Universität Potsdam
Lehrstuhl
Didaktik der Informatik

Nebenläufigkeit im Schulfach Informatik

- Legitimation und Vermittlung -

Marco Thomas

Inhalt:

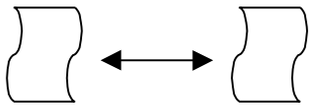
- Zum Begriff "Nebenläufigkeit"
- Eine fundamentale, allgemeinbildende Idee
- Vermittlung im Unterricht
- Beispiele

Nebenläufigkeit

Def. 1

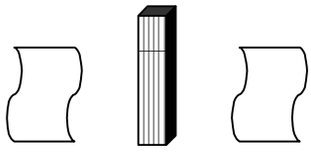
Zwei Ereignisse/Vorgänge heißen **nebenläufig**, wenn sie

- zeitweilig voneinander unabhängig auftreten bzw. ablaufen können und
- abhängigkeitserzeugende Wirkungszusammenhänge zwischen ihnen bestehen können.



Def. 2

Nebenläufige Ereignisse oder Vorgänge heißen **parallel** oder gleichzeitig, wenn zwischen ihnen keine Wirkungszusammenhänge bestehen, die die Unabhängigkeit beeinflussen.



Beispiele

- Während eines Vortrags können die Zuhörer parallel irgendwelchen anderen Tätigkeiten nachgehen.
- Ein zum Vortrag nebenläufiges Gespräch zwischen zwei Zuhörern zu einer aufgelegten Vortragsfolie kann gestört werden, wenn eine andere Vortragsfolie auf den Projektor aufgelegt wird.

- Zwei Prozesse auf einem Rechner mit einer CPU können nebenläufig, aber nicht parallel ablaufen.
- Zwei nicht-vernetzte Rechner arbeiten vollständig parallel (gleichzeitig).

Nebenläufigkeit ist eine fundamentale Idee der Informatik[1] !

- Horizontalkriterium (d. h. in vielen Bereichen der Wissenschaft erkennbar):
  - Petri-Netze (Theoret. Informatik)
  - Betriebssysteme und Programmiersprachen (Prakt. Informatik)
  - Rechnerarchitektur und Prozeßdatenverarbeitung (Techn. Informatik)
  - Informationssysteme (Angew. Informatik)
  - Projektunterricht (Didaktik der Informatik)
- Vertikalkriterium (d. h. auf jedem intellektuellem Niveau vermittelbar)

  Primarstufe:
  - gemeinsame Nutzung eines Druckers, des Rechners oder Internets
  - nebenläufiges Sortieren durch Mischen

  Unter-/Mittelstufe:
  - Erstellung und Programmierung von multimedialen Dokumentationen
  - Analyse und Konstruktion von Systemen mithilfe grafikorientierter Modellbildungswerkzeuge[2] der Informatik
  - Petri-Netze mittels entsprechender Unterrichtshilfen

  Oberstufe:
  - Analyse und Konstruktion von nebenläufigen Prozessen mithilfe von Programmiersprachen
  - nebenläufige Algorithmen und Rechnerarchitekturen
- Sinnkriterium (d. h. einen Bezug zum Alltag und zur Lebenswelt besitzt)
  - Nebenläufigkeit ist ein Alltagsprinzip
  - die explizite Modellierung komplexer Systeme fördert die Fähigkeit zur Bewältigung
  - nebenläufige Prozesse in der technisierten Lebenswelt
- Zeitkriterium (d.h. in der historischen Entwicklung der Wissenschaft längerfristig aufzeigbar)

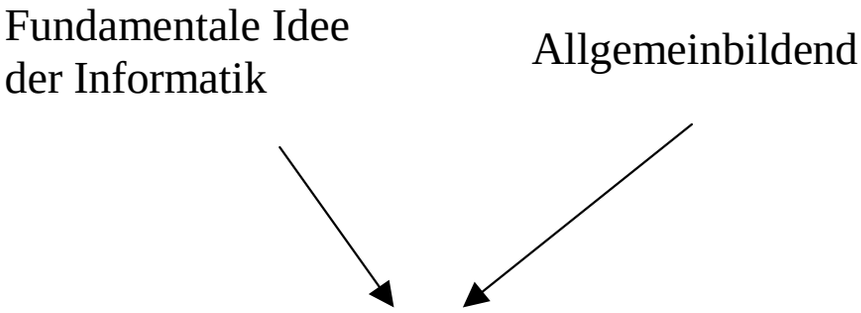
  1600 Rechenmaschinen von Schickard, Pascal, Leibniz, ...
  1960 Korroutinen, Semaphor, Petri-Netze
  1970 OOP, Monitore, Verteilte Systeme
  1980 parallele Programmiersprachen, Parallelcomputer
  1990 Neuronale Netze, Massiv parallele Programmierung

[1] im Sinne Schwills [Schwill, A. 1994]
[2] Grafikorientierte Modellbildungswerkzeuge, die echte Nebenläufigkeit ermöglichen, sind noch zu entwickeln.

Hypothese: **Nebenläufigkeit ist ein allgemeinbildungswürdiges Konzept[3] !**

1. **Vorbereitung auf zukünftige Lebenssituationen**
   - Nebenläufigkeit ist ein wichtiges Element natürlicher und künstlicher Systeme.
   - Das Arbeiten mit nebenläufigen Modellen zur Bewältigung vernetzter Systeme wird zukünftig an Bedeutung gewinnen.
   - Vorherrschendes lineares und eingeschränktes Denken bei geringem Abstraktionsvermögen wird bei Schülern und Ausgelernten beklagt
2. **Stiftung kultureller Kohärenz**
   - ergibt sich aus der Eigenschaft der Nebenläufigkeit eine fundamentale Idee der Informatik zu sein und der zunehmenden Bedeutung informatischer Methoden und Werkzeuge im alltäglichen Leben
   - Nebenläufigkeit ist schon immer ein wichtiges Element in Situationen des alltäglichen Lebens gewesen
3. **Aufbau einer zeitgemäßen Weltorientierung**
   - die Lebenswelt kann durch das Verständnis von nebenläufigen Prozessen und der dabei auftretenden Phänomene besser erklärt und geplant werden
   - Förderung vernetzten, globalen Denkens und Handelns
4. **Anleitung zum kritischen Vernunftgebrauch**
   - die Übertragbarkeit wissenschaftlicher Erkenntnisse und Methoden zu nebenläufigen Prozessen auf die reale Lebenswelt, als auch die von künstlichen auf natürliche Systeme, und jeweils umgekehrt, ist kritisch zu reflektieren, um Unterschiede und Gemeinsamkeiten bewusst zu machen.
5. **Entfaltung eines verantwortlichen Umgangs mit den erworbenenen Kompetenzen**
   - s. unter (4)
6. **Stärkung des Schüler-Ichs**
   - selbständiges Problemlösen durch z. B. Transfer erworbener Kenntnisse auf Nebenläufigkeit enthaltene Alltagssituationen
   - rasche Erfolgskontrolle und dadurch Motivation zur Auseinandersetzung mit weiteren komplexen und nebenläufigen Systemen
   - produktorientiertes Planen nebenläufiger Abläufe

[3] Begründung anhand der Postulate allgemeiner Bildung nach Bussmann, Heymann in [Engbring, D. 1996]



Folgerung:
Nebenläufigkeit ist ein obligatorisch zu behandelnder Gegenstand im Informatikunterricht.

Feststellung:

- Bis vor kurzem gab es kaum Möglichkeiten Nebenläufigkeit im Unterricht gezielt zu thematisieren.
- In den Lehrplänen wird Nebenläufigkeit meist nicht explizit erwähnt.

Vermittlung von Nebenläufigkeit im Unterricht

Prinzip: exemplarische Beispiele

Fundamentale Ideen, wie die Nebenläufigkeit und damit verbundene Begriffe, sind nach Nievergelt[4] anhand von klassischen Beispielen (besser: an exemplarischen Problemen) zu vermitteln, die sich zur Einbettung in die Wissensstruktur der Schüler bewährt haben *(Quod est demonstrandum!)*

Prinzip: Bezug zur Lebenswelt

Andererseits sollten Inhalte und Gegenstände in einem übergreifendem Zusammenhang (Lernbereich) gestellt werden, so dass an vorhandenen "Knoten" der Wissensstruktur angeknüpft werden kann.

Klassische Beispiele aus der Fachwissenschaft Informatik:

- Bus-/Flugreservierungen
- Philosophenproblem
- Leser-Schreiber-Problem
- Verbraucher-Erzeuger-Problem

Groblernziele:

- Entwicklung und Anwendung (rechnergestützter) Methoden zur Beherrschung und effizienten Nutzung nebenläufiger Prozesse
- Entwicklung und Bewertung von Strategien für Probleme, die bei nebenläufigen Vorgängen auftreten können.
- Fähigkeit zur Modellierung nebenläufiger Vorgänge.

Übersicht zu den Beispielen:

1	Multitasking Betriebsysteme	- Java
2	Das Philosophenproblem	- ITG-Pascal / Java
3	Die Brücke	- Unterrichtshilfe
4	Wartung, Aufzug, u. a.	

[4] in [Schwill, A. 1991]

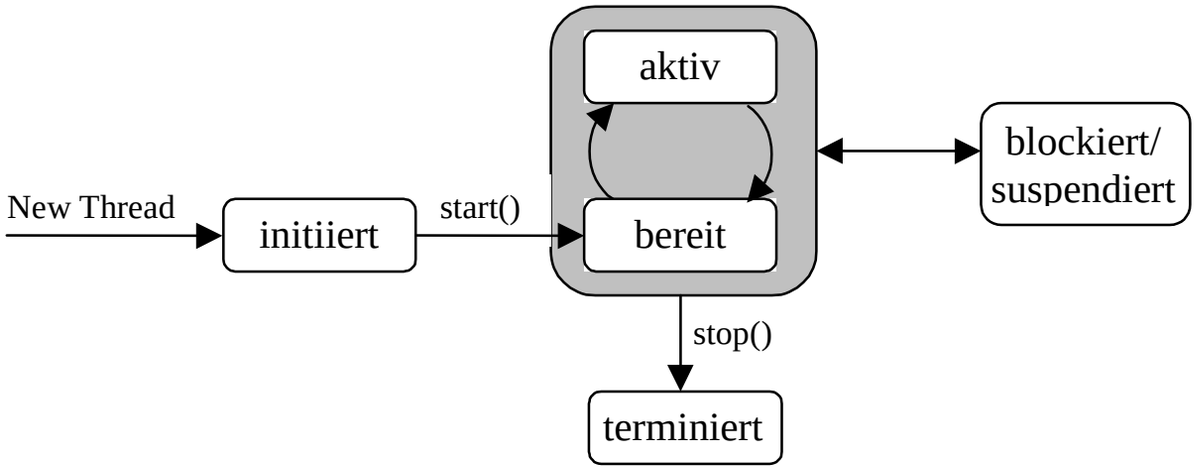
Multitasking Betriebssystem

Kontext:
Nicht alle Betriebssysteme unterstützen nebenläufige Prozesse. Mit Hilfe einer entsprechenden Programmiersprache (hier: Java) lässt sich diese Eigenschaft experimentell ermitteln.

Problem:
Mit den vorgegebenen Programmen soll das Betriebssystem getestet werden, indem die Schleifendurchläufe und die Anzahl der Prozesse variiert werden.

Lösungsansatz:
Klar! Das BS ist geeignet, wenn die Prozesse relativ gleichmäßig ablaufen können.

Lernziel:
Erarbeitung der programmiersprachlichen Grundlagen zur Implementierung nebenläufiger Prozesse.
Lebenszyklus und Zustände eines Prozesses.



Programme:
Bsp0.java; Prozess.java

Anmerkung:
Java unterstützt time-slicing, garantiert es aber nicht.

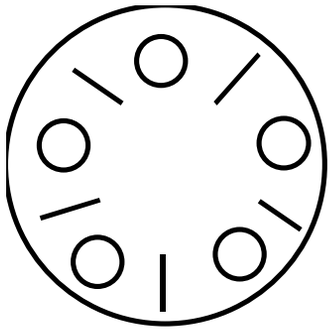
D. h. ein Prozess bekommt keine Zeit zugeteilt, wenn ein Prozess gleicher Priorität läuft und nicht am Weiterlaufen gehindert wird!

```
public class Bsp0{
  public static void main (String[] args){
    Prozess a = new Prozess();     /* Thread-Objekt der Klasse Prozess erzeugen   */
    a.start();                     /* und starten, d.h. vom Zustand INITIERT in den    */
                                   /* Zustand BEREIT/AKTIV versetzen          */

    Prozess b = new Prozess();     /* Thread-Objekt der Klasse Prozess */
    b.setName("Prozess 2");        /* einen anderen Namen vergeben */
    b.start();                     /* und starten */
  }
}
```

```
public class Prozess implements Runnable{
  private Thread myThread = null;
  private String prozessName = "Prozess";

  public void setName(String name){        /* Methode zur Namensaenderung  */
    this.prozessName = name;               /* des Objekts, welche ggf. die */
    if(myThread != null){                  /* entsprechende Methode des    */
      myThread.setName(prozessName);       /* THreads aufruft.          */
    }
  }

  public void start(){                     /* Start - Methode des Objekts */
    if(myThread == null){
      myThread = new Thread(this,prozessName);
      myThread.start();
    }
  }

  public void run() {                      /* die Hauptroutine des Threads */

     System.out.println(myThread.getName() + " startet!");
     for (int i=0; i<10;i++){
       System.out.println(i + myThread.getName() + " is running");
       try {
         Thread.sleep((int)(Math.random() * 1000));
       } catch (InterruptedException e) {}
     }
     System.out.println("DONE!");
    }
                                           /** Ende der Hauptroutine */

  public void stop(){                /* Stop - Methode des Objekts */
    myThread.stop();                 /* versetzt den Prozess in den Zustand TERMINIERT */
  }
}
```

Das Philosophenproblem

Fünf Philosophen sitzen an einem runden Tisch. Vor jedem Philosphen befindet sich ein Teller mit Reis. Zwischen zwei Tellern liegt jeweils ein Stäbchen. Bevor ein Philosoph mit dem Denken aufhört und essen kann, muss er beide Stäbchen neben seinem Teller bekommen.



- Jeder Philosoph "ist" ein individueller *Prozess*.
- Die Stäbchen sind gemeinsame *Betriebsmittel*.
- Der Zugriff auf die Betriebsmittel ist ein *kritischer Abschnitt*.

Problem!

Wenn alle Philosophen jeweils zur gleichen Zeit hungrig werden und sich gleichzeitig ein Stäbchen nehmen, in der Hoffnung, anschließend das zweite Stäbchen zu bekommen, kommt es zu einer *Verklemmung*.

Exkurs:
Vier notwendige Bedingungen für eine Verklemmung (Coffman et al. 1971)

- Prozesse haben exklusiven Zugriff auf Betriebsmittel
- Prozesse haben Betriebsmittel und warten auf andere Betriebsmittel
- Betriebsmittel können nicht entzogen werden, sondern müssen von den Prozessen freigegeben werden.
- Es besteht eine geschlossene Kette von Prozessen, die Betriebsmittel anfordern, die der Nachfolger in der Kette hält.

➔ Der Zugriff der Philosophen muss geschickt synchronisiert werden.

Lösungsansatz mit Semaphoren in ITG-Pascal[5]

(aus [LOGIN 5/93])

Semaphor-Konzept (Dijkstra)	ITG-Pascal:
P(i:semaphor)	WarteAufSignal(i:signal)
V(i:semaphor)	SendeSignal(i:signal)
semaphor	signal

Initialisierung

```
VAR g0, ... , g4: Signal;
    SendeSignal(g0); ...; SendeSignal(g4);
```

Philosoph i

```
REPEAT
    denken;
    hungrig werden;
    WarteAufSignal(gi);
    Greife die linke Gabel;
    WarteAufSignal(g(i+1) mod 5);
    Greife die rechte Gabel;
    essen;
    Leg die rechte Gabel hin;
    SendeSignal(g(i+1) mod 5);
    Leg die linke Gabel hin;
    SendeSignal(gi);
UNTIL FALSE
```

Bewertung der Lösung:

+ zwei benachbarte Philosophen können nie gleichzeitig essen
- werden alle Philosophen gleichzeitig hungrig, tritt eine Verklemmung ein

Lösung: Speichere den Zustand des Philosophen i in einer gemeinsamen Variablen c_i und verhindere einen gleichzeitigen Zugriff auf diese Variablen (wechselseitiger Ausschluss mit Semaphoren).

[5] ITG-Pascal ist eine nebenläufige Prozesse unterstützender Programmiersprache, die als kostenlose Test-Version beim LOGIN-Verlag bezogen werden kann.

Lösungsansatz mit Monitoren in Java

Monitor-Konzept (Hoare, Hansen)

- Ein Monitor ist eine Sammlung von Prozeduren, Variablen und Datenstrukturen, die in einem Modul zusammengefasst sind.
- Prozesse können nicht auf die internen Variablen und Strukturen zugreifen.
- Pro Monitor kann nur ein Prozeß aktiv sein
- Ist ein Monitor besetzt, wird der aufrufende Prozess suspendiert.
- Der Compiler garantiert den wechselseitigen Ausschluss im Monitor

In JAVA werden Objekte zu Monitoren, wenn sie Methoden enthalten, die mit dem Schlüsselwort `synchronized` gekennzeichnet.
Nur eine der gekennzeichneten Methoden kann pro Objekt aufgerufen werden!

Ein Philosoph darf nur in den Zustand "Essen", wenn keiner seiner Nachbarn ißt.

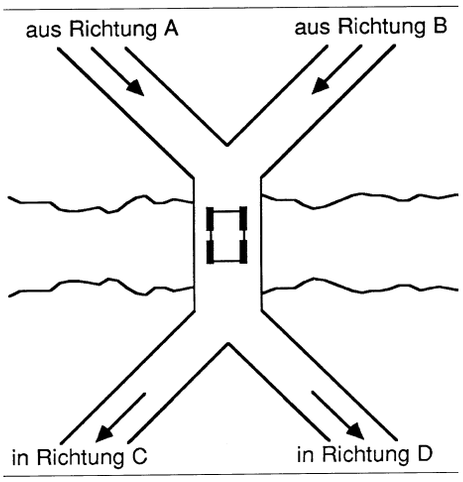
```
class Manager {
    ...
    private zustand[] c = new zustand[4];

    Initialisieren der c[i]:=denken

    synchronized public void Test (i:Nr){
        wenn die benachbarten Philosophen nicht essen,
        dann setze den Aufrufer in den Zustand essen
    }
    synchronized public void Hunger(i:Nr) {
        c[i]:= hungrig;
    }
    synchronized public void Satt(i:Nr) {
        c[i]:= denken;
    }
    ...
}
```

Anm.

Unter http://www.informatik.hu-berlin.de/~rabiyato/Java_seminar.html findet sich eine Implementierung unter JAVA, bei der jedoch hungrige Philosophen nicht mehr denken können.

Die Brücke

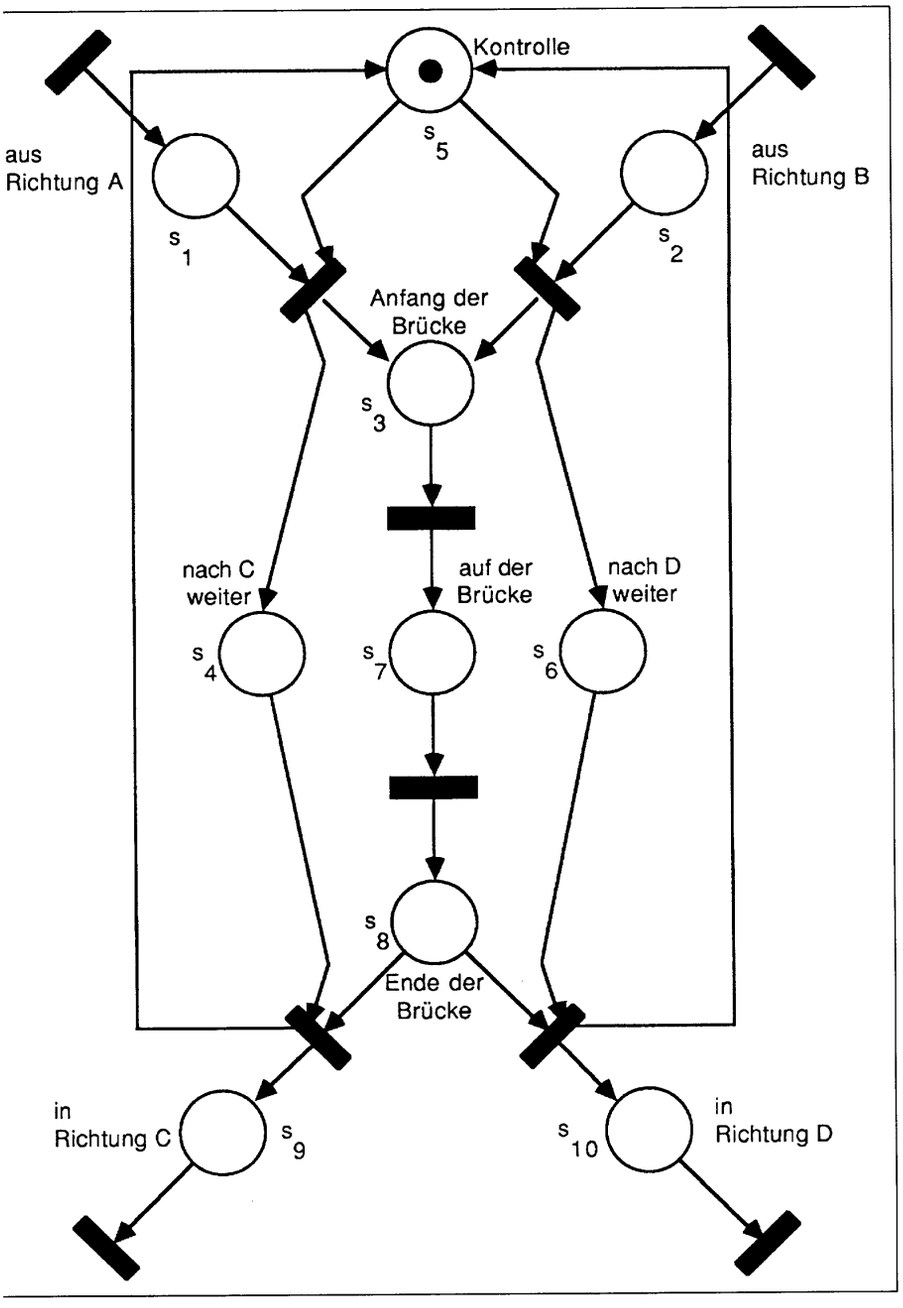
(aus LOGIN 5/93)



Aus Richtung A kommende Fahrzeuge fahren nach C, von B kommende nach D.

Ziel:
Analyse und Erkenntnis-gewinn über das System

Terminierung?
Verklemmungen?
...
=>
Ergänzung des Systems



Simulation mit einer Unterrichtshilfe

- als Marken werden Geldstücke verwendet
- Mit einem Würfel wird die aktive Transition ausgewählt, die im nächsten Schritt schaltet (Nichtdeterminismus)

Anm.

- Anschließend kann das Petri-Netz mit einer sequentiellen Programmiersprache implementiert und simuliert werden.
- Petri-Netze sind nur für Probleme mit fester Grundstruktur geeignet.

Bsp. 0.1 (Nachricht an einen Prozess senden)

Kontext:

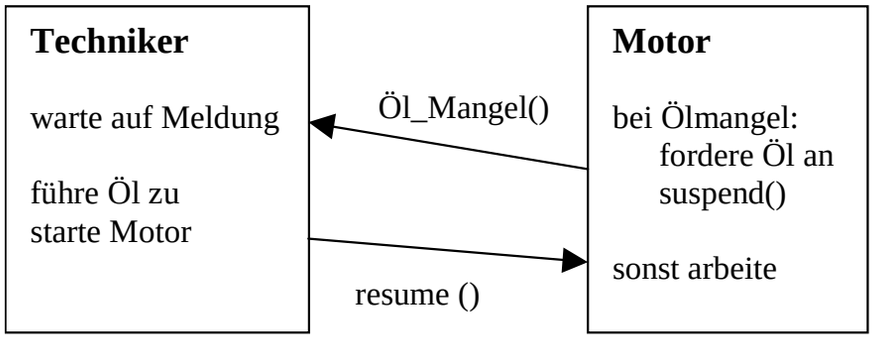
Techniker Heinrich Passauf ist für die Wartung einer motorgetriebenen Anlage zuständig. Wenn einem Motor das Öl ausgeht, stoppt er seine Aktivität und ruft den Techniker, damit dieser Öl nachfüllt (sonst kommt es zu einem Kolbenfresser). Der Techniker schaltet anschließend den Motor wieder ein, so dass die Anlage an der unterbrochenen Tätigkeit fortfahren kann.

Problem:

Der Techniker soll durch ein automatisches Schmieröl-Steuerungsprogramm ersetzt werden. Aus Sicherheitsgründen muss der Motor auch weiterhin vorher kurzzeitig gestoppt werden.

Lösungsansatz:

Der Motor (Prozess 1 oder mehr) suspendiert sich bei Ölmangel von seiner derzeitigen Tätigkeit und teilt dies dem Steuerungsprogramm (Prozess 2) mit. Dieses führt dem entsprechendem Motor Öl zu (muss nicht implementiert werden) und startet diesen erneut.



Lernziel:

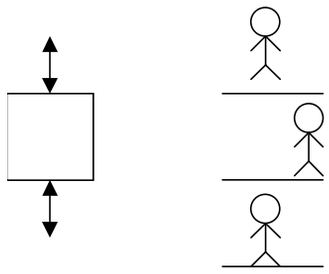
Kommunikation zwischen Prozessen durch Aufruf von Methoden.
Unterscheidung der Zustände suspendiert und terminiert.

Anmerkung:

Das Problem kann zunächst nur mit einem Motor betrachtet werden, um Konflikte zu vermeiden. Bevor Öl nachgefüllt wird, könnte der Techniker noch überprüfen, ob sich der Motor wirklich abgeschaltet hat.
Erweiterung sind möglich und werden sicherlich von den Schülern angesprochen.

Bsp. 1 (Resourcenverteilung - 1 BM, Prozesse gleicher Priorität)

Kontext:
In einem Hochhaus mit x Stockwerken wurde ein Aufzug eingebaut, der mit einem automatischen Steuerungsprogramm ausgestattet werden soll. Von jedem Stockwerk aus kann der Aufzug per Knopfdruck angefordert werden.



Problem:
Nach welcher bzgl. der Wartezeit fairen Strategie fährt der Aufzug die einzelnen Stockwerke an?

Lösungsansatz:
mögliche Strategien: FIFO, SSF, Aufzug
Ein Prozess simuliert den Aufzug mit einer gewählten Strategie, die Anfragen werden von einem weiteren Prozess zufällig erzeugt. Für jede Anfrage wird die Zeit bis zu ihrer Bearbeitung festgehalten, nach jeder Bearbeitung wird die durchschnittliche Bearbeitungszeit festgehalten.
Eventuell kann ein mathematischer Beweis zur Bewertung der Strategien angeschlossen werden.

Lernziel:
Nachrichten-Übermittlung zwischen Prozessen
Strategien zur Resourcenverteilung

Anmerkung:
Der Aufzug-Algorithmus wird auch zur Positionierung von Festplattenarmen verwendet.

Bsp. 2 (Resourcenverteilung - 1 BM,Prozesse gleicher Priorität, bekannte Bearbeitungszeit, gleich große Zeitscheiben)

Kontext:
Eine Fenster-Firma kann aufgrund ihrer Erfahrung einschätzen, wieviel Zeit Kundenaufträge bestimmter Art benötigen (kleine Reparaturen in unmittelbarer Nähe contra Fensterfassadeneinbau in weiter Ferne).

Problem:
Es soll eine Strategie gefunden werden, die die Kunden gleich behandelt und die mittlere Bearbeitungszeit für jeden Tag optimiert.

Lösungsansatz:
mögliche Strategien: SPT, Shortest Remaining Prozessing Time First
Ein Programm (2 Prozesse)

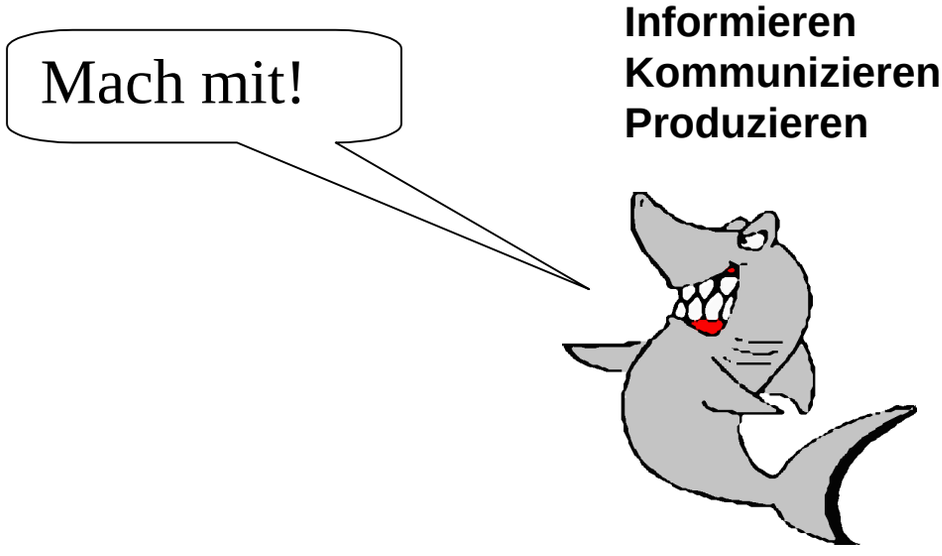
Anm.
Lesen/Schreiben auf Platten
Windows<3.11 (Zeitscheiben, unterschiedliche Prioritäten)

Weitere Themenfelder

- Kettenreaktion
- Erzeuger-Verbraucher-Problem (Regallager)
- Zugriff auf eine Datenbank (Flugbuchung)
- Zuteilung von CPU-Zeit (auch auf Verteilten Systemen)
- Verklemmung beim Linksabbiegen
- Restaurant-Betrieb
- Regallager-Paketpacken
- Flaschenabfüllanlage [Duden]

- Resourcenverteilung
- Wechselseitiger Ausschluss
- Verklemmungs-Vermeidungsstrategien
- Parallele Programmierung(Schachcomputer, [LOGIN 5/93])

Aufruf



http://www.didaktik.cs.uni-potsdam.de/HyFISCH

Literatur

[Duden] Schülerduden Informatik. Dudenverlag, Mannheim 1997

[Engbring, D. 1996] "Allgemeinbildende Informatik - Annäherung an ein Paradoxon". FIFF-Kommunikation 1996

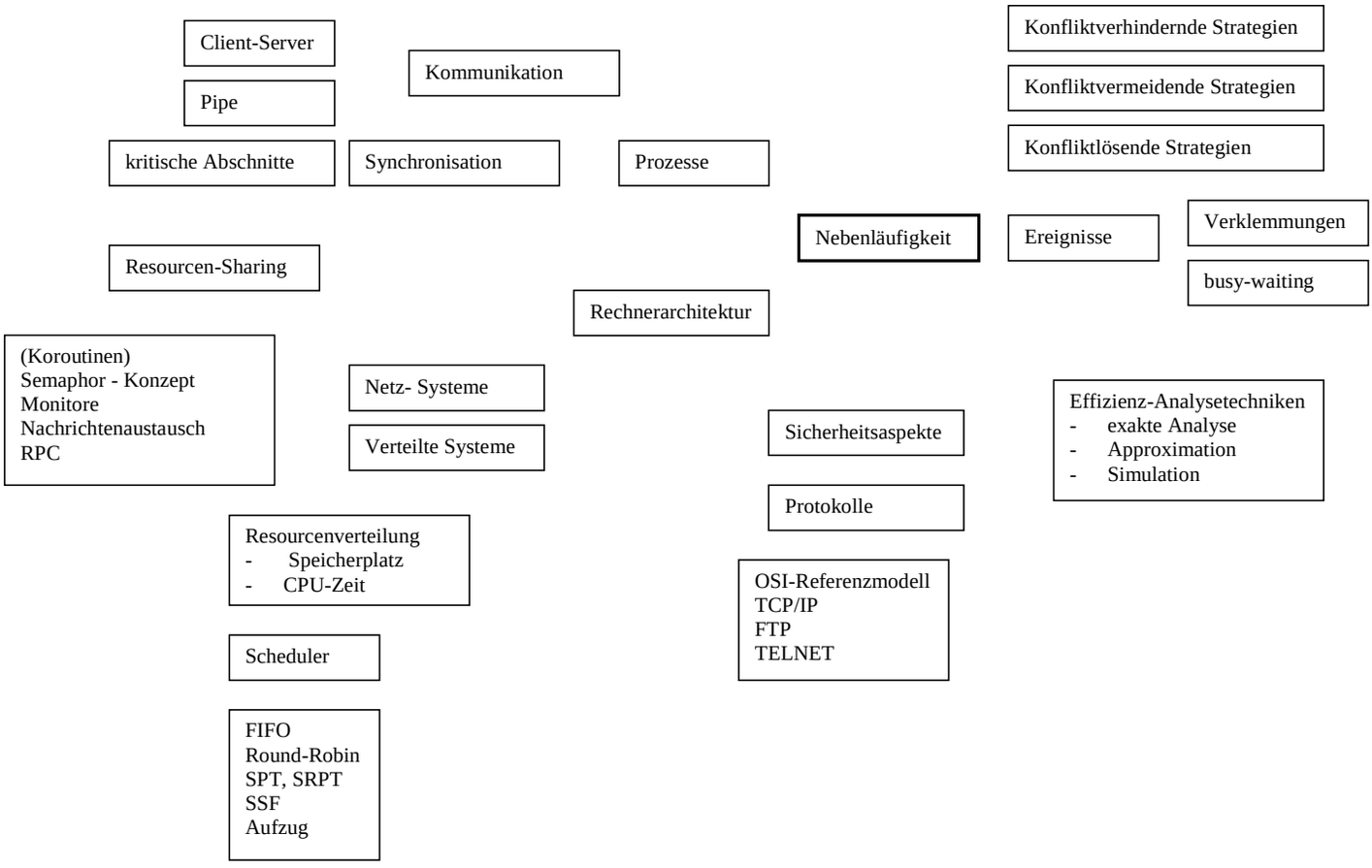
[LOGIN 5/93] Themenheft "Parallelverarbeitung" LOG IN 5 (1993)

[Schneider, H.-J- 1991] Lexikon der Informatik und Datenverarbeitung. Oldenbourg Verlag, München 1991

[Schwill, A. 1991] Didaktik der Informatik - Skriptum zur gleichnamigen Vorlesung im Hauptstudium. Universität/GH Paderborn 1991

[Schwill, A. 1994] "Fundamentale Ideen in Mathematik und Informatik". in "Fundamentale Ideen" - Bericht über die 12. Tagung des Arbeitskreises Mathematik und Informatik. Franzbecker Verlag, Hildesheim 1994

Einige interessante Web-Adressen:

"Die speisenden Phiosophen"
http://www.informatik.hu-berlin.de/~rabiyato/Java_seminar.html
"The Java-Tutorial"
http://java.sun.com/docs/books/tutorial/index.html

Anhang: Begriffsnetz zur Nebenläufigkeit