$2^3 \cdot 2^{10} = 2^{13}$

Kapitel 3: Hauptspeicherverwaltung

Inhalt und Wiederholung

Randbedingungen der Hauptspeicherverwaltung

- Der Prozessor (CPU) kann Programme nur dann ausführen, wenn sie im **Hauptspeicher (RAM**) vorliegen.
- **Der Zugriff auf den Hauptspeicher muss sehr effizient möglich sein**, da es eine sehr häufige Operation ist.
  - Falls spezielle (Berechnungs-)Operationen erforderlich sind, dann müssen diese durch die Hardware direkt durchgeführt werden. **(MMU)**
- *Das **Betriebssystem darf** durch andere **laufende Prozesse nicht manipuliert werden** können.*
- *Ein **Prozess darf** durch andere **laufende Prozesse nicht manipuliert** werden können.*

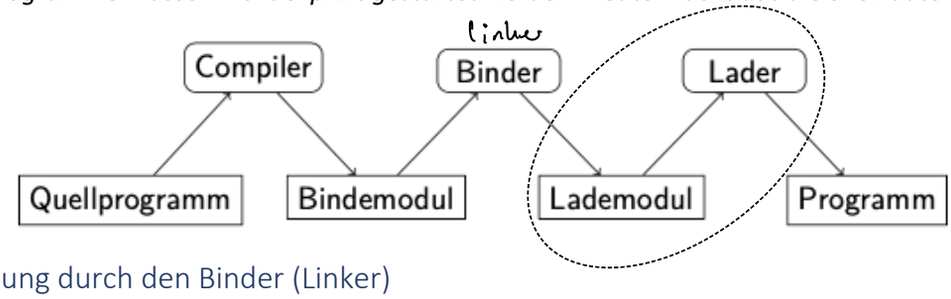
Adresstypen

- Speicherzellen werden durch ihre sog. Adresse identifiziert. Adressen sind positive ganze Zahlen.
- **Hardware-Adressen** (physische Adresse):
  - Sie beschreibt wo im RAM die Daten gespeichert sind.
  - In der CPU werden sie in Adressregistern gespeichert.
- **Programm-Adressen** (logische Adresse):
  - Kommen in der Assembler-Programmierung explizit vor.
  - Mit Zeigern, bzw. Referenzen können Adressen bearbeitet werden.

Adressumwandelung bei Programmerstellung

- **Compiler** ersetzen Variablennamen durch zugehörige numerische Programmadresse.
- Mehrere Objektdateien werden vom **linker** zusammengebunden und dabei werden Adressen angepasst.
- Beim Start des Programms wird es in den Hauptspeicher geladen vc und dabei findet die letzte Adressumsetzung statt.

Hinweis: Ladeprogramme müssen nicht explizit gestartet werden. Heute macht das die Shell automatisch.



Adressanpassung durch den Binder (Linker)

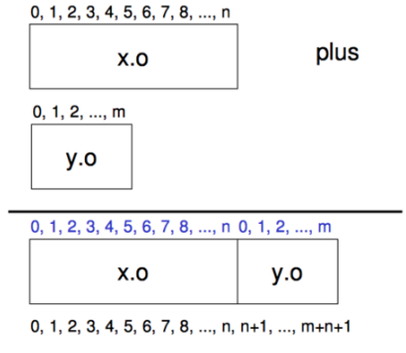
- Gegeben seien zwei Bindemodule *(meist .o-Dateien)* in denen alle Adressen bei 0 beginnen.
- Beim zusammenkopieren müssten alle Adressen des hinteren Moduls um die Größe des vorderen Moduls erhöht werden.
  - Binder muss Programmcode von Daten unterscheiden können.
  - Das Umrechnen ist viel zu viel Rechenaufwand für den Binder.

Lösung: Benutze **Relative Adressen**

- In jedem Modul wird ein **Basisregister benutzt**.
- Im vorderen Modul wird eine 0 ins Basisregister geschrieben.
- Im hinteren Modul wird die Größe des vorderen Moduls ins Basisregister geschrieben.
- Bei jedem Speicherzugriff wird jetzt der Wert des Basisregisters zur Adresse addiert und auf diese neue Adresse wird dann zugegriffen.

Binden von zwei Modulen

- Alle Adressen in x.o und y.o beginnen bei 0.
- Wird y.o nun hinter x.o gehangen, dann verschieben sich alle Adressen in y.o um die Länge von x.o
- In beiden Modulen wird nun eine Basisadresse definiert. In x.o ist sie 0 und in y.o ist sie n+1.
- Beim Zugriff auf logischen Adressen (0. . . i) eines Moduls wird der Inhalt der zugehörigen Basisregisters addiert und erst dann liegt die richtige Adresse vor.

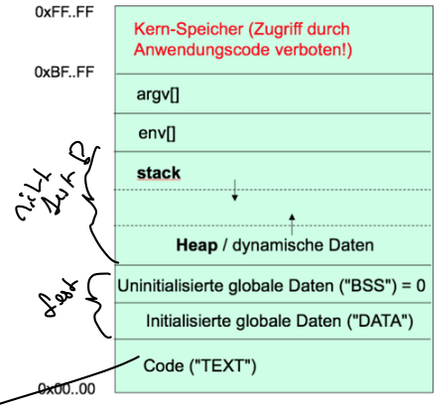


Adressanpassungen durch den Lader

- Hauptspeicheradressen beginnen bei 0, aber ein Programm sollte an einer beliebigen Stelle des Hauptspeichers ausgeführt werden können.
- Wird ein Programm in den Hauptspeicher geladen, dann müssen **logische Adressen** in **physische Adressen** umgewandelt werden.
    - Benutze relative Adressierung und passe den Wert des Basisregisters richtig an. Eigenschaften:
- Es lässt sich mehr als ein Programm laden und ausführen.
    - Ein Programm muss immer komplett im RAM stehen.
    - Der Speicherbedarf von Programmen muss vorab feststehen.
    - Ein Programm kann auf den Speicherbereich anderer Programme zugreifen.

Speicheraufteilung von Prozessen

- Platzbedarf für Programmcode und globale Daten steht nach dem Übersetzen fest.
- Platzbedarf für dynamische Daten steht nicht fest!
    - Lokale Variablen, die bei Methodenaufruf instantiiert werden. Datenstruktur: Stapel (engl. **stack**)
    - Dynamisch angeforderter (z. B. durch new oder malloc) Speicher. Datenstruktur: „Halde“ (engl. heap)



Zusammenhängende Speicherverwaltung

- Im Einprogammbetrieb wird dem Programm der gesamten noch freie Hauptspeicher zugeordnet.
- Für Mehrprogrammbetrieb muss der Hauptspeicher in Bereiche eingeteilt werden, in denen dann die einzelnen Programme ausgeführt werden.
- Probleme:
    - Wie viele Bereiche (Programme) sollen vorgesehen sein?
    - Wie groß dimensioniert werden die Bereiche?
    - Wie groß sollte der Bereich für ein Programm sinnvollerweise sein?
    - Wie wird mit Lücken im RAM umgegangen?
- Programme durften nicht mehr RAM brauchen, als im Rechner eingebaut war.

Nicht zusammenhängende Speicherverwaltung

- Eigentlich müssen Programme nicht immer komplett im Hauptspeicher vorliegen.
- Es reicht, wenn der Teil, der gerade benötigt wird im Hauptspeicher liegt.
- Aus welchen Teilen besteht ein Programm und welche Zerlegungseinheiten können benutzt werden?
    - **Segmente**
    - **Seiten**

Einige Fachbegriffe

- **Segment**: Ein logischer Bestandteil des Speicherbereichs eines Programms, z. B. Programmcode, statischer Datenbereich, usw.
- **Seite** (page): Ein Ausschnitt aus dem logischen Adressbereich mit fester Größe, beispielsweise 1 KB.
- **Rahmen** (frame) oder **Kachel**: Ein Ausschnitt aus dem physischen Adressbereich (also aus dem RAM) mit fester Größe, beispielsweise 1 KB
- **Block**: Der Nutzinhalt eines Sektors auf der Festplatte.
- **Memory Management Unit (MMU):** Bestandteil der CPU, der logische Adressen als Eingabe bekommt und daraus die zugehörige physische Adresse berechnet.

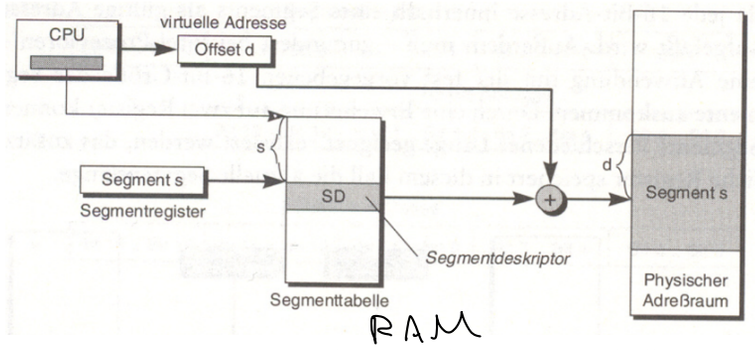
Seiten = Rahmen!!!

Wenn Seitengröße, Rahmengröße und Blockgröße gleich sind, dann lassen sich Seiten einfach in eine Kachel kopieren oder ein Kachelinhalt auf der Festplatte sichern oder . . .

Segment- und Seitenbasierte Adressierung

Segmentbasierte Adressierung (GDI)

- **Logische Adressen** (virtuelle Adressen) gehören immer zu einem Segment und werden über eine Segmenttabelle auf physische Adressen abgebildet.
- **Segmente**: Programmcode, statische Daten, dynamische Daten, usw. Heap, Stack etc.



- Das **Segmentregister** ist in der CPU, die Segmenttabelle wird im Hauptspeicher abgelegt *(und in der MMU zwischengespeichert).*
- Der **Segmentdeskriptor** enthält Informationen über:
  - Basisadresse des Segments, also wo das Segment im physischen Adressraum liegt.
  - Größe des Segments
  - Weitere Status-Bits (Zugriffsrechte, eingelagert vs. ausgelagert, usw.)

Vorteile:

- Segmente können einfach ein- und ausgelagert werden.
- Zugriffsschutz kann realisiert werden.
- Gemeinsame Nutzung von „Nur Lese“ Segmenten (z.B. Programmcode) durch mehrere Prozesse ist möglich.

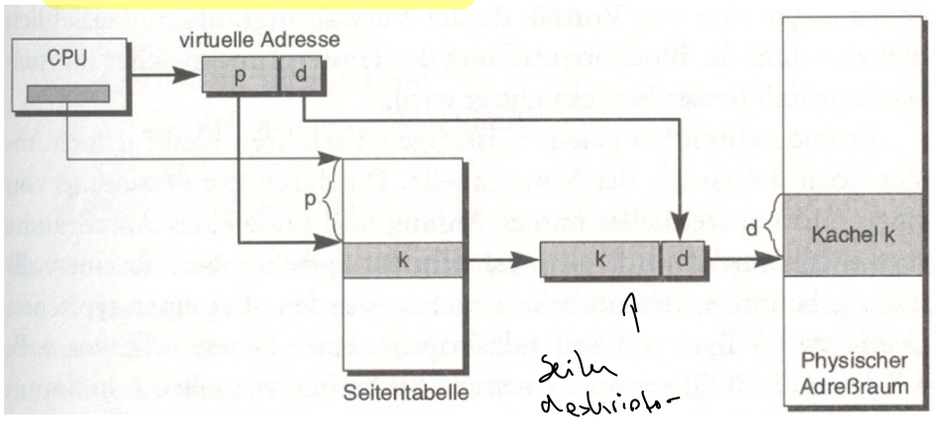
Nachteile:

- Fragmentierung des Speichers, da Segmentgrößen variabel. Hoher Aufwand beim ersten Zugriff in einem Segment.

Seitenbasierte Adressierung

Eine **logische (virtuelle**) Adresse besteht aus zwei Teilen:

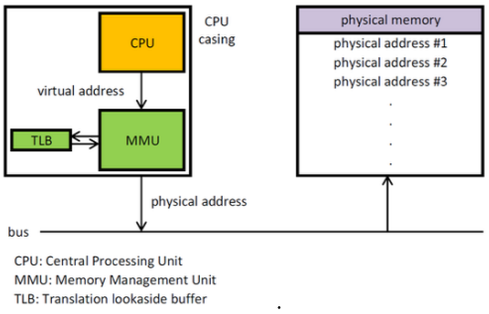
1. **Seiteninformation** (oder Adresse der Seite) p.
2. **Adresse** innerhalb der Seite d.



- Die Seitentabelle steht im Hauptspeicher und wird in der MMU zwischengespeichert (engl. to cache).
- **Seitendeskriptor** enthält Informationen über:
  - Seitenadresse
  - weitere Status-Bits (Schutzbits, Presence Bit, Reference Bit, Dirty Bit, usw.)
- **Vorteile**:
  - Keine Segmentregister erforderlich
  - wenig interne Fragmentierung.
- **Nachteile**:
  - Seitentabelle wird sehr groß (Abhilfe: Mehrstufiges Paging)
  - Spezielle Hardware auf der MMU erforderlich, damit Zugriffe schnell genug möglich sind (**Translation Lookaside Buffer, TLB**)

Memory Management Unit (MMU)

- Hardwarekomponente, die den Zugriff auf den Arbeitsspeicher verwaltet: Rechnet virtuelle Adressen in physische Adressen um.
- ~~Ursprünglicher eigener Baustein auf der Hauptplatine~~
- Heute in die CPU integriert
    - Führt Speicherschutzaufgaben aus
    - Prozesse dürfen nicht wahlfrei auf Speicher anderer Prozesse zugreifen (auch horizontale Trennung genannt); Voraussetzung für eine funktionierende Prozessisolation (engl. process isolation)
    - Anwendungsprozesse dürfen nicht beliebig auf den Speicher zugreifen, in welchem das Betriebssystem liegt (auch vertikale Hierarchie genannt); Voraussetzung für ein stabil laufendes Betriebssystem



- Schreib- und Lesezugriffe werden durch die MMU auf Gültigkeit geprüft

Translation Lookaside Buffer (TLB)

- speichert letzte Adressübersetzungen zwischen

Segment- oder seitenbasierte Adressierung?

- Ursprünglich wurden beide Konzepte umgesetzt
- Moderne Betriebssysteme verwenden meist seitenbasierte Adressierung

Auslagerungen

- Wird mehr Hauptspeicher benötigt als zur Verfügung steht, dann kann das Betriebssystem Teile eines Adressraums auf die Festplatte auslagern und dann für andere Prozesse benutzen.
- Auslagern ganzer Adressräume (Prozesse) wird auch **swapping** genannt.
- Beim Auslagern von Segmenten oder Seiten wird von **paging** gesprochen.
- Das Betriebssystem braucht einen Bereich auf der Festplatte, der hierfür reserviert ist.
    - Bei MS-Windows wird eine Datei pagefile.sys hierfür benutzt.
    - Bei UNIX-Systemen wird eine sogenannte Swap-Partition angelegt.
- **Vorteil**: Es können Programme ausgeführt werden, die mehr Hauptspeicher brauchen, als RAM im Rechner verbaut ist.

Fragen beim Paging

- ~~Wann werden einem gestarteten Programm Hauptspeicher-Kacheln zugewiesen und wann werden die ersten Seiten des Programms eingelagert?~~
    - Bei Bedarf (engl. demand paging) vs. vorab (engl. pre paging)
- Wie viele Kachel sollten einem Programm zugeordnet werden?
    - Arbeitsmenge (engl. working set)
- Welche Seite sollte ausgelagert werden, wenn alle Kacheln belegt sind und eine neue Seite eingelagert werden muss?
    - Optimale Strategie vs. LRU
        - Beispiele: Linux verwendet LRU, Windows verwendet FIFO

Demand Paging

Wenn beim Zugriff auf eine Adresse festgestellt wird, dass die zugehörige Seite nicht eingelagert ist wird das ein Seitenfehler (engl. page fault) genannt.

Dann passiert folgendes:

1. Die CPU löst einen Seitenfehler-Interrupt aus. Dauer: tI .
2. Das laufende Programm wird unterbrochen (blockiert) und das Betriebssystem muss evtl. eine Kachel auslagern, um Platz zu schaffen. Dauer: t_{AUS}
3. Das Betriebssystem lädt die referenzierte Seite von der Festplatte in den Hauptspeicher. Dauer: tEIN.
4. Die Seitentabelle wird aktualisiert (engl. to update) und das unterbrochene Programm kann fortgesetzt werden (wird wieder bereit oder rechnend). Dauer: t_{AKT}

Gesamtverzögerung ist im wesentlichen durch t_{EIN} und t_{AUS} bestimmt. Größenordnung: einige Millisekunden.

Seitenauslagerungsstrategien

- Optimale Strategie nach **Belady**: Lagere die Seite aus, die in Zukunft am längsten nicht mehr referenziert wird. (Allerdings: Betriebssystem können nicht hellsehen!)
    - Vorteil: Es muss insgesamt am wenigsten ein-/ausgelagert werden.
    - Nachteil: Es ist nicht bekannt, welche Seite am längsten nicht mehr benutzt wird.
- **Least Recently Used (LRU):** Lagere die Seite aus, die in der Vergangenheit am längsten nicht mehr benutzt wurde.
    - Annahme: Diese Seite wird wohl auch in Zukunft nicht mehr gebraucht werden.
- **First In First Out (FIFO)**: Lagere die Seite aus, die schon am längsten eingelagert ist.

Abstraktion für Programmierer*innen

- Die Adressaufteilung eines Prozesses und die Ausstattung eines Rechners mit RAM sind für den Programmierer egal. Variablen werden definiert und bei Bedarf wird weiterer Speicherplatz mit den Mechanismen der Programmiersprache angefordert.
- Trotzdem bleibt es wichtig, dass Programmierer*innen genau darauf achten, angeforderten Speicher auch wieder frei zu geben. Faustregel:
    - Zu jedem malloc in C gibt es auch ein free.
    - Zu jedem new in C++ gibt es auch ein delete.
- Als Programmierer*innen sollte trotzdem sparsam mit Speicher umgegangen werden und kein überflüssiger Speicher angefordert werden. Daten sollten außerdem nicht unnötig hin und her kopiert werden.
- Bei mehrdimensionalen arrays sollten Programmierer auch auf die richtige Schachtelung der Schleifen beim Durchlaufen achten.

Linux Hauptspeicherverwaltung

- Bei der Erstinstallation eines Linux-Systems wird eine sog. swap-Partition angelegt, wo der Kernel RAM-Rahmen auslagern kann.
- Mit dem Kommando free lässt sich die aktuelle Speicherauslastung anschauen. Beispiel:

```
wohlfeil@SSH:~$ free
             total       used       free     shared    buffers     cached
Mem:       4060972    1257956    2803016      82684      90852     995244
-/+ buffers/cache:     171860    3889112
Swap:       223228        112     223116
```

Zusammenfassung

- Prozesse benutzen logischen (virtuellen) Adressraum (0 . . . 2n − 1).
- Das Betriebssystem weist jedem Prozess einen Teil des physischen Speichers (RAM) zu, in dem dann der Prozess ausgeführt wird.
- Umrechnung der logischen Adressen in physische Adressen ist erforderlich. Sie wird von der Memory Management Unit (MMU) durchgeführt.
- Bei der nicht zusammenhängenden Speicherverwaltung stehen nur Teile eines Prozesses im RAM:
- Segment: logischer Teil des virtuellen Adressraums
- Seite: fester Ausschnitt (1 bis 8 KB) des virtuellen Adressraums Kachel: fester Ausschnitt aus dem RAM
- Reicht das RAM nicht, werden Seiten auf Festplatte ausgelagert (paging). Zuordnung von Kacheln zu Prozessen und gute Auslagerungsstrategien sind zu finden.

Inodes: spezielle Datenstruktur, welcher alle Metadaten über Dateien enthält & auf deren Blöcke verweist.
  Die Anzahl an Blöcken die ein inode referenzieren kann ist begrenzt.
  Dann wird der inode auf ein weitere Tabelle referenzieren, um mehr Blöcke anzusprechen
Vorteil: flexibel einsetzbar für kleine & große Dateien