

# Treiberentwicklung, Echtzeit- und Betriebssysteme

Prof. Dr. Peter Gerwinski

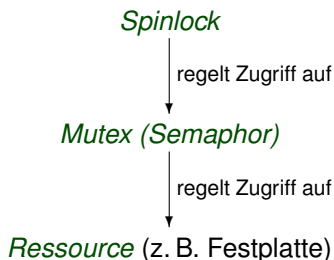
12. Mai 2014

# 3 Echtzeit

## 3.3 Ressourcen

### Ressourcen reservieren

- *Semaphor*  
gemeinsame Variable mehrerer Prozesse  
zur Regelung des Zugriffs auf eine Ressource  
Ressource belegt → Kontextwechsel
- *Mutex*  
Mechanismus, damit immer nur ein Prozeß gleichzeitig  
auf eine Ressource zugreifen kann  
spezieller binärer Semaphor: nur „Besitzer“ darf freigeben
- *Spinlock (busy waiting)*  
leichtgewichtige Alternative zu Kontextwechsel
- *Kritischer Abschnitt – critical section*  
Programmabschnitt zwischen Reservierung  
und Freigabe einer Ressource  
→ sollte immer so kurz wie möglich sein



# 3 Echtzeit

## 3.3 Ressourcen

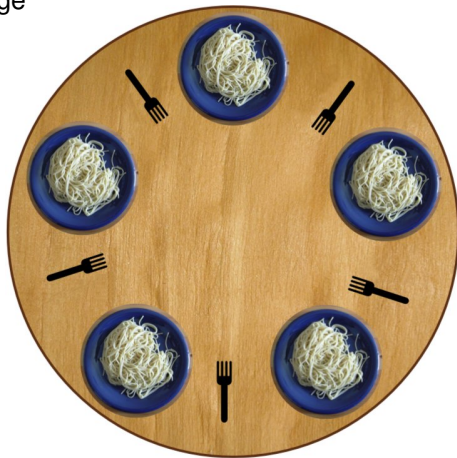
**Verklemmungen:** Gegenseitiges Blockieren von Ressourcen

- **Deadlock:** Prozeß wartet
- **Livelock:** Prozeß macht andere Dinge (z. B. *busy waiting*)

Beispiel: Philosophenproblem

- 5 Philosophen, 5 Gabeln
- 2 Gabeln zum Essen notwendig
- Wer essen will, nimmt eine Gabel und wartet notfalls auf die zweite.
- Keiner legt eine einzelne Gabel wieder zurück.

Jeder hält 1 Gabel → **Verklemmung**  
schweigen → **Deadlock**  
philosophieren weiter → **Livelock**



# 3 Echtzeit

## 3.3 Ressourcen

*Verklemmungen*: Gegenseitiges Blockieren von Ressourcen

- *Deadlock*: Prozeß wartet
- *Livelock*: Prozeß macht andere Dinge  
(z. B. *busy waiting*)

Beispiel: Philosophenproblem

Bedingungen für Verklemmungen:

- |                        |   |
|------------------------|---|
| • Exklusivität         | → Spooling  |
| • <i>hold and wait</i> | → simultane Zuteilung                             |
| • Entzug nicht möglich | → Prozesse suspendieren, beenden, <i>Rollback</i> |
| • zirkuläre Blockade   | → Reihenfolge abhängig von Ressourcen             |

# 3 Echtzeit

## 3.4 Prioritäten

Linux 0.01

- Timer-Interrupt: Zähler des aktuellen Prozesses wird dekrementiert; Prozeß mit höchstem Zähler bekommt Rechenzeit.
- Wenn es keinen laufbereiten Prozeß mit positivem Zähler gibt, bekommen alle Prozesse gemäß ihrer *Priorität* neue Zähler zugewiesen.
- *keine* harte Echtzeit

→ *dynamische Prioritätenvergabe*:  
Rechenzeit hängt vom Verhalten des Prozesses ab

Echtzeitbetriebssysteme

- Prozesse können einen festen Anteil an Rechenzeit bekommen.
- Bei Ereignissen können Prozesse hoher Priorität Prozesse niedriger Priorität unterbrechen, aber nicht umgekehrt.

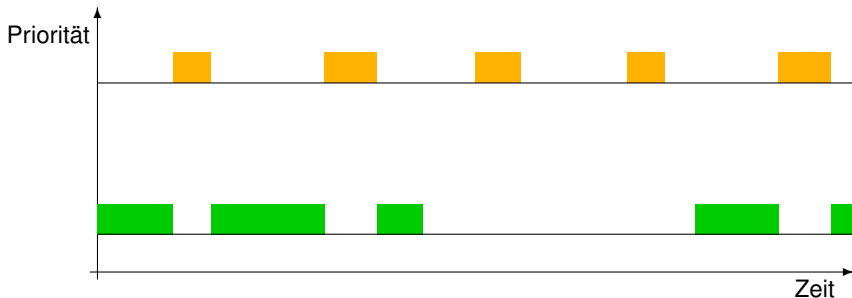
→ *statische Prioritätenvergabe*

# 3 Echtzeit

## 3.4 Prioritäten

### Echtzeitbetriebssysteme

- Prozesse können einen festen Anteil an Rechenzeit bekommen.
- Bei Ereignissen können Prozesse hoher Priorität Prozesse niedriger Priorität unterbrechen, aber nicht umgekehrt.

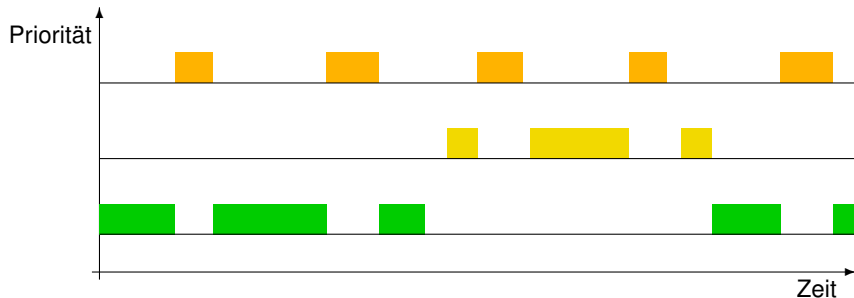


# 3 Echtzeit

## 3.4 Prioritäten

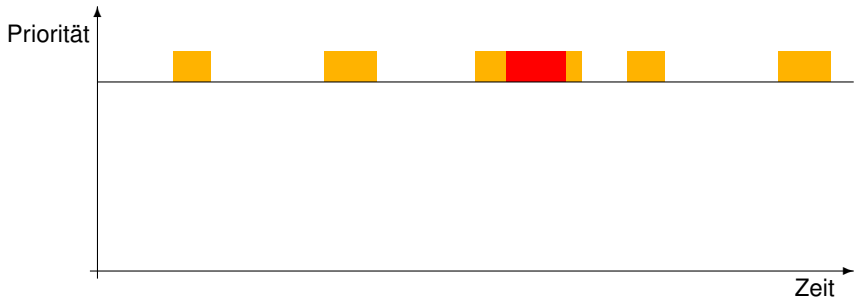
### Echtzeitbetriebssysteme

- Prozesse können einen festen Anteil an Rechenzeit bekommen.
- Bei Ereignissen können Prozesse hoher Priorität Prozesse niedriger Priorität unterbrechen, aber nicht umgekehrt.



# 3 Echtzeit

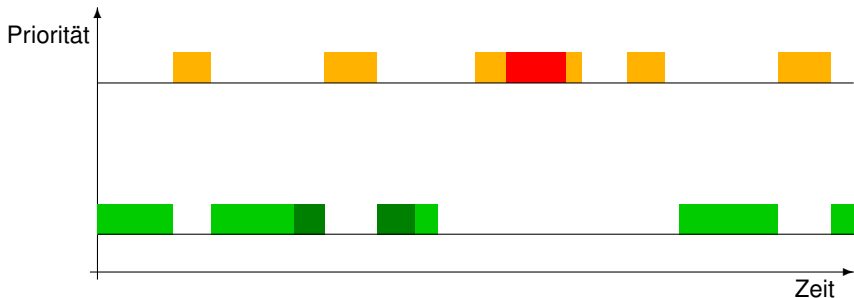
## 3.4 Prioritäten und Ressourcen





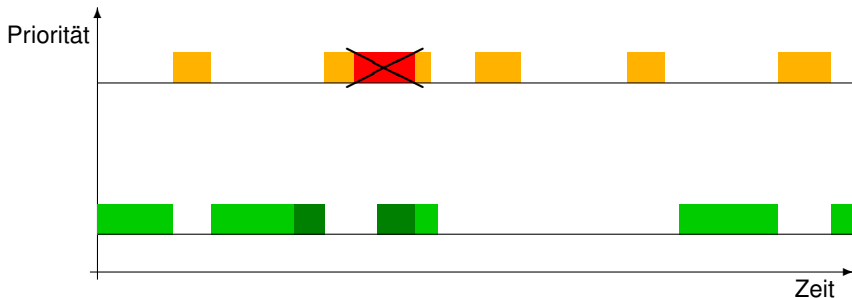
# 3 Echtzeit

## 3.4 Prioritäten und Ressourcen



# 3 Echtzeit

## 3.4 Prioritäten und Ressourcen



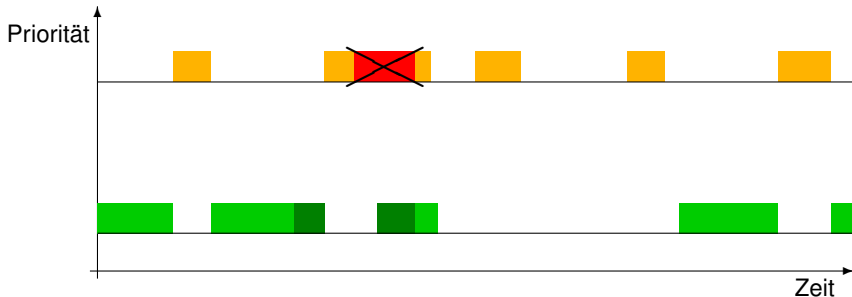
## 3 Echtzeit

### 3.4 Prioritäten und Ressourcen

Der höher priorisierte Prozeß bewirkt selbst, daß er eine Ressource verspätet bekommt.

→ *begrenzte Prioritätsinversion*

maximale Verzögerung: Länge des kritischen Bereichs



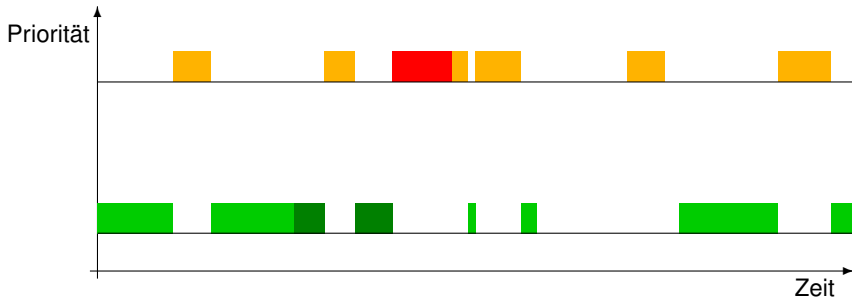
# 3 Echtzeit

## 3.4 Prioritäten und Ressourcen

Der höher priorisierte Prozeß bewirkt selbst, daß er eine Ressource verspätet bekommt.

→ *begrenzte Prioritätsinversion*

maximale Verzögerung: Länge des kritischen Bereichs



## 3 Echtzeit

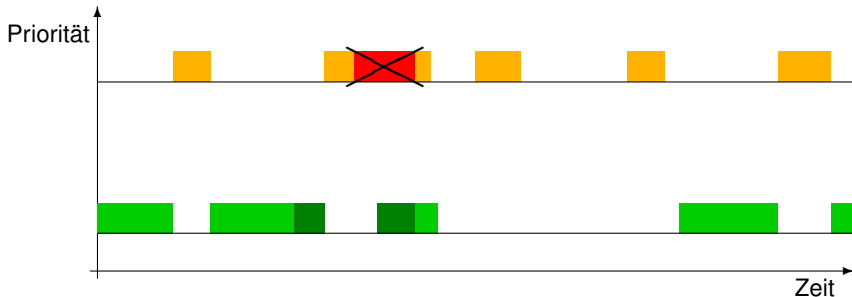
### 3.4 Prioritäten und Ressourcen

*unbegrenzte Prioritätsinversion*

## 3 Echtzeit

### 3.4 Prioritäten und Ressourcen

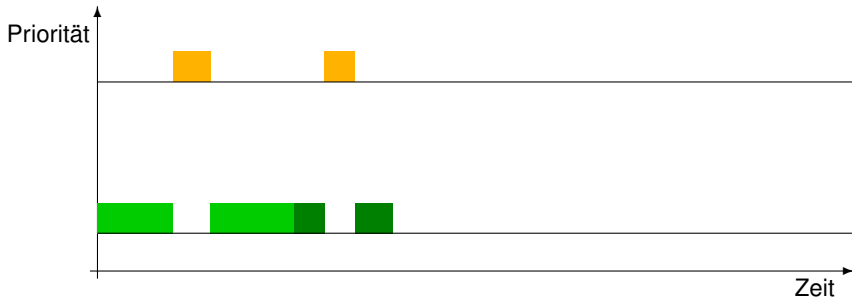
*unbegrenzte Prioritätsinversion*



## 3 Echtzeit

### 3.4 Prioritäten und Ressourcen

*unbegrenzte Prioritätsinversion*

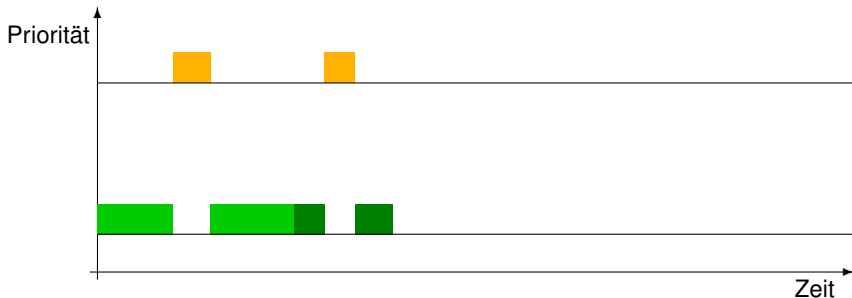


## 3 Echtzeit

### 3.4 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*



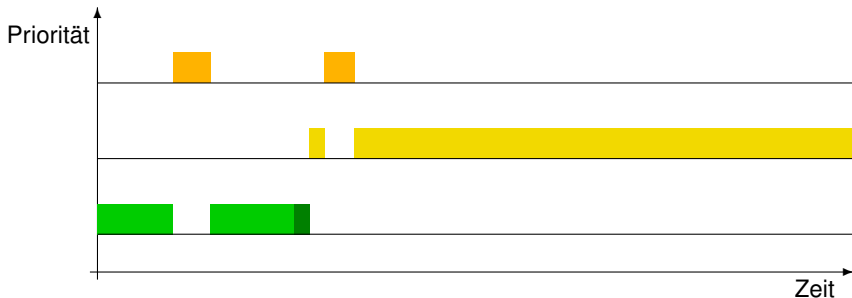


## 3 Echtzeit

### 3.4 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*



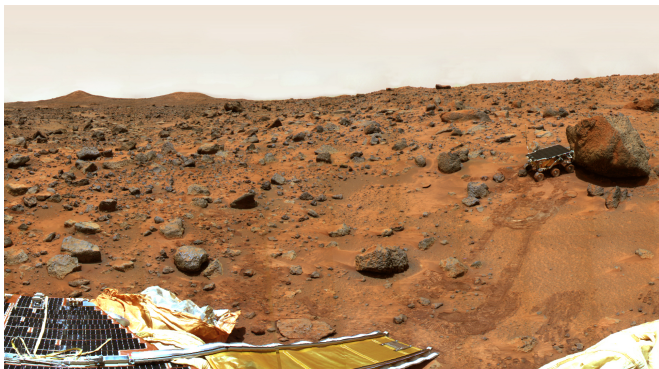
# 3 Echtzeit

## 3.4 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997



## 3 Echtzeit

### 3.4 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997

Gegenmaßnahmen

- *Priority Inheritance – Prioritätsvererbung*  
Der Besitzer des Mutex erbt die Priorität des Prozesses, der auf den Mutex wartet.
- *Priority Ceiling – Prioritätsobergrenze*  
Der Besitzer des Mutex bekommt sofort die Priorität des höchstmöglichen Prozesses, der evtl. den Mutex benötigen könnte.
- *Priority Aging*  
Die Priorität wächst mit der Wartezeit.

## 3 Echtzeit

### 3.4 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997

Gegenmaßnahmen

- *Priority Inheritance – Prioritätsvererbung*

Der Besitzer des Mutex erbt die Priorität des Prozesses, der auf den Mutex wartet.

- *Priority Ceiling – Prioritätsobergrenze*

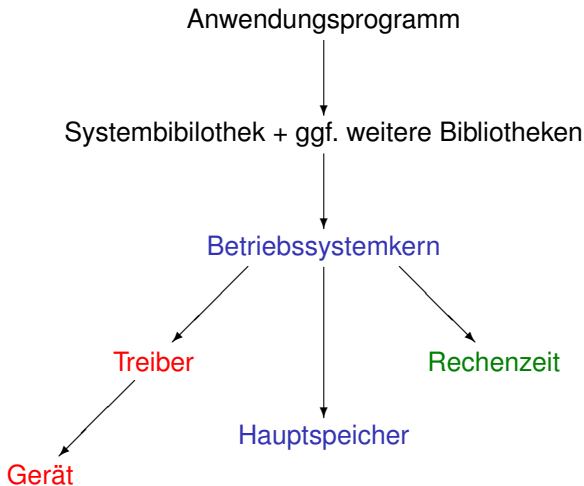
Der Besitzer des Mutex bekommt sofort die Priorität des höchstmöglichen Prozesses, der evtl. den Mutex benötigen könnte.

- *Priority Aging*

Die Priorität wächst mit der Wartezeit.

} nur möglich, wenn  
Mutexe im Spiel sind

# Treiberentwicklung, Echtzeit- und Betriebssysteme



## 4 Speicherverwaltung

### Aufgaben

- Zuteilung
- Speicherschutz
- virtueller Speicher

### Realisierung

- Hardware: Bank Switching
- gar nicht (Mikro-Controller, frühe Betriebssysteme)
- auf Prozeßebene: Segmentierung
- virtuell: Seitenverwaltung (Paging)

# 4 Speicherverwaltung

## 4.1 Bank Switching

Apple II (1977),  
Commodore 64 (1982), ...

- Physikalisch adressierbar: 64 kB
- Bedarf nach mehr Speicher
- Lösung: Über Output-Port Teile des Speichers umschalten
- Das Programm muß sich derweil vollständig in einem anderen Teil des Speichers befinden.



# 4 Speicherverwaltung

## 4.2 Speichersegmentierung

IBM PC (1981),  
IBM PC/AT (1984), ...

- Physikalisch adressierbar:  
1 MB bis 16 MB
- Problem: 16-Bit-Register  
können nur 64 kB adressieren
- Lösung: Segment- und Offset-Adressen
- Zusätzlich: Länge der Segmente einstellbar (Deskriptortabelle)  
→ Speicherschutz
- Konzept übernommen für 32-Bit-Prozessoren  
Physikalisch adressierbar: 4 GB  
→ Segmente dienen nur noch dem Speicherschutz





## 4 Speicherverwaltung

### 4.3 Virtuelle Speicherverwaltung

- zusätzlich zur Segmentierung (nachgeschaltet)
- Aufteilung des Speichers in gleich große Kacheln – *Pages*
- Speicher muß nicht wirklich vorhanden sein

—→ bei Zugriff: *Page Fault* („Interrupt“)

—→ Betriebssystem kann Speicher auf Platte auslagern

—→ weitere Anwendung: Dateizugriff über „Speicherbereich“