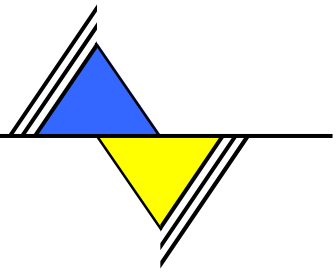
FACHHOCHSCHULE FÜR ÖFFENTLICHE VERWALTUNG UND RECHTSPFLEGE IN BAYERN

Fachbereich Allgemeine Innere Verwaltung



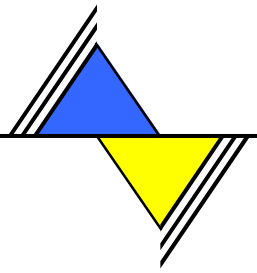
Betriebssysteme

8. Betriebsmittelverwaltung

Lehrveranstaltung im Studienschwerpunkt Verwaltungsinformatik

erstellt durch:

Name:	Karl Wohlrab
Telefon:	09281 / 409-279
Fax:	09281 / 409-55279
Email:	mailto: Karl.Wohlrab@fhvr-aiv.de



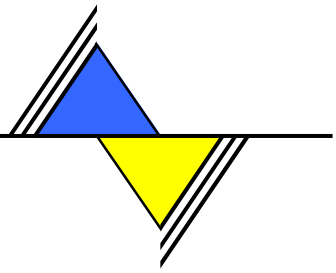
Inhalt

1 Betriebsmittelverwaltung 3
1.1 Vergabe von Betriebsmitteln ... 3

2 Speichermanagement 4
2.1 Swapping ... 6
2.1.1 Speicherverwaltung mit Bitmaps ... 7
2.1.2 Speicherverwaltung mit verketteten Listen ... 8
2.2 Virtueller Speicher ... 10
2.2.1 Paging ... 10
2.2.2 Seitenersetzungsalgorithmen ... 13
2.2.2.1 Der Not-Recently-Used Algorithmus (NRU) ... 14
2.2.2.2 Der First-In-First-Out-Algorithmus (FIFO) ... 15
2.2.2.3 Der Second-Chance-Algorithmus ... 15
2.2.2.4 Der Last-Recently-Used Algorithmus (LRU) ... 16
2.2.2.5 Seitenflattern ... 16

3 Ein- und Ausgabe 17
3.1 E/A-Geräte ... 17
3.2 Steuereinheiten und Controller ... 19
3.3 E/A-Software ... 20
3.4 Schichten der E/A-Software ... 20
3.4.1 Interrupt-Handler ... 21
3.4.2 Gerätetreiber ... 21
3.4.3 Geräteunabhängige Ein-/Ausgabe-Software ... 22
3.4.4 E/A-Software im Benutzeradressraum ... 22

4 Ausblick: Deadlocks 23
4.1 Bedingungen für Deadlocks ... 23
4.2 Deadlockerkennung und Behebung ... 23
4.3 Deadlockvermeidung ... 23



1 Betriebsmittelverwaltung

1.1 Vergabe von Betriebsmitteln

Für die Vergabe von Betriebsmitteln gelten im Grunde die gleichen Regeln wie sie bereits bei der Prozessverwaltung besprochen wurden. Das Thema soll deshalb hier nicht weiter vertieft werden.

Stichworte:

- TimeSharing
- Zyklische Vergabeverfahren
- Prioritätenregelung

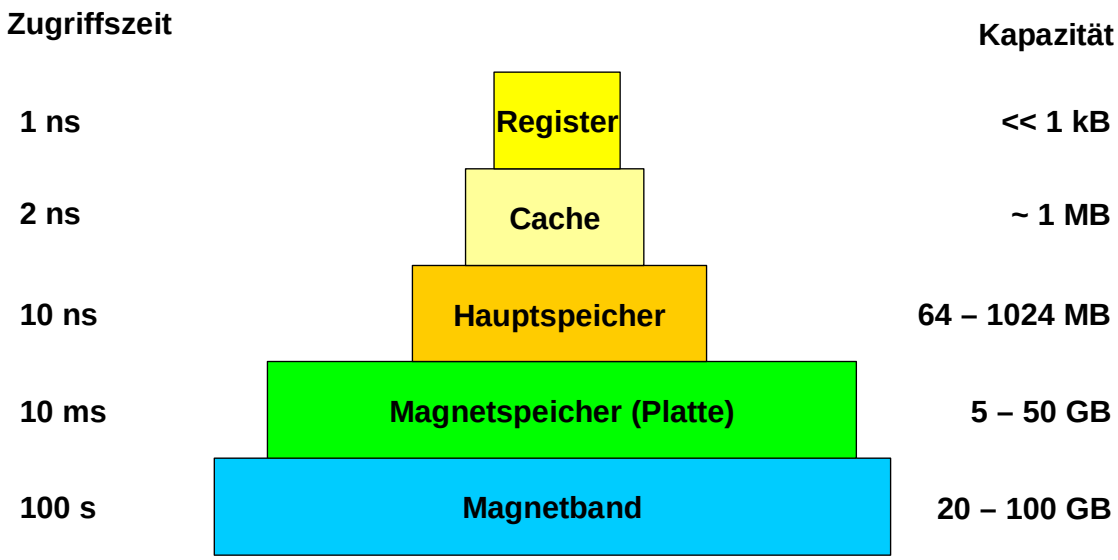
2 Speichermanagement

Literaturhinweise:

⌦ [Tann02] Kap. 4, Seite 209 ff

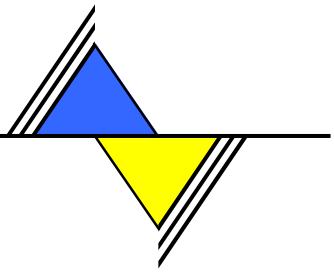
Der Speicher, genauer gesagt der Arbeitsspeichern ist neben der CPU eine der wichtigsten Ressourcen eines Rechensystems. Jeder Programmierer hätte natürlich am Liebsten ausreichend (d.h. unbegrenzt) Arbeitspeicher zur Verfügung. Dieser sollte auch noch beliebig schnell, nicht flüchtig und natürlich auch noch billig sein. Die Realität sieht aber leider ganz anders aus, Deshalb wird in den meisten Computern eine Speicherhierarchie aufgebaut, um die real existierenden Einschränkungen zu mildern und weitgehend zu verbergen.

Speicherhierarchie



Quelle: Tannenbaun 2002

Die Magnetspeicher (Platte und Magnetband) sind dabei vergleichsweise preiswert und in "nahezu" unbegrenzter Speicherkapazität verfügbar – der Adressraum ist sehr groß (potentiell unendlich). Die beiden Speichermedien gehören zu den nicht flüchtigen Speichertypen, d.h. die gespeicherten Daten bleiben nach Stromabschaltung weiterhin erhalten. Leider ist hier die Zugriffszeit nicht für eine schnelle Programmverarbeitung und damit ausreichende CPU-Auslastung geeignet.



Die Register (an der Spitze der Speicherhierarchie) haben die gewünschte Zugriffszeit (sie sind ja Teil der CPU) – sie sind jedoch flüchtig, in ihrer Kapazität sehr begrenzt, nicht erweiterbar und sehr teuer.

Als guter Kompromiss wurde deshalb der Arbeitsspeicher (RAM) eingeführt, der einen halbwegs schnellen Datenzugriff bei noch erschwinglichen Preisen ermöglicht.

Um den Anwendern bzw. Anwendungsprogrammen einen potentiell beliebig großen Arbeitsspeicher "vorzugaukeln" wird als Teil des Betriebssystems die Komponente "**Speicherverwaltung**" realisiert.

Die Speicherverwaltung verwaltet die Speicherhierarchie und teilt den Prozessen je nach Bedarf den benötigten Speicher zu bzw. gibt nicht mehr benötigten Speicher wieder frei. Die zentrele Komponente ist hierbei der Arbeitsspeicher (RAM)

BS-relevante Merkmale: Hauptspeicher

- **Absolute Größe**
- **für Anwender verfügbare Größe**
- **Zugriffsgeschwindigkeit**
- **Speicherhierarchie**

Für die Speicherverwaltung gibt es verschiedene Strategien.

2.1 Swapping

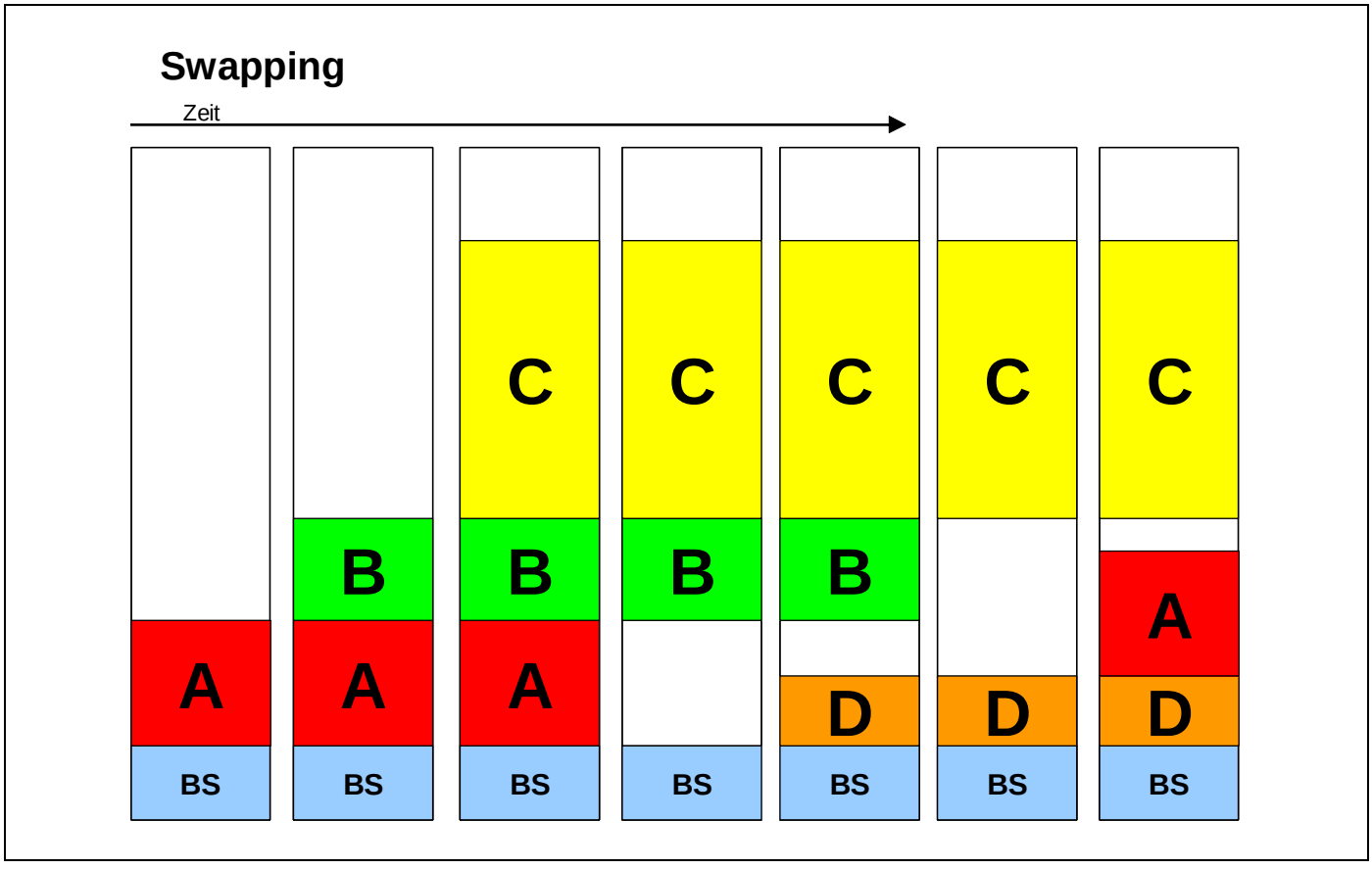
Literaturhinweise:

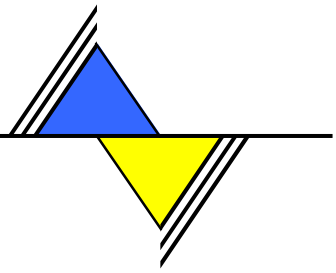
- [Tann02] Kap. 4.2, Seite 216 ff

Bei Swapping handelt es sich um eine relativ einfache Speicherverwaltungsstrategie:

- Der Arbeitsspeicher wird in feste Partitionen aufgeteilt.
- Jeder Auftrag wird in eine Partition geladen (Voraussetzung ist natürlich, dass der Auftrag am Anfang der Warteschlage steht)
- Der Auftrag bleibt solange in der Speicherpartition, bis er beendet ist.

Solange genügend Aufträge im Arbeitsspeicher Platz finden, um die CPU auszulasten gibt es keinen Anlass für eine komplexere Strategie. Bei Timesharing-Systemen und PCs mit graphischer Oberfläche ist jedoch im Normalfall nie ausreichend Arbeitsspeicher vorhanden, sodass hier entsprechende Speichervergabestrategien Verwendung finden müssen.





Swapping birgt jedoch einige Probleme:

- Beim Swapping wird jeder Prozess komplett in den Speicher geladen, darf eine gewisse Zeit laufen und wird dann komplett wieder auf die Festplatte ausgelagert. Dabei kann es aber vorkommen, dass durch die Auslagerung eines Prozesses Lücken entstehen, die durch den danach eingelagerten Prozess nicht komplett gefüllt werden. Bei längerer Anwendung des Verfahrens, entstehen so immer mehr kleine Lüchen, die für die weitere Nutzung nicht mehr brauchbar sind. Durch die so genannte **Speicherverdichtung** (memory compaction) können die vielen kleinen Löcher im Speicher wieder zu einem großen Loch zusammengefügt werden – dieses kann dann wieder für Prozesse genutzt werden. Aufgrund des vergleichsweise hohen Rechenzeitbedarfs wird Speicherverdichtung normalerweise nicht eingesetzt.
- Der genaue Speicherbedarf eines Prozesses ist normalerweise nicht genau zu prognostizieren. Als Abhilfe kann man jedem Prozess immer etwas mehr Speicher zuteilen, als aktuell benötigt – der Prozess hat also Platz zum Wachsen. Nachteilig ist dabei aber auch, dass häufig zu viel Speicher zugeteilt wird, der gar nicht genutzt wird und gleichzeitig evtl. an anderer Stelle fehlt. Beim Auslagern muss man dann natürlich dafür sorgen, dass nur der wirklich genutzte Speicher auf die Platte ausgelagert wird.

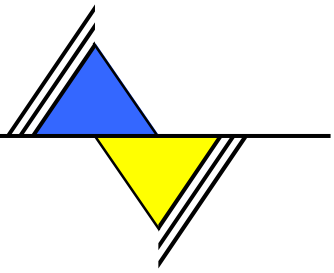
Abhilfe kann hier die dynamische Zuteilung des Speichers bringen.

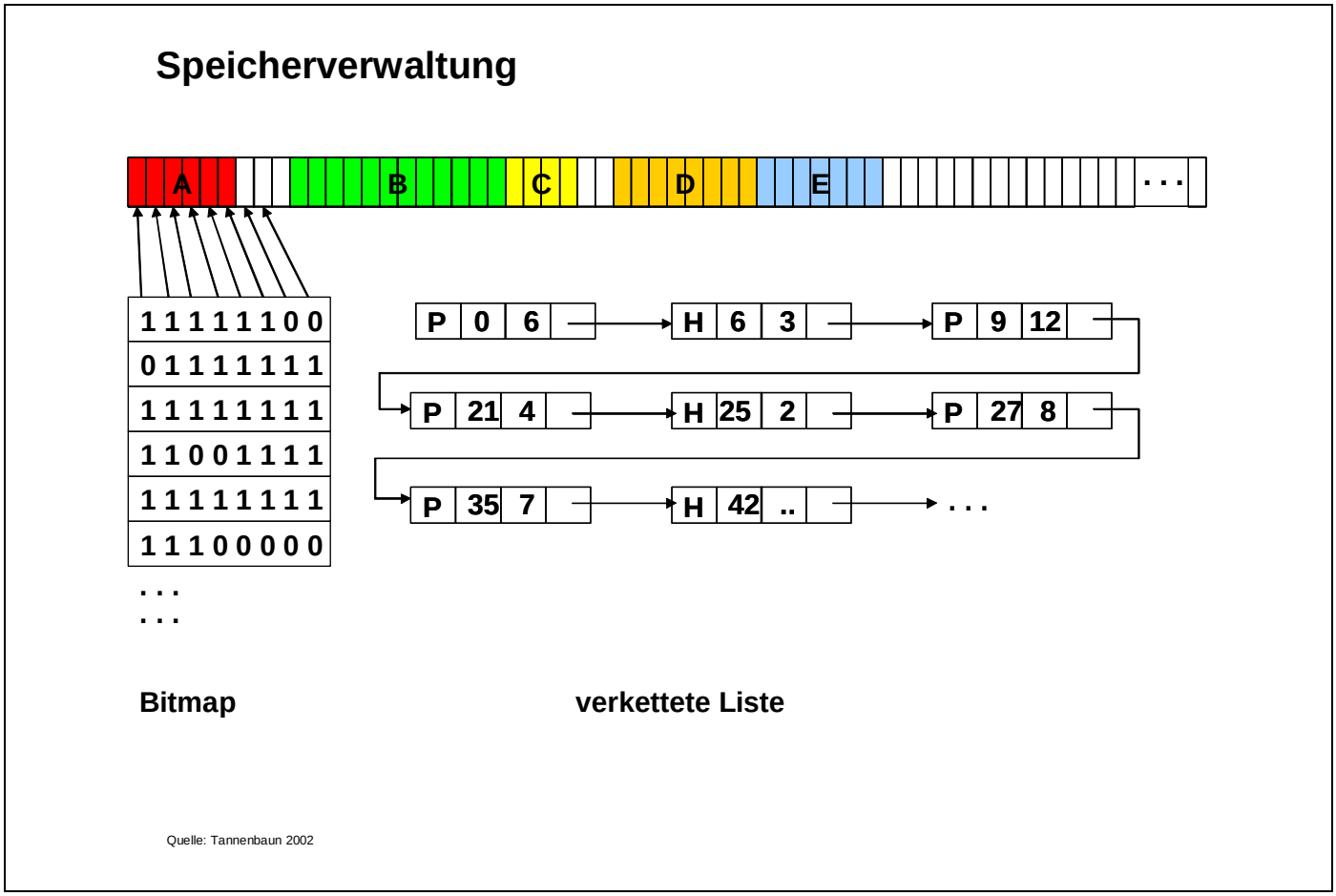
Die Verwaltung des Seichers erfolgt entweder durch
- Bitmaps oder mit Hilfe von
- Freispeicherlisten

2.1.1 Speicherverwaltung mit Bitmaps

Literaturhinweise:
- [Tann02] Kap. 4.2.1, Seite 219 ff

- Der Speicher wird in (gleich große) Allokationseinheiten eingeteilt
- Jeder Einheit entspricht ein Bit in einer Bitmap
  (0 bedeutet Einheit ist frei; 1 bedeutet Einheit ist belegt)
- Jedes mal wenn ein Prozess, der k Einheiten Platz benötigt in den ASP geladen werden soll, muss die gesamte Bitmap nach einer Folge mit k aufeinander folgenden freien Einheiten durchsucht werden.



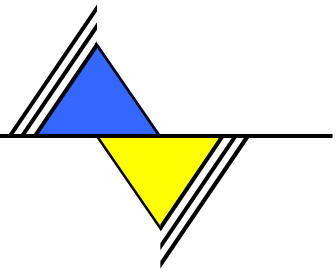


2.1.2 Speicherverwaltung mit verketteten Listen

Literaturhinweise:

- [Tann02] Kap. 4.2.2, Seite 220 ff

- Der Speicher wird zweckmäßigerweise in (gleich große) Allokationseinheiten eingeteilt
- Der Speicher wird als verkettet Liste von Speichersegmenten dargestellt.
- Jeder Eintrag der Liste kennzeichnet, ob es sich um ein Loch (H=hole) oder einen Prozess (P=process) handelt. Weiterhin enthält jeder Eintrag die Startadresse und Länge des Segmentes sowie einen Zeiger auf den nächsten Eintrag.



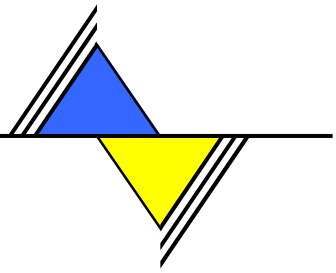
Wenn die Liste nach Speicheradressen aufsteigen geordnet ist gibt es verschiedene Vergabestrategien für die Zuweisung von Arbeitsspeicher. Gehen wir davon aus, dass die Größe des erforderlichen Arbeitsspeichers bei der Anforderung bekannt ist. Dann kann man folgende Vergabealgorithmen anwenden:

- **First Fit**
  Das erste freie Loch wird geteilt; der erste Teil wird dem anfordernden Prozess passend zugewiesen und der Rest wird als neues (kleineres) Loch in die Liste aufgenommen.
- **Next Fit**
  Eine Variation von First Fit; hier beginnt die Suche jedoch nicht immer vom Anfang der Liste siodern am gerade aktuellen Standort.
- **Best Fit**
  Hier wird bei jeder Anforderung die gesamte Liste durchsucht und das kleinste passende Loch für die Speicherzuteilung herangezogen.
- **Worst Fit**
  Analog zu Best fit, jedoch wird hier immer das größte zur Verfügung stehende Loch herangezogen

Bei allen vier Algorithmen lässt sich die Suche noch beschleunigen, indem getrennte Listen für freie und benutzte Speicherbereiche verwendet werden; Der Algorithmus für die Zuweisung bzw. Freigabe von Speicherkapazität wird dadurch jedoch komplexer.

Ein weitere Zuteilungsalgorithmus **Quick Fit** verwendet getrennte Liste für Löcher in gängigen Größen. Dadurch ist schon eine Vorselektion und damit eine kürzere Suchzeit möglich.

Die Verschmelzung mehrerer kleiner Löcher, die z.B.: bei der Terminierung oder Auslagerung vom Prozessen entstehen zu einem größeren Loch ist bei allen der obigen Algorithmen problematisch und aufwändig.



2.2 Virtueller Speicher

Literaturhinweise:
- ⌦ [Tann02] Kap. 4.3, Seite 222 ff

Die bisher genannten Speicherverwaltungsverfahren basieren alle auf der Grundvoraussetzung, dass der insgesamt vorhandene Arbeitsspeicher ausreicht, um ein Programm einschließlich aller seiner Module (Text, Data, Stack) aufzunehmen. Die Praxis zeigt jedoch, dass häufig Programme eine erheblich größeren Speicherbedarf haben, als durch den vorhandenen RAM abgedeckt werden kann.

Eine Lösung war es, die Programme in mehrere Abschnitte (sog. **Overlays**) aufzuteilen, die einzeln in den Arbeitsspeicher geladen werden konnten. Die Overlays wurden einzeln auf der Festplatte gespeichert und nur bei Bedarf (z.B.: durch einen entsprechenden Aufruf in einem anderen Overlay) in den ASP geladen. Die Aufteilung der Programme in Overlays mussten die Programmierer von Hand vornehmen – die gewählte Aufteilung war statisch.

Durch die Methode **virtueller Speicher** wurden die Programmierer von der manuellen Aufteilung entlastet – die Aufteilung übernahm das Betriebssystem.

Das Programm wurde einfach in gleich große Teile unterteilt und die gerade benötigten Teile bei Bedarf in den ASP geladen bzw. wieder entladen.

Dieses Verfahren eignet sich auch hervorragend für Multiprogrammingsysteme, denn wenn ein Programm auf die Einlagerung eines Teils wartet, wartet es auf I/O und kann somit unterbrochen werden und ein anderer Prozess kann zwischenzeitlich die CPU nutzen.

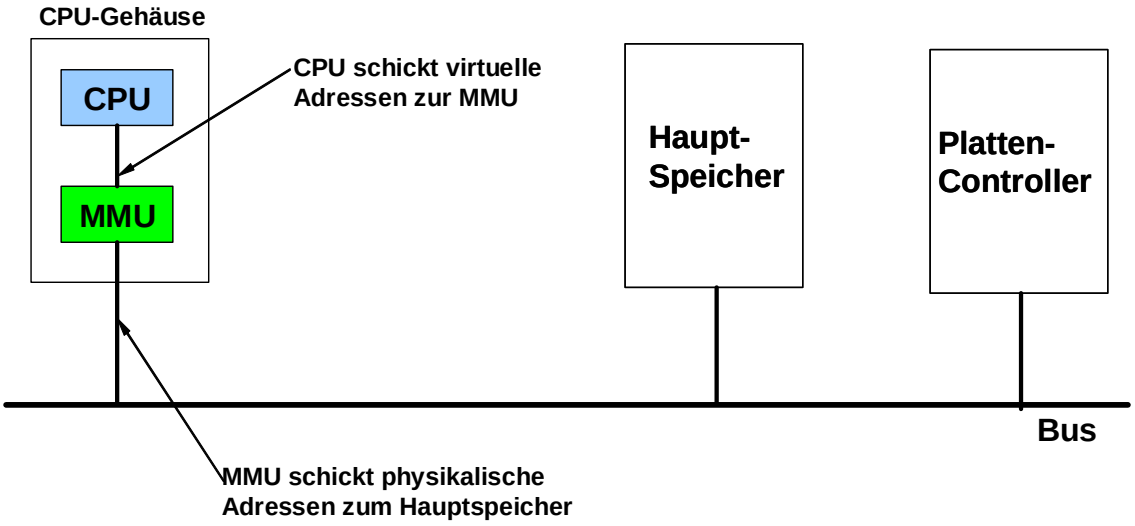
Die meisten Systeme mit virtueller Speicherverwaltung verwenden eine Technik namens Paging

2.2.1 Paging .

Literaturhinweise:
- ⌦ [Tann02] Kap. 4.3.1, Seite 222 ff

Beim Paging-Verfahren werden Speicheradressen mit Hilfe von Indizierung, Basisregistern und Segmentadressen generiert. Dies vom Programm generierten Adressen heißen **virtuelle Adressen**, sie bilden den **virtuellen Adressraum** des Programms. Eine spezielle Hardwarekomponente, die **MMU** (**Memory Management Unit**) bildet beim Zugriff auf einen Speicherplatz die virtuelle Adresse auf die **physische Adresse** des Arbeitspeichers ab, Die MMU ist meist mit auf dem CPU-Chip integriert (sie ist aber nicht Bestandteil der CPU; bei früheren Systemen war die MMU als eigenständiger Chip realisiert.

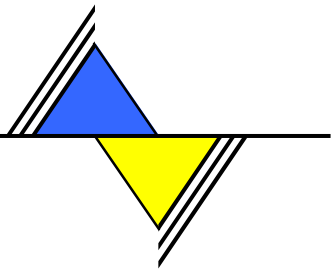
CPU und MMU



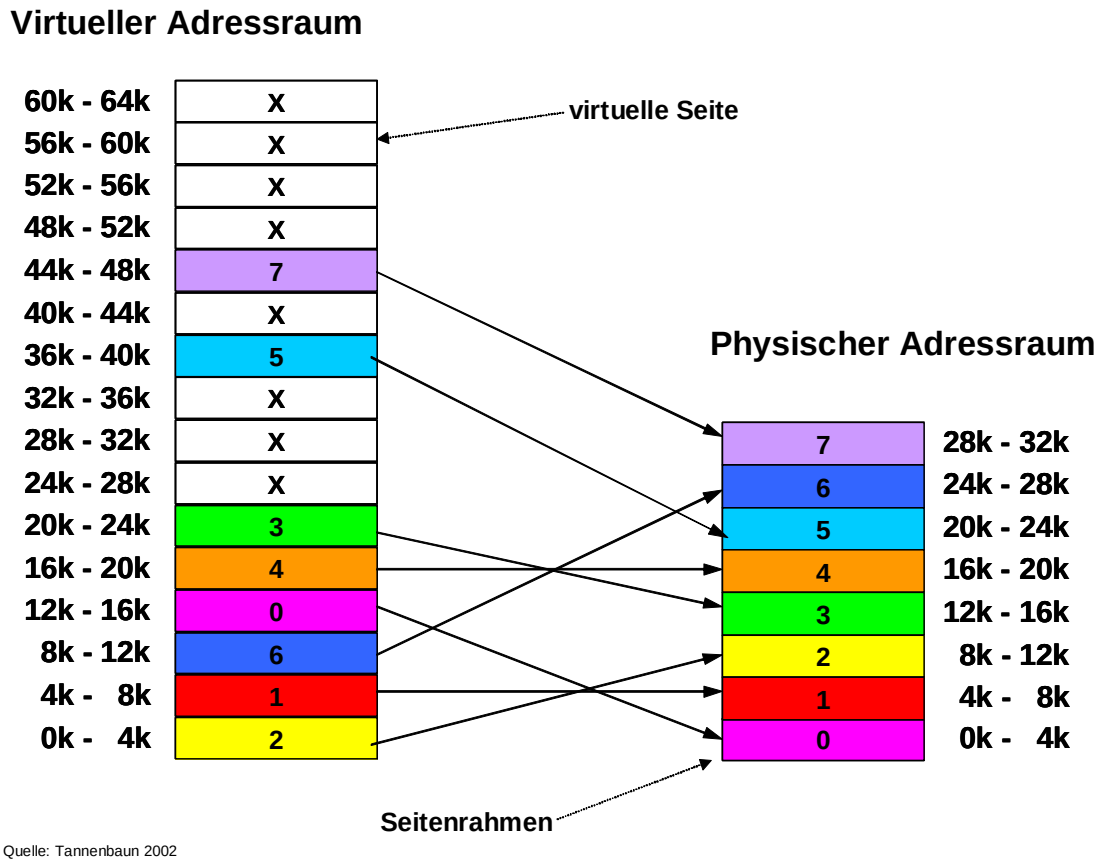
Quelle: Tannenbaun 2002

Grundzüge das Paging-Verfahrens:

- Der virtuelle Adressraum ist in gleich große Einheiten, die **Seiten** (**pages**) unterteilt
- Die den Seiten entsprechenden Einheiten im physischen Speicher heißen **Seitenrahmen** oder **Seitenkacheln** (**page frames**)
- Seitenrahmen und Seitenkacheln sind immer gleich groß
  Übliche Werte aus der Praxis liegen zwischen 512 Byte und 64 KB pro Seite/Seitenrahmen
  (für die weiteren Betrachtungen gehen wir von einer Seitengröße von 4 kB aus).
- Zwischen Arbeitsspeicher und Festplatte werden immer ganze Seiten übertragen.
- Die Beziehung zwischen virtuellen und physischen Adressen wird in einer Seitentabelle festgelegt
- … Fortsetzung siehe unten



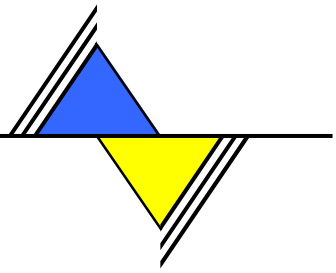
Paging: Seitentabelle



Quelle: Tannenbaun 2002

Beispiel:

- Ein Programm will auf die virtuelle Adresse 0 zugreifen, also wird die virtuelle Adresse 0 am die MMU geschickt.
- Die MMU stellt fest, dass die virtuelle Adresse zur Seite 0 gehört (0-4095)
- Das entsprciht dem Seitenrahmen mit der Nummer 2 (1892-12287)
- Die virtuelle Adresse 0 wird von der MMU auf die physische Adresse 8192 abgebildet und gibt den Wert 8192 als Adresse der zu lesenden Speicherstelle auf den Speicherbus.
- Der Speicher führt den gewünschten Zugriff auf die Speicherstelle 8192 aus.



Grundzüge das Paging-Verfahrens (Fortsetzung):

- Falls das Programm versucht, auf eine Seite zuzugreifen, die sich nicht im physischen Speicher befindet stellt dies die MMU fest und löst einen Systemaufruf aus. Dieser Systemaufruf wird **Seitenfehler** genannt.
- Das Betriebssystem sucht einen (evtl. wenig genutzten) Seitenrahmen aus und schreibt seinen Inhalt auf die Platte zurück. Man spricht auch von "**Auslagerung** der Seite"
- Danach wird die gewünschte Seite in den jetzt freigewordenen Seitenrahmen geladen, die Seitentabelle aktualisiert und der unterbrochene Befehl noch einmal ausgeführt.
- In der Realität wird in der Seitentabelle ein zusätzliches Bit (das **present/absent-Bit**) mitgeführt, das anzeigt, ob eine Seite gerade im Arbeitsspeicher eingelagert ist oder auch nicht.
- Die Seitennummer wird als Index für die Seitentabelle benutzt

2.2.2 Seitenersetzungsalgorithmen

Literaturhinweise:

- [Tann02] Kap. 4.4, Seite 234 ff

Ein nicht triviales Problem stellt der Algorithmus dar, der im Bearfsfall (bei Seitenfehler) auswählt, welche Seite ausgelagert werden muss. Hierbei sind einige Aspekte von nicht unerheblicher Bedeutung:

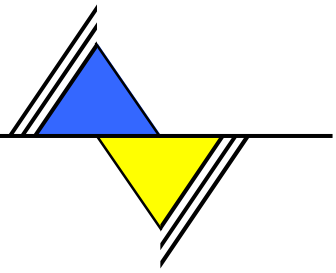
- Wenn die auszulagernde Seite seit ihrer Einlagerung modifiziert wurde (z.B.: durch einen Schreibzugriff), so muss sie auf die Platte zurückgeschrieben werden (Zeitbedarf).
- Wurde die Seite seit ihrer Einlagerung nicht verändert (z.B.: unveränderbarer Programmcode), dann kann das Zurückschreiben entfallen.
- Es ist ungünstig Seiten auszulagern, die im nächsten Arbeitstakt wieder benötigt werden. Viel benutze Seiten sollten demzufolge möglichst nicht ausgelagert werden

Das Problem der Seitenersetzung tritt nicht nur bei der Verwaltung des Arbeitsspeichers auf sondern auch an deren Stellen bei der Computerverwaltung (z.B.: beim Cache-Memory)

Der optimale Seitenalgorithmus ist zwar leicht zu beschreiben, seine Implementierung ist jedoch nahezu unmöglich:

- Wenn ein Seitenfehler auftritt befindet sich ein bestimmte Menge von Seiten im ASP.
- Für jede dieser Seiten kann (theoretisch) bestimmt werden, nach wie vielen Befehlszyklen auf sie zugegriffen wird.
- Jede der Seiten wird mit der Anzahl der Befehle markiert, die noch vergehen, bis auf die Seite zugegriffen wird.
- Der optimale Algorithmus entfernt die Seite mit der höchsten Zahl der Befehle, um den Seitenfehler der beim Zugriff auf die auszulagernde Seite entsteht möglichst lange hinauszuschieben.

Der Algorithmus ist jedoch nicht realisierbar, da im Moment des Seitenfehlers, das Betriebssystem noch nicht weiß, wann auf welche Seite das nächste Mal zugegriffen wird. In realen Systemen werden deshalb Algorithmen, eingesetzt die mit großer Wahrscheinlichkeit dem optimalen Algorithmus möglichst nahe kommen.



2.2.2.1 Der Not-Recently-Used Algorithmus (NRU)

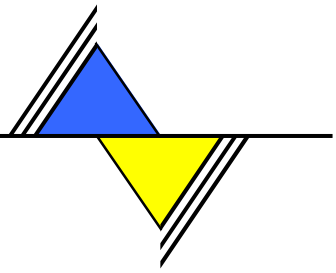
Literaturhinweise:
- ⌦ [Tann02] Kap. 4.4.2, Seite 236 ff

Hier wird eine Seite zur Auslagerung ausgewählt, die nicht kürzlich genutzt wurde.

Kurzbeschreibung:
- ⌦ Im Rechner werden mit Hilfe von 2 Statusbits die Zugriffe auf Seiten überwacht. Diese Statusbits sind Teil des Seitentabelleneintrags, sie werden von der Hardware bei jedem Zugriff auf die betreffende Seite gesetzt.
- ⌦ Das R-Bit wird gesetzt, wenn auf die Seite zugegriffen wird (Schreib- und Lesezugriff)
- ⌦ Das M-Bit wird gesetzt wenn auf die Seite schreibend zugegriffen wird (Modified)
- ⌦ Wenn ein Prozess gestartret wird, setzt das Betriebssystem die R- und M-Bits für alle Seiten auf 0.
- ⌦ In bestimmten Zeitabständen (z.B.: bei jeder Timerunterbrechung) werden alle R-Bits gelöscht. Damit wird sichergestellt, dass nur bei Seiten die seit der letzten Timerunterbrechung benutzt wurden das R-Bit gesetzt ist.
- ⌦ Bei einem Seitenfehler teilt das Betriebssystem alle Sieten nach den Zustand der R- und M-Bits in vier Klassen ein:
  - Klasse 0: R=0, M=0: nicht referenziert, nicht modifiziert
  - Klasse 1: R=0, M=1: nicht referenziert, modifiziert
  - Klasse 2: R=1, M=0: referenziert, nicht modifiziert
  - Klasse 3: R=1, M=1: referenziert, modifiziert

  Achtung: Klasse 1 bedeutet, dass die Seite in der letzten Periode nicht referenziert wurde, sie wurde aber bereits in einer früheren Periode modifiziert,
- ⌦ Der NRU-Algorithmus entfernt eine zufällige Seite aus der niedrigsten nicht leeren Klasse.

Vorteile des NRU-Algorithmus:
- \+ Leicht verständlich
- \+ Leicht und effizient zu implementieren
- \+ Zwar nicht optimal aber in vielen Fällen ausreichend



2.2.2.2 Der First-In-First-Out-Algorithmus (FIFO)

Literaturhinweise:
- ⌦ [Tann02] Kap. 4.4.3, Seite 237 ff

Hier wird diejenige Seite zur Auslagerung ausgewählt, die bereits am längsten im ASP verweilt.

Kurzbeschreibung:
- ⌦ Das BS verwaltet eine Liste mit allen Seiten im Speicher.
- ⌦ Am Kopf der Liste steht die die älteste Seite, am Ende der Liste die Seite, die zuletzt eingelagert wurde.
- ⌦ Bei einem Seitenfehler wird die Seite die am Kopf steht entfernt und die neue Seite am Ende der Liste angehängt.

Nachteile des FIFO-Algorithmus:
- Auch aktuell wichtige Seiten (Seiten, die ständig benötigt werden) werden ausgelagert
- Wird nur in seltenen Fällen angewandt

2.2.2.3 Der Second-Chance-Algorithmus

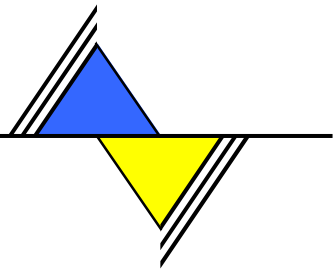
Literaturhinweise:
- ⌦ [Tann02] Kap. 4.4.4, Seite 237 ff

Dies ist eine Variante des FIFO-Algorithmus, die das Problem vermeidet, dass Seiten ausgelagert werden, obwohl sie häufig benutzt werden.

Kurzbeschreibung:
- ⌦ Bei einem Seitenfehler wird zunächst das R-Bit der ältesten Seite überprüft.
- ⌦ Ist das R-Bit nicht gesetzt, wird die Seite nicht nur alt sondern auch schon längere Zeit nicht genutzt und wird ausgelagert
- ⌦ Ist das R-Bit der ältesten Seite gesetzt, wird es gelöscht und die Seite an das Ende der Liste verschoben und die Ladezeit so gesetzt, als wäre die Seite soeben neu geladen worden. Danach wird die Suche fortgesetzt

Nachteile des Second-Chance-Algorithmus:
- Wenn alle Seiten referenziert wurden (R-Bit gesetzt) wird zunächst für jeden einzelne Seite der obige Algorithmus angewandt.
- Die ursprünglich am Anfang stehende Seite steht dann wieder am Anfang (aber jetzt mit zurückgesetzten R-Bit) und wird ausgelagert.
- Für die Anwendung des Algorithmus wurde jedoch ein nicht unerheblicher Anteil an Rechenzeit verbraucht.

Eine weitere (etwas optimalere) Variante ist der Clock-Algorithmus; hier werden die Seiten in einer Ringliste verwaltet und ein (Uhr-)Zeiger zeigt immer auf die älteste Seite.



2.2.2.4 Der Last-Recently-Used Algorithmus (LRU)

Literaturhinweise:
- ⌦ [Tann02] Kap. 4.4.6, Seite 239 ff

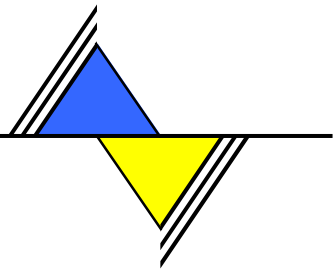
Dieser Algorithmus geht von der Annahme aus, dass eine Seite, die von den letzten paar Befehlen häufig benutzt wurde auch von den nächsten Befehlen benötigt wird. Umgekehrt werden Seiten, die schon lange nicht benutzt wurden mit großer Wahrscheinlichkeit auch nicht in nächster Zeit benötigt. Bei einen Seitenfehler ist also immer diejenige Seite zu entfernen, die am längsten nicht benutzt wurde.

Kurzbeschreibung:
- ⌦ Das BS verwaltet eine Liste mit allen Seiten im Speicher.
- ⌦ Am Kopf der Liste steht die Seite, die zuletzt genutzt wurde, am Ende der Liste die an längsten nicht mehr genutzte Seite.
- ⌦ Bei jeden Speicherzugriff muss die Liste wie folgt aktualisiert werden:
  - Die Seite in der Liste suchen,
  - die Seite an der gefundenen Stelle löschen und
  - die Seite an den Anfang der Liste verschieben.
- ⌦ Die Realisierung ist sehr aufwändig (und damit teuer)

2.2.2.5 Seitenflattern

Wenn die Seiten eines Prozesses ständig entladen und wieder neu geladen werden müssen (z.B.: weil der Rechner sehr stark ausgelastet ist und häufig Seitenfehler auftreten) wird die meist Zeit für die Ein- und Auslagerung von Seiten verbraucht; für die eigentliche Berechnung steht kaum noch Zeit zur Verfügung. Man spricht in diesem Fall von **Seitenflattern** (engl. **Thrashing**), da hier die Seiten ständig zwischen Arbeitsspeicher und Festplatte "hin und her flattern". Die fatale Konsequenz des Seitenflattern ist, dass sich der Effekt immer mehr verstärkt und so der Anteil an nutzbarer Rechenleistung immer geringer wird.

Als pragmatische Abhilfemaßnahme kann man einen Prozess (an Besten den der die häufigsten Seitenfehler produzuiert) anhalten (oder notfalls auch abbrechen) und einige Zeit warten, bis sich die Engpasssituation wieder auflöst und das Seitenflattern nachlässt.



3 Ein- und Ausgabe

Literaturhinweise:

⌦ [Tann02] Kap. 4, Seite 292 ff

Neben der Steuerung von Prozessen und der Verwaltung des Arbeitsspeichers ist natürlich auch die Ein- und Ausgabe eine wichtige Aufgabe des Betriebssystems.

3.1 E/A-Geräte

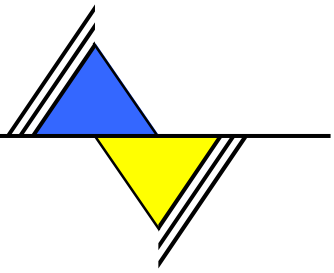
E/A-Geräte werden in zwei Hauptklassen eingeteilt:

- blockorientierte Geräte und
- zeichenorientierte Geräte

Hierbei sind jedoch auch Ausnahmen möglich. So gibt es Geräte, die keiner der beiden Klassen zuzuordnen sind (z.B.: Uhren, die lediglich in festen Zeitabständen Unterbrechungen auslosen). Derartige Sonderfälle sollen jedoch im Rahmen der vorliegenden Veranstaltung nicht weiter betrachten werden.

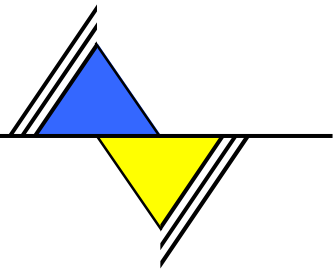
blockorientierte E/A-Geräte

- **Informationen werden in Blöcken mit fester Größe gespeichert**
- **Jeder Block hat eine eigene (feste) Adresse**
- **Jeder Block kann unabhängig von allen anderen Blöcken gelesen werden**
- **Schreib-/Leseoperationen betreffen immer einen ganzen Block**
- **gängige Blockgrößen: 512 Byte … 32.768 Bytes**
- **Typische Vertreter: Plattenspeicher**



zeichenorientierte E/A-Geräte

- **Informationen werden zeichenweise gespeichert**
- **Daten werden als Zeichenstrom gelesen bzw. gespeichert**
- **Keine Berücksichtigung einer Blockstruktur**
- **Der Datenstrom ist nicht adressierbar und kennt keine Suchoperationen**
- **Typische Vertreter: Zeilendrucker, Netzwerkkarten, graph. Zeigegeräte (z.B.: Maus), Tastatur, …**



3.2 Steuereinheiten und Controller

Die E/A-Komponenten eines Rechner bestehen im Normalfall aus einer mechanischen und einer elektronischen Komponente. Da es häufig möglich ist, diese beiden Komponenten zu trennen, kann man die E/A-Einheiten modular aufbauen.

Die elektronische Komponente bezeichnet man meist als Controller oder Adapter, die bei PCs meist als Einsteckkarte (oder onBoard) realisiert wird. Beispiele sind:

- Video-Karte / Controller
- ISDN- oder DSL-Karte
- Festplatten-Controller (inzwischen meist onBoard)
- …

Die Controllerkarte ist meist über ein Kabel mit dem Gerät verbunden; häufig kpnnen von einen Controller 2, 4 oder 8 Geräte verwaltet werden.

Die Schnittstelle zwischen Controller und dem Gerät ist meist standardisiert (ANSI, IEEE, ISO) oder zumindest nach einen de-facto-Standard (z.B.: DIE, EIDE, SCSI, USB, …) realisiert. Damit können dann eine Vielzahl unterschiedlicher Geräte von dem Controller bedient werden.

Beispiel Festplatte lesen:

- Spur mit 256 Sektoren zu je 512 Bytes
- Von der Festplatte kommt ein serieller Bit-Strom
  - Vorspann (Zylinder- und Sektornummer, Sektorgröße, …)
  - 4096 Bit (=512 Bytes) des Sektors
  - Prüfsumme oder fehlerkorrigierender Code (ECC)
- Controller konvertiert den Bit-Strom in Byte-Blöcke und führt ggf. erforderliche Fehlerkorrekturen durch. Dies findet typischerweise in in einem Puffer innerhalb des Controllers statt.
- An das Betriebssysytem wird der fertige Block (512 Bytes) übergeben.

3.3 E/A-Software

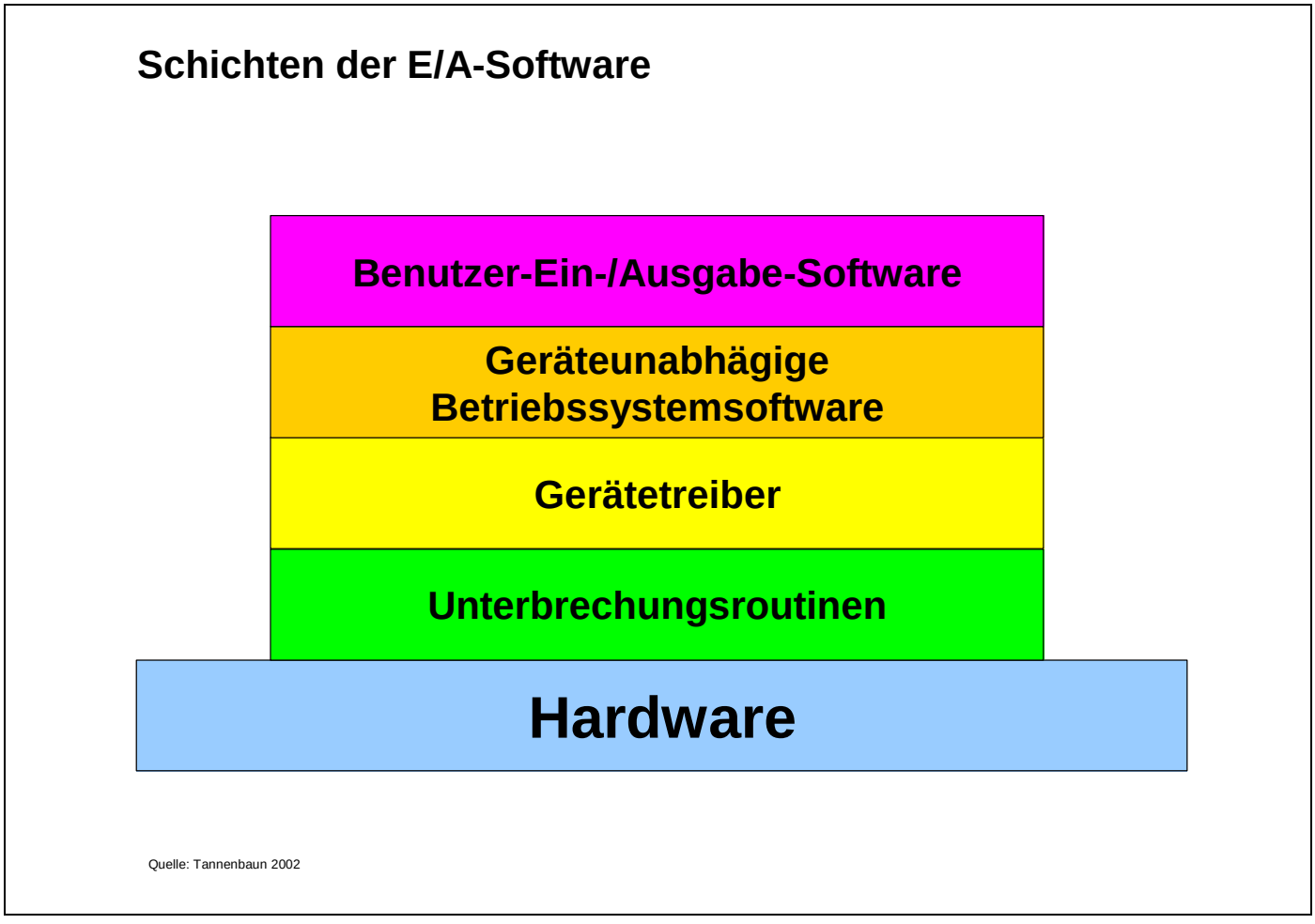
Es ist das Ziel, die Ein-/Ausgabe möglichst geräteunabhängig zu realisieren.

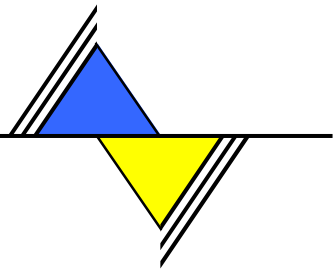
Beispiel: Es sollte möglich sein, Programme zu schreiben, denen es egal ist, ob eine Datei auf Festplatte, Diskette, CD-ROM oder Magnetband steht.

Es ist Aufgabe des Betriebssystems diese unterschiedlichen Anforderungen zu behandeln und dem Anwenderprogramm eine einheitliche (und standardisierte) Zugriffsmöglichkeit zu bieten. Hierfür wird spezielle E/A-Software entwickelt die auch die Gerätetreiber enthält.

3.4 Schichten der E/A-Software

Die E/A-Software wird i.R. in vier Schichten realisiert:





3.4.1 Interrupt-Handler

Unterbrechungen (Interrupts) werden erforderlich, wenn z.B.: Störungen am Gerät auftreten (Beispiel: Papier fehlt beim Drucker). Diese müssen erkannt und bearbeitet werden.

Der Ablauf einer Unterbrechungsbehandlung ist im Grundsatz wie folgt:

Interrupt-Behandlung

1. **Hardware sichert Befehlszähler und Registerinhalte im Kellerspeicher**
2. **Hardware holt neuen Befehlszähler vom Interruptvektor**
3. **Assemblerroutine speichert Register in Prozesskontrollblock**
4. **Assemblerroutine erzeugt neuen Kellerspeicher**
5. **Unterbrechungsroutine wird abgearbeitet**
6. **Scheduler sucht den nächsten zur Verarbeitung anstehenden Prozess**
7. **Assemblerroutine lädt Befehlszähler vom Prozesskontrollblock wieder in CPU**
8. **neuer aktueller Prozess wird gestartet**

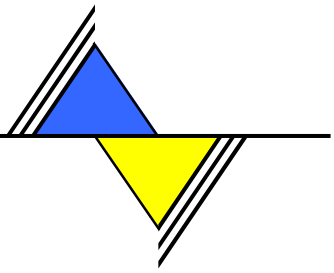
Quelle: Tannenbaun 2002

3.4.2 Gerätetreiber

Die Gerätetreiber (Device-Files) sind die gerätespezifischen Steuerprogramme. Gerätetreiber werden i.R. vom Gerätehersteller programmiert und mit dem Gerät zusammen ausgeliefert. Für eine Vielzahl von Standard-Geräten sind die Gerätetreiber auch bereits im Lieferumfang des Betriebssystems enthalten.

Der Gerätetreiber muss Zugriff auf die Hardware und die Register des Controllers haben, d.h. im Normalfall muss er im Kernel des Betriebssystems integriert werden. Fehlerhafte Treiber sind deshalb auch eine häufige Quelle für Systemabstürze.

Die Gerätetreiber bieten gegenüber der geräteunabhängigen E/A-Software eine definerte und standardisierte Schnittstelle an.



3.4.3 Geräteunabhängige Ein-/Ausgabe-Software

Die geräteunabhängige E/A-Software nutzt die standardisierte Schnittstelle zum Gerätetreiber um Ein- und Ausgaben mit dem gewünschten Gerät durchzuführen. Aufgaben der geräteunabhängigen E/SOftware sind:

Geräteunabhängige E/A-Software

- **Uniforme Schnittstelle für Gerätetreiber**
- **Pufferung**
- **Fehlerreport**
- **Anforderung und Freigabe von Geräten**
- **Service: geräteunabhängige Blockgröße**

3.4.4 E/A-Software im Benutzeradressraum

Dies sind Programme und/oder Programmbibliotheken, die zu Benutzerprogrammen hinzugebunden werden. Aufgabe dieser Programme ist es, die Parameter für den Systemaufruf (d.h. den Aufruf der zugeordneten Softwarekomponente des Betriebssystems beritzustelle und dann die gewünschte Komponente aufzurufen.

4 Ausblick: Deadlocks

4.1 Bedingungen für Deadlocks

4.2 Deadlockerkennung und Behebung

4.3 Deadlockvermeidung

Semaphore

####################################
####################################