Informatik II

Dr. Henrik Brosenne
Georg-August-Universität Göttingen
Institut für Informatik

Sommersemester 2016

Inhalt

Betriebssysteme
- Einführung
- Prozessverwaltung
- Scheduling
  - First Come First Served (FCFS)
  - Shortest Job First (SJF)
  - Round-Robin-Scheduling
- Prozess-Synchronisation
  - Mutex
- Speicherverwaltung
  - Swapping

Literatur

Andrew S. Tanenbaum,
Moderne Betriebssysteme,
2te Auflage, Pearson Studium, 2002.

Carsten Vogt,
Betriebssysteme,
Spektrum Akademischer Verlag, 2001.

Abraham Silberschatz, Peter B. Galvin,
Operating System Concepts (5th Edition),
John Wiley and Sons, 1999.

William Stallings,
Betriebssysteme - Prinzipien und Umsetzung (4te Auflage),
Prentice Hall, 2002.

Allgemein

Computersoftware gliedert sich in zwei Gruppen.

- **Systemprogramme** ermöglichen den Betrieb des Computers.
- **Anwenderprogramme** erfüllen die Anforderungen der Anwender.

Das **Betriebssystem** ist das wichtigste Systemprogramm.

Definition eines Betriebssystems nach DIN 44300

- Betriebssystem: Die Programme eines digitalen Rechnersystems, die zusammen mit den Eigenschaften der Rechenanlage die Basis der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen.

Hardwaresicht

Man kann bei Rechnersystemen zwei Sichten einnehmen.

Hardwaresicht. Ein Rechnersystem besteht aus einer Menge kooperierender Hardwarekomponenten.

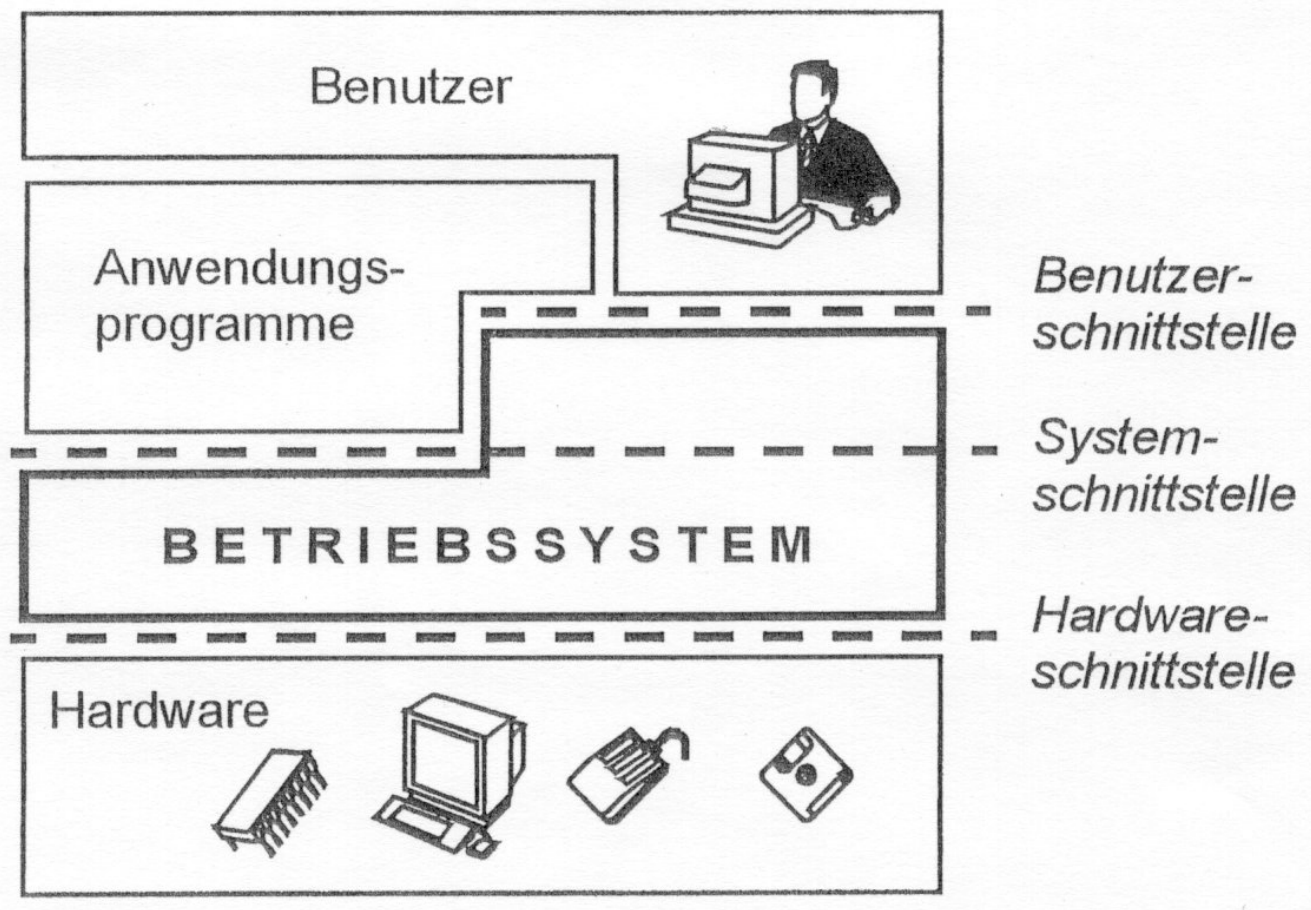
- Prozessor (CPU). Ausführung von Maschinenprogrammen
- Hauptspeicher. *Kurzfristige* Speicherung einer begrenzte Menge von Daten.
- Hintergrundspeicher (Festplatte, Diskette, CD, DVD). *Langfristige* Speicherung größerer Datenmengen.
- Eingabegeräte (Tastatur, Maus).
- Ausgabegeräte (Bildschirm, Drucker).
- Netzwerkkarte. Verbindung an ein Kommunikationsnetz.
- ...

Anwendersicht

Anwendersicht. Ein Rechnersystem stellt (benutzerfreundliche) Konzepte bereit, mit denen Daten und Informationen verarbeitet werden können.

- Dateisystem. Klar strukturiert mit Dienstprogrammen.
- Programmierumgebung. Schreiben und Übersetzen von Programmen.
- Ein- und Ausgabedienste. Zugriff auf Daten, Internet Angebote, etc.
- Systemverwaltung.
- Multi-User-Fähigkeit. Unterstützung mehrere Benutzer.
- ...

Position des Betriebssystems



Betriebssystem

Das Betriebssystem (engl. *operating system*) ist die Softwarekomponente, die die abstrakte Anwendersicht auf der Grundlage der realen Hardwaresicht umsetzt.

Das Betriebssystem

- macht die Hardware für den Anwender benutzbar.
- setzt auf der Hardware auf, steuert diese und bietet *nach oben* benutzer- und programmierfreundliche Dienste an.
- enthält interen Programme und Datenstrukturen.

Definitionen

Ressourcen (Betriebsmittel). Die Ressourcen (Betriebsmittel) eines Betriebssystems sind alle Hard- und Softwarekomponenten, die für die Programmausführung relevant sind. Z.B. Prozessor, Hauptspeicher, I/O-Geräte, Hintergrundspeicher, etc.

Betriebssystem als Ressourcenverwalter. Ein Betriebssystem bezeichnet alle Programme eines Rechensystems, die die Ausführung der Benutzerprogramme, die Verteilung der Ressourcen auf die Benutzerprogramme und die Aufrechterhaltung der Betriebsart steuern und überwachen.

Aufgaben eines Betriebssystems

Hauptaufgaben eines Betriebssystems

- Prozessverwaltung
- Speicherverwaltung
- Verwaltung des Dateisystems
- Geräteverwaltung

Prozessverwaltung

Prozessverwaltung (Ein Prozess oder auch Task ist ein in Ausführung befindliches Programm)

- Erzeugen und Löschen von Prozessen.
- Prozessorzuteilung (Scheduling).
- Prozesskommunikation.
- Synchronisation nebenläufiger Prozesse, die gemeinsame Daten benutzen.

Speicherverwaltung

Speicherverwaltung

- Zuteilung des verfügbaren physikalischen Speichers an Prozesse.
  - Segementierung (= Unterteilung des benutzten Speicheradressraums in einzelne Segmente).
  - ...
- Einbeziehen des Hintergrundspeichers (z.B. Festplatte).
  - Paging (= Bereitstellung von virtuellem Speicher).
  - Swapping (= Ein-/Auslagern von Prozessen).
  - ...

Verwaltung des Dateisystems

Verwaltung des Dateisystems

- Logische Sicht auf Speichereinheiten (Dateien).
  - Benutzer arbeitet mit Dateinamen. Wie und wo die Dateien gespeichert werden, ist ihm egal.
- Systemaufrufe für Dateioperationen.
  - Erzeugen, Löschen, Öffnen, Lesen, Schreiben, Kopieren, etc.
- Strukturierung mittels Verzeichnissen (engl. directories).
- Schutz von Dateien und Verzeichnissen vor unberechtigtem Zugriff

Geräteverwaltung

Geräteverwaltung

- Auswahl und Bereitstellung von I/O-Geräten.
- Anpassung an physikalische Eigenschaften der Geräte.
- Überwachung der Datenübertragung.

Weitere wichtige Konzepte

Fehlertoleranz.

- Graceful Degradation. Beim Ausfall einzelner Komponenten läuft das System mit vollem Funktionsumfang mit verminderter Leistung weiter.
- Fehlertoleranz wird durch Redundanz erkauft.

Realzeitbetrieb.

- Betriebssystem muss den Realzeit-kritischen Prozessen die Betriebsmittel so zuteilen, dass die angeforderten Zeitanforderungen eingehalten werden.
- Für zeitkritische Systeme. Meßsysteme, Anlagensteuerungen, etc.

Benutzeroberflächen.

- Betriebssystem kann eine Benutzerschnittstelle für die eigene Bedienung enthalten.
- Betriebssystem kann Funktionen bereitstellen, mit denen aus Anwendungsprogrammen heraus auf die Benutzerschnittstelle zugegriffen werden kann.

Betriebsart

Die **Betriebsart** ist ein wichtiges Kennzeichen für die Leistungsfähigkeit und den Anwendungsbereich eines Betriebssystems.

Rechensysteme habe im Allgemeinem mehrere Aufträge gleichzeitig zu bearbeiten. Die Betriebsart bestimmt (im wesentlichen) den zeitlichen Ablauf der Ausführung dieser Aufträge.

Die Spanne möglicher Abläufe reicht von **streng sequentiell** (= hintereinander) bis **voll nebenläufig** (= gleichzeitig).

Betriebsarten

Einzelbenutzerbetrieb

> **Ein Benutzer** belegt das **gesamte Rechensystem** und erteilt Aufträge, die **streng sequentiell** abgearbeitet werden.

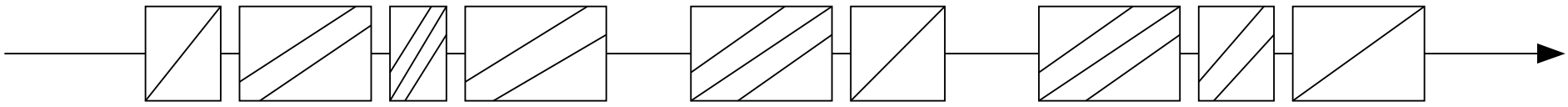
Batchbetrieb

> Ein Auftrag (engl. *job*) wird durch eine Reihe von **Steuerkommandos an das Betriebssystem**, formuliert in einer *job control language*, eingeleitet und abgeschlossen. Zwischen den Steuerkommandos befindet sich das eigentliche Programm.

Mehrprogrammbetrieb

> Mehrer Aufträge werden **nebenläufig** (engl. *concurrent*) bearbeit, wobei der Prozessor (im schnellen Wechsel) umgeschaltet wird.

Mehrprogrammbetrieb



= Prozessor arbeitet

Mehrprogrammbetrieb

- Mehrer Aufträge werden **nebenläufig** (engl. *concurrent*) bearbeit, wobei der Prozessor (im schnellen Wechsel) umgeschaltet wird.
- Flexible Prozessorzuteilung. Der Prozessor kann auch während des Abarbeiten eines Auftrag zu einem anderen wechseln, weil
  - der gerade ausgeführte Auftrag wartet.
  - ein dringenderer Auftrag bearbeitet werden soll.
  - gleichberechtigte Aufträge gleichmäßig (fair) bearbeiten werden sollen.
- Ermöglicht **Timesharing**. Mehrere Benutzer können gleichzeitig mit dem Rechensystem arbeiten.

Inhalt

Betriebssysteme
- Einführung
- Prozessverwaltung
- Scheduling
  - First Come First Served (FCFS)
  - Shortest Job First (SJF)
  - Round-Robin-Scheduling
- Prozess-Synchronisation
  - Mutex
- Speicherverwaltung
  - Swapping

Prozesse

Auf modernen Rechensystemen können mehrere Programme nebenläufig bearbeitet werden.

- Benutzerprogramme, Lesen/Schreiben von/auf Platten, Drucken von Dateien, etc.
- Ermöglicht bessere Nutzung der Ressourcen.

Ein **Prozess** (engl. *process*, **task**) ist die Abstraktion eines laufenden Programms.

Ein Prozess benötigt **Betriebsmittel** (Prozessorzeit, Speicher, Dateien, etc.) und ist selbst ein Betriebsmittel.

Ein Prozessor führt in jeder Zeiteinheit maximal einen Prozess aus. Laufen mehrere Prozesse, dann finden Prozesswechsel statt.

Kontext eines Prozesses

Prozesse werden vom Betriebssystem verwaltet.

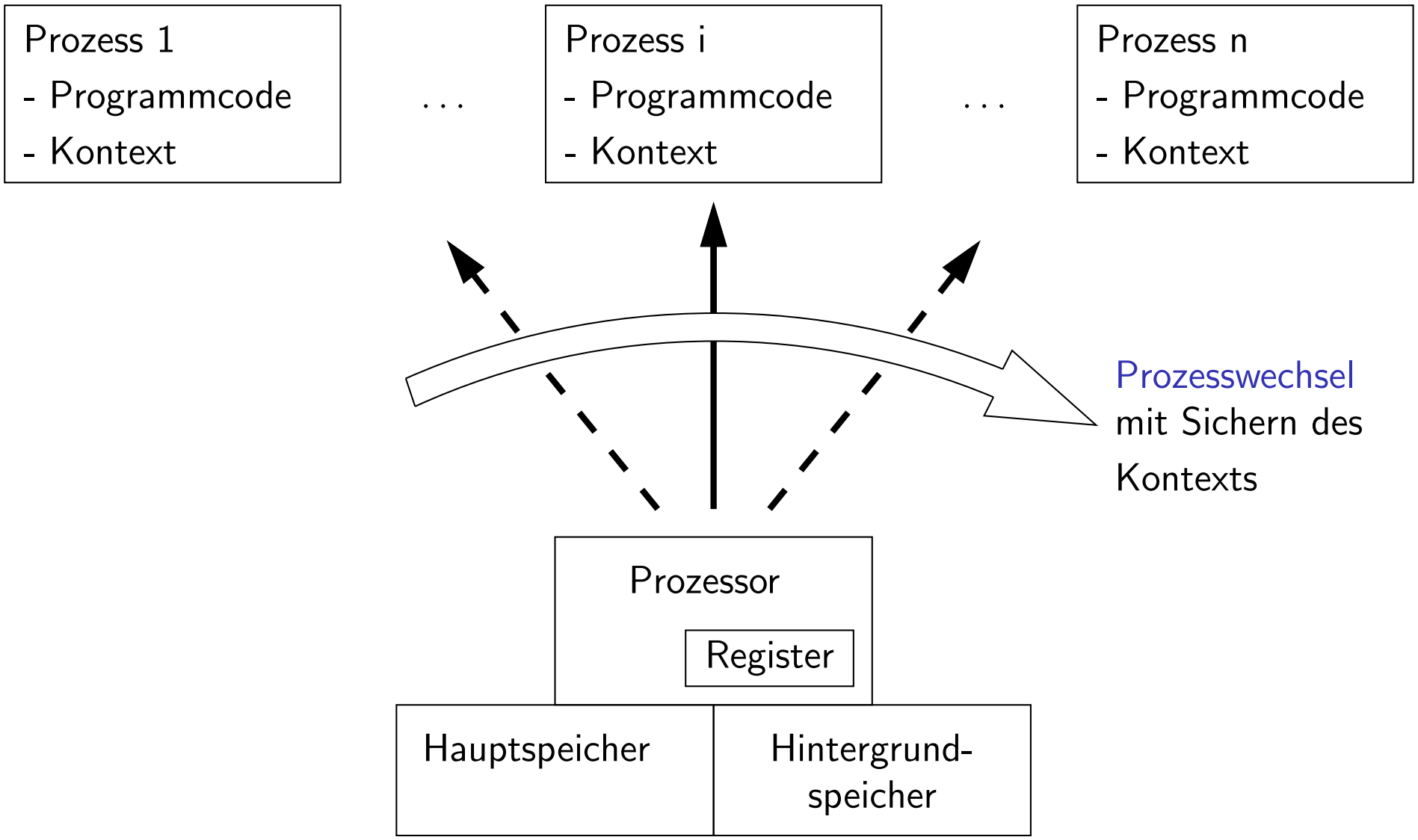
Eine Aufgabe des Betriebssystems besteht darin, den verschiedenen Prozessen Prozessorzeit zuzuteilen (engl. **scheduling)**.

Um das Scheduling zu ermöglichen, besteht ein Prozess aus dem **auszuführenden Programmcode** (inkl. Daten) und einem **Kontext**.

Zum **Kontext** eines Prozesses gehören

- die **Registerinhalte des Prozessors**,
- dem Prozess zugeordnete **Bereiche des direkt zugreifbaren Speichers**,
- durch den Prozess **geöffnete Dateien**,
- dem Prozess **zugeordnete Peripheriegeräte**,
- **Verwaltungsinformationen** über den Prozess.

Prozesswechsel



Prozesszustände

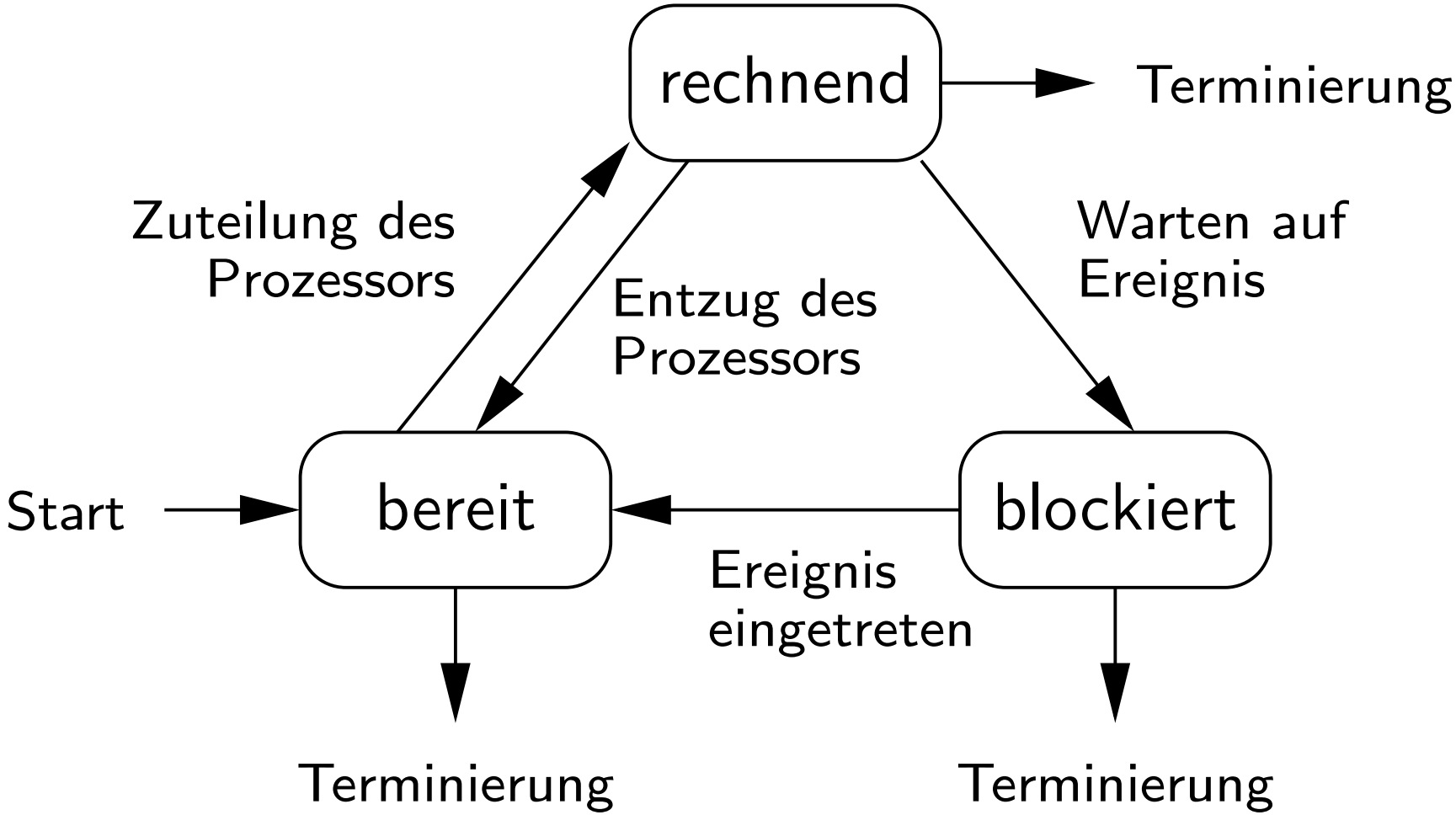
Ein Prozess befindet sich zu jedem Zeitpunkt in einem bestimmten **Zustand**. Es finden **dynamisch Zustandsübergänge**, also Veränderungen des Prozesszustands statt.

Prozesszustände

- **rechnend** (*running*). Prozess wird momentan ausgeführt.
- **bereit** (*ready*). Prozess ist ausführbar und wartet auf die Zuteilung des Prozessors.
- **blockiert** (*blocked*). Prozess kann momentan nicht ausgeführt werden und wartet auf das Eintreten eines Ereignisses (z.B. Nachricht von einem E/A-Prozess).

I.d.R. existieren noch weitere Zustände, z.B. *new* (Prozess wird gerade erzeugt) oder *exit* (Prozess wird gerade beendet) sowie evtl. weitere Verfeinerungen der obigen Zustände.

Zustandsübergänge



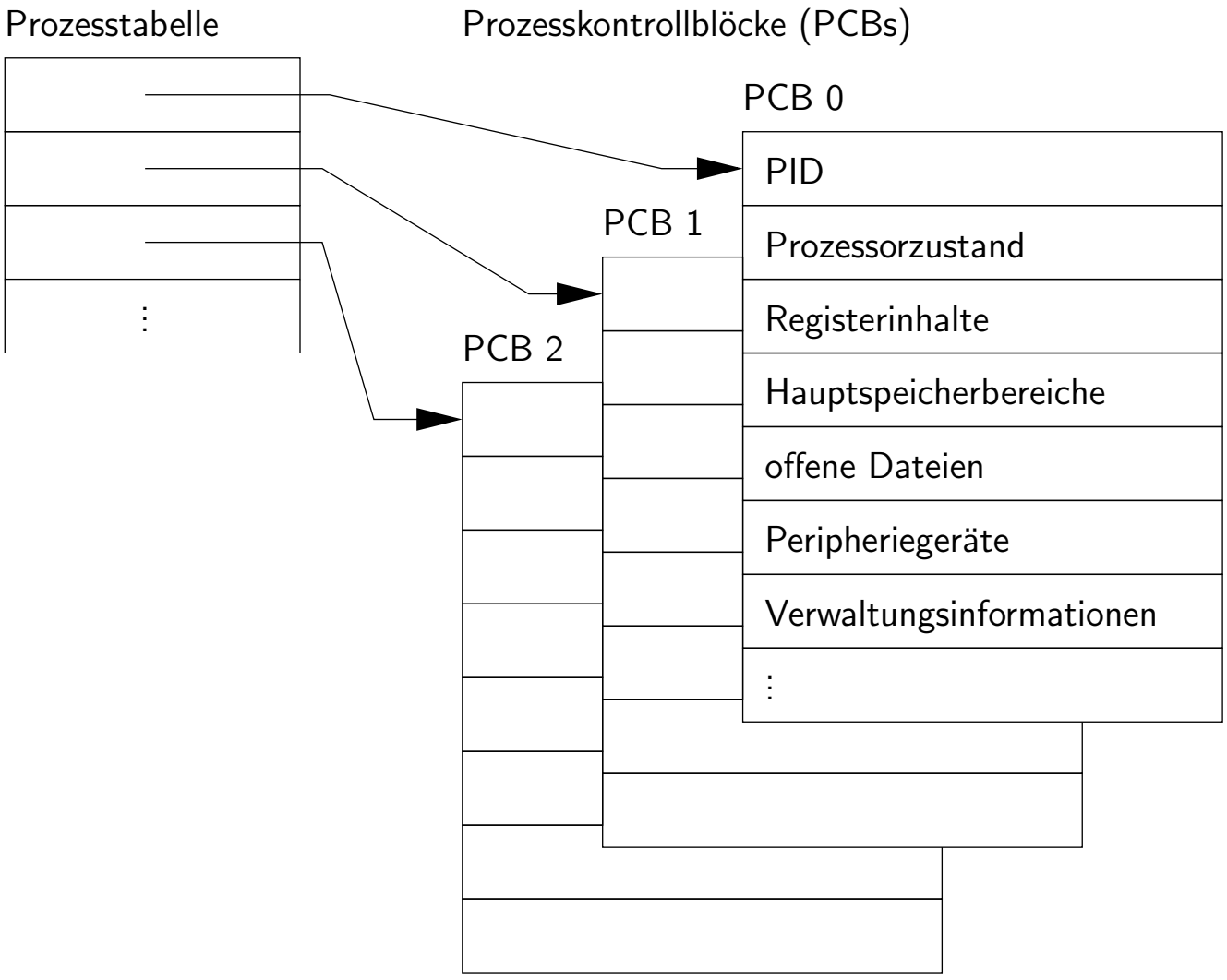
Prozesstabelle und Prozesskontrollblöcke (1/2)

Das Betriebssystem verwaltet Prozesse mit Hilfe einer **Prozesstabelle**, die **Prozesskontrollblöcke** (engl. *process control blocks*, **PCBs**), bzw. Verweise auf PCBs, für alle existierenden Prozesse enthält.

Ein PCB enthält

- den Kontext des Prozesses,
- für das Scheduling benötigte Informationen,
- weitere Verwaltungsinformationen.

Prozesstabelle und Prozesskontrollblöcke (2/2)



Prozesswechsel (Dispatching) (1/2)

Dispatcher (deutsch *Prozessumschalter*)

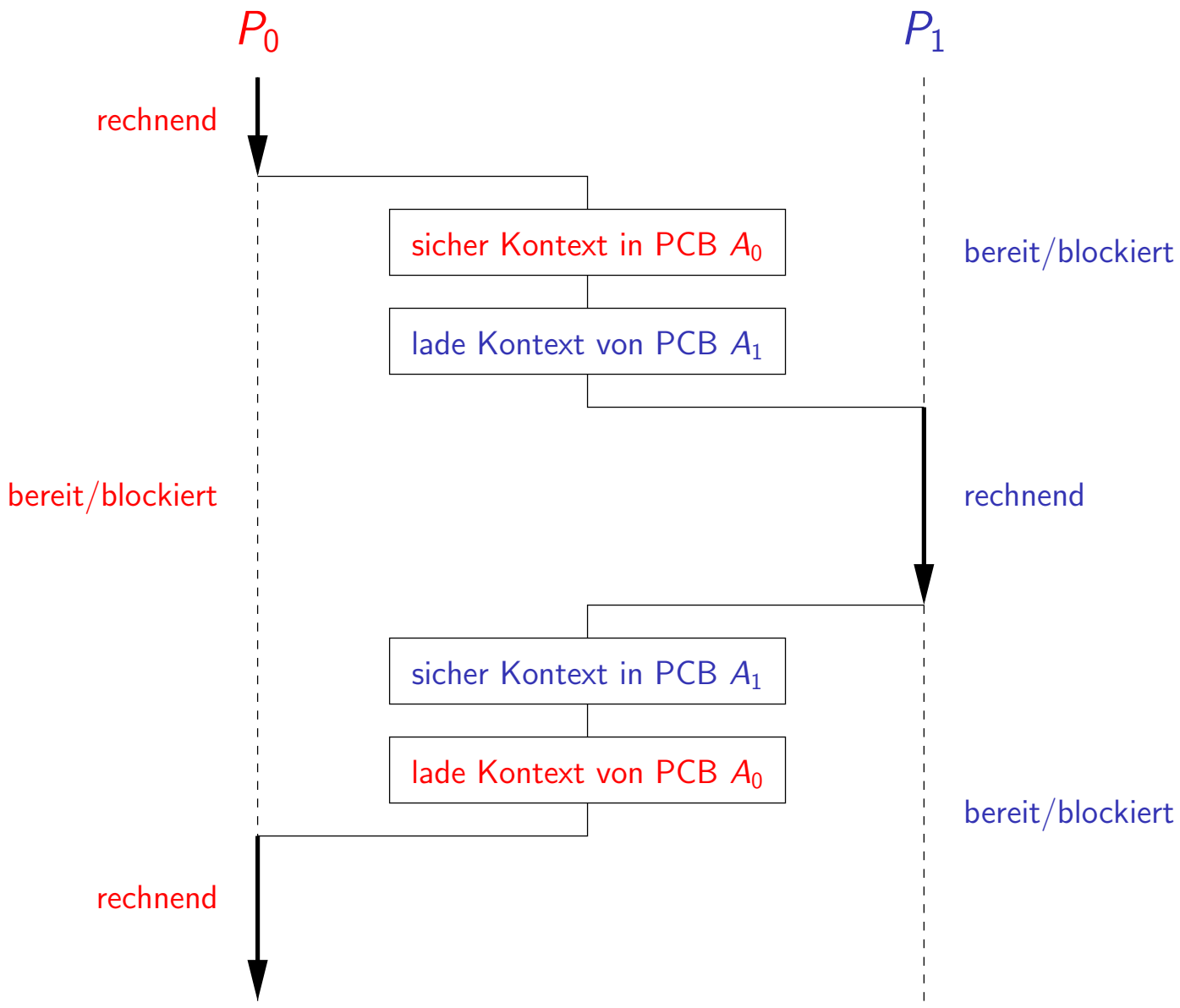
Prozesswechsel

- Der aktuelle Kontext eines Prozesses P_0 wird im einem PCB gesichert.
- Der Kontext eines anderen Prozesses P_1 wird aus einem PCB geladen.
- Der Zustand beider PCBs muss aktualisiert werden.
- **Prozesswechsel** = **Kontextwechsel**

Prozesswechsel sind relativ teuer (benötigen viel Zeit).

Prozesswechsel wird häufig von spezieller Hardware unterstützt.

Prozesswechsel (Dispatching) (2/2)



Inhalt

Betriebssysteme

- Einführung
- Prozessverwaltung
- Scheduling
  - First Come First Served (FCFS)
  - Shortest Job First (SJF)
  - Round-Robin-Scheduling
- Prozess-Synchronisation
  - Mutex
- Speicherverwaltung
  - Swapping

Scheduling

Scheduling ist die Zuteilung von Prozessorzeit an die Prozesse.

Komponenten des Scheduling.

- **Prozesswechselkosten.** Prozesswechsel sind relativ teuer, weil der Kontextes der Prozesse gesichert/gelanden werden muß.
- **Warteschlangenmodell.** Wartende Prozesse werden in internen Warteschlangen gehalten, die Auswahlstrategie der Warteschlangen haben wesentlichen Einfluss auf das Systemverhalten.
- **Scheduling-Verfahren.**

Fragen

- Wann erfolgt der Kontextwechsel?
- Nach welchen Kriterien wird der Prozess ausgewählt, der als nächstes bearbeitet wird?

Anforderungen (1/2)

Alle Systeme

- *Fairness*. Jeder Prozess bekommt Rechenzeit der CPU.
- *Policy Enforcement*. Durchsetzung der Verfahrensweisen, keine Ausnahmen.
- *Balance*. Alle Teile des Systems sind (gleichmäßig) ausgelastet.
- *Data Protection*. Keine Daten oder Prozesse gehen verloren.
- *Scalability*. Mittlere Leistung wird bei wachsender Last (Anzahl von Prozessen) beibehalten. D.h. es gibt keine Schwelle, ab der das Scheduling nur noch sehr langsam oder gar nicht mehr funktioniert.

Anforderungen (2/2)

Batch-Systeme (Stapelverarbeitungssysteme)

- *Throughput* (Durchsatz). Maximiere nach Prozessen pro Zeiteinheit.
- *Turnaround Time*. Minimiere die Zeit vom Start bis zur Beendigung eines Prozesses.
- *Processor Load*. Belege die CPU konstant mit Jobs.

First Come First Served (FCFS)

Prinzip

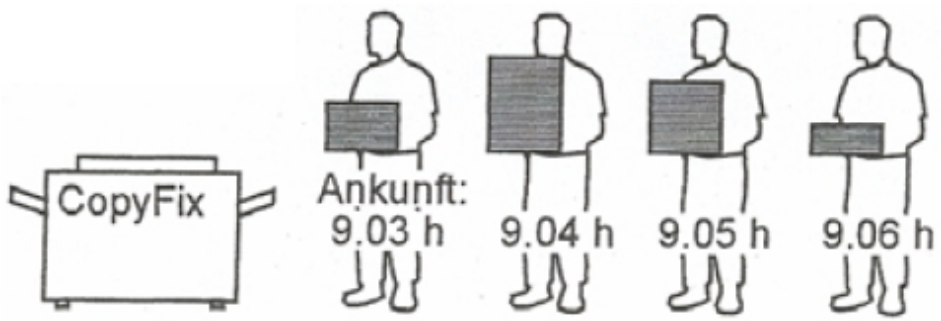
- Prozesse bekommen den Prozessor entsprechend ihrer Ankunftsreihenfolge zugeteilt.
- Keine Abhängigkeiten zwischen den Prozessen.
- Laufende Prozesse werden nicht unterbrochen.

Eigenschaften

- Fair (jeder Prozess kommt dran).
- Einfache Implementierung.

Bemerkungen

- Die mittlere Wartezeit kann unter Umständen sehr hoch werden.



Shortest Job First (SJF)

Prinzip

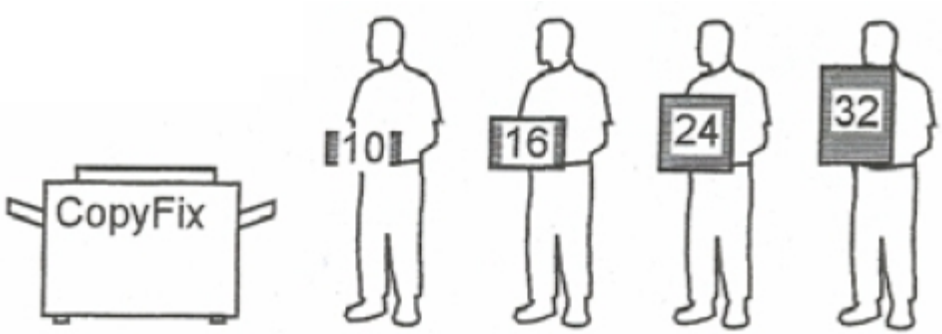
- Es wird jeweils der Prozess mit der kürzesten Rechenzeit als nächstes gerechnet.
- Keine Abhängigkeiten zwischen den Prozessen.
- Laufende Prozesse werden nicht unterbrochen.

Eigenschaften

- Nicht fair (kurze Prozesse können lange Prozesse überholen).

Problem

- Wie wird die Rechenzeit eines Prozesses ermittelt?



Scheduling und Mehrprogrammbetrieb

Auf Sytemen im Mehrprogrammbetrieb sind normalerweise immer mehrere Prozesse zu einem Zeitpunkt rechenbereit.

Verfahren

- Man zerlegt die Rechenzeit in Zeitscheiben (gleicher oder variabler Länge) und ordnet diese nach bestimmten Kriterien (z.B. Fairness, Prioritäten, Rechenzeit, etc.) den rechenbereiten Prozessen zu.
- Hat ein Prozess seine Zeitscheibe verbraucht wird er unterbrochen und muß auf eine neue Zuteilung warten.

Round-Robin-Scheduling

Prinzip

- Die Rechenzeit wird in gleichlange Zeitscheiben/-schlitze (*time slices*) aufgeteilt.
- Prozesse werden in einer Warteschlange eingereiht und in FIFO-Ordnung (*first in, first out*) ausgewählt.
- Ein rechnender Prozess wird nach Ablauf einer Zeitscheibe unterbrochen und wieder hinten in die Warteschlange eingestellt (Rundlauf, *round robin*).
  Bemerkung
  Wird ein Prozess blockiert oder beendet er sich bevor dessen Zeitscheibe komplett aufgebraucht ist, wird sofort der nächste Prozess ausgewählt und kann eine Zeitscheiben lang rechnen.

Eigenschaften

- Die Prozessorzeit wird nahezu gleichmässig auf die vorhandenen Prozesse aufgeteilt.

Ankunfts-/Rechnenzeit

Die **Ankunftszeit** eines Prozesses ist der Zeitpunkt ab dem der Prozess vom Scheduling berücksichtigt wird. Der Prozess ist rechenbereit und wenn zu diesem Zeitpunkt der Prozessor nicht belegt ist, bekommt der Prozess sofort Rechenzeit zugeteilt.

Die **Rechenzeit** eines Prozesses ist die Anzahl an Zeiteinheiten, für die der Prozess Rechenzeit zugeteilt bekommt muss, um vollständig abzulaufen, d.h. sich zu beenden.

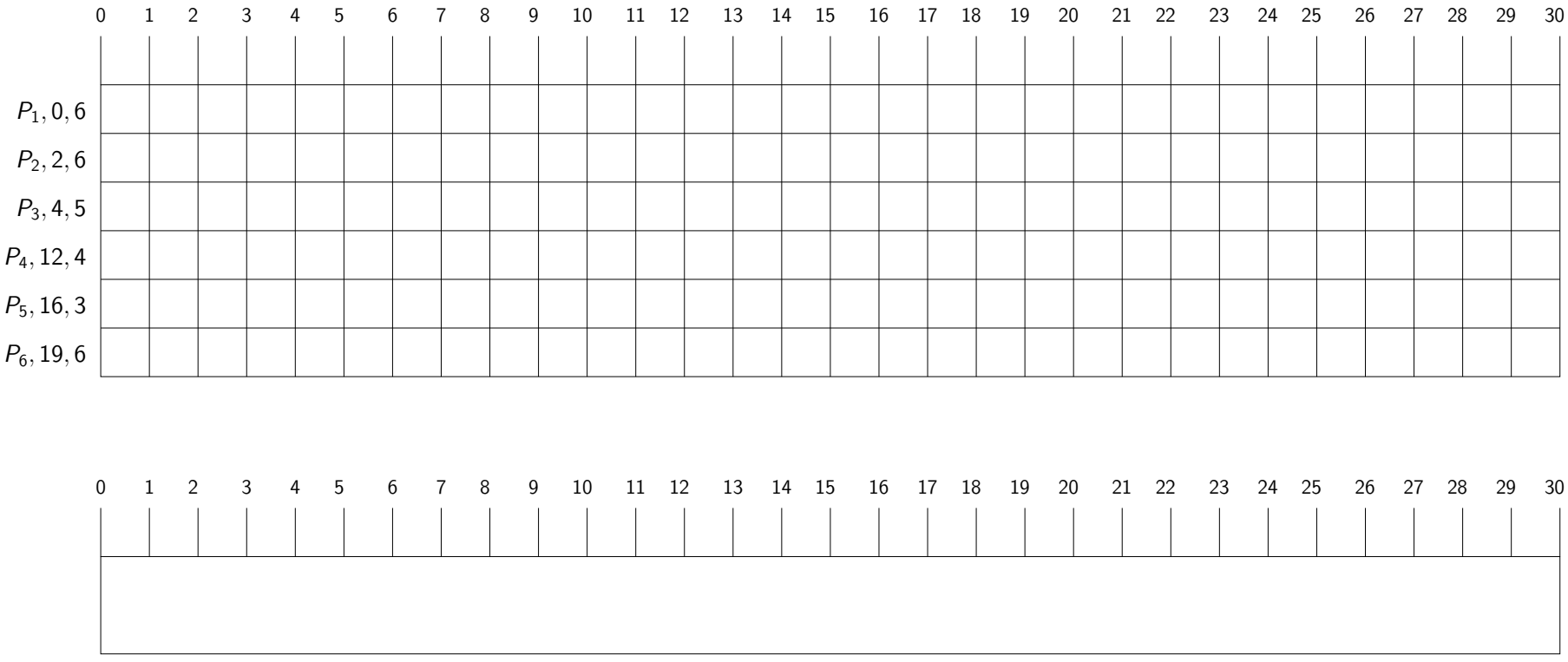
Beispiel

Gegeben seien die Prozesse P_1 bis P_6 mit folgenden *Ankunftszeiten* a_i und *Rechenzeiten* t_i.

Prozesse	P_1	P_2	P_3	P_4	P_5	P_6
Ankunftszeit a_i	0	2	4	12	16	19
Rechenzeit t_i	6	6	5	4	3	6

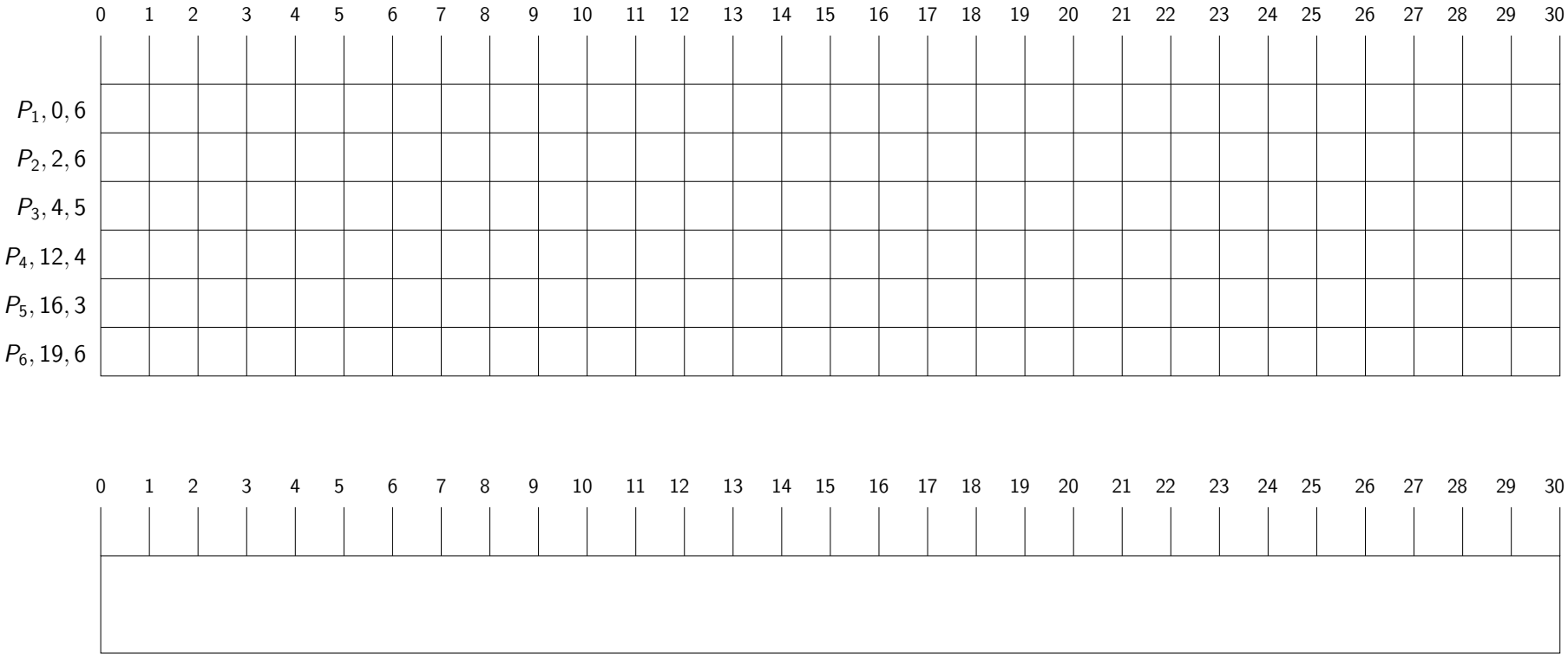
Darstellung des Schedules als *Gantt Chart* (nach Henry L. Gantt 1861-1919) oder *Balkenplan*.

Beispiel, FCFS

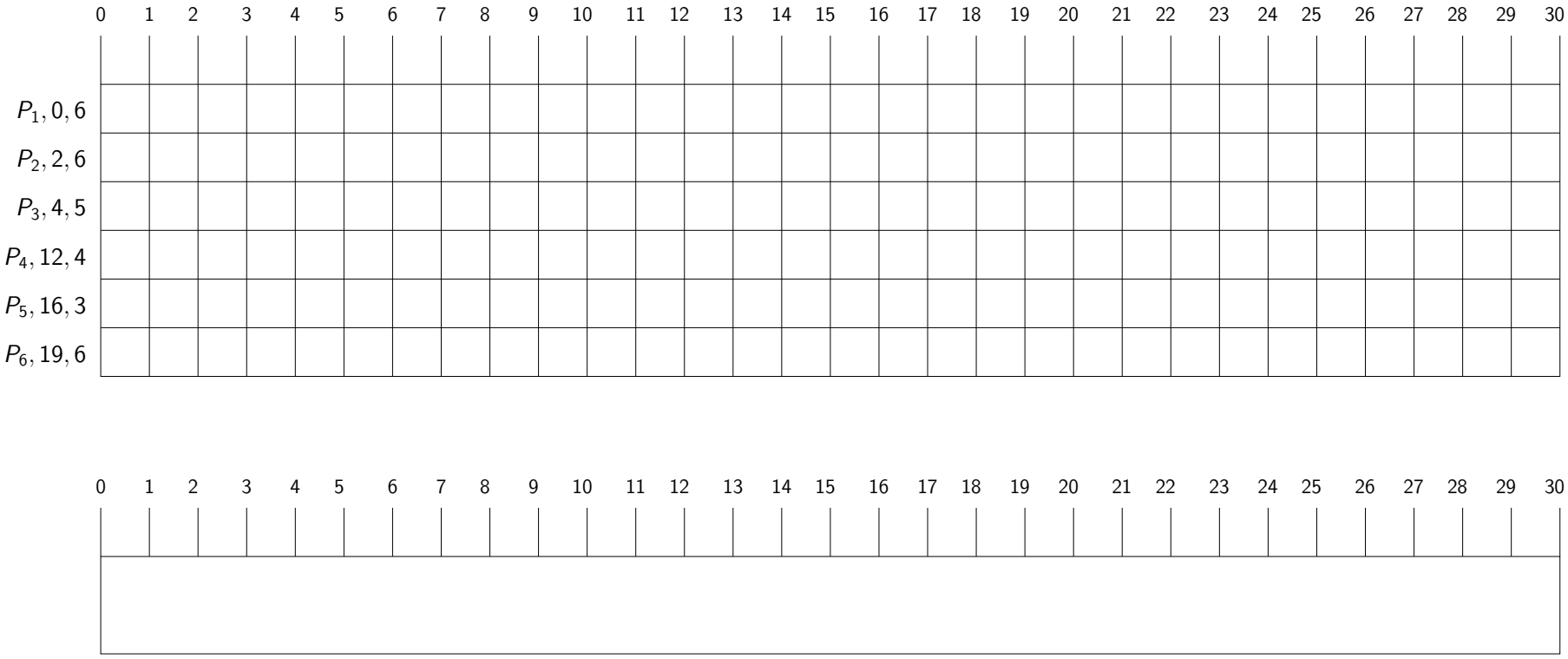


Darstellung des Schedules als *Gantt Chart* (nach Henry L. Gantt 1861-1919) oder *Balkenplan*.

Beispiel, SJF



Beispiel, Round-Robin (Zeitscheibe 5)



Inhalt

Betriebssysteme
- Einführung
- Prozessverwaltung
- Scheduling
  - First Come First Served (FCFS)
  - Shortest Job First (SJF)
  - Round-Robin-Scheduling
- Prozess-Synchronisation
  - Mutex
- Speicherverwaltung
  - Swapping

Gemeinsame Ressourcen

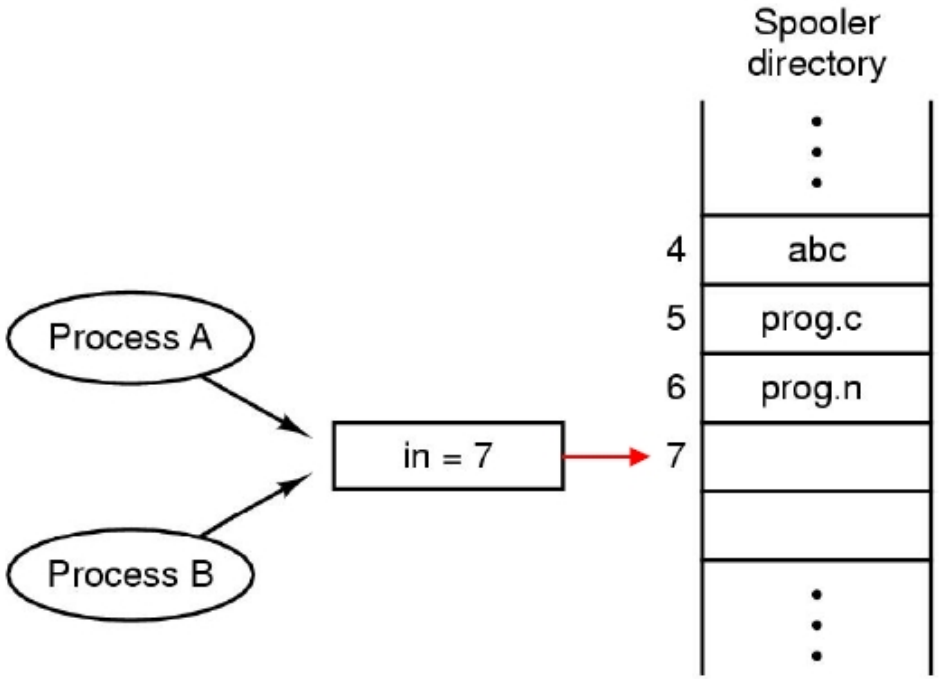
Nebenläufig ablaufende Prozesse (Mehrprogrammbetrieb) haben häufig **gemeinsame Ressourcen**.

- **Geräte.** Drucker, Platten, usw.
- **Daten.** Dateien, Shared Memory, usw.

Zugriffe auf gemeinsame Ressourcen müssen geordnet erfolgen. Um zu vermeiden, dass die Ergebnisse abhängig sind von der **Reihenfolge** der Abarbeitungsschritte der einzelnen Prozesse (**Race Condition**).

Beispiel. Race Condition (1/3)

Zwei Prozesse A und B schreiben Druckaufträge in einen Druckerspooler.
Die Prozesse verwenden hierzu die Kontrollvariable **`in`** (nächster freier Slot im Spoolerdirectory) des Druckerspoolers.



Beispiel. Race Condition (1/3)

```
1  // main process
2
3  start_concurrent(A, B);
```

```
1  // A
2
3  in = spooler->in;
4  spool(spooler, in, jobA);
5  in = in + 1;
6  spooler->in = in;
```

```
1  // B
2
3  in = spooler->in;
4  spool(spooler, in, jobB);
5  in = in + 1;
6  spooler->in = in;
```

Beispiel. Race Condition (2/2)

Prozess A

- Liest die Variable **`in`**.
- Schreibt den Auftrag **`jobA`** in den durch **`in`** angegebenen Slot (= 7) des Spoolers.
- Berechnet den Wert (= 8), mit dem **`in`** aktualisiert werden soll.
- Wird vom Scheduler unterbrochen und von Prozess B aus dem Prozessor verdrängt.

Prozess B

- Liest Variable **`in`**.
- Schreibt den Auftrag **`jobB`** den durch **`in`** angegebenen Slot (= 7) des Spoolers Überschriebt damit den Auftrag **`jobA`**.
- Aktualisiert Variable **`in`** (neuer Wert = 8) und terminiert.

Prozess A

- Nimmt die Bearbeitung wieder auf.
- Aktualisiert Variable **`in`** (neuer Wert = 8). In der (falschen) Annahme das niemand zwischenzeitlich auf den Spooler zugegriffen hat und terminiert.

Der Auftrag von Prozess A wird nie bearbeitet.

Problem

Ein Prozess befindet sich in seinem **kritischen Abschnitt**, wenn er auf gemeinsame Ressourcen zugreift.

Vermeidung von Race Conditions.

- **Wechselseitiger Ausschluss** (*mutual exclusion*). Keine zwei Prozesse dürfen sich gleichzeitig in ihren kritischen Abschnitten befinden.
- Kein Prozess, der außerhalb seines kritischen Abschnitts läuft, darf andere Prozesse blockieren.
- Es dürfen keine Annahmen über Hardware (z.B. Geschwindigkeit und Anzahl der CPUs) und Betriebssystem (z.B. Scheduling-Algorithmus) gemacht werden.
- Kein Prozess sollte ewig darauf warten müssen, in seinen kritischen Abschnitt einzutreten.

Mutex (1/2)

Ein **Mutex** (von *mutual exclusion*) kann zur Synchronisation von nebenläufigen Prozessen benutzt werden.

Ein Mutex **ist** eine neue (geschützte) Variablenart auf der nur **unteilbaren (atomaren) Operationen** ausgeführt werden können.

- Datenstruktur

```
mutex {
  boolean          free;
  prozess_queue    queue;
}
```

- Deklaration und Initialisierung

```
1  mutex m;
2  mutex m = true;
3  mutex m = false;
```

Der Mutex `m` wird deklariert und wie folgt initialisiert

- 1: `m.free=true` 2: `m.free=true` 3: `m.free=false`
- `m.queue` ist eine (leere) Datenstruktur, die Prozesse (Verweise auf Prozesskontrollblöcke) aufnehmen kann.

Mutex (2/2)

Operationen auf einem Mutex.

- **down** (wait, P)

```
down(m) {
  if (m.free)
    m.free = false;
  else {
    block(this_process);
    insert(m.queue, this_process);
  }
}
```

- **up** (signal, V)

```
up(m) {
  if (is_empty(m.queue))
    m.free = true;
  else {
    process = remove(m.queue);
    wake_up(process);
  }
}
```

Beispiel, ohne Race Condition

```
1  // main process
2  global mutex m;
3  start_concurrent(A, B);
```

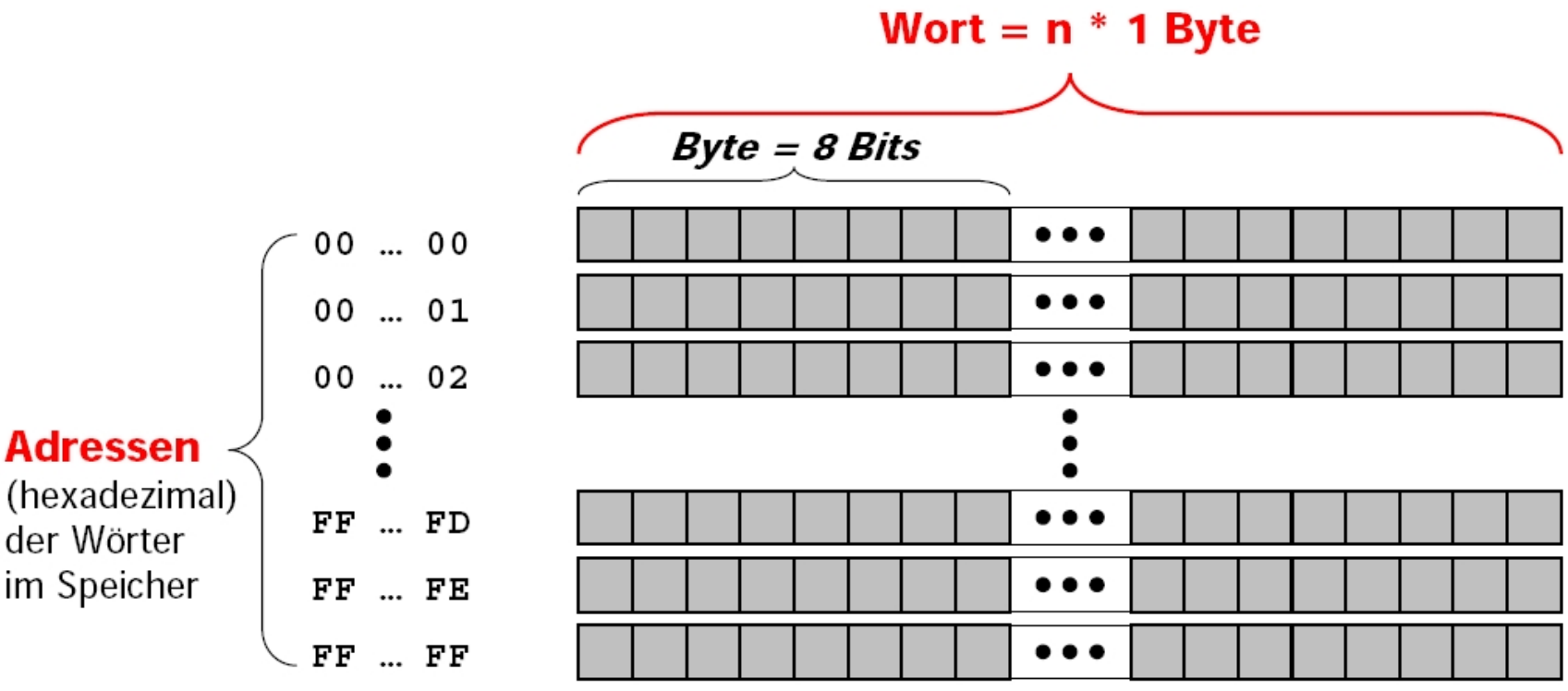
```
1  // A
2  down(m);
3  in = spooler->in;
4  spool(spooler, in, jobA);
5  in = in + 1;
6  spooler->in = in;
7  up(m);
```

```
1  // B
2  down(m);
3  in = spooler->in;
4  spool(spooler, in, jobB);
5  in = in + 1;
6  spooler->in = in;
7  up(m);
```

Inhalt

Betriebssysteme
- Einführung
- Prozessverwaltung
- Scheduling
  - First Come First Served (FCFS)
  - Shortest Job First (SJF)
  - Round-Robin-Scheduling
- Prozess-Synchronisation
  - Mutex
- Speicherverwaltung
  - Swapping

Speicherorganisation



Speicheraufteilung

Die einfachste Speicherverwaltungsstrategie ist den Speicher zwischen Prozesses und Betriebssystem aufzuteilen.

Dann wird entweder einem Prozess der gesamten, für Prozesse reservierten, Speicher zur Verfügung gestellt oder der Speicher wird unter mehreren Prozessen aufgeteilt.

Dabei wird jeder Prozess immer **vollständig** im Hauptspeicher gehalten.

Adressraum

Wird der Speicher zwischen Betriebssystem und einem oder mehreren Prozessen aufgeteilt, ergibt sich ein wesentliches Problem.

Prozesse sprechen grundsätzlich direkt physikalische Adressen an.

- **Relokation.** Ein Programm enthält absolute Adressen, diese müssen relativ zur Lage des Prozesses im Speicher umgesetzt werden.
- **Schutz.** Jeder Prozess soll nur die Adressen des Speicherbereichs ansprechen, der im zugeteilt ist.

Diese Anforderungen werden durch die Einführung des **Adressraums** (*adress space*), einer naheliegenden Speicherabstraktion, erfüllt.

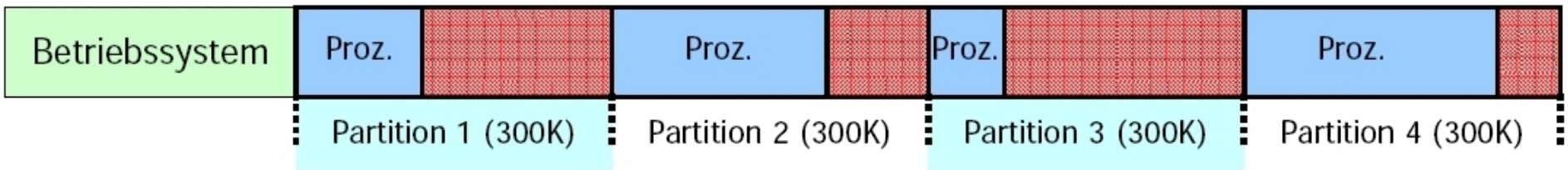
Ein Adressraum ist eine Menge von Adressen, die ein Prozess zur Adressierung des Speichers nutzen kann.

Physikalisch ist der Adressraum, im einfachsten Fall, ein zusammenhängender Speicherbereich (Partition) fester Größe.

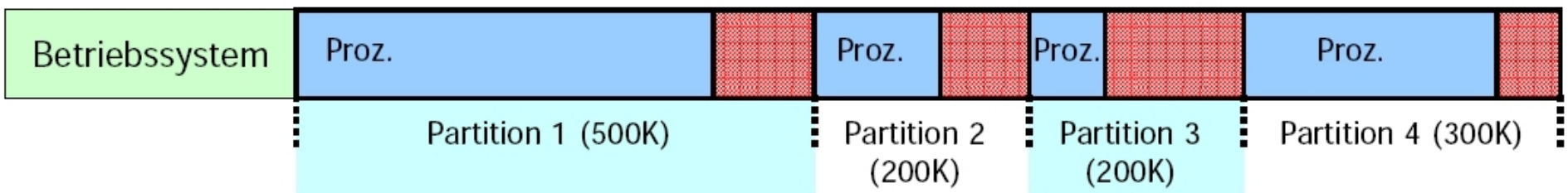
Aus Sicht des Prozesses ist der Adressraum der ganze zur Verfügung stehende Speicher, der in der Regel mit der Adresse 0 beginnt. Die Endadresse ist abhängig von der Größe des zugeteilen Speicherbereichs.

Partition

Feste Partition/Adressräume gleicher Größe.



Feste Partition/Adressräume unterschiedlicher Größen.



Dynamische Relokation

Der Adressraum jedes Prozeses wird auf einen feste Partition des Speichers abgebildet.

Der Prozessor hat zwei zusätzliche Register, das **Basisregister** (`base`) und das **Limitregister** (`limit`). Diese Register gehören zum Kontext des Prozesses.

Beim Erzeugen eines Prozesses wird die physikalische Anfangsadresse der dem Prozess zugeteilten Partition in das Basisregister geladen und die Größe der Partition wird in das Limitregister geladen.

Wenn ein Prozess Speicher referenziert, um eine Befehl zu holen, einen Datenwort zu lesen oder zu schreiben, etc. benutzt der Prozessor automatisch Limit- und Basisregister.

- **Schutz**. Für jede Adresse prüft der Prozessor ob der Wert kleiner oder gleich dem Wert im Limitregister ist, wenn nicht wird der Zugriff verweigert.
- **Relokation**. Zu jeder Adresse addiert der Prozessor automatisch den Wert im Basisregister und schreibt die berechnete Adresse auf den Adressbus.

Bemerkung

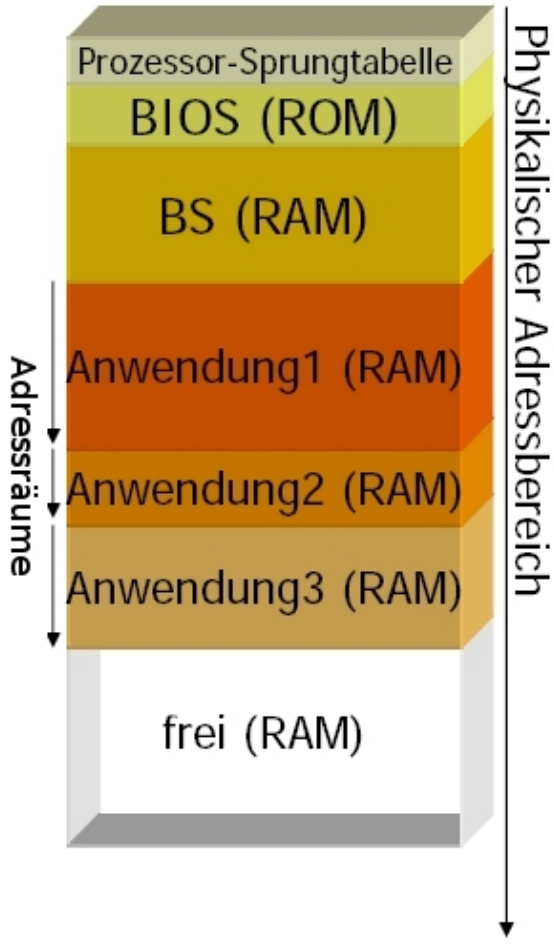
Dynamische Relokation wurde noch im Intel 8088 eingesetzt.

Beispiel UNIX mit dynamischer Relokation

Mehrere Prozesse, jeder hat **eigenen Adressraum**.

Bei jedem Kontextwechsel werden Basis- und Limitregister mitgeführt, sodass jeweils der Adressraum des aktuellen Prozesses gestellt wird.

- **Relokation**. Alle Programme werden für den gleichen Adressraum kompiliert.
- **Schutz**. Fremder Speicher ist gar nicht sichtbar.



Moderne Speicherverwaltung

Die Anforderungen an die Verwaltung des Speichers ergeben sich aus den Design modernen Rechensysteme.

Es laufen soviele Prozesse, dass der physikalische Speicher nicht groß genug ist um alle gleichzeitig aufzunehmen.

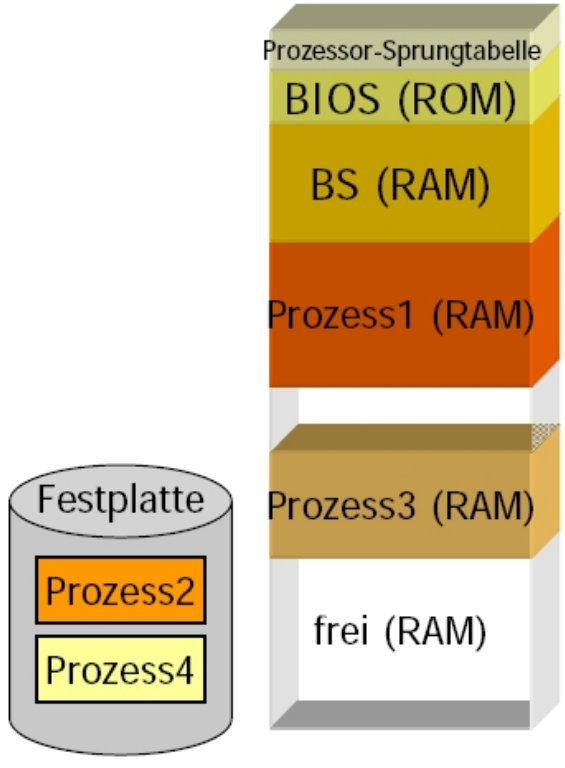
Swapping ist ein grundlegender Ansatz, wie man der Überlastung des Speichers begegnen kann, wobei der Adressraum eines Prozesses entweder vollständig im Speicher gehalten wird oder vollständig auf den Hintergrundspeicher (z.B. Festplatte) ausgelagert wird.

Swapping, Prinzip

Swapping beschreibt das Ein-/Auslagern des **vollständigen** Adressraumes der Prozesses.

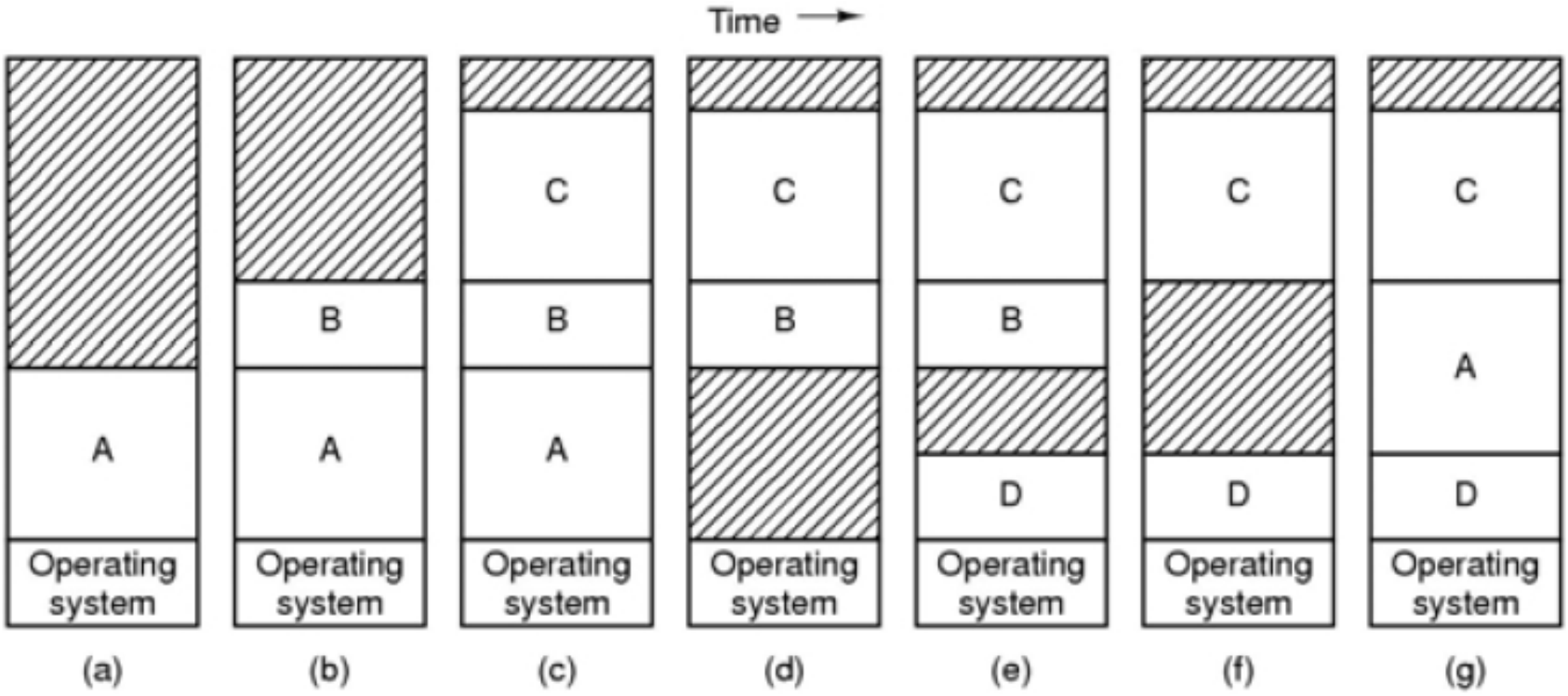
- Auslagern von (z.B. blockierten) Prozessen auf die Festplatte.
- Einlagern von (z.B. bereiten) Prozesses in den Speicher.

Es ist keine spezielle Hardware notwendig. Der Scheduler hat Überblick über den rechnenden, die bereiten und blockierte Prozesse und kann die Ein-/Auslagerung veranlassen.



Swapping, Beispiel

Speicherzuteilung für einzelne Prozesse ändert sich bei der Ein- und Auslagerung (freier Speicher = schraffierte Bereiche).

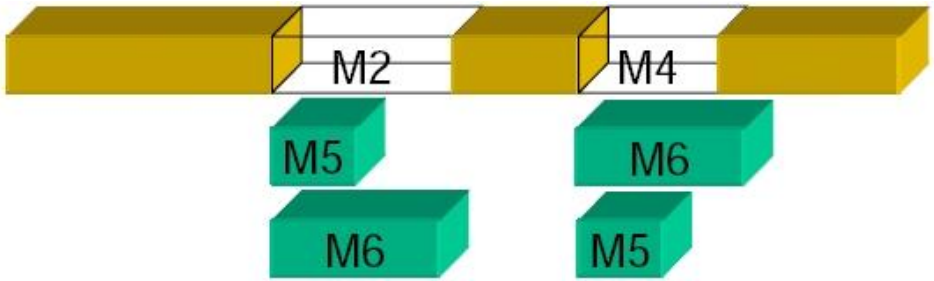


Beim Einlagern kann sich der physiklaische Adressraum (die Partition) des Prozesses ändern. Das heißt bei dynamischer Relokation müssen beim Einlagern eines Prozesses u.U. die Werte im Basis- und Limitregister angepasst werden.

Swapping, Probleme

Positionierung

M2, M4 werden freigegeben, wo M5, M6 platzieren?



Fragmentierung

Anforderung hat nur selten genau die Größe eines freien Speicherbereichs. Nach einiger Zeit entstehen viele kleine *Löcher* im Speicher.

Fazit.

Verschiedene **Speicherbelegungsstrategien** sind möglich.

Speicherbelegungsstrategien

First Fit.
Der erste ausreichend große freie Speicherbereich wird belegt.

Next Fit.
Wie First Fit, aber Suche beginnt an der Stelle, wo zuletzt ein passendes Speicherbereich gefunden wurde. D.h. wenn beim letzten Mal das Loch nicht vollständig geschlossen, sondern lediglich verkleinert wurde, beginnt die Suche bei diesem Loch.

Best Fit.
Es wird der kleinste freie Speicherbereich belegt, der die Anforderung noch erfüllen kann.

Worst Fit.
Es wird der größte freie Speicherbereich belegt.