

# Vorlesung 8: Concurrency

Peter B. Ladkin

[ladkin@rvs.uni-bielefeld.de](mailto:ladkin@rvs.uni-bielefeld.de)

Wintersemester 2001/2002

# Prozess Synchronisierung Puzzle I

- Process 1: (x: integer)  
begin x <- 0; x <- x+1; stop; end
- Process 2: (x: integer)  
begin read x; stop; end
- Was ist der gelesene Wert von x, wenn diese Programme concurrent laufen?

# Puzzle 1

- Sequenzierung (auf einem Multitasking-CPU)

`x <- 0; x <- 0+1; read x`

`x <- 0; read x; <- 0+1`

- Alles?

# Puzzle 1

- Nein, natürlich nicht!

```
read x; x <-0; x <- 0+1
```

- Allerdings: 0, 1, und allesmögliche

# Prozess Synchronisierung Puzzle 2

- Prozess 1: (x: integer)  
begin x <- 0; x <- x+1; stop; end
- Prozess 2: (x,y: integer)  
begin y <- 0; y <- x+1; stop; end
- Voraussetzung: Memory Platz x ist dergleiche als Memory Platz y
- Werte von x, y an Ende?

## Puzzle 2

- $x$  und  $y$  gleich; sagen wir Platz  $A$
- $x \leftarrow 0; x \leftarrow 0+1; y \leftarrow 0; y \leftarrow 0+1$   
 $A = 1$
- $X \leftarrow 0; y \leftarrow 0; x \leftarrow 0+1; y \leftarrow 1+1$   
 $A = 2$

## Puzzle 2

- $x \leftarrow 0; y \leftarrow 0; y \leftarrow 0+1; x \leftarrow 1+1$   
 $A = 2$
- usw
- $A = 1$  oder  $A = 2$

# Prozess Synchronisierung Puzzle 3

- Wert von der Variabel  $z$  ist 1, falls es existieren 20 Blöcke freiverfügbarem Speicher;
- ...ist 2, falls es  $\dots < 20$  Blöcke....
- Wert von  $z$  ist 1
- Prozess 1 braucht 15 Blöcke, Prozess 2 auch
- Beide lesen  $z$  gleichzeitig
- Was passiert?



# Prozess Synchronisierung Puzzle 4

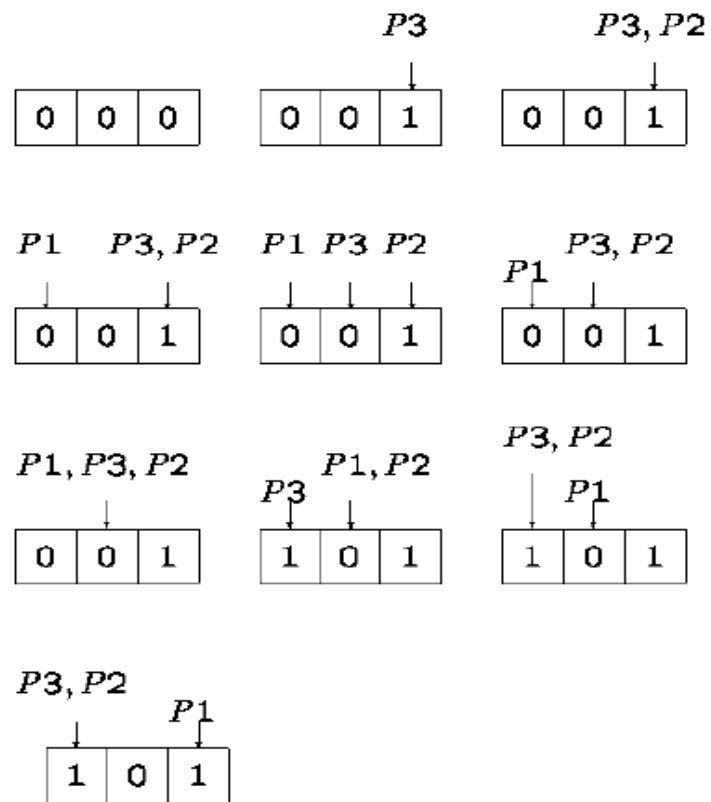
- Programm 1 und Programm 2 lesen Variabel turn
- turn könnte von Programm 3 geschrieben werden
- turn hat 3 Bits
- turn = 001 bedeutet, Prog 1 kann den Drucker benutzen
- Turn = 101 bedeutet, Prog 2 .....

## Puzzle 4

- Prog 3 schreibt Wert 101, von rechts nach links
- Prog 2 liest von rechts nach links
- Prog 1 liest von links nach rechts
- In der folgenden Ausführung.....

# Puzzle 4

- P3 schreibt B3  
P2 liest B3  
P1 liest B1  
P3 schreibt B2  
P2 liest B2  
P3 schreibt B1  
P2 liest B1  
P1 liest B3



## Puzzle 4

- P1 liest 001
- P2 liest 101
- Beide gehen daran
- Kunstliche Ausgabe, vielleicht, aber nicht gerade die erwartete
- Stell mal vor, falls dies mit einem Digital Flight Control System passieren würde!

## Puzzle 4: Lösung

- Sicherstellen, dass nur ein Prozess Anschluss an die Variabel zu einer Zeit hat
- Dies heisst: **mutex (mutual exclusion)**
- **Problemlösung stammt von Edsger Dijkstra**
- **Turing-Preissieger, Designer des THE Betriebssystems (Eindhoven, 1968)**

# Mutex

- Dijkstra ist Empfänger des Semaphores
- **Wie ein Spielmarke, die nur ein Prozess zu einer Zeit hat**
- **Beispiel: Lösung zu Puzzle 4. Zu jeder Zeit: P1 hat Anschluss, P2 und P3 nicht; oder P2 hat Anschluss, P1 und P3 nicht; oder P3 hat Anschluss, P1 und P2 nicht**

# Semaphore

- Vorteil: mutex sichergestellt
- Nachteil: ineffizient
- Allerdings muss man nur sicherstellen, dass nicht gelesen wird wenn geschrieben wird
- P1 und P2 könnten ohne Gefahr gleichzeitig lesen; nur wenn und dass P3 nicht schreibt



# Semaphore: Übung

- Wie kann die effizientere Lösung mit Hilfe von Semaphoren implementiert werden?
- Wie könnten Semaphoren mit Hilfe von Interrupt-Masking programmiert werden?

# Semaphoren

- Alle Prozessen können gleichzeitig versuchen, den Semaphor zu holen
- Nur ein Prozess könnte den Semaphore "bekommen"
- Die anderen müssen *warten* (z.B. auf eine Warteschlange) bis der Prozess fertig ist
- *Was passiert, wenn der Prozess scheitert?*

# Semaphoren

- Technische ausgesehen ist ein Semaphor ein *Shared Variable* , deren Anschluss kontrolliert ist
- *Zwei Operationen nur: holen und freigeben*
- Ein *Interlock* , der verhindert, dass ein Prozess *in seinen Critical Section* hineingeht, wenn der Semaphor schon gesetzt wird

# Semaphoren

- Wie ein einziges Bit
- Gesetzt:  $P$  ("passeren")
- *Freigegeben:  $V$  ("vrijgeven")*
- *Nur ein Prozess kann zu einem Zeitpunkt eine Operation ausführen*
- *Andere sind blockiert bis  $V(S)$  ausgeführt wird*
- *d.h.,  $P$  und  $V$  sind atomäre Operationen (atomic Operationen)*

# Semaphoren

- Semaphoren und andere atomäre Operationen werden normalerweise im Betriebssystem implementiert

# Einfache Mutex

- Mask(Interrupts);  
Critical Section;  
Unmask(Interrupts)
- Ineffizient
- z.B., P1 und P2 möchten Drucker1 benutzen  
P3 und P4 möchten Drucker2 benutzen
- P1 muss nur P2 ausschliessen, P2 nur P1,  
P3 nur P4, und P4 nur P3
- P1 könnte gegen P3 ausgetauscht werden, usw

# Programmierung mit Semaphoren

- var integer x,y = 0;  
semaphore sem = 1;

```
cobegin
  loop P(sem); CS1; V(sem) endloop
[]
  loop P(sem); CS2; V(sem) endloop
coend
```

# Notation

- cobegin .... [] ..... coend
- Die zwei Hälfte laufen gleichzeitig
- Es könnte mehrere Klausel geben
- cobegin ... [] ... [] ... coend
- Notation von Dijkstra
- Prozedurale Sprache - man könnte die *Zustände* des Programms nicht beschreiben



# Implementierung von Semaphoren

- Später