2.4 Synchronisationsbedarf und -probleme

✎ Synchronisationsbedarf

- Vermeiden von Nichtdeterminiertheit

➩Schreib/Schreib-Konflikt

```
conc
    buf = produce() ||
    buf = produce()
end conc;
```

<u>Lösung (in Java):</u>

```
public class Buffer {
    BufferItem buf;
    public BufferItem produce() {
        ...
    }
    public synchronized void write(BufferItem bi) {
        ...
    }
}

Buffer p;
    T1: ...; b1 = p.produce(); p.write(b1); ...
    T2: ...; b2 = p.produce(); p.write(b2); ...
```


- Vermeiden von Inkonsistenz

➩ Konkurrenzsituation im Eisenbahnverkehr

<u>Lösung (in C++):</u>

```
sema_t crit_s;
sema_init(&crit_s, 1, UNSYNC_THREAD, NULL);
Train1:   ... sema_wait(&crit_s); ...; sema_post(&crit_s); ...
Train2:   ... sema_wait(&crit_s); ...; sema_post(&crit_s); ...
```

➩ Schreib/Lese-Konflikt

```
public class Bank {
    static int CASH = 0;
    static int CHECK = 1;
    int total = 0;
    int checks = 0;
    public void deposit(int sort, int m) {
        if (sort==CHECK) {
            checks = checks + m;
        }
        total = total + m;
    }
    public void balance() {
        System.out.println("Total: " + total);
        System.out.println("Check: " + checks);
} }
```


Bank möller;
H. Möller möller.deposit(scheck, 200)
F. Möller möller.balance()

H.Möller	F.Möller	total	check
		100	0
checks+200		100	200
	balance	100	200
total+200		300	200

<u>Lösung (in Java):</u>

```
public synchronized void deposite(int sort, int m) {...}
public synchronized void balance() {...}
```

- Gewährleisten einer logischen Abfolge

➩ C / C++

```
ifstream &onedata;  char *st[100];

P1:                             P2:
...                             ...
onedata.open(name);             onedata >> st;
...                             ...
```

<u>Lösung (in C++):</u>

```
sema_t await;
sema_init(&await, 0, UNSYNC_THREAD, NULL);
P1: ...; onedata.open(name); sema_post(&await); ...
P2: ...; sema_wait(&await); onedata >> st; ...
```


✎ Typische Synchronisationsprobleme

- Betriebsmittelnutzung
  - Die exklusive Nutzung eines Betriebsmittels bildet einen kritischen Abschnitt der Ordnung 1.
  - Stehen von einer bestimmten Sorte von exklusiven Betriebsmitteln N gleichwertige zur Verfügung, bildet die Nutzung aller Betriebsmittel zusammen einen kritischen Abschnitt der Ordnung N.
  - Für die Nutzung von Betriebsmitteln sind mehrseitige Synchronisationen erforderlich. Aufgrund dieser wichtigen Anwendung nennt man die mehrseitige Synchronisation auch <u>ressourcenorientierte Synchronisation</u> (*resource-oriented synchronization*).
- Leser und Schreiber
  - Prozesse, die gemeinsame Variablen schreibend zugreifen können, nennt man <u>Schreiber</u> (*writer*).
  - Prozesse, die gemeinsame Variablen lesend zugreifen können, nennt man <u>Leser</u> (*reader*).
  - **Beliebig** viele Leser können nebenläufig auf die Variablen zugreifen, aber Schreiber benötigen **exklusiven** Zugriff auf die Variablen.

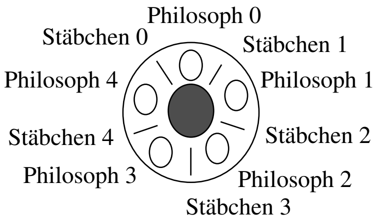
➩ Erstes Leser/Schreiber-Problem

Kein Leser soll beim Betreten seines kritischen Abschnitts warten müssen, wenn nicht bereits ein Schreiber seinen kritischen Abschnitt betreten hat.

➩ Zweites Leser/Schreiber-Problem

Die Schreiber sollen so schnell wie möglich Zugang zum kritischen Abschnitt erhalten. Alle Leser, die in dem Moment, wo der Schreiber seinen Anspruch an dem Kritischen Abschnitt anmeldet, im kritischen Abschnitt arbeiten, dürfen ihre Arbeit dort beenden.

• Speisende Philosophen



- Ein Philosoph muß warten, bis ein Stäbchen auf dem Tisch liegt. Dies ist eine einseitige Synchronisation.
- Zwei Philosophen dürfen nicht gleichzeitig zum selben Stäbchen greifen, wozu eine mehrseitige Synchronisation erforderlich ist.
- Man muß darauf achten, daß nicht jeder Philosoph das rechts neben seinem Teller liegende Stäbchen aufnimmt und dann auf das seines Nachbarn wartet. (gegenseitige Behinderung)
- Gerechtigkeit ist auch wichtig, damit alle Philosophen gleich satt werden und keiner verhungert.

2.5 Verklemmungen

✎ **Verklemmungen**

Eine Situation, in der Prozesse auf Ereignisse warten, die nicht mehr eintreten können, wird als <u>Verklemmung</u> (*deadlock*) bezeichnet.

➩ Speisende Philosophen

➩ Zwei Studenten benötigen zur Vorbereitung auf die Prüfung ein zweibändiges Lehrbuch.

Student A ist mit dem Bearbeiten von Band 1 fertig. Er will Band 1 noch nicht zurückgeben, da er beim Durcharbeiten des ersten Kapitels von Band 2 den erst Band noch häufig zum Nachschlagen benötigen wird.

Student B hat “vorsorglich” Band 2 ausgeliehen, als er entdeckt, daß Band 1 bereits nicht mehr vorhanden ist.

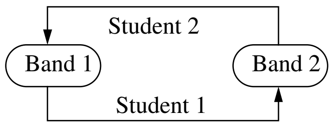
✎ **Verklemmungsbedingungen**

Verklemmungen beim Streit um kritische Abschnitte können nur dann auftreten, wenn die folgenden vier **notwendigen und hinreichenden** Bedingungen erfüllt sind:

- Die umstrittenen kritischen Abschnitte sind nur exklusiv nutzbar.
- Die umstrittenen kritischen Abschnitte können nicht entzogen werden.
- Die Prozesse belegen die schon zugewiesenen kritischen Abschnitte auch dann, wenn sie auf die Zuweisung weiterer kritischer Abschnitte warten.

- Es gibt eine zyklische Kette von Prozessen, von denen jeder mindestens einen kritischen Abschnitt besitzt, den der nächste Prozeß in der Kette benötigt.

➩



✎ **Verklemmungsvermeidung**

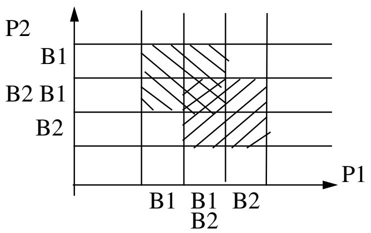
- Regeln

  Wenn man für die Vergabe von kritischen Abschnitten Regeln festlegt, die dafür sorgen, daß immer mindestens eine der im letzten Abschnitt genannten Bedingungen nicht erfüllt ist, kann keine Verklemmung auftreten.

  - gemeinsame Nutzung
  - Zurückgeben bei Bedarf
  - einmalige Anforderung
  - Anforderungsordnung

- Bedarfsanalyse

➩



• Algorithmen

➩ Bankiersalgorithmus (Banker’s algorithm) (Dijkstra 1968)

<u>Vorraussetzungen</u>

a. Prozesse benötigen nur eine Art von Betriebsmitteln.
b. Die maximalen Betriebsmittelanforderungen jedes Prozesses sind im voraus bekannt.

<u>Problembeschreibung</u>

Ein Bankier besitzt einen bestimmten Geldbetrag und verleiht davon Geld an seine Kunden. Alle Kunden schließen mit dem Bankier einen Vertrag ab, der vier Punkte enthält:

1. Die kunden Verpflichten sich, ihre Geschäfte in endlicher Zeit abzuschließen und spätestens dann das gesamte geliehene Geld zurückzugeben.
2. Der Kredit jedes Kunden ist nicht größer als das Gesamtkapital des Bankiers.
3. Alle Kunden setzen bei Vertragsabschluß ihren Kreditrahmen fest, der bei einer Kreditname nie überschritten wird.
4. Die Kunden können jederzeit Geld bis zur Höhe ihres jeweiligen Kreditrahmens fordern, müssen dabei jedoch in Kauf nehmen, daß der Bankier diese Forderungen erst nach einer gewissen Zeit erfüllt.

$$\forall\ (1 \le i \le N) : \text{bedarf}[s[i]] - \text{geliehen}[s[i]] \le \text{verfügbar} + \text{geliehen}[s[0]] + ... + \text{geliehen}[s[i-1]]$$

```
prozesse: 1, . . ., N
int bargeld = kapital;
int geliehen[N], bedarf[N];

bool sicher() {

   int verfügbar = bargeld;
   int zweifelhaft[N] = {true, . . ., true};
   int p =0;

   while (p < N) {
      if (zweifelhaft[p] &&
                        bedarf[p]-geliehen[p]) <= verfügbar) {
         zweifelhaft[p] = false;
         verfügbar = verfügbar + geliehen[p];
         p = 0;
      }
      else p = p+1;
   }

   return (verfügbar==gesamte_kapital)
}
```