Inhalt

Betriebssysteme

Im engeren Sinne nur der Kernel

- **Kernel**: sehr klein, enthält Grundfunktionen
- **Pakete**: Rest der Betriebssysteminstallation

Beispiel: Bestandteile Linux

- **Linux-Kernel**: hardwarenahe Software für Scheduling, Multitasking, Gerätetreiber, Speicherverwaltung etc.
- **GNU-Paketen**: Shell, Compiler, Bibliotheken usw.

Aufgaben

- Prozessverwaltung
  - **Prozess**: ausführbares Programm – besteht aus Code, Daten und Ressourcen
  - Überwacht Prozesse
  - Erzeugt neue Prozesse
  - Regelt Kommunikation unter den Prozessen
- Speicherverwaltung
  - Verwaltung des Hauptspeichers
  - Speicherzuteilung zu Prozessen
- Verwaltung des Dateisystems
  - Erzeugung eines Namensraums mit dem auf das Dateisystem zugegriffen werden kann
- Verwaltung von Geräten
  - Zugriff auf Ein- und Ausgabegeräte mittels überwachter Aufrufe
  - Effiziente Aufteilung der E/A Geräte zu Prozessen

Ringe der CPU

Ring: Sicherheitsstufe eines Prozesses. Bestimmt nutzbaren Befehlssatz und Speicherbereich

- **Kernel-Mode**: Ring 0
- **User-Mode**: Ring 1-3

Ringe:

- 0: meisten Berechtigungen, direkte Hardwarezugriffe & Zugriff auf alle RAM-Bereiche
- 1: wird nicht verwendet
- 2: wird nicht verwendet
- 3: Anwenderprogramme: eingeschränkte Rechte (z.B. keine Interrupts – Programmunterbrechungen – erlaubt)

Arten

Allgemein

- **Batch-System**: z.B. DOS
- **Dialogsystem**: z.B. Windows, Linux, Unix
- **Echtzeitsystem**: für Industrieanwendungen

Anzahl der parallel bedienbaren Benutzer

- **Singleuser-System**: nur ein Nutzer
- **Multiuser-System**: mehrere Nutzer

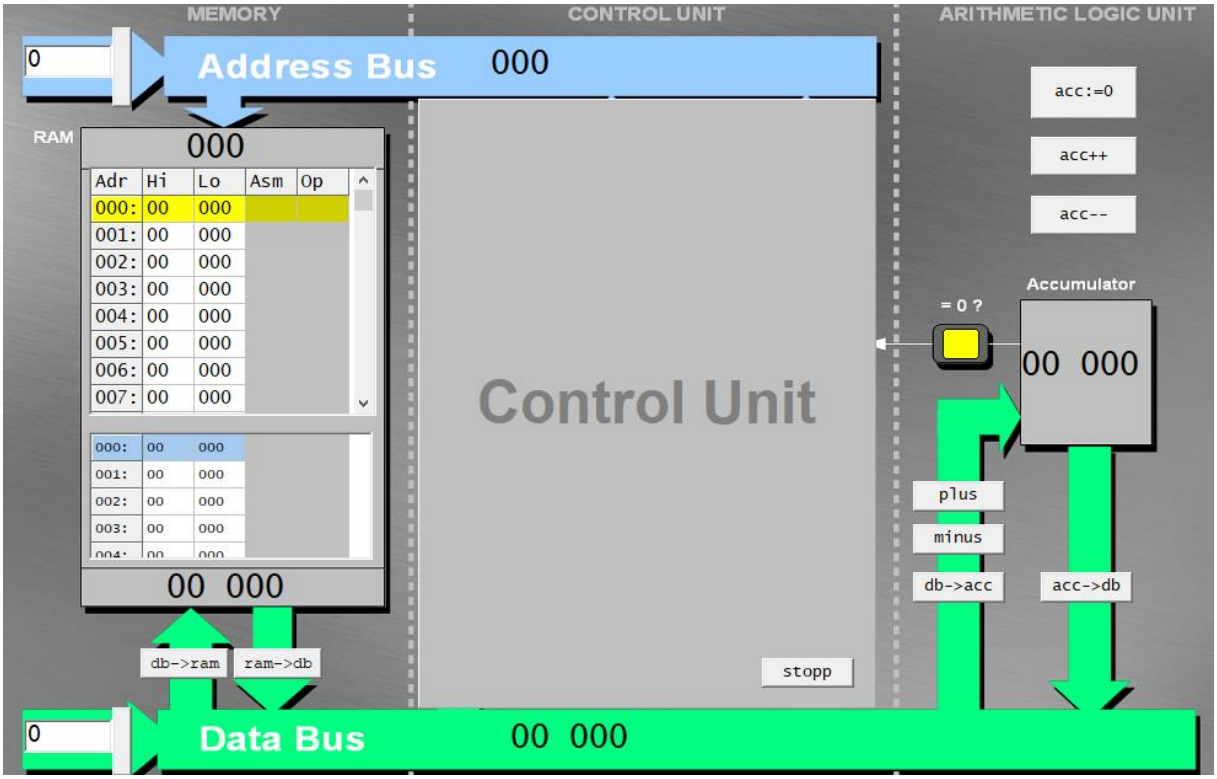
Anzahl der parallel bearbeitbaren Aufträge

- **Ein-Prozess-Betriebssysteme** (Single Tasking): DOS
- **Mehr-Prozess-Betriebssysteme** (Multi-Tasking): Linux, Windows

Rechnerarchitektur

Von-Neumann

Code und Daten des Programms liegen im gleichen Speicherbereich



Komponenten

- **Rechenwerk (ALU):** Ausführung von Befehlen (Addition, logische Operationen)
- **Steuerwerk**: Koordiniert die Abarbeitung des Programms (Laden von Befehlen, Operanden holen)
- **Bussystem**: Kommunikation zwischen den einzelnen Komponenten
  - *Adressbus*: Auf welche Adresse soll zugegriffen werden?
  - *Datenbus*: Liefert Daten
- **Speicher**: Arbeitsspeicher mit Daten und Code
- **Ein-/Ausgabewerk**: Schnittstellen zum Anwender oder anderen Systemen

Zyklus

5 Phasen:

- **Fetch**: Nächsten Befehl holen
- **Decode**: Befehl dekodieren (Mit Mikrocode – Befehlssatz der CPU)
- **Fetch Operands**: Benötigte Operanden holen
- **Execute**: Befehl ausführen
- **Write back**: Ergebnis zurückschreiben

Simplifizierte Rechnerarchitektur

Komponenten

- **ALU**: Führt Rechenoperationen aus
- **Mehrzweck- und Gleitkommaregister**: Spezielle Speicher mit denen die CPU rechnen kann
- **Befehlsregister**: Spezielles Register, in dem der Maschinencode des aktuell ausgeführten Befehls steht
- **Programmstatuswort/Statusregister**: Enthält Flags die ALU setzt (Überlauf, negatives Ergebnis etc.)
- **Stackpointer**: Spezielles Register, das auf die Adresse des obersten Elements des Programmstacks zeigt
- **Befehlszähler**: Register, das auf die Adresse des nächsten Befehls zeigt, der geholt werden soll
- **Memory Management Unit (MMU):** Verwaltet Zugriff auf den Arbeitsspeicher
- **L1/L2/L3 Cache:** Sehr schneller Pufferspeicher zwischen RAM und CPU – Speichert häufig benutze Daten

Maschinensprache

- Programmiersprache, die aus Elementen besteht, die der Prozessor direkt ausführen kann
- **Befehlssatz**: Menge der Befehle eines Prozessors
- **Mnenomics**: Für bessere Lesbarkeit von Maschinencode

Betriebssystemaufrufe

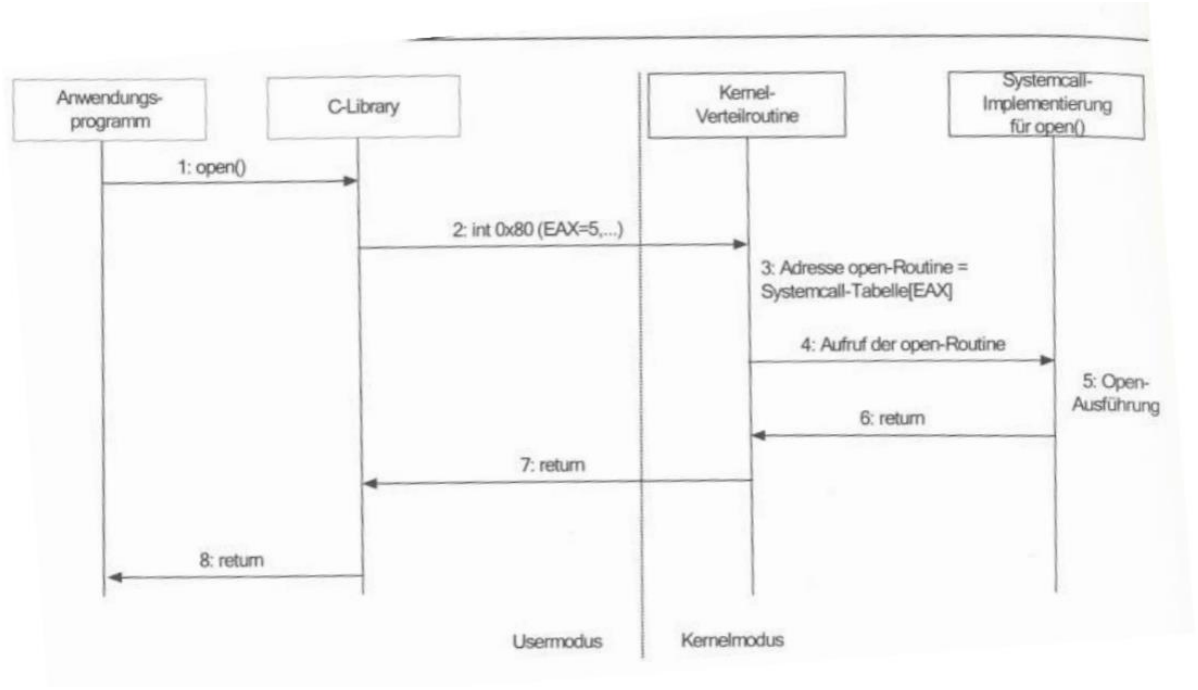
Ablauf

- Die zugehörige Interrupt-Service-Routine wird ausgeführt (dadurch wird Betriebssystem-Code ausgeführt)
- Es wird in den Kerne-Mode geschaltet
- Notwendige Überprüfungen werden durchgeführt (z.B.: dürfen nicht quasi-gleichzeitig mehrere Prozesse in eine Datei schreiben)
- Der Datenzugriff wird entweder erlaubt und durchgeführt oder verweigert
- Es wird in den User-Mode zurückgeschaltet
- Der zuvor angehaltene Prozess wird wieder gestartet
- Dabei wird ihm ein Rückgabewert mit dem Ergebnis des Systemaufrufs zur Verfügung gestellt

Interrupt: Signal, das von Gerät oder Thread an das OS geschickt wird, um rauszufinden, was als Nächstes zu tun ist:

- Software: Programm fordert Betriebsdienste an oder Gerätetreiber mit System interagieren wollen
- Hardware: Hardware benötigt OS (bewegen der Maus, drücken einer Taste)

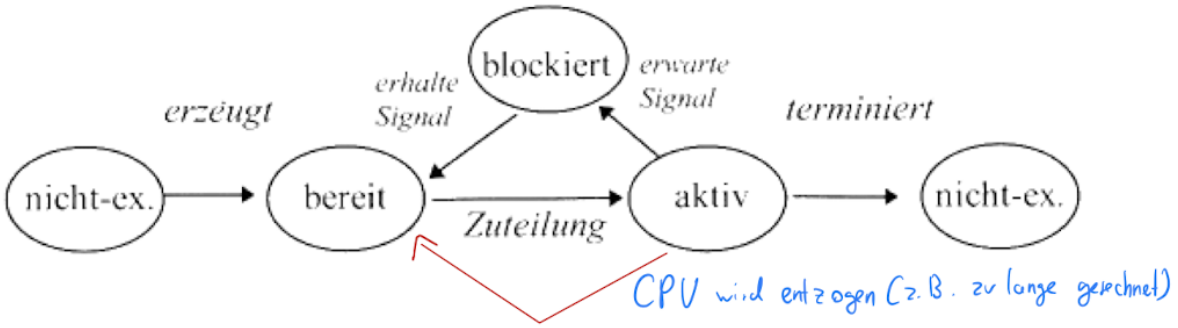
Aufruf unter Linux



Prozesse

- **Prozess/Task**: Ablaufumgebung für ein Programm auf einem Rechnersystem
- Prozess ist Programm zur Laufzeit
- *Multitasking*: Mehrere Prozesse konkurrieren um Betriebsmittel wie CPU und Speicher
- *Single-Task*: Immer nur ein Prozess
- Zuteilung/Scheduling: Wird vom Betriebssystem übernommen

Prozesslebenszyklus



- **Bereit**: Prozess wird erzeugt
- **Aktiv**: Prozess arbeitet gerade und ist einer CPU zugeteilt
- **Blockiert**: Prozess wartet auf Ressource/Betriebsmittel während der Ausführung
- **Nicht-existent**: Nicht erzeugt oder bereits beendet

Prozesskontext

Informationen zu einem Prozess

Drei Bereiche des Prozesskontexts:

- **Benutzerkontext**: Daten des Prozesses
- **Hardwarekontext**: Inhalte der CPU-Register, Stackpointer, Befehlszähler, Programmstatuswort

- **Systemkontext**: Informationen aus Sicht des Betriebssystems (Prozessnummer, geöffnete Dateien…)

Kontextwechsel

- CPU wird dem aktuell laufenden Prozess entzogen und einem anderen Prozess zugeteilt
- Prozesskontext des alten Prozesses muss gesichert werden
- Prozesskontext des neuen Prozesses muss geladen werden

Prozesstabelle & Process Control Block

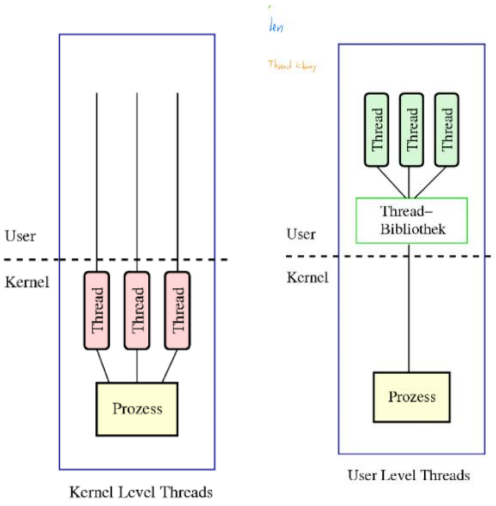
- **Prozesstabelle**: Alle Prozesse eines Betriebssystems
- **Process Control Block**: Datenstruktur in der Prozesstabelle, die wichtige Informationen eines Prozesses speichert
    - Programmzähler
    - Prozesszustand
    - Priorität
    - Verbrauchte Prozessorzeit
    - Eigene Prozessnummer
    - Prozessnummer des erzeugenden Prozesses (Elternprozess)
    - Zugeordnete Betriebsmittel (geöffnete Dateien)
    - Registerinhalte, Stackpointer, Programmstatuswort
- Beim Kontextwechsel werden Daten aus dem alten Prozess in einen PCB gespeichert und aus dem PCB des laufenden Prozesses geladen.

Threads (Leichtgewichtprozesse)

- Kontextwechsel bei einem Prozess ist zeitaufwändig
    - Innerhalb eines Prozesses kann es nebenläufige Aufgaben geben, die sich Betriebsmittel teilen, sog. Threads
    - Wechsel zwischen Threads ist schneller als zwischen Prozessen unter anderem aufgrund von geteilten Betriebsmitteln zwischen Threads:
        - Codesegment
        - Datensegment
        - Dateideskriptoren (geöffnete Dateien)
- Beispiel Word:
    - Je ein Thread für Eingabeverarbeitung, Darstellung, Druckaufträge

Kernel-Level-Threads (Heavy-Weight-Threads) & User-Level-Threads

Art	**Kernel Level Threads**	**User Level Threads**
Verwaltung	Werden durch das Betriebssystem verwaltet: - Erzeugen, Beenden - Scheduling	Betriebssystem kennt die Threads nicht. Werden durch Threadbibliothek verwaltet: - Erzeugen, Beenden - Scheduling
Basisinfo	- Aktuelle Werte eines Threads werden in einem Thread Control Block (TCB) gespeichert - Äquivalent zu PCB	- Threadbibliothek wird durch den Prozess eingebunden - Threadbibliothek teilt CPU auf Threads auf
Verhalten	Warten auf ein Signal blockiert andere Threads nicht	Warten auf ein Signal blockiert andere Threads
Kontextwechsel	Fast wie bei Prozessen	Sehr einfach
Beispiel	Windows	Java-Threads



CPU-Scheduling

Nebenläufige Prozesse müssen sich die CPU teilen

- Zuteilung der CPU wird durch das CPU-Scheduling geregelt

Arten des CPU-Scheduling

- **Non-preemptiv** (nicht verdrängend): Prozess wird nicht unterbrochen, bis er selbst die CPU frei gibt
- **Preemptiv** (verdrängend): Einem Prozess kann die CPU durch das Betriebssystem entzogen werden

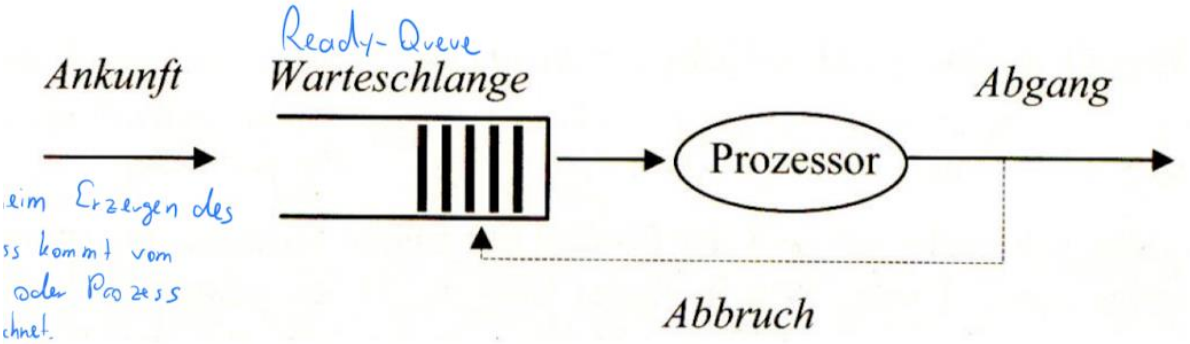
Bestandteile des CPU-Scheduling

- **Prozessmanager**: Verwaltet Prozesse
- **Scheduler**: Teil des Prozessmanagers – regelt Planung der CPU-Zuteilung
    - Bereite Prozesse stehen in der „Ready-Queue“. Scheduler legt die Reihenfolge fest
- **Dispatcher**: Führt Prozesswechsel aus

Verschiedene Schedulingziele

- **Fairness**: Garantierte Mindestzuteilung
- **Effizienz**: Möglichst hohe CPU-Auslastung
- **Minimale Antwortzeit**: Reaktionszeit auf ein Ereignis soll minimiert werden
- **Durchschnittliche Wartezeit**
- **Durchsatz**: Möglichst viele Prozesse bearbeiten
- **Durchlaufzeit**: Mittlere Zeit, bis ein Prozess abgearbeitet ist, soll möglichst gering sein
- **Kalkulierbarkeit**: Zeit, bis zu der ein Prozess abgearbeitet ist, ist berechenbar

Arten des Scheduling



- **Kurzfristig (Short-Term)**: Regelt, welcher Prozess aus der Warteschlange als nächstes die CPU bekommt
- **Mittelfristig (Medium-Term)**: Kann Prozesse aus dem Hauptspeicher auf die Festplatte auslagern, wenn der Prozess lange nicht im Zustand „Bereit“ war
- **Langfristig (Long-Term):** Regelt, welche Prozesse überhaupt in die Warteschlange dürfen. Bei Überlastung des Systems können Prozesse abgewiesen werden

Strategien

- **First Come First Serve:** Abarbeitung nach Ankunftszeit
- **Shortest Job First**: Kürzeste Bearbeitungszeit zuerst
- **Priority Scheduling**: Höchste Priorität zuerst
- **Shortest Remaining Time First**: Kürzeste Restbearbeitungszeit zuerst – Prozess mit kürzerer Restbearbeitungszeit als laufender Prozess kommt an -> laufender Prozess wird unterbrochen
- **Round Robin**: Nach bestimmtem Zeitabschnitt wird der Prozess gewechselt
- **Dynamic Priority Round Robin**: Statische Priorität zu Prozessbeginn, kann sich aber zur Laufzeit ändern

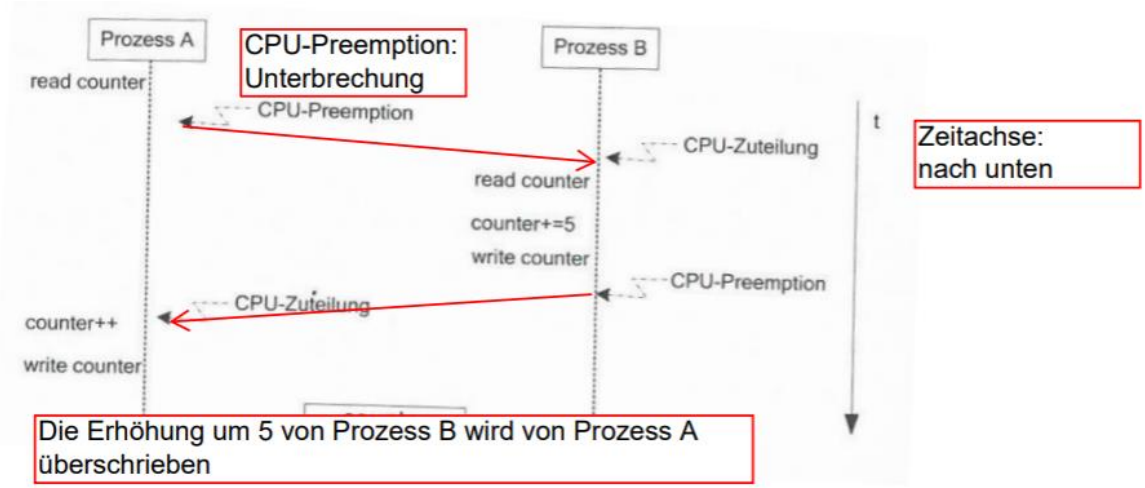
Prozesssynchronisation

- Auf Mehr-Programm-Betriebssystemen werden Prozesse oder Threads quasi-parallel ausgeführt (Nebenläufigkeit)
- Durch präemptives (verdrängendes) Scheduling kann einem Prozess der CPU entzogen werden, wenn ein anderer Prozess aktiviert werden muss
- Prozess kann nicht wissen wann er unterbrochen wird und es nur teilweise beeinflussen
    - Bearbeitet ein Prozess ein Betriebsmittel, das mit anderen geteilt wird, muss es in einem konsistenten Zustand hinterlassen werden
    - **Atomare Aktionen**: Codebereiche, die an einem Stück bearbeitet werden müssen um Betriebsmittel konsistent zu hinterlassen

Lost-Update-Problem

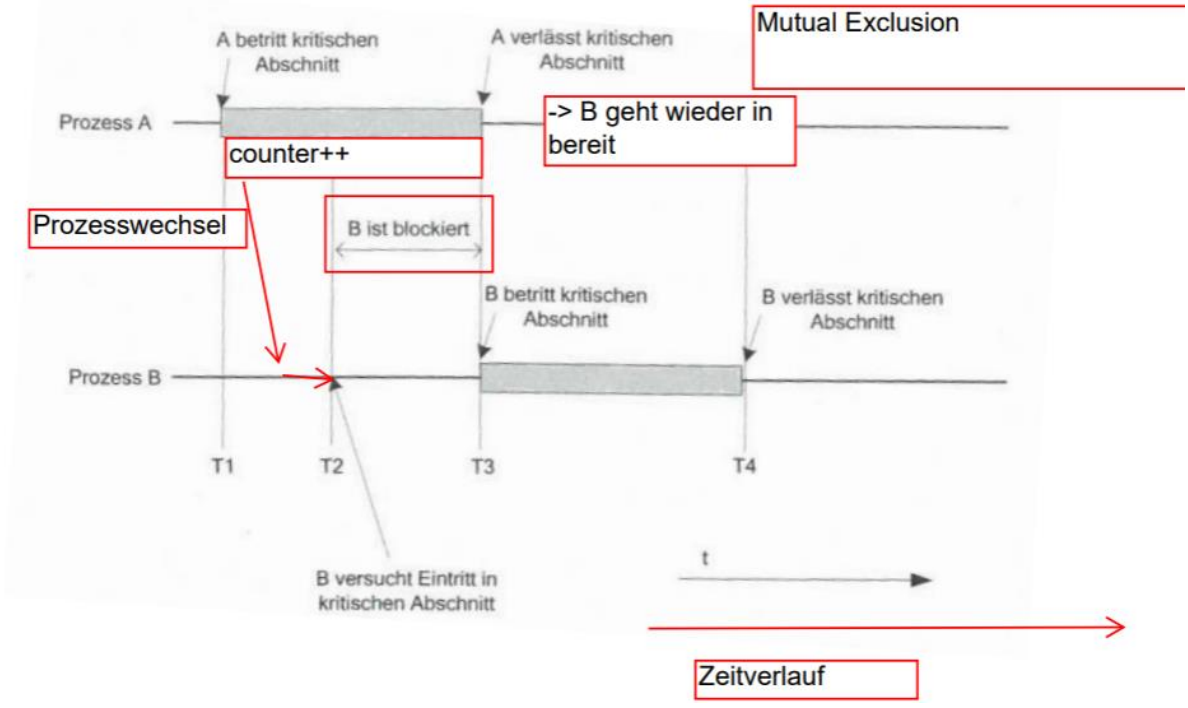
- Zwei Prozesse wollen auf dieselbe Variable zugreifen
- Prozess A erhöht die Variable um 1, Prozess B um 5
    - Aktion „Erhöhen um x“ auf Maschinenbefehlsebene:
        1. Counter lesen
        2. Counter um x erhöhen
        3. Counter schreiben
- Wird Prozess A zwischen 1. Und 2. Von Prozess B unterbrochen, so wird das Ergebnis von Prozess B überschrieben, sobald Prozess A wieder CPU-Zeit hat, da Prozess A noch mit dem Variablenwert vor der Ausführung von Prozess B arbeitet.

- Das Ergebnis der beiden Prozesse hängt von ihrer zeitlichen Reihenfolge ab
- **Race Condition**: Ergebnis zweier Prozesse mit dem gleichen Betriebsmittel, das von der zeitlichen Reihenfolge abhängt



Lösung: wechselseitiger Ausschluss

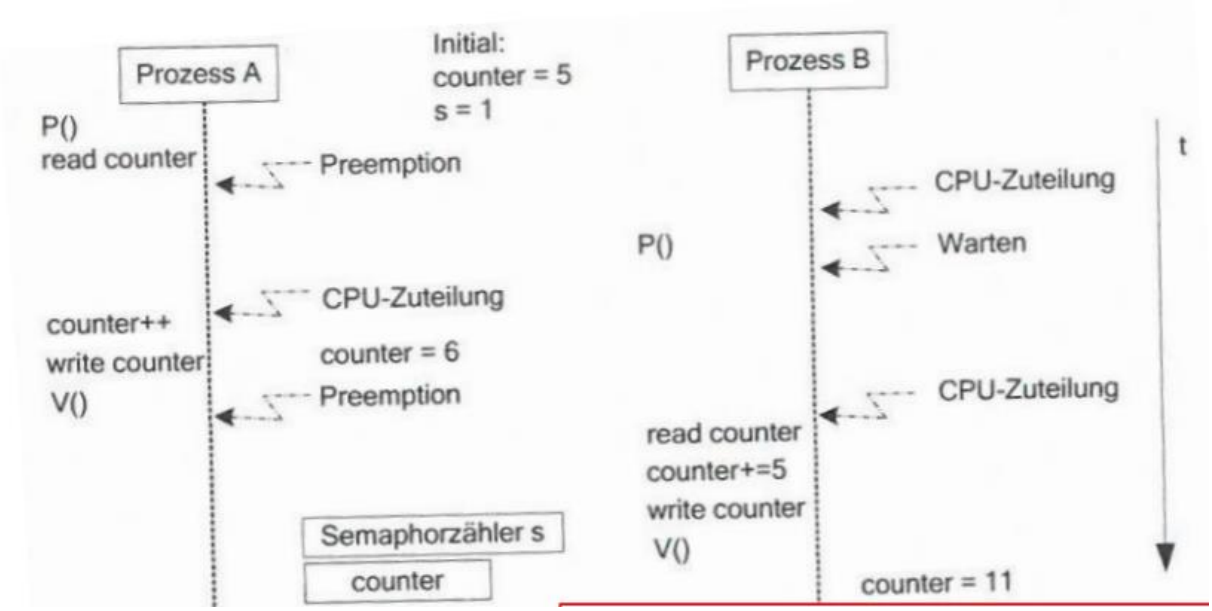
- **Kritischer Abschnitt**: Programmteil, der eine atomare Aktion beinhält
- **Wechselseitiger Ausschluss**: Sorgt dafür, dass sich in einem kritischen Abschnitt immer nur ein Prozess befindet (z.B. mit Hilfe einer Semaphore)



Praktische Implementierung: (binäre) Semaphoren

- **Semaphorenzähler**: Bestimmt wie viele Prozesse in einem kritischen Abschnitt sein dürfen
  - Binäre Semaphore: Maximal ein Prozess in einem kritischen Abschnitt
- Eintritt in kritischen Abschnitt:
  - Funktion P() wird aufgerufen
    - Sperrt kritischen Bereich
    - Reduziert Semaphorenzähler um 1, sofern er größer als 0 ist

        - Semaphorenzähler ist 0: Eintritt in kritischen Bereich wird verwehrt – Prozess verliert CPU
- Verlassen eines kritischen Abschnitts:
    - Funktion V() wird aufgerufen
        - Gibt kritischen Bereich frei
        - Semaphorenzähler wird um 1 erhöht
- Jeder kritische Abschnitt/jedes Betriebsmittel hat eine zugeordnete Semaphore



Probleme mit nebenläufigen Prozessen

Blockieren

- Prozess 1 belegt ein Betriebsmittel
- Prozess 2 benötigt das gleiche Betriebsmittel
- Prozess 2 ist blockiert

Verhungern

Ein Prozess erhält ein Betriebsmittel nie, da immer andere Prozesse das Betriebsmittel belegen

Deadlock

Mehrere Prozesse versuchen auf Betriebsmittel zuzugreifen, die von jeweils einem der anderen Prozesse blockiert sind.