Betriebssysteme

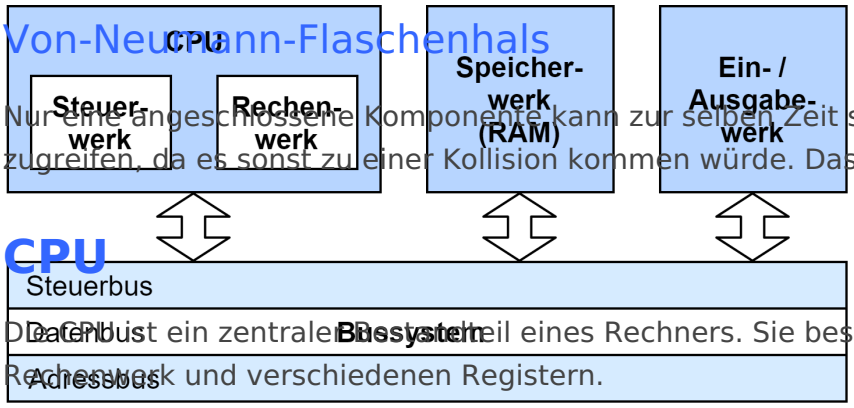
- Computerarchitektur
- Betriebssysteme
- Prozessverwaltung
- Speicherverwaltung
- Geräteverwaltung
- Dateiverwaltung

Computerarchitektur

Die Computerarchitektur ist ein Teilgebiet der Informatik, welche sich mit dem internen sowie externen Aufbau eines Computersystems beschäftigt.

Von-Neumann-Architektur

Die Von-Neumann-Architektur bildet ein Referenzsystem für Computer und besteht aus einer CPU mit Steuerwerk und Rechenwerk, einem Ein- / Ausgabewerk, einem Speicherwerk und einem Bus-System. Das Bus-System verbindet die Komponenten miteinander.



Von-Neumann-Flaschenhals

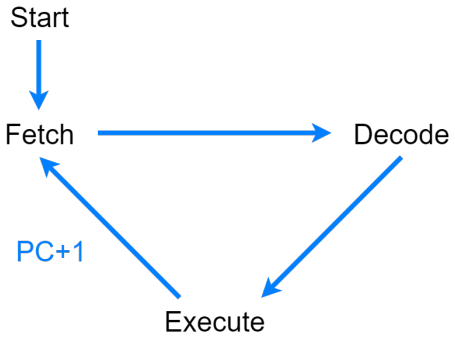
Nur eine angeschlossene Komponente kann zur selben Zeit schreibend auf das Bussystem zugreifen, da es sonst zu einer Kollision kommen würde. Das macht das Gerät insgesamt langsam.

CPU

Die CPU ist ein zentraler Bestandteil eines Rechners. Sie besteht aus dem Steuerwerk, dem Rechenwerk und verschiedenen Registern.

Steuerwerk

Das Steuerwerk dient der sequentiellen Abarbeitung von Programmen.



Fetch

- Adresse aus Befehlszähler auslesen → RAM
- Befehl aus Steuerwerk → RAM
- RAM liefert Daten zurück
- Daten → Befehlsregister

Decode

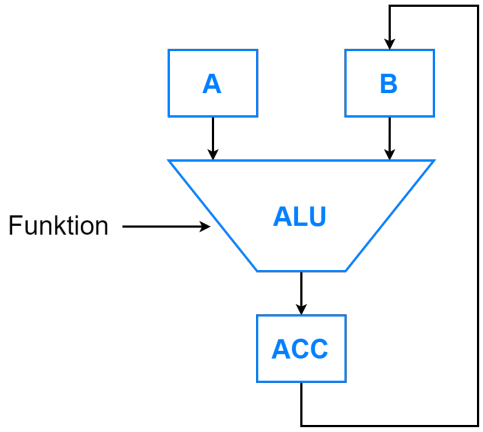
→ Befehl decodieren, Weichen stellen

Execute

→ Befehl ausführen

Rechenwerk

Das Rechenwerk führt vom Steuerwerk in Auftrag gegebene Berechnungen durch.



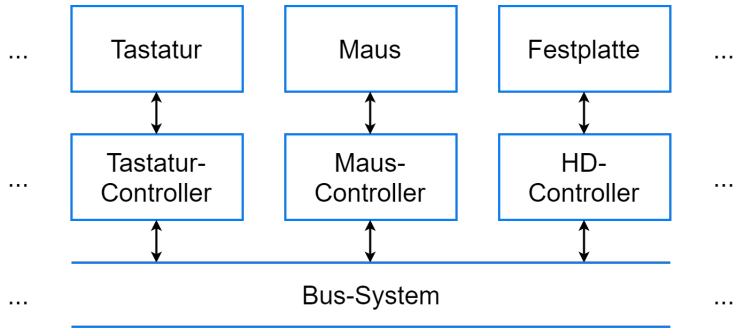
In der ALU werden Funktionen durchgeführt. Die Werte werden schließlich im Akkumulator zwischengespeichert, wo sie ausgelesen werden können.

Speicherwerk

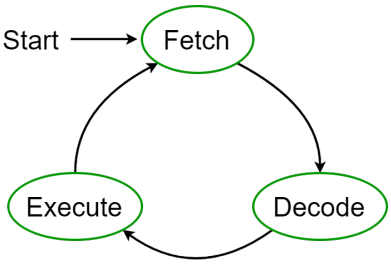
Das Speicherwerk ist in eine endliche Anzahl gleichgroßer Speicherzellen unterteilt. Jede Zelle verfügt über eine eindeutige Adresse. Es kann den Wert einer adressierten Speicherstelle auslesen oder einen Wert in eine Speicherzelle schreiben.

Ein- / Ausgabewerk

Das Ein- / Ausgabewerk dient der Ein- und Ausgabe von Informationen über verschiedene Peripheriegeräte (Maus, Tastatur, Monitor, ...). Für jedes Gerät gibt es einen Controller.



Von-Neumann-Zyklus



Fetch:

- Sende Adresse des Befehls an das Speicherwerk
- Empfange aktuellen Befehl vom Speicherwerk
→ Hole aktuellen Befehl

Decode:

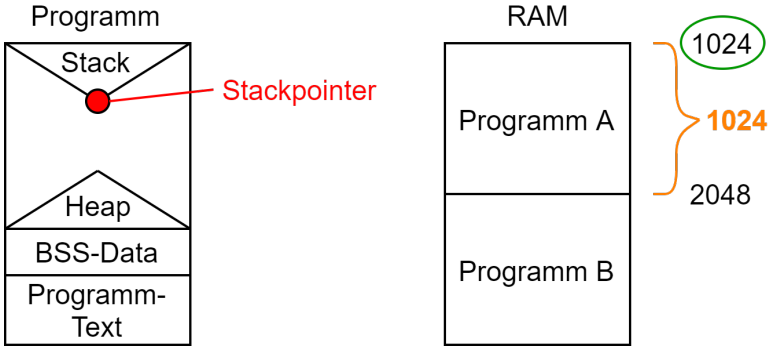
- Analysiere Befehl und treffe Vorbereitungen
- Erhöhe Programcounter (um 1)

Execute:

- Führe den Befehl aus

Weitere Komponenten der Von-Neumann-Architektur

Register



Im Stackregister wird der

Stackpointer

, der oberste Punkt des Stacks, gespeichert.

Das **Basisregister** legt die Adresse fest, zu der das Programm beginnt.

Im **Limitregister** wird die Größe des Speicherbereiches gespeichert.

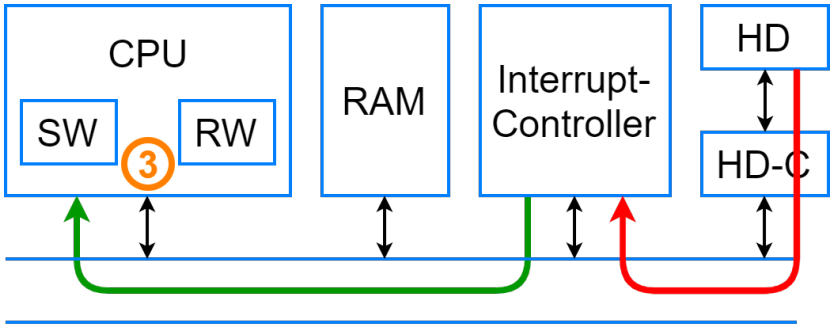
Um mehrere Programme im RAM zu halten, werden das Basis- und Limitregister benötigt.

Interrupt-Controller

Der Interrupt-Controller nimmt Interruptsignale entgegen und informiert die CPU über das Vorliegen eines Interrupts.
Die CPU arbeitet den Interrupt mit der Interruptbehanldungsroutine ab.

Die Interruptbehandlungsroutine ist eine Reihe von Anweisungen, die einem bestimmten Interrupt zugeordnet sind und auf der CPU abgearbeitet werden können.

Was passiert bei einem Interrupt?



1. HD löst Interrupt aus.

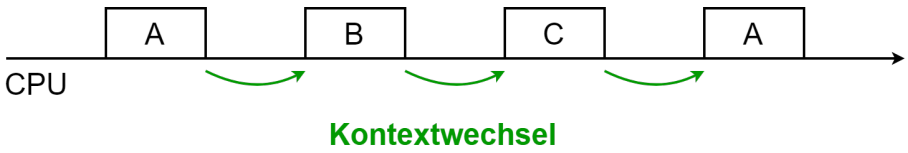
Interrupt wird an Interrupt-Controller gesendet und in Queue gespeichert.

2. Der Interrupt-Controller meldet den Interrupt dem Steuerwerk.

3. Nach der letzten Execute-Phase des aktuellen Prozesses werden aktuelle Registerinhalte gesichert. Die Interruptbehandlungsroutine wird ausgeführt.

Kontextwechsel

Damit mehrere Prozesse quasi gleichzeitig auf der CPU laufen können, müssen regelmäßig Kontextwechsel durchgeführt werden.

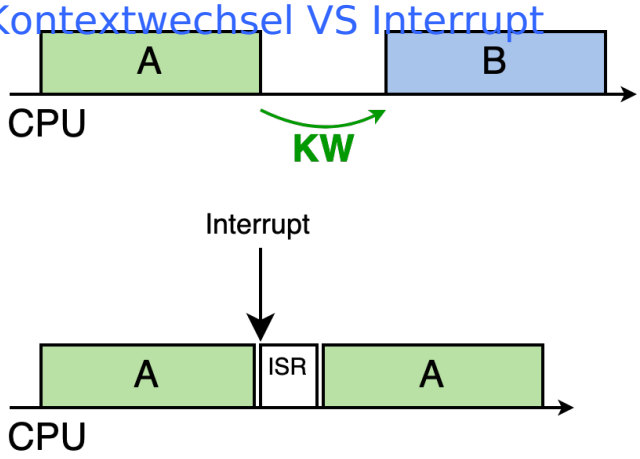


Während eines Kontextwechsels werden Registerinhalte des vorherigen Prozesses gesichert und Registerinhalte des neuen Prozesses geladen.

Zwischengespeichert werden die Registerinhalte auf dem Stack des jeweiligen gerade ausgeführten Programmes.

Der Stackpointer wird in einer bestimmten Verwaltungseinheit vom Betriebssystem hinterlegt.

Kontextwechsel VS Interrupt



Bei einem Kontextwechsel werden alle Registerinhalte eines Prozesses zwischengespeichert. Bei hunderten Registern ist das entsprechend zeitaufwendig.

Ein Interrupt setzt einen Prozess nur kurzzeitig aus und lässt die wesentlichen Registerinhalte im Prozessor unangetastet. Es werden nur eine Hand voll Registerinhalte kurzzeitig ausgetauscht und zwischengespeichert. Die Interruptserviceroutine läuft durch und anschließend wird der vorher laufende Prozess wird fortgesetzt.

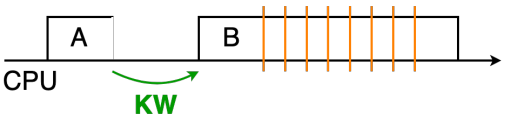
DMA-Controller

Der Direct-Memory-Access-Controller dient dazu, die CPU bei dem Datentransfer (z.B. HD zu RAM) zu entlasten.

Prozess ohne DMA

~~LHD~~ → 1. Teil laden (~~XXX~~ → Kontextwechsel)
STORE → Datenwort in RAM ablegen
~~LHD~~
...

Die CPU ist ca. 1 Mio mal schneller als die HD, daher wird ein **Kontextwechsel** zu einem anderen Prozess gemacht. Sobald ein Programmteil geladen wurde, wird ein **Interrupt** gesendet. Das führt zu vielen Interrupts in Prozess B.



Der DMA-Controller löst das Problem. Er agiert als Mini-CPU und kontrolliert den Datentransfer. Sobald das Programm vollständig in den RAM geladen wurde, sendet er einen Interrupt.
Solange kann die CPU an einem anderen Prozess weiterarbeiten.

Er benötigt dabei Quellsystem, Zielsystem, die jeweilige Startadresse und die zu übertragende Menge (Datenwörter).

Betriebssysteme

Ein Betriebssystem ist ein Programm, das dem Benutzer und Anwendungsprogrammen elementare Dienste bereitstellt. Es steuert und überwacht die Abwicklung von Programmen und regelt den Betrieb des Rechnersystems. Die zentrale Aufgabe eines Betriebssystems ist die Betriebsmittelverwaltung. Unter einem Betriebsmittel versteht man eine beliebige Hardware- oder Software-Ressource. Betriebsmittel sind Prozessen zugeordnet.

Kernel-Mode → Alle Befehle sind zugelassen

User-Mode → Nur bestimmte Befehle sind zugelassen, Zugriff auf bestimmte Dinge über **Systemaufrufe** (BS führt Befehle stellvertretend für Anwendungsprogramm aus)

Prozessverwaltung

> Ein **Prozess** ist ein Programm in Ausführung.

Prozesse erzeugen

Bei der Erzeugung eines Prozesses bekommt dieser eine eindeutige ID.

Windoof

- Die createProcess-Methode wird aufgerufen
- In 6 Phasen wird der Prozess initialisiert

Linux

- Die fork-Methode erzeugt eine exakte Kopie des aktuellen Prozesses
- Der Kindprozess wird mit eigenen Inhalten gefüllt, bekommt eine eigene P-ID → aushöhlen und neu befüllen

Prozesskontrollblock

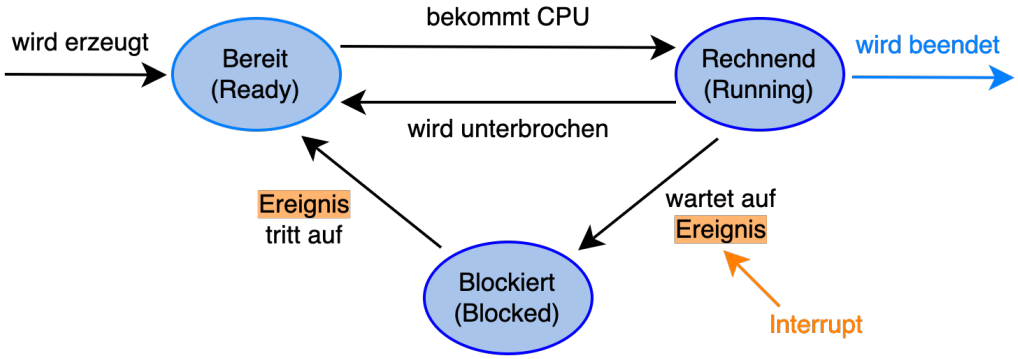
In einem Prozesskontrollblock fasst das Betriebssystem alle zu einem einzelnen Prozess gehörenden Informationen zusammen.
ID, Besitzer, Programm-Ausführdatei, Zustand, Liste der geöffneten Dateien

Prozesstabelle

In einer Prozesstabelle fasst das Betriebssystem alle Informationen aller erzeugten Prozesse zusammen. Es stehen alle Prozesskontrollblöcke untereinander. (z.B. Task-Manager)

Prozesszustände

Wir unterscheiden drei Prozesszustände.



Der Scheduler

entscheidet, welcher Prozess die CPU als nächstes bekommt.

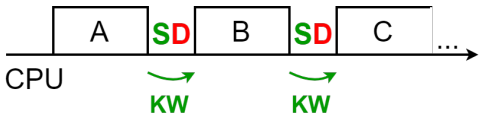
Thread

Unter einem Thread versteht man einen Teil eines Prozesses, der einen unabhängigen Kontrollfluss repräsentiert.

Threads wechseln sich durch **Threadkontextwechsel** immer wieder auf der CPU ab.
Dabei gibt es auch einen Threadscheduler.
Jeder Prozess hat mindestens einen Thread.

Scheduling

Unter Scheduling versteht man die Tätigkeit des Aufteilens der verfügbaren Prozessorzeit (CPU-Zeit) auf alle Prozesse.



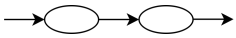
Der **Schedular** führt die Scheduling-Tätigkeit durch.

Der **Dispatcher** entzieht bei einem Kontextwechsel dem derzeit aktiven Prozess die CPU und teilt sie dem nächsten Prozess zu.

Die Zeit, die ein Prozess zusammenhängend auf der CPU hat, wird als **Quantum** bezeichnet. Als Programmierer hat man keine Ahnung, wie lang ein solches Quantum auf einer jeweiligen CPU dauert und wie viele Befehle man innerhalb von einem Quantum abarbeiten kann.

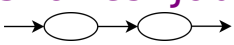
Scheduling-Verfahren

First Come-First Serve



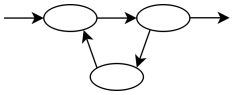
→ Die Prozesse laufen nacheinander vollständig auf der CPU ab.

Shortest Job First



→ Der kürzeste Prozess wird als erstes abgearbeitet. Die Zeit des Prozesses muss dafür vom Programmierer angegeben werden.

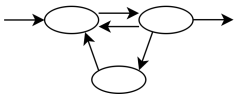
Shortest Remaining Time Next



→ Es bekommt der Prozess die CPU, der die kürzeste Restlaufzeit besitzt (Kontextwechsel, falls

Prozess warten muss).

Round-Robin



→ Nach Ablauf eines Quantums wird ein Kontextwechsel zu einem Prozess mit der nächst-höheren P-ID und dem Zustand "Ready" gemacht.
→ Basiert auf einer ringförmig verketteten Liste (Next des letzten Elements zeigt wieder auf das erste).

Priority Scheduling

→ gleiches Verhalten wie beim Round Robin, nur werden zusätzlich Prioritäten eingeführt.
→ Es werden zuerst Prozesse mit hoher Priorität abgearbeitet.
→ Prioritäten werden nach bestimmter Zeit runtergestuft.

Synchronisation

Bei der Synchronisation geht es um Mechanismen zur Vermeidung von Problemen, die bei der nebenläufigen Ausführung von Prozessen oder Threads in Verbindung mit gemeinsam genutzten Betriebsmitteln entstehen können.

Ein **Betriebsmittel** ist eine beliebige Hard- oder Software-Ressource, z.B. eine Variable, ein Drucker oder eine Datei.

Unter **Nebenläufigkeit** versteht man die quasi-parallele Ausführung von Befehlen unterschiedlicher Prozesse oder Threads auf der CPU.

Unter **Race Conditions** versteht man Situationen, bei denen zwei oder mehr Prozesse ein oder mehrere Betriebsmittel gemeinsam nutzen, und das Ergebnis der Ausführung von der zeitlichen Reihenfolge der Zugriffe der beteiligten Prozesse oder Threads auf das Betriebsmittel abhängt.

→ z.B. nutzen einer gleichen Variable (Kap. 3.2.11.1.2)

Unter einem **kritischen Abschnitt** versteht man Programmteile, die während ihrer Ausführung auf der CPU nicht durch kritische Abschnitte anderer Prozesse oder Threads unterbrochen werden dürfen, sofern die beteiligten Prozesse oder Threads auf gemeinsam genutzte Betriebsmittel zugreifen.

Aktives Warten

Aktives Warten ist das ständige Abfragen eines Sperrkennzeichens am Eingang eines kritischen Abschnitts.

```
while (lock == 1);  // Tue nichts
```

```
lock = 1;
// kritischer Abschnitt!
lock = 0;
```

Problem:
Ein Kontextwechsel im ungünstigsten Moment *(nach* `while (lock == 1);` *und vor Setzen der lock-Variable* `lock = 1;` *)* führt dazu, dass es trotzdem zu Fehlern kommen kann. Der kritische Abschnitt ist entsperrt, die Variable aber noch nicht.

TSL

Mit dem TSL-Befehl passieren zwei Dinge als **atomare Aktion**:

> **Atomare Aktion:** Nicht teilbar! KW kann entweder vor oder nach TSL Befehl passieren. Nicht zwischen dem Holen des Werts einer Speicherzelle und dem Überschreiben der Zelle mit dem Wert 1.

→ Wert aus Speicherzelle X in Akkumulator kopieren
→ Wert an Speicherzelle X mit 1 überschreiben (`lock = 1`)

> Das Problem des **ungünstigsten Moments** ist gelöst.

> Beim aktiven Warten wird CPU-Zeit vergeudet: Der Prozess ist 100% damit beschäftigt, die Bedingung zu prüfen und nichts zu tun.

Semaphore

Unter einem Semaphor versteht man eine Datenstruktur, welche einen ganzzahligen Zähler sowie eine Warteschlange bereitstellt. Zusätzlich sind zwei **atomare** Operationen `P()` und `V()` auf diese Datenstruktur definiert.

Semaphor (0, 1)
Mutex, binär
ein Betriebsmittel

Semaphor (x > 1)
Zählsemaphor
x Betriebsmittel

`P()`

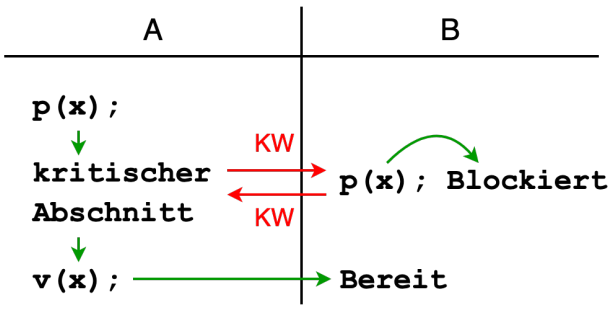
- Hat der Zähler einen positiven Wert, verringere ihn um 1
- Ansonsten blockiere den aufrufenden Prozess und füge ihn der Warteschlange hinzu

`V()`

- Ist die Warteschlange nicht leer, entblockiere den ersten Prozess
- Ansonsten erhöhe den Zähler um 1

Wechselseitiger Ausschluss

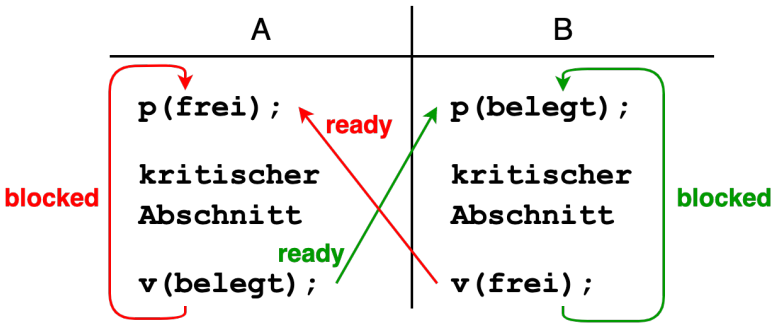
```
Semaphor x = new Semaphore(1);
```



Da A sich im kritischen Abschnitt befindet, wird B durch x blockiert (Zähler von x ist auf 0). Nachdem A den kritischen Abschnitt verlässt, wird B durch `v(x)` entblockiert.

Reihenfolge durch Setzung

```
Semaphor frei = new Semaphor(1);
Semaphor belegt = new Semaphor(0);
```



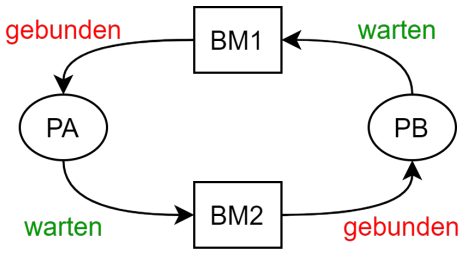
Bei zwei voneinander abhängigen Prozessen werden zwei Semaphore benötigt, sodass sich die Prozesse gegenseitig entblockieren.

Deadlocks

Eine Menge von Prozessen befindet sich in einem Deadlock-Zustand, wenn jeder Prozess aus der Menge auf ein Ereignis wartet, das nur ein anderer Prozess der Menge auflösen kann.

Beispiel: Prozess A und B benötigen BM1 und BM2. A hat bereits BM1 an sich gebunden, B hat BM2. Beide Prozesse warten nun auf das jeweils andere Betriebsmittel.

Betriebsmittelgraph



Vier Bedingungen nach Coffman

1. Mutual exclusion condition
   → Gegenseitiger Ausschluss
   → es steht nur ein BM zur Verfügung
2. Wait for condition
   → warten auf ein anderes BM
3. No preemption condition
   → Den Prozessen können bereits gebundene BM nicht entrissen werden
4. Circular wait condition
   → siehe Betriebsmittelgraph

> Theoretisch könnten Deadlocks vermieden werden, indem jedes BM eine Nummer bekommt und immer das BM mit der niedrigeren Nummer zuerst angefordert wird. (Bsp. Bank: 3.2.12.4 Aufg. 3)

> In der Praxis funktioniert das nicht: Zu Beginn ist nicht klar, welche BM vom Prozess benötigt werden.

> Deadlocks kommen sehr selten vor und es ist aufwendig, sie zu erkennen (kostet viel Rechenzeit).
> Daher werden sie in gängigen Betriebssystemen ignoriert.

Interprozesskommunikation

In Betriebssystemen müssen Möglichkeiten geschaffen werden, damit Prozesse und Threads miteinander kommunizieren können:

- gemeinsame Daten- und Speicherbereiche
- Sockets → Netzwerkverbindung; Pipes → Ausgabe von P1 = Eingabe von P2

Speicherverwaltung

Die Speicherverwaltung ist Teil des Betriebssystems und erledigt alle erforderlichen Arbeiten zur Verwaltung des physischen und des virtuellen Speichers eines Computers.

Grundlagen

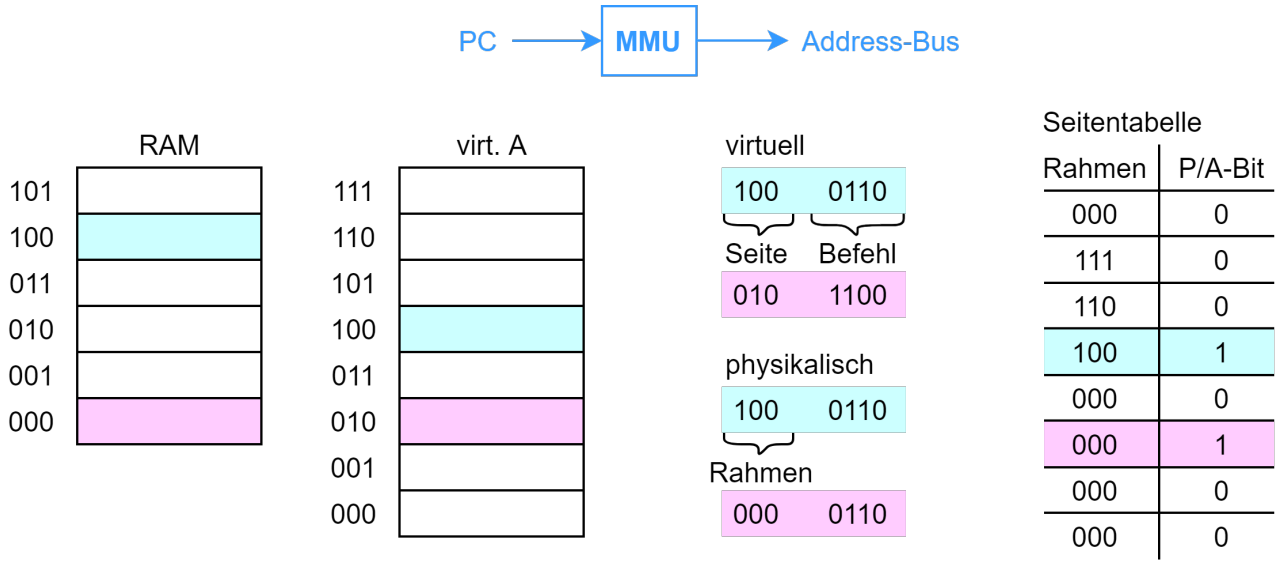
Virtuelle Speicherverwaltung

Prozesse werden in einem virtuellen Speicher gehalten, um im RAM nur tatsächlich benötigte Programmteile zu speichern. Die MMU (Memory Management Unit) rechnet die virtuellen Speicheradressen in die physischen Adressen um.

> Ein **Seitenrahmen** ist ein zusammenhängender Block von Speicherzellen des virtuellen Speichers (RAM).
>
> Eine **Seite** ist ein zusammenhängender Block von Speicherzellen des physischen Speichers (RAM). Die Größe einer Seite entspricht der Größe eines Seitenrahmens.

Seitentabellen

Zur Umrechnung der virtuellen Speicheradresse in die physische werden Seitentabellen benötigt.



Seitentabelle

Rahmen	P/A-Bit
000	0
111	0
110	0
100	1
000	0
000	1
000	0
000	0

Die **MMU** prüft in der Seitentabelle des aktuellen Prozesses den Eintrag an der entsprechenden Stelle aus der virtuellen Adresse.
Ist das `P/A-Bit = true (1)`, kann die Rahmennummer aus der Tabellenzeile entnommen werden.
Ist das `P/A-Bit = false (0)`, kommt es zu einem **Page-Fault (Seitenfehler)**. Die Seite muss zunächst vom DMA-Controller in den RAM-Speicher geladen werden. Ist der RAM bereits voll, kommen **Seitenersetzungsverfahren** ins Spiel.

> **Swapping**
>
> Kompletter Prozess wird ausgetauscht (Basis- & Limitregister, ...)
> Wenn der RAM voll ist wird ein gesamter Prozess in den HD Speicher verschoben, sodass Platz im RAM für einen neuen Prozess geschaffen wird.
> Ein kompletter Prozess ist entweder ganz eingelagert (RAM) oder ganz ausgelagert (HD)
> (Altes Verfahren)

Seitenersetzungsverfahren → Paging

Unter Paging versteht man das Ein- bzw. Auslagern von Teilen eines Prozesses.

Optimale Seitenersetzung

→ Welche Seite wird in der Zukunft am längsten nicht benötigt?
Die optimale Seitenersetzung kann nicht programmiert werden.

No Recently Used

→ Welche Seite wurde in der Vergangenheit am längsten nicht genutzt?
Es wird davon ausgegangen, dass diese Seite auch in Zukunft lange nicht benötigt wird.
Es wird ein Referenziert-Bit und ein Modifiziert-Bit eingeführt.

Das `R-Bit` wird `true (1)` gesetzt, wenn die Seite verwendet wurde. Es wird in regelmäßigen Abschnitten auf `false (0)` gesetzt.
Das `M-Bit` wird `true (1)` gesetzt, wenn die Seite verändert wurde.

In folgender Reihenfolge werden Seiten gesucht und ersetzt. Wenn:

1. `R = 0 && M = 0` (Seite nicht verwendet)
2. `R = 0 && M = 1` (Seite modifiziert, lange nicht verwendet)
3. `R = 1 && M = 0` (Seite verwendet, nicht modifiziert)
4. `R = 1 && M = 1` (Seite verwendet und modifiziert)

First-In-First-Out

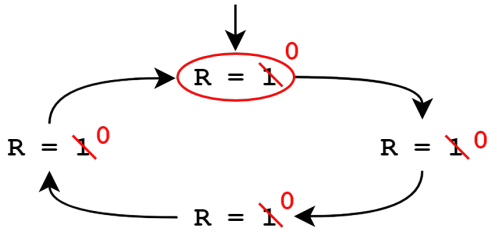
→ Die Seiten, die zuerst im RAM gespeichert wurden, werden als erstes überschrieben.

Working Set

→ Die wichtigsten Seiten, das "Working-Set" eines Prozesses werden eingelagert.

Second Chance

→ In einem kreisförmigen Durchlauf wird die Schlange der zuerst eingelagerten Seiten durchlaufen. Ist das `R-Bit = true (1)` wird es auf `false (0)` gesetzt und die Seite bekommt eine zweite Chance.



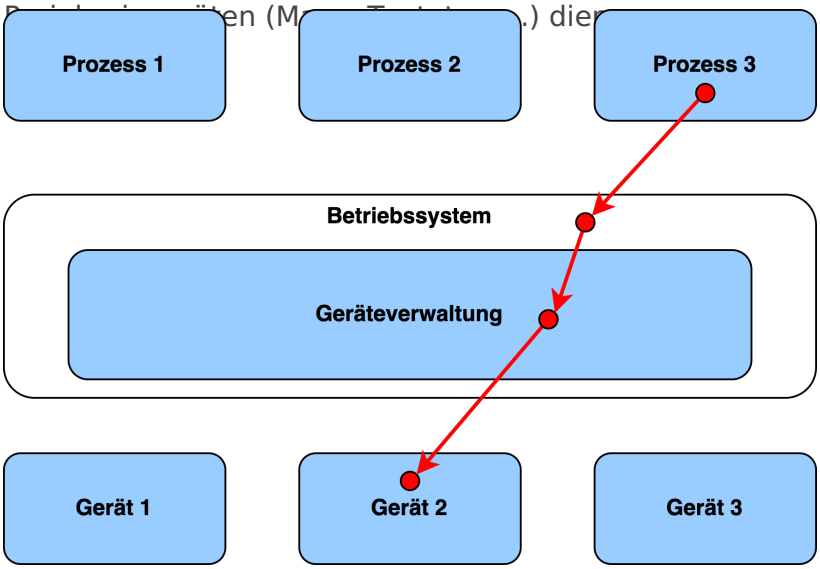
Sind alle `R-Bit = 1` wird der erste Prozess ersetzt (die Seite).

Shared Memory

Zur Interkommunikation können gemeinsame Speicherbereiche sinnvoll sein. Dazu wird der Seitenrahmen des RAMs in allen Seitentabellen aller beteiligten Prozesse hinterlegt.

Geräteverwaltung

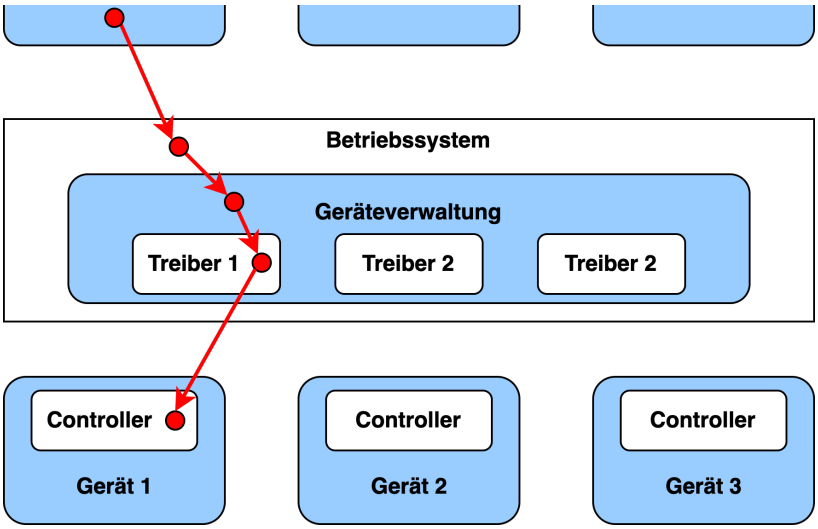
Unter der **Geräteverwaltung** fasst man alle Aufgaben und Tätigkeiten des Betriebssystems zusammen, welche einer optimierten Zusammenarbeit zwischen dem Betriebssystem und den Peripheriegeräten (M... T...) dien...



Geräte können nur über **Systemaufrufe** miteinander kommunizieren.

Gerätetreiber

Unter einem Gerätetreiber versteht man eine Software-Komponente, welche zur Geräteverwaltung des Betriebssystems gehört und Interaktionen zwischen Betriebssystem und Controller eines bestimmten Peripheriegeräts steuert.



Aufgaben des Gerätetreibers

- Initialisierung des Gerätecontrollers

- Gerät dem Betriebssystem bekannt machen

- Interruptbehandlung für ein Gerät

- Bereitstellen einer Schnittstelle zum Gerätecontroller und zur Geräteverwaltung

---

Geräteklassen

Bei der Implementierung der Geräteverwaltung unterscheidet das Betriebssystem unterschiedliche Geräteklassen.

Blockorientierte Geräte...

...übertragen Daten jeweils in ganzen Blöcken, was sowohl beim Lese- auch auch beim Schreibvorgang gilt. Jeder Datenblock ist direkt adressierbar.

Für die Zusammenarbeit mit der Geräteverwaltung implementiert der Gerätetreiber eines blockorientierten Gerätes folgende Funktionen:

- `initDevice()` → Gerät initialisieren
- `readBlock()` → Block lesen
- `writeBlock()` → Block schreiben
- `handleInterrupt()` → Interruptbehandlung

> Bsp. Festplatte: → Sektorgröße lesen und schreiben

Zeichenorientierte Geräte...

...erzeugen oder empfangen einen Datenstrom. Die einzelnen Zeichen sind dabei nicht adressierbar.

Für die Zusammenarbeit mit der Geräteverwaltung implementiert der Gerätetreiber eines zeichenorientierten Gerätes folgende Funktionen:

- `initDevice()` → Gerät initialisieren
- `readChar()` → Zeichen lesen
- `writeChar()` → Zeichen schreiben
- `handleInterrupt()` → Interrupt behandeln

> Bsp. Maus & Tastatur

Dateiverwaltung

Unter einer **Datei** versteht man einen Bestand an zusammengehörigen digitalen Daten, die dauerhaft auf einem geeigneten Speichermedium gespeichert sind.

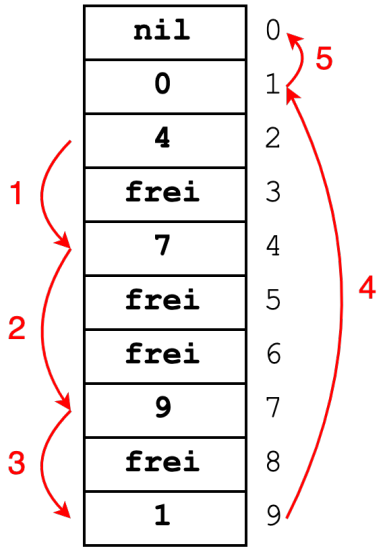
Dateisystem

Unter einem Dateisystem versteht man den Teil eines Betriebssystems, der die geordnete Ablage und das leichte Wiederfinden von Dateien auf geeigneten Speichermedien ermöglicht, sowie die erforderlichen Zugriffsmöglichkeiten auf die Dateien bereitstellt.

Aufgaben eines Dateisystems

- Verwaltung des freien und belegten Speicherplatzes
- Verwalten von Verzeichnissen
- Verwaltung von Zugriffsrechten und Attributen
- Strukturiertes Ablegen von Dateien

Dateisysteme unter Windows

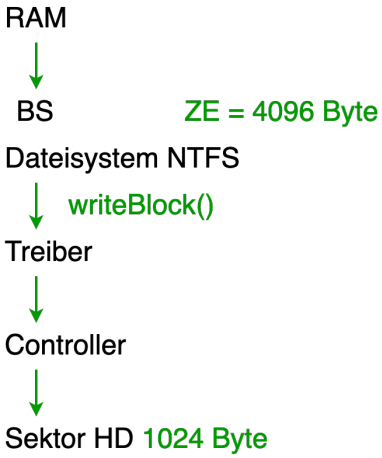


FAT - File Allocation Table

FAT ist eine Dateizuordnungstabelle die den Dateien Zuordnungseinheiten zuordnet und freie und belegte ZE zeigen kann.

1. Datei beginnt in ZE 2. Dies steht im Verzeichniseintrag des BS.
2. Datei geht weiter in ZE 4. Danach ZE 7 usw.
3. Datei endet in ZE 0.

NTFS - New Technology File System



Im Master File Table speichert das NTFS-Dateisystem Informationen zu allen Dateien und Verzeichnissen eines NTFS formatierten Volumen.

Eine Datei wird immer in Zuordnungseinheiten eingeteilt und gespeichert. Ist die Datei sehr klein, wird sie direkt im MFT gespeichert. Je kleiner die ZE, desto weniger Speicherplatz wird "verschwendet".