Segmenttabelle im Speicher, die für jedes Segment mindestens drei Angaben enthält:
  - Ist das Segment im Hauptspeicher?
  - Anfangsadresse des Segments im Hauptspeicher
  - Länge des Segments

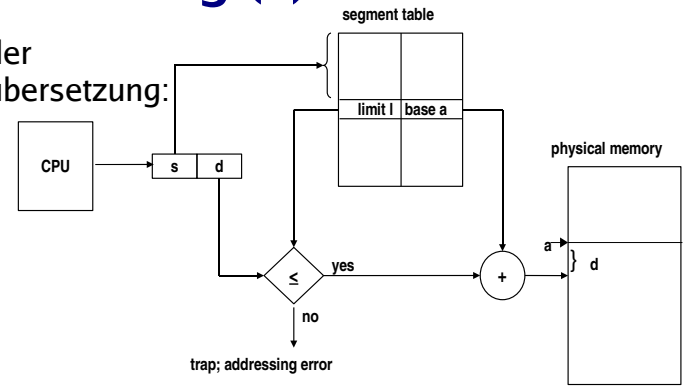
Segmentierung (4)

- Die Adresse der Segmenttabelle steht in einem Segmenttabellenregister.



Segmentierung (5)

- Ablauf der Adressübersetzung:



- Für jeden Hauptspeicherzugriff ist ein zusätzlicher Speicherzugriff auf die Segmenttabelle nötig. Dies müssen Register in der Hardware beschleunigen

Segmentierung (6)

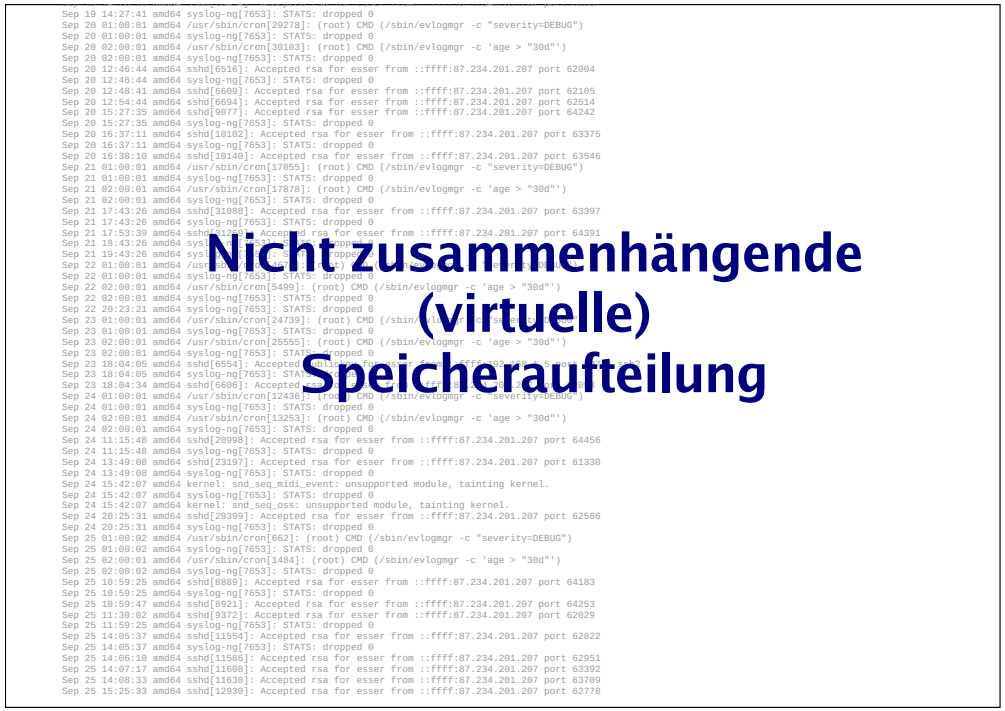
- Zugriff auf ein Segment, das nicht im Hauptspeicher ist -> Segment Fault: Betriebssystem muss das Segment erst in den Hauptspeicher laden.

- Schutz eines Segments:
  Die Einträge in der Segmenttabelle enthalten zusätzlich einen Schutzcode (z. B. für die Zugriffe Lesen, Schreiben, Ausführen), der vom Programmierer (oder Compiler/Linker) festgelegt wird.

Segmentierung (7)

- Sharing eines Segments im Hauptspeicher ist einfach:
  Man lässt die Einträge in den Segment-tabellen mehrerer Prozesse auf die gleiche Hauptspeicheradresse zeigen.

- Fragmentierung:
  Da jedes Segment im Hauptspeicher zusammenhängend abgelegt wird, tritt externe Fragmentierung auf.



Nicht zusammenhängende (virtuelle) Speicheraufteilung

Ausgangslage

- 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

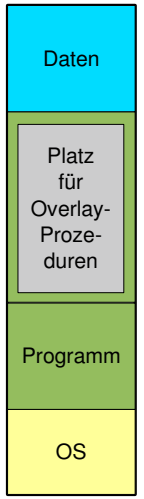
Turbo Pascal, um 1985-90:

```
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

Virtuelle Speicherverwaltung (Paging)

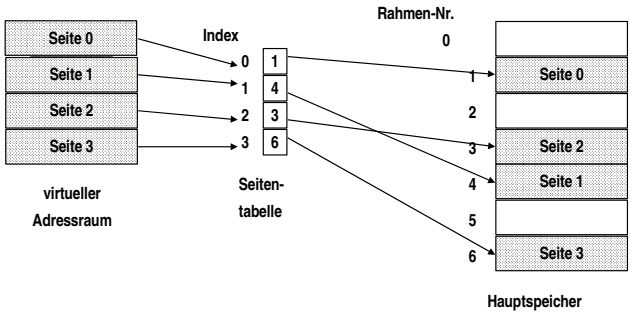
- Aufteilung des Adressraums in Seiten (pages) fester Größe und des Hauptspeichers in Seitenrahmen (page frames) gleicher Größe.
  - Typische Seitengrößen: 512 Byte bis 8192 Byte (immer Zweierpotenz).
- Der lineare, zusammenhängende Adressraum eines Prozesses („virtueller" Adressraum) wird auf beliebige,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

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.
  - Keine externe Fragmentierung, geringe interne Fragmentierung.
  - Kein Aufwand für den Programmierer.

Virtueller Adressraum (1)

- Beim Paging wird der Zusammenhang zwischen Programmadresse und physikalischer Hauptspeicheradresse erst zur Laufzeit mit Hilfe der Seitentabellen hergestellt.



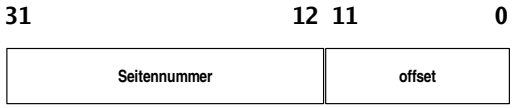
Virtueller Adressraum (2)

- 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

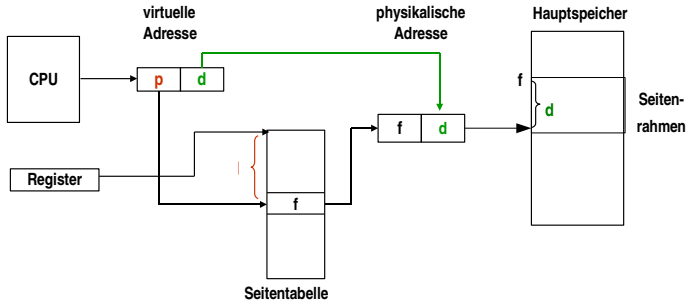
Beispiel: 32-bit-Adresse bei einer Seitengröße von 4096 (=2^12) Byte:



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.
- Ein spezielles Register 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)

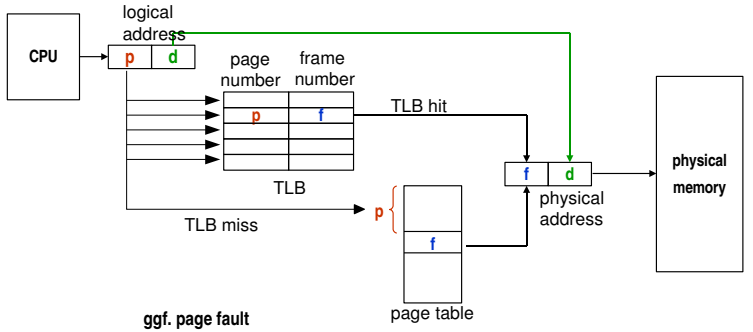
- allgemeiner Vorgang:
  - Nur Teile des Prozesses befinden sich im physikalischen Speicher
  - falls Zugriff auf eine Adresse, die ausgelagert ist:
    - BS setzt den Prozess auf blockiert
    - BS setzt eine Disk-I/O-Leseanfrage ab
    - Nach Laden des fehlenden Stücks (Seite oder Segment) wird ein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

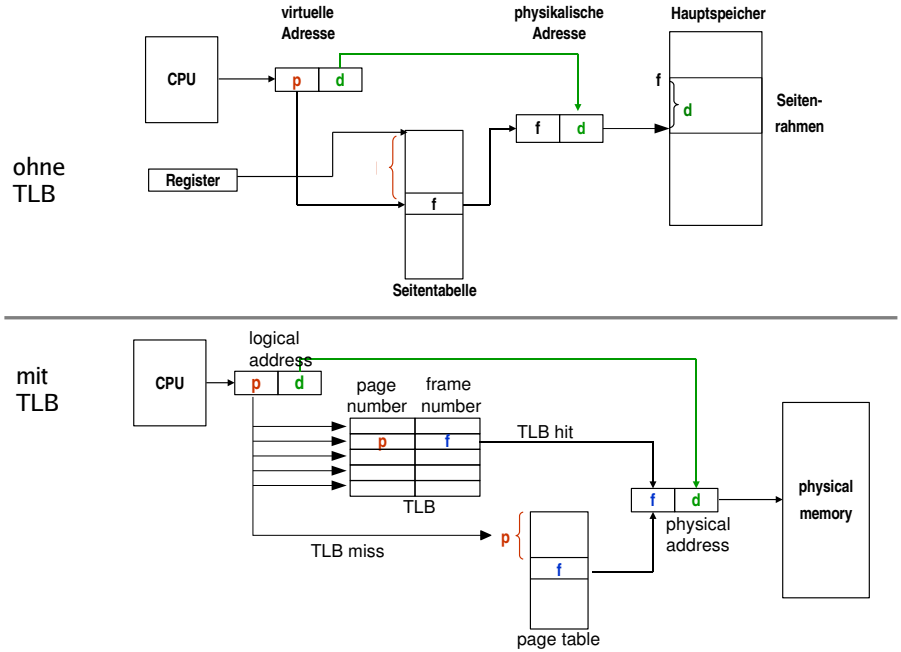
Virtueller Speicher allgemein (3)

- „*thrashing*“ (siehe später): 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 Lookaside Buffer (TLB): schneller Hardware-Cache, mit den zuletzt benutzten Seitentabelleneinträgen
- 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%).

Translation Look-Aside Buffer (3)

- Inhalt des TLB ist prozessspezifisch!
  Zwei Möglichkeiten:
  - Jeder Eintrag im TLB enthält ein „valid bit“.
    Bei Prozesswechsel (Context Switch) wird der gesamte Inhalt des TLB invalidiert.
  - 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 und Speicher-Cache