

# Systemprogrammierung

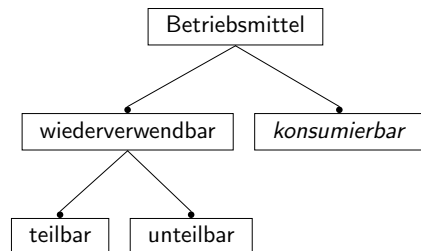
## Betriebsmittelverwaltung

Wolfgang Schröder-Preikschat

Lehrstuhl Informatik 4

4. Juni 2012

## Wiederholung: Betriebsmittel und Betriebsmittelarten



### Hardware

- CPU, Speicher
- Geräte (Peripherie)
- *Signale*

### Software

- Dateien, E/A-Puffer
- Seitenrahmen
- Deskriptoren, ...
- *Signale, Nachrichten*

## Wettbewerb um Betriebsmittel (engl. *resource contention*)

### konsumierbare Betriebsmittel

- **unbegrenzte Anzahl**
- einseitige Synchronisation

### wiederverwendbare Betriebsmittel

- **begrenzte Anzahl**
- mehrseitige Synchronisation

## Gliederung

### 1 Betriebsmittel

- Klassifikation
- Verwaltung

### 2 Verklemmungen

- Grundlagen
- Beispiel
- Gegenmaßnahmen
  - Vorbeugung
  - Vermeidung
  - Erkennung und Erholung

### 3 Zusammenfassung

Prozesse befinden sich untereinander im **Konflikt**, wenn:

- 1 eine **begrenzte Anzahl gemeinsamer Betriebsmittel** vorrätig ist
- 2 diese Betriebsmittel **unteilbar** und von derselben Art sind
- 3 ein Zugriff darauf gleichzeitig geschieht: **gleichzeitige Prozesse** [3]

Prozesse sind im **Streit** (engl. *contention*) um ein Betriebsmittel, wenn einer das Betriebsmittel anfordert, das ein anderer bereits besitzt

- der anfordernde Prozess blockiert und wartet auf die Freigabe des Betriebsmittels durch den Prozess, der das Betriebsmittel belegt
- der das Betriebsmittel belegende Prozess löst den auf die Freigabe des Betriebsmittels wartenden Prozess aus, deblockiert ihn wieder

## Ziele

- konfliktfreie Abwicklung der anstehenden Aufträge
- korrekte Bearbeitung der Aufträge in endlicher Zeit
- gleichmäßige, maximierte Auslastung der Betriebsmittel
- hoher Durchsatz, kurze Durchlaufzeit, hohe Ausfallsicherheit
- ⋮
- Betriebsmittelanforderung frei von Verhungerung/Verklemmung

### Allgemein

- Durchsetzung der vorgegebenen Betriebsstrategie
- eine optimale Realisierung in Bezug auf relevante Kriterien

## Verfahrensweisen

### Statisch

- vor Laufzeit oder vor einem Laufzeitabschnitt
- Anforderung aller (im Abschnitt) benötigten Betriebsmittel
- Zuteilung der Betriebsmittel erfolgt ggf. lange vor ihrer eigentlichen Benutzung
- Freigabe aller belegten Betriebsmittel mit Laufzeit(abschnitt)ende

- Risiko einer nur **suboptimalen Auslastung** der Betriebsmittel

### Dynamisch

- zur Laufzeit, in beliebigen Laufzeitabschnitten
- Anforderung des jeweils benötigten Betriebsmittels bei Bedarf
- Zuteilung des jeweiligen Betriebsmittels erfolgt „im Moment“ seiner Benutzung
- Freigabe eines belegten Betriebsmittels, sobald kein Bedarf mehr besteht

- Risiko der **Verklemmung** von abhängigen Prozessen

## Aufgaben

- **Buchführung** über die im Rechensystem vorhandenen Betriebsmittel
  - Art, Klasse
  - Zugriffsrechte, Prozesszuordnung, Nutzungszustand und -dauer
- **Steuerung** der Verarbeitung von Betriebsmittelanforderungen
  - Entgegennahme, Überprüfung (z.B. der Zugriffsrechte)
  - Einplanung der Nutzung angeforderter Betriebsmittel durch Prozesse
  - Einlastung (Zuteilung) von Betriebsmittel
  - Entzug oder Freigabe von Prozessen benutzter Betriebsmittel

### Betriebsmittelentzug

- Zurücknahme (engl. *revocation*) der Betriebsmittel, die von einem „aus dem Ruder geratenen“ Prozess belegt werden
- bei **Virtualisierung** zusätzlich:
  - Rückforderung und Neuzuteilung eines realen Betriebsmittels, wobei das zugehörige virtuelle Betriebsmittel dem Prozess zugeteilt bleibt

## Gliederung

- 1 Betriebsmittel
  - Klassifikation
  - Verwaltung
- 2 Verklemmungen
  - Grundlagen
  - Beispiel
  - Gegenmaßnahmen
    - Vorbeugung
    - Vermeidung
    - Erkennung und Erholung
- 3 Zusammenfassung

## Parallele und Funktionale Programmierung, 2. Semester

Ergänzung, Verfeinerung bzw. Vertiefung von PFP



- Teil I, 5. **Lebendigkeitsprobleme** [6], speziell. . .
  - ✓ Verklemmung
  - ✓ Bedingungen
  - ✓ Gegenmaßnahmen

... bei Verwendung blockierender Synchronisation, genauer:

- in Semaphore verborgene „Untiefen“ in Maschinenprogrammen
- zur sequentiellen Ausführung kritischer Abschnitte auf Ebene<sub>3</sub>
  - verdeutlicht durch Programme der Ebene<sub>5</sub> (C)

## Stillstand von Prozessen bei aktivem Warten

live-lock ist ...

- ein *deadlock*-ähnlicher Zustand, in dem die involvierten Prozesse zwar nicht blockieren, sie aber auch keine wirklichen Fortschritte in der weiteren Programmausführung erreichen
- wenn die an der Verklemmung beteiligten Prozesse **wechselseitig aktiv** auf die Bereitstellung von Betriebsmitteln **warten**:
  - ohne Prozessorabgabe  $\mapsto$  *busy waiting*
  - mit Prozessorabgabe, in Laufbereitschaft bleibend  $\mapsto$  *lazy waiting*
- das „größere Übel“, da dieser Zustand nicht eindeutig erkennbar ist und damit die Basis zur „Erholung“ fehlt
  - die verklemmten Prozesse haben die **Einplanungszustände** „laufend“ oder „bereit“ (d.h., jeden anderen außer „blockiert“)
  - die Unterscheidung von unverklemmten Prozessen ist kaum möglich

## Stillstand von Prozessen bei passivem Warten

### dead-lock [5]

- 1 a standstill resulting from the action of equal and opposed forces; stalemate
- 2 a tie between opponents in the course of a contest
- 3 DEADBOLT — to bring or come to a deadlock

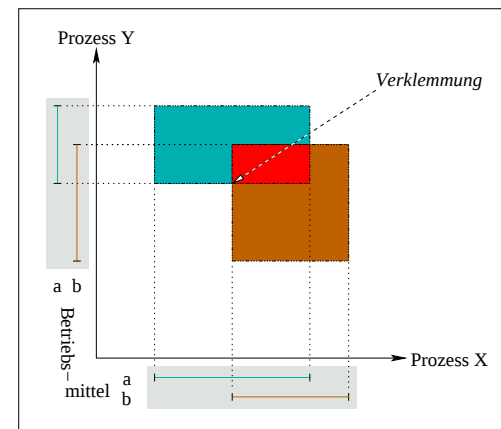
Der Begriff bezeichnet (in der Informatik)

[...] einen Zustand, in dem die beteiligten Prozesse wechselseitig auf den Eintritt von Bedingungen warten, die nur durch andere Prozesse in dieser Gruppe selbst hergestellt werden können. [4]

- das „geringere Übel“ (im Vergleich zum *livelock*), da dieser Zustand eindeutig erkennbar ist und so die Basis zur „Erholung“ gegeben ist
  - die verklemmten Prozesse sind im **Einplanungszustand** „**blockiert**“

## Entstehung von Verklemmungen

Überlappender Zugriff auf gemeinsame unteilbare Betriebsmittel



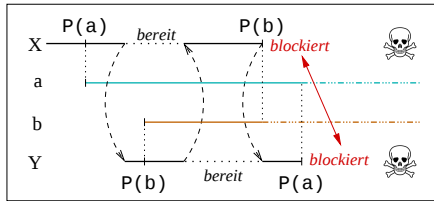
Alles hängt davon ab,

- 1 ob sich die Prozesse einander überlappen und
- 2 welche Auswirkung die Überlappung auf den Zugriff auf gemeinsame Betriebsmittel hat.

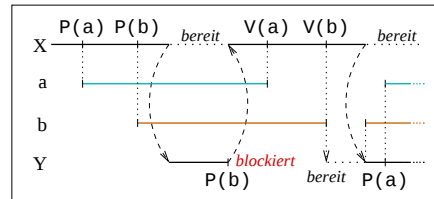
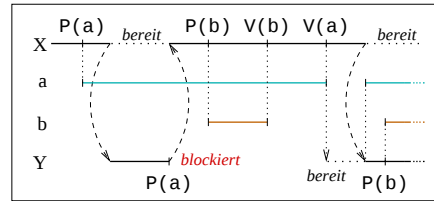
Keine Verklemmung, falls. . .

$P_X$   $B_a$  erst nach  $P_Y$  oder  
 $P_Y$   $B_b$  erst nach  $P_X$  belegt.

## Gleichzeitige Prozesse und Betriebsmittelanforderungen

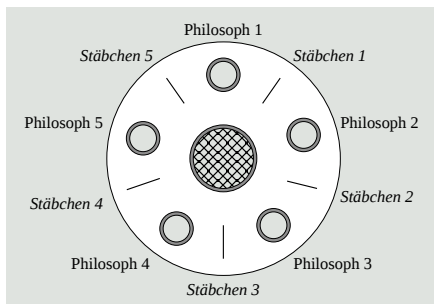


Verklemmungen sind durch geschickte Prozesseinplanung verhinderbar — vorausgesetzt, die Prozesse wie auch ihre Betriebsmittelanforderungen sind alle bekannt.



- kooperative Ausführung von  $P_X$  und  $P_Y \leadsto$  Verklemmungsfreiheit

## Speisende Philosophen: Szenario



Prozess  $\mapsto$  Philosoph  
 Betriebsmittel  $\mapsto$  Stäbchen

- unteilbar

Fünf Philosophen, die nichts anderes zu tun haben, als zu denken und zu essen, sitzen an einem runden Tisch. Denken macht hungrig — also wird jeder Philosoph auch essen. Dazu benötigt ein Philosoph jedoch stets beide neben seinem Teller liegenden Stäbchen.

### Verklemmungsfall...

Die Philosophen nehmen zugleich das eine Stäbchen auf und greifen anschließend auf das andere zu.

## Voraussetzungen für Verklemmungen

Notwendige und hinreichende Bedingungen

notwendige Bedingungen (müssen erfüllt sein, damit die Aussage zutreffen kann)

- exklusive Belegung von Betriebsmitteln („*mutual exclusion*“)
  - die umstrittenen Betriebsmittel sind nur unteilbar nutzbar
- Nachforderung von Betriebsmitteln („*hold and wait*“)
  - die umstrittenen Betriebsmittel sind nur schrittweise belegbar
- kein Entzug von Betriebsmitteln („*no preemption*“)
  - die umstrittenen Betriebsmittel sind nicht rückforderbar

hinreichende Bedingung (muss erfüllt sein, damit die Aussage zutrifft bzw. „bewiesen“ ist)

- zirkulares Warten (engl. *circular wait*)
  - Existenz einer geschlossenen Kette wechselseitig wartender Prozesse

## Speisende Philosophen: Umsetzung — mit Problemen

```
void phil(int who) {
    for (;;) {
        think();
        grab(who);
        munch();
        drop(who);
    }
}
```

```
void think() { ... }
void munch() { ... }
```

```
semaphore_t rod[5] = {
    {1, 0}, {1, 0}, {1, 0}, {1, 0}, {1, 0}
};

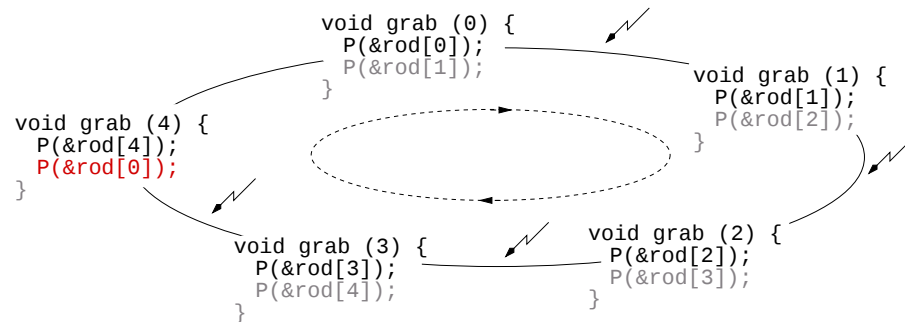
void grab(int who) {
    P(&rod[who]);
    P(&rod[(who + 1) % NPHIL]);
}

void drop(int who) {
    V(&rod[who]);
    V(&rod[(who + 1) % NPHIL]);
}
```

$P()$  fordert zu einem Zeitpunkt nur ein Stäbchen (engl. *rod*) an  
 $V()$  gibt ein Stäbchen frei

## Speisende Philosophen: Verklemmung

„Ende einer Dienstfahrt“:  $\text{grab}(i-1) \models \text{Philosoph}_i, \text{rod}[i-1] \models \text{Stäbchen}_i$



- 1 Philosoph<sub>1</sub> nimmt Stäbchen<sub>1</sub> auf
- 2 Philosoph<sub>2</sub> überlappt Philosoph<sub>1</sub>, nimmt Stäbchen<sub>2</sub> auf
- 3 Philosoph<sub>3</sub> überlappt Philosoph<sub>2</sub>, nimmt Stäbchen<sub>3</sub> auf
- 4 Philosoph<sub>4</sub> überlappt Philosoph<sub>3</sub>, nimmt Stäbchen<sub>4</sub> auf
- 5 Philosoph<sub>5</sub> überlappt Philosoph<sub>4</sub>, nimmt Stäbchen<sub>5</sub> auf, **fordert Stäbchen<sub>1</sub> an**

## Speisende Philosophen: Kritischer Abschnitt

```

semaphore_t lock = {1, 0};

void grab(int who) {
  P(&lock);
  P(&rod[who]);
  P(&rod[(who + 1) % NPHIL]);
  V(&lock);
}

```

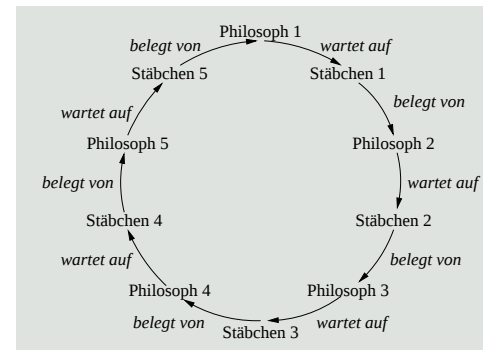
Philosoph<sub>who</sub> fordert die benötigten Stäbchen „atomar“ (unteilbar) an

- ein **binärer Semaphor** (lock) zum wechselseitigen Ausschluss
- „hold and wait“ vorgebeugt

**Problemfall: Verdrängung in munch(), Überlappung mit grab(int)**

- Philosoph<sub>who+1</sub> muss auf Stäbchen<sub>who+1</sub> (rechte Hand) warten
  - blockiert im kritischen Abschnitt, ohne diesen freizugeben
- Philosoph<sub>who+2</sub> muss auf Freigabe des kritischen Abschnitts warten
  - ebenso ergeht es den beiden anderen Philosophen
- schlimmstenfalls kann immer nur ein Philosoph essen

## Speisende Philosophen: Zirkulares Warten



**Betriebsmittelgraph** zeigt für jedes Betriebsmittel, welcher Prozess es belegt

**Wartegraph** verbucht für jeden Prozess, auf welches Betriebsmittel er wartet

- ein geschlossener Kreis (im Wartegraphen) erfasst all die Prozesse, die sich zusammen im **Deadlock** befinden.
- es muss sichergestellt sein, dass ein solcher Kreis entweder nicht entstehen oder dass er erkannt und „durchbrochen“ werden kann

## Speisende Philosophen: Synchronisationsaspekte [2]

### mehrseitige Synchronisation

- **wechselseitiger Ausschluss** beim Gebrauch wiederverwendbarer, unteilbarer (nicht entziehbarer) Betriebsmittel
- keine zwei benachbarten Philosophen können gleichzeitig dasselbe Stäbchen gemeinsam benutzen

### einseitige Synchronisation

- ein Philosoph muss warten, bis seine beiden Nachbarn ihm ein Stäbchen zur Verfügung gestellt haben

### Randbedingungen

- jeder Philosoph fordert die Stäbchen **nacheinander** an
  - kein Philosoph legt ein Stäbchen zurück, wenn er feststellt, dass das andere bereits vom Nachbarn aufgenommen worden ist
- ein Philosoph kann seinem Nachbarn ein bereits aufgenommenes Stäbchen **nicht entziehen**

## Speisende Philosophen: Praxisrelevanz

⋮

- auf mehreren Magnetbändern vorliegende Daten sortieren
- einen kontinuierlichen Strom kodierter Informationen umkodieren
- Nachrichten von einem Eingangsport auf einen Ausgangsport leiten
- Pakete auf einem Ringnetz zur übernächsten Station durchschleusen
- Daten von dem einen *Backup Medium* auf ein anderes transferieren

- überall dort, wo eine von mehreren Prozessen benutzte Routine (mindestens) zwei wiederverwendbare Betriebsmittel zugleich benötigt, diese aber nur nacheinander angefordert werden können

## Verklemmungsvermeidung (engl. *deadlock avoidance*)

Verhinderung von Wartezyklen durch **strategische Maßnahmen**:

- keine der drei notwendigen Bedingungen wird entkräftet
- fortlaufende **Bedarfsanalyse** schließt zirkulares Warten aus

Prozesse und ihre Betriebsmittelanforderungen sind zu steuern:

- das System wird (laufend) auf „**unsichere Zustände**“ hin überprüft
- Zuteilungsablehnung im Falle ungedeckten Betriebsmittelbedarfs
- anfordernde Prozesse nicht bedienen bzw. frühzeitig suspendieren
- Betriebsmittelnutzung einschränken  $\leadsto$  „**sichere Zustände**“

- *à priori* Wissen erforderlich: maximaler Betriebsmittelbedarf

## Verklemmungsvorbeugung (engl. *deadlock prevention*)

**indirekte Methoden** entkräften eine der Bedingungen 1–3

- 1 nicht-blockierende Verfahren verwenden
- 2 Betriebsmittelanforderungen unteilbar (atomar) auslegen
- 3 Betriebsmittelentzug durch **Virtualisierung** ermöglichen
  - virtueller Speicher, virtuelle Geräte, virtuelle Prozessoren

**direkte Methoden** entkräften Bedingung 4

- 4 lineare/totale Ordnung von Betriebsmittelklassen einführen:
  - Betriebsmittel  $B_i$  ist nur dann erfolgreich vor  $B_j$  belegbar, wenn  $i$  linear vor  $j$  angeordnet ist (d.h.  $i < j$ ).

- Regeln, die das Eintreten von Verklemmungen verhindern
- Methoden, die zur Entwurfs- bzw. Implementierungszeit greifen

## Sicherer/Unsicherer Zustand: Speisende Philosophen

**Ausgangspunkt** fünf Stäbchen sind insgesamt vorhanden

- jeder der fünf Philosophen braucht zwei Stäbchen zum Essen

**Situation**  $P_1$ ,  $P_2$  und  $P_3$  haben je ein Stäbchen; zwei Stäbchen sind frei

- $P_4$  fordert ein Stäbchen an  $\rightarrow$  ein Stäbchen wäre dann noch frei
  - **sicherer Zustand**: einer von dann 4 Philosophen könnte essen
  - die Anforderung von  $P_4$  wird akzeptiert
- $P_5$  fordert ein Stäbchen an  $\rightarrow$  kein Stäbchen wäre dann mehr frei
  - **unsicherer Zustand**: keiner der Philosophen könnte essen
  - die Anforderung von  $P_5$  wird abgelehnt,  $P_5$  muss warten
- haben vier Philosophen je ein Stäbchen, wird der fünfte gestoppt

## Sicherer/Unsicherer Zustand: Leistungsvermittlung

**Ausgangspunkt** ein Vermittlungsrechner mit 12 Kommunikationskanälen

- Prozess  $P_1$  benötigt max. 10 Kanäle,  $P_2$  vier und  $P_3$  neun

**Situation:**  $P_1$  belegt fünf Kanäle,  $P_2$  und  $P_3$  je zwei; drei Kanäle sind frei

- $P_3$  fordert einen Kanal an, zwei bleiben frei → **unsicherer Zustand**
  - $P_3$  könnte noch sechs Kanäle anfordern:  $6 > 2$
  - die Anforderung von  $P_3$  wird abgelehnt,  $P_3$  muss warten
- $P_1$  fordert zwei Kanäle an, einer bleibt frei → **unsicherer Zustand**
  - $P_1$  könnte noch drei Kanäle anfordern:  $3 > 1$
  - die Anforderung von  $P_1$  wird abgelehnt,  $P_1$  muss warten
- **sichere Prozessfolge:**  $P_2 \rightarrow P_1 \rightarrow P_3$

## Verklemmungserkennung (engl. *deadlock detection*)

Verklemmungen werden (stillschweigend) in Kauf genommen...

- nichts im System verhindert das Auftreten von Wartezyklen
- keine der vier Bedingungen wird entkräftet

Ansatz: **Wartegraph** erstellen und auf Zyklen hin untersuchen  $\leadsto O(n^2)$

- zu häufige Überprüfung verschwendet Betriebsmittel/Rechenleistung
- zu seltene Überprüfung lässt Betriebsmittel brach liegen

**Zyklensuche** geschieht zumeist in großen Zeitabständen, wenn...

- Betriebsmittelanforderungen zu lange andauern
- die Auslastung der CPU trotz Prozesszunahme sinkt
- die CPU bereits über einen sehr langen Zeitraum untätig ist

## Verklemmungsfreiheit

Verhinderung unsicherer Zustände

**sicherer Zustand** ist, wenn eine Folge der Verarbeitung vorhandener

Prozesse existiert, in der alle Betriebsmittelanforderungen erfüllbar sind

**unsicherer Zustand** ist, wenn eine solche Folge nicht existiert; Erkennung dieses Zustands z.B. durch:

- **Betriebsmittelbelegungsgraph** (engl. *resource allocation graph*)
  - damit Vorhersage über das Eintreten von Zyklen treffen  $\leadsto O(n^2)$
  - bei jeder Betriebsmittelanforderung den Graphen überprüfen
- **Bankiersalgorithmus** (engl. *banker's algorithm* [1])
  - 1 Prozesse beenden ihre Operationen in endlicher Zeit
  - 2 Betriebsmittelbedarf aller Prozesse übersteigt Gesamtbestand nicht
  - 3 Prozesse definieren einen verbindlichen **Kreditrahmen**
  - 4 Betriebsmittelzuteilung erfolgt variabel innerhalb dieses Rahmens
- die Verfahrensweisen führen Prozesse dem *long-term scheduling* zu

## Verklemmungsauflösung (engl. *deadlock resolution*)

Erholungsphase nach der Erkennungsphase

**Prozesse abbrechen** und dadurch Betriebsmittel frei bekommen

- verklemmte Prozesse schrittweise abbrechen (gr. Aufwand)
  - mit dem „effektivsten Opfer“ (?) beginnen
- alle verklemmten Prozesse terminieren (gr. Schaden)

**Betriebsmittel entziehen** und mit dem „effektivsten Opfer“ (?) beginnen

- der betreffende Prozess ist zurückzufahren/wieder aufzusetzen
  - Transaktionen, *checkpointing/recovery* (gr. Aufwand)
- ein Aushungern der zurückgefahrenen Prozesse ist zu vermeiden

### Gratwanderung zwischen Schaden und Aufwand

- Schäden sind unvermeidbar und die Frage ist, wie sie sich auswirken

## Gliederung

- 1 Betriebsmittel
  - Klassifikation
  - Verwaltung
- 2 Verklemmungen
  - Grundlagen
  - Beispiel
  - Gegenmaßnahmen
    - Vorbeugung
    - Vermeidung
    - Erkennung und Erholung
- 3 Zusammenfassung

## Resümee

- **Betriebsmittel** zeigen sich als Entitäten von Hardware und Software
  - wiederverwendbar
  - konsumierbar
  - begrenzt verfügbar: teilbar, unteilbar
  - unbegrenzt verfügbar
- Ziele, Aufgaben und Verfahrensweise der **Betriebsmittelverwaltung**
  - Betriebsmittelanforderung frei von Verhungering/Verklemmung
  - Buchführung der Betriebsmittel, Steuerung der Anforderungen
  - statische/dynamische Zuteilung von Betriebsmitteln
- für eine Verklemmung müssen **vier Bedingungen** gleichzeitig gelten
  - exklusive Belegung, Nachforderung, kein Entzug von Betriebsmitteln
  - zirkulares Warten der die Betriebsmittel beanspruchenden Prozesse
  - nicht zu vergessen: Verklemmung bedeutet „**deadlock**“ oder „**livelock**“
- die **Gegenmaßnahmen** sind:
  - Vorbeugen, Vermeiden, Erkennen & Erholen
  - die Verfahren können im Mix zum Einsatz kommen

## Nachlese . . .

Verfahren zum **Vermeiden/Erkennen** sind eher praxisirrelevant

- sie sind kaum umzusetzen, zu aufwändig und damit nicht einsetzbar
- zudem macht die — nach wie vor stark ausgeprägte — Vorherrschaft sequentieller Programmierung diese Verfahren wenig notwendig

**Vorbeugung** ist besser als heilen — notwendige Bedingungen entkräften:

- 1 durch **nicht-blockierende Synchronisation** kritischer Abschnitte
  - keine exklusive Belegung von kritischen Abschnitten durchführen
- 2 durch **Virtualisierung**, wenn 1. nicht möglich/unpraktisch ist
  - Prozesse beanspruchen/belegen nur **virtuelle Betriebsmittel**
  - der Trick besteht darin, in kritischen Momenten den Prozessen (ohne ihr Wissen) **physische Betriebsmittel** entziehen zu können
  - dadurch wird die Bedingung der Nichtentziehbarkeit entkräftet

## Literaturverzeichnis

- [1] DIJKSTRA, E. W.: Cooperating Sequential Processes / Technische Universiteit Eindhoven. Eindhoven, The Netherlands, 1965 (EWD-123). – Forschungsbericht. – (Reprinted in *Great Papers in Computer Science*, P. Laplante, ed., IEEE Press, New York, NY, 1996)
- [2] DIJKSTRA, E. W.: Hierarchical Ordering of Sequential Processes. In: *Acta Informatica* 1 (1971), Jun., Nr. 2, S. 115–138
- [3] HANSEN, P. B.: *Operating System Principles*. Prentice Hall International, 1973
- [4] NEHMER, J. ; STURM, P. : *Systemsoftware: Grundlagen moderner Betriebssysteme*. dpunkt Verlag GmbH, 2001. – ISBN 3-898-64115-5
- [5] NEUFELD, V. E. (Hrsg.): *Webster's New World Dictionary*. Third College. Simon & Schuster, Inc., 1988. – ISBN 0-13-947169-3
- [6] PHILIPPSEN, M. : *Parallele und Funktionale Programmierung*. <http://www2.cs.fau.de/Lehre/SS20??/PPP/material/>, jährlich. – Vorlesungsfolien