

# Echtzeitsysteme

Übungen zur Vorlesung

Betriebsmittelprotokolle

**Simon Schuster**   **Peter Wägemann**

Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)  
Lehrstuhl für Informatik 4 (Verteilte Systeme und Betriebssysteme)  
<https://www4.cs.fau.de>

Sommersemester 2022





## Evaluation der Veranstaltung

- Eure Meinung (Lob/Kritik) ist uns wichtig!
- Eure Rückmeldung hat Konsequenzen
- Bitte evaluiert Vorlesung und Übungen





## Evaluation der Veranstaltung

- Eure Meinung (**Lob/Kritik**) ist uns wichtig!
- Eure Rückmeldung hat Konsequenzen
- Bitte evaluiert **Vorlesung** und **Übungen**



Typische Rückläuferquote  $\mapsto$  **2 – 10%**

- Zu wenig für eine sinnvolle Einschätzung
- Aber: typische Rückläuferquote in EZS  $\mapsto$  **60 – 80%**





## Evaluation der Veranstaltung

- Eure Meinung (**Lob/Kritik**) ist uns wichtig!
- Eure Rückmeldung hat Konsequenzen
- Bitte evaluiert **Vorlesung** und **Übungen**



Typische Rückläuferquote  $\mapsto$  **2 – 10%**

- Zu wenig für eine sinnvolle Einschätzung
- Aber: typische Rückläuferquote in EZS  $\mapsto$  **60 – 80%**



- 1 Organisatorisches
- 2 Übernahmeprüfung**
- 3 Zugriffskontrolle
- 4 Zugriffskontrolle in eCos
- 5 Hinweise zu Aufgabe 7

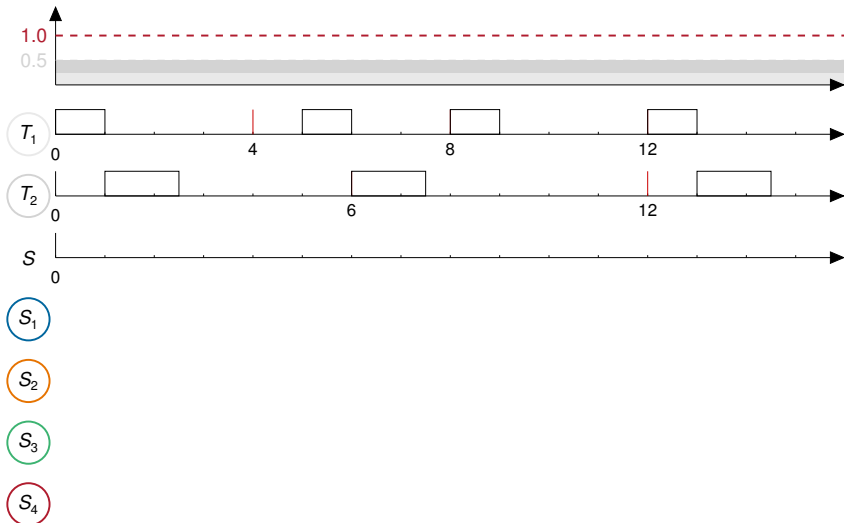


## Wiederholung: Übernahmeprüfung bei terminbasierter Einplanung



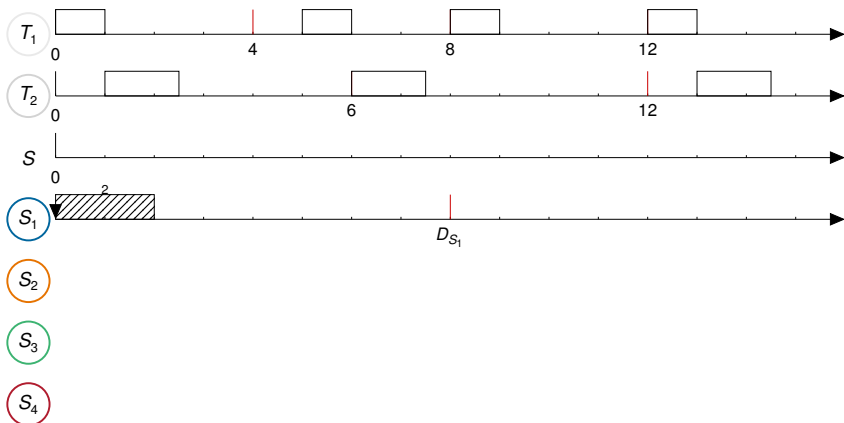
## Beispiel: Dichte-basierter Akzeptanztest [1, S. 252]

$T_1 = (4, 1)$ ,  $T_2 = (6, 1.5) \rightsquigarrow \Delta = 0.5$ , EDF-Ablaufplanung



# Beispiel: Dichte-basierter Akzeptanztest [1, S. 252]

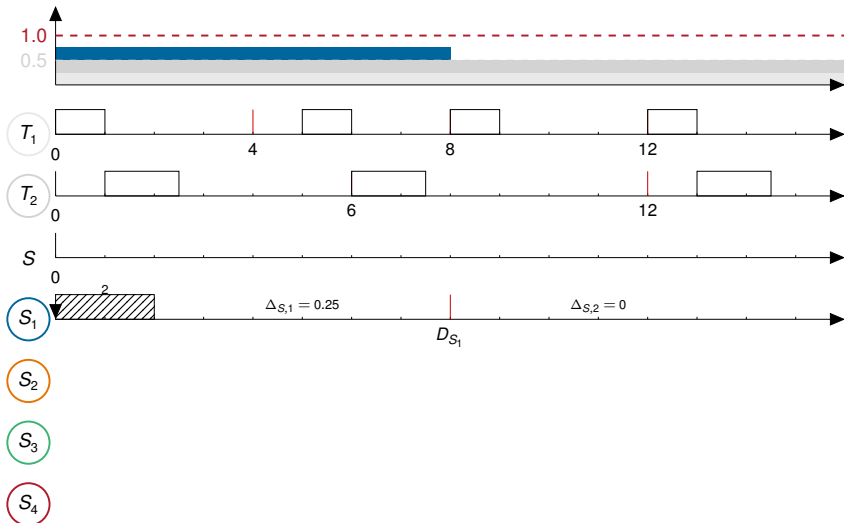
$T_1 = (4, 1)$ ,  $T_2 = (6, 1.5) \rightsquigarrow \Delta = 0.5$ , EDF-Ablaufplanung





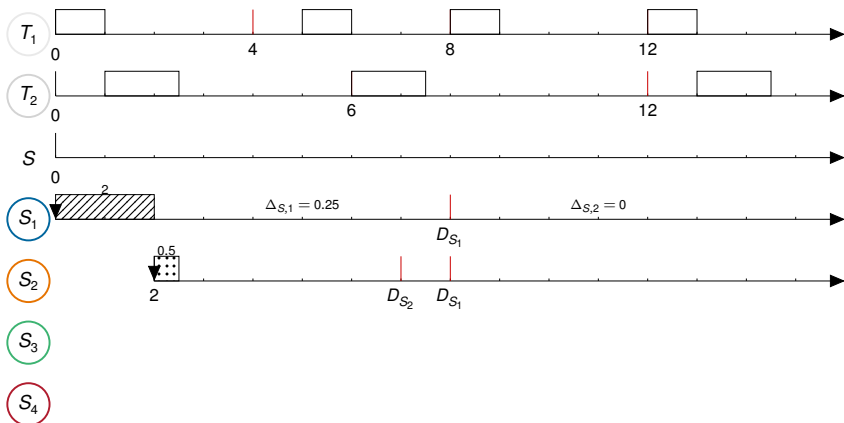
# Beispiel: Dichte-basierter Akzeptanztest [1, S. 252]

$T_1 = (4, 1)$ ,  $T_2 = (6, 1.5) \rightsquigarrow \Delta = 0.5$ , EDF-Ablaufplanung



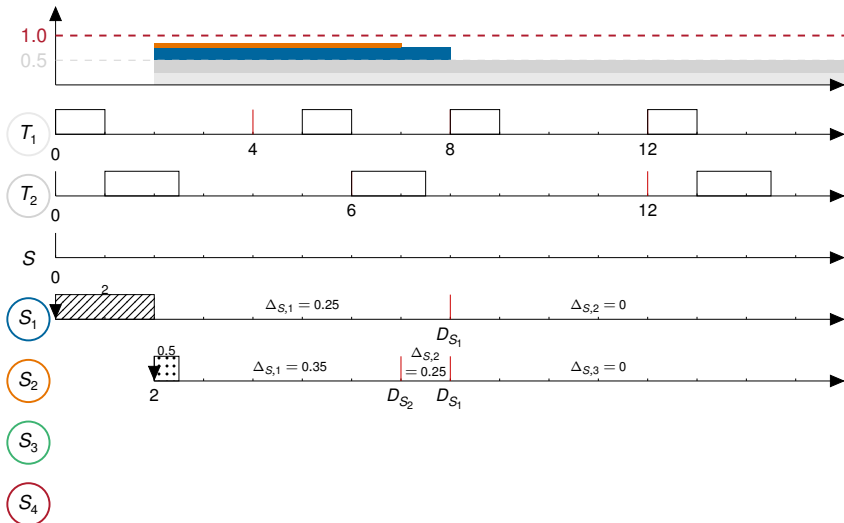
# Beispiel: Dichte-basierter Akzeptanztest [1, S. 252]

$T_1 = (4, 1)$ ,  $T_2 = (6, 1.5) \rightsquigarrow \Delta = 0.5$ , EDF-Ablaufplanung



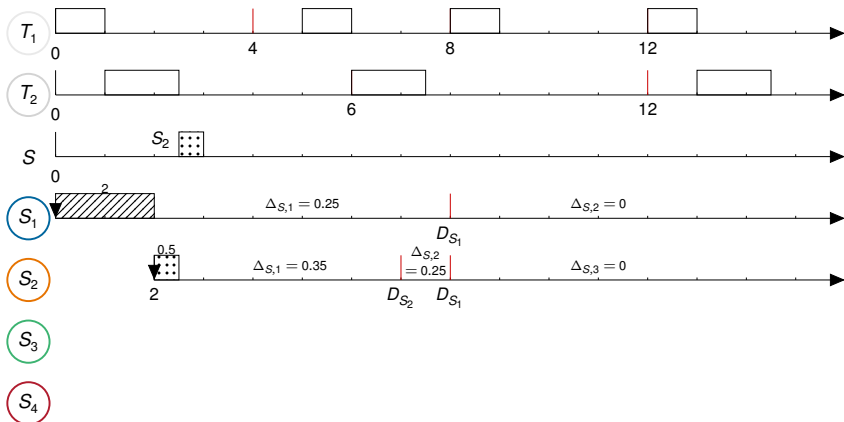
# Beispiel: Dichte-basierter Akzeptanztest [1, S. 252]

$T_1 = (4, 1)$ ,  $T_2 = (6, 1.5) \rightsquigarrow \Delta = 0.5$ , EDF-Ablaufplanung



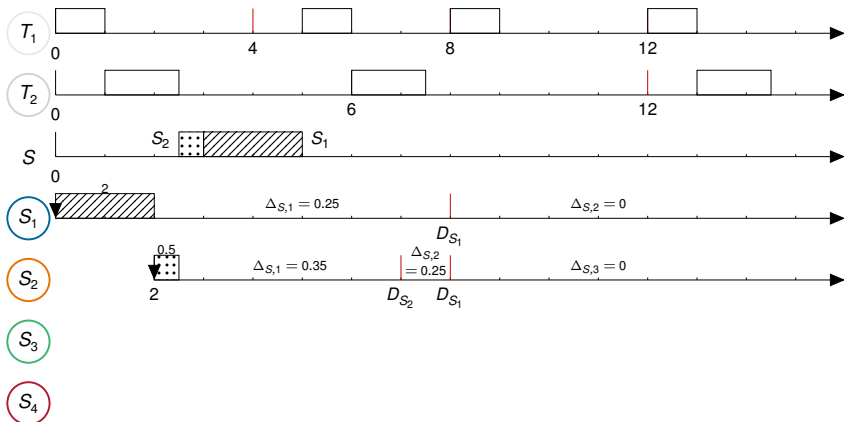
# Beispiel: Dichte-basierter Akzeptanztest [1, S. 252]

$T_1 = (4, 1)$ ,  $T_2 = (6, 1.5) \rightsquigarrow \Delta = 0.5$ , EDF-Ablaufplanung



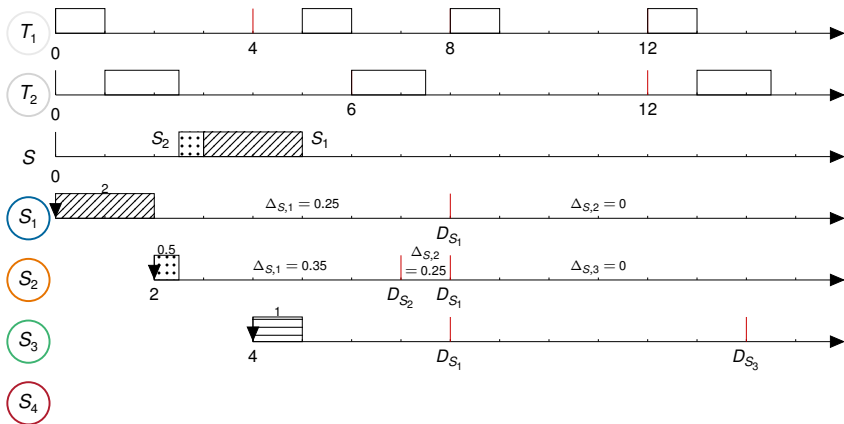
# Beispiel: Dichte-basierter Akzeptanztest [1, S. 252]

$T_1 = (4, 1)$ ,  $T_2 = (6, 1.5) \rightsquigarrow \Delta = 0.5$ , EDF-Ablaufplanung



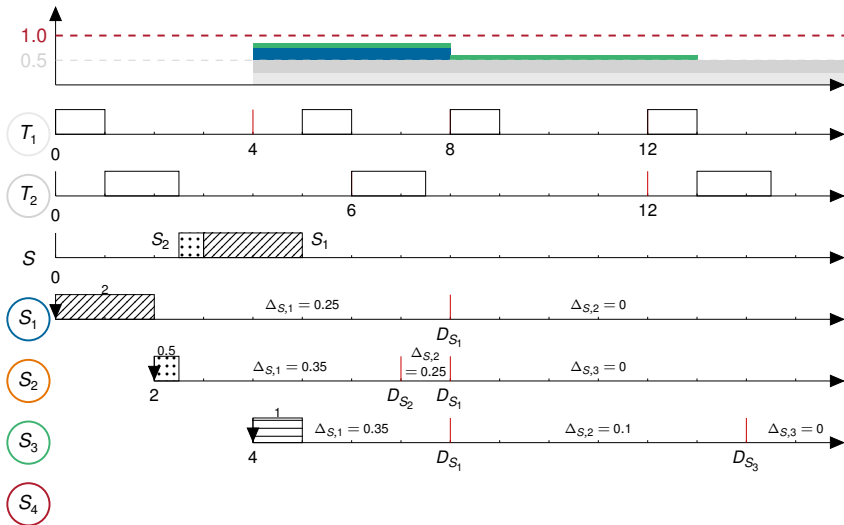
# Beispiel: Dichte-basierter Akzeptanztest [1, S. 252]

$T_1 = (4, 1)$ ,  $T_2 = (6, 1.5) \rightsquigarrow \Delta = 0.5$ , EDF-Ablaufplanung



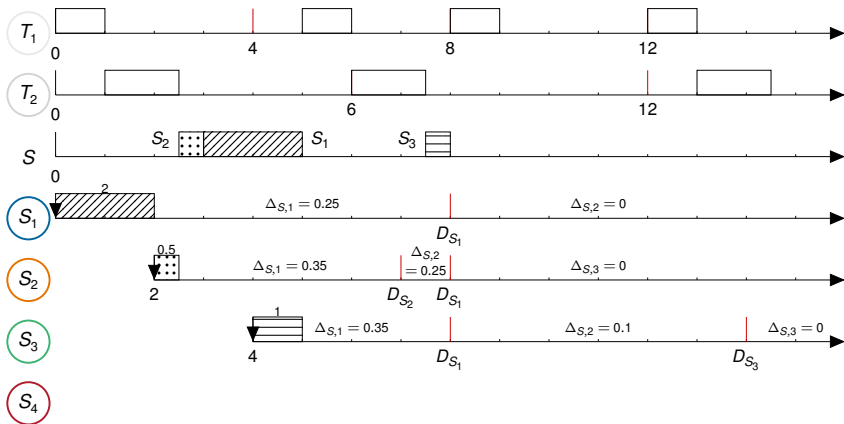
# Beispiel: Dichte-basierter Akzeptanztest [1, S. 252]

$T_1 = (4, 1)$ ,  $T_2 = (6, 1.5) \rightsquigarrow \Delta = 0.5$ , EDF-Ablaufplanung



# Beispiel: Dichte-basierter Akzeptanztest [1, S. 252]

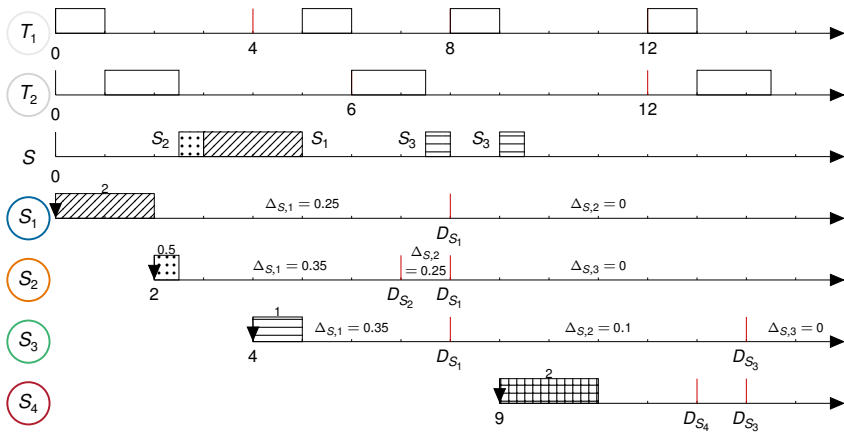
$T_1 = (4, 1)$ ,  $T_2 = (6, 1.5) \rightsquigarrow \Delta = 0.5$ , EDF-Ablaufplanung





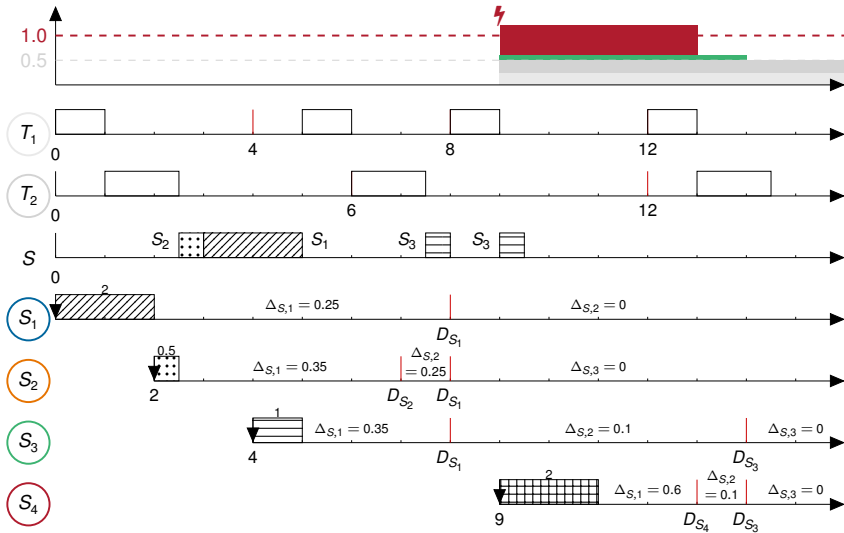
# Beispiel: Dichte-basierter Akzeptanztest [1, S. 252]

$T_1 = (4, 1)$ ,  $T_2 = (6, 1.5) \rightsquigarrow \Delta = 0.5$ , EDF-Ablaufplanung



# Beispiel: Dichte-basierter Akzeptanztest [1, S. 252]

$T_1 = (4, 1), T_2 = (6, 1.5) \rightsquigarrow \Delta = 0.5$ , EDF-Ablaufplanung



- 1 Organisatorisches
- 2 Übernahmeprüfung
- 3 Zugriffskontrolle**
- 4 Zugriffskontrolle in eCos
- 5 Hinweise zu Aufgabe 7



## Konkurrenz und Koordination

- Betriebsmittelarten  $\rightsquigarrow$  einseitige/mehrseitige Synchronisation
- Konkurrenz  $\rightsquigarrow$  Vergabe/Freigabe (P/V)
- Konflikt  $\rightsquigarrow$  Streit um begrenzte bzw. unteilbare Betriebsmittel (BM)



## Konkurrenz und Koordination

- Betriebsmittelarten  $\rightsquigarrow$  einseitige/mehrseitige Synchronisation
- Konkurrenz  $\rightsquigarrow$  Vergabe/Freigabe (P/V)
- Konflikt  $\rightsquigarrow$  Streit um begrenzte bzw. unteilbare Betriebsmittel (BM)

## Synchronisation

- $\rightsquigarrow$  Nichtfunktionale Eigenschaft
- Prioritätsumkehr  $\rightsquigarrow$  kontrolliert vs. unkontrolliert
  - Verklemmung (Deadlock)



## Konkurrenz und Koordination

- Betriebsmittelarten  $\rightsquigarrow$  einseitige/mehrseitige Synchronisation
- Konkurrenz  $\rightsquigarrow$  Vergabe/Freigabe (P/V)
- Konflikt  $\rightsquigarrow$  Streit um begrenzte bzw. unteilbare Betriebsmittel (BM)

## Synchronisation

- $\rightsquigarrow$  Nichtfunktionale Eigenschaft
- Prioritätsumkehr  $\rightsquigarrow$  kontrolliert vs. unkontrolliert
- Verklemmung (Deadlock)

## Synchronisationsprotokolle

- Verdrängungssteuerung
- Prioritätsvererbung & Prioritätsobergrenzen
- Blockadezeit  $\rightsquigarrow$  direkt vs. durch Vererbung



## Verdrängungssteuerung (NPCS)

- Unterbindet Verdrängung im kritischen Abschnitt
- Blockadezeit  $\leadsto \max(cs)$
- + Deadlock Prevention  $\leadsto$  Kein „hold and wait“
- + Kein à priori Wissen nötig
- + Einfach; gut für wenige BM
- Verzögerung höher priorer Jobs ohne Konflikt



## Verdrängungssteuerung (NPCS)

- Unterbindet Verdrängung im kritischen Abschnitt
- Blockadezeit  $\leadsto \max(cs)$
- + Deadlock Prevention  $\leadsto$  Kein „hold and wait“
- + Kein à priori Wissen nötig
- + Einfach; gut für wenige BM
- Verzögerung höher priorer Jobs ohne Konflikt

## Prioritätsvererbung (Priority Inheritance)

- Priorität zeitweise erhöhen (von höherprioreren Fäden erben)
- Blockadezeit  $\leadsto \min(n, k) \cdot \max(cs)$
- + Verbessert Verzögerung von Jobs ohne Konflikt
- Transitive Blockierung möglich; Deadlocks möglich





## Prioritätsobergrenzen (Priority Ceiling Protocol)

- Variante der PV mit Prioritätsobergrenzen
- BM-Obergrenze  $\rightsquigarrow \max(p_i)$  aller Jobs die das BM nutzen
- Systemobergrenze  $\rightsquigarrow$  höchstprioreres, belegtes BM (zur *Laufzeit*)
- Betriebsmittelvergabe  $\rightsquigarrow$  BM-Graph (lineare Ordnung)
- Blockadezeit  $\rightsquigarrow \max(cs)$  (wie NPCS)
- + **Deadlock Avoidance**  $\rightsquigarrow$  Kein „cyclic wait“
- + Vermeidet transitive Blockierung
- à priori Wissen nötig; aufwendig; avoidance blocking



## Prioritätsbergrenzen (Priority Ceiling Protocol)

- Variante der PV mit Prioritätsbergrenzen
- BM-Obergrenze  $\rightsquigarrow \max(p_i)$  aller Jobs die das BM nutzen
- Systemobergrenze  $\rightsquigarrow$  höchstprioreres, belegtes BM (zur *Laufzeit*)
- Betriebsmittelvergabe  $\rightsquigarrow$  BM-Graph (lineare Ordnung)
- Blockadezeit  $\rightsquigarrow \max(cs)$  (wie NPCCS)
- + **Deadlock Avoidance**  $\rightsquigarrow$  Kein „cyclic wait“
- + Vermeidet transitive Blockierung
- à priori Wissen nötig; aufwendig; avoidance blocking

## Stackbasierte Prioritätsbergrenzen

- Vereinfachung des klassischen PCP  $\rightsquigarrow$  Stack-based PCP
- Implementiert z. B. in OSEK; Keine Selbstsuspendierung erlaubt!



- 1 Organisatorisches
- 2 Übernahmeprüfung
- 3 Zugriffskontrolle
- 4 Zugriffskontrolle in eCos**
- 5 Hinweise zu Aufgabe 7



Kerneldatenstrukturen durch Sperren des Schedulers geschützt

~> **Big Kernel Lock** (BKL)

- **Sperre:** `void cyg_scheduler_lock(void);`
  - Sofortiges Anhalten des Scheduling
  - Verzögerung der DSR-Ausführungen
  - **ISRs werden weiterhin zugestellt!**
- **Freigabe:** `void cyg_scheduler_unlock(void);`
  - Sofortige Abarbeitung angelaufener DSRs
- Alle **Systemaufrufe** werden per NPCCS synchronisiert
- Anwendungen sollten Mutexe, Semaphore, etc. nutzen
  - **Ausnahme:** Synchronisation zwischen DSR und Thread

<sup>1</sup><http://ecos.sourceware.org/docs-latest/ref/kernel-schedcontrol.html>

# Gegenseitiger Ausschluss – eCos-NPCS<sup>1</sup>

Nicht-preemptiver kritischer Abschnitt durch Sperren des Schedulers

Kerneldatenstrukturen durch Sperren des Schedulers geschützt

~> **Big Kernel Lock** (BKL)

- **Sperre:** `void cyg_scheduler_lock(void);`
  - Sofortiges Anhalten des Scheduling
  - Verzögerung der DSR-Ausführungen
  - **ISRs werden weiterhin zugestellt!**
- **Freigabe:** `void cyg_scheduler_unlock(void);`
  - Sofortige Abarbeitung angelaufener DSRs
- Alle **Systemaufrufe** werden per NPCCS synchronisiert
- Anwendungen sollten Mutexe, Semaphore, etc. nutzen
  - **Ausnahme:** Synchronisation zwischen DSR und Thread

Was sind die Vor- bzw. Nachteile des BKL Konzepts?

<sup>1</sup><http://ecos.sourceware.org/docs-latest/ref/kernel-schedcontrol.html>

## ■ Initialisierung

```
void cyg_mutex_init(cyg_mutex_t* mutex);
```

## ■ Protokoll auswählen:

```
void cyg_mutex_set_protocol(cyg_mutex_t* mutex,
                           enum cyg_mutex_protocol protocol);
```

- CYG\_MUTEX\_NONE keine Prioritätsvererbung
- CYG\_MUTEX\_INHERIT erbe Priorität des aktuellen Inhabers
- CYG\_MUTEX\_CEILING erbe Prioritätsobergrenze

## ■ nur bei CYG\_MUTEX\_CEILING: Prioritätsobergrenze setzen

```
void cyg_mutex_set_ceiling(cyg_mutex_t* mutex,
                           cyg_priority_t priority);
```

## ■ *Prioritätsobergrenze höherprior als Thread: in eCos -1*

<sup>2</sup><http://ecos.sourceware.org/docs-latest/ref/kernel-mutexes.html>

## ■ Mutex belegen

```
cyg_bool_t cyg_mutex_lock(cyg_mutex_t* mutex);
```

### Rückgabewert

- true falls Belegen erfolgreich
- false sonst

## ■ Mutex freigeben:

```
void cyg_mutex_unlock(cyg_mutex_t* mutex);
```



```
1 static cyg_mutex_t s_mutex;
2
3 void cyg_user_start(void) {
4     // Mutex initialisieren
5     cyg_mutex_init(&s_mutex);
6
7     // Protokoll auswaehlen
8     cyg_mutex_set_protocol(&s_mutex, CYG_MUTEX_CEILING);
9
10    // Prioritaetsobergrenze festlegen
11    cyg_mutex_set_ceiling(&s_mutex, 3);
12
13    // Tasks, Alarme etc.
14 }
15
16 void task_entry(cyg_addrword_t data) {
17     cyg_mutex_lock(&s_mutex); // auf Freigabe warten
18     // kritischer Abschnitt
19     cyg_mutex_unlock(&s_mutex); // Mutex freigeben
20 }
```





- 1 Organisatorisches
- 2 Übernahmeprüfung
- 3 Zugriffskontrolle
- 4 Zugriffskontrolle in eCos
- 5 Hinweise zu Aufgabe 7**



## Aufgabensysteme

- 1 3 Aufgaben, 1 Betriebsmittel
- 2 4 Aufgaben, 3 Betriebsmittel
- 3 3 Aufgaben, 2 Betriebsmittel

## Problematische Konstellationen für einzelne Vergabeprotokolle

- Deadlocks
- Pathfinder-Beispiel
- Transitive Blockierung

## Implementierung von 1–3, Basisaufgabe:

- aufgabe\_1.c  $\rightsquigarrow$  Verdrängungssteuerung
- aufgabe\_2.c  $\rightsquigarrow$  Prioritätsvererbung
- aufgabe\_3.c  $\rightsquigarrow$  Prioritätsobergrenzen



(R, 3, 4)

- R: verwendetes Betriebsmittel
- 3: relativer Anforderungszeitpunkt in ms
- 4: Zeit, die Betriebsmittel gehalten wird in ms

## Beispiel: Task mit WCET von 9 ms

```
1 void task() {  
2     non_critical_work(); // 3 ms  
3  
4     cyg_mutex_lock(&R); // acquire  
5     critical_work(); // 4 ms  
6     cyg_mutex_unlock(&R); // release  
7  
8     non_critical_work(); // 2 ms  
9 }
```



# Betriebsmittelnotation

(R, 3, 4)

- R: verwendetes Betriebsmittel
- 3: relativer Anforderungszeitpunkt in ms
- 4: Zeit, die Betriebsmittel gehalten wird in ms

## Beispiel: Task mit WCET von 9 ms

```
1 void task() {  
2     non_critical_work(); // 3 ms  
3  
4     cyg_mutex_lock(&R); // acquire  
5     critical_work(); // 4 ms  
6     cyg_mutex_unlock(&R); // release  
7  
8     non_critical_work(); // 2 ms  
9 }
```

## Phasenversatz im Tracing

- Aufgabensysteme beginnen mit Phasenversatz von 1 ms
- Traces versetzt (Tracenullpunkt entspricht 1 ms)



- [1] Jane W. S. Liu.  
*Real-Time Systems.*  
2000.

