

# Betriebsmittelprotokolle

**Tobias Klaus   Florian Schmaus   Peter Wägemann**

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

23. Januar 2017

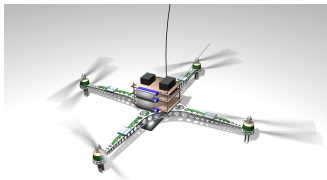


## Wiederholung: Übernahmeprüfung bei terminbasierter Einplanung



- 1 Übernahmeprüfung
- 2 Exkurs: Zustandsautomaten**
- 3 Zugriffskontrolle
- 4 Zugriffskontrolle in eCos
- 5 Hinweise zu Aufgabe 7

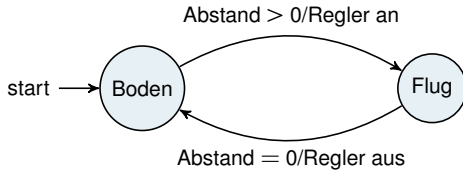




- I4Copter grundsätzlich instabil
- ↳ Fluglageregelung zwingend erforderlich
- Im Flug: Regelkreis geschlossen
- **Aber:** Am Boden Regelkreis offen
- ↳ Regler darf am Boden nicht laufen
  - Andernfalls **Verfälschung des Reglerzustands**

⇒ Zustandsmaschine mit zwei Zuständen





```
1  enum FlightState {
2      Landed,
3      InFlight
4  };

5  enum Event {
6      GroundDistanceGreaterThanZero,
7      GroundDistanceZero
8  };

9  static FlightState g_flightState;
```



```
1 static void state_init(void) {  
2     calibrateSensors();  
3     initializeController();  
  
4     g_flightState = Landed;  
5 }
```

```
1 static void event_loop(void) {  
2     state_init();  
3     while (true) {  
4         Event event = waitForEvent();  
5         state_transition(event);  
6     }  
7 }
```

- In Zustand z.B. zyklischer Ablaufplan
- Analyse einzelner Zustände



# Zustandsübergang

```
1  static void state_transition(Event event) {
2      switch (g_flightState) {
3          case Landed:
4              state_transition_landed(event);
5              break;
6          case InFlight:
7              state_transition_inFlight(event);
8              break;
9      }
10 }
11 static void state_transition_landed(Event event) {
12     if (event == GroundDistanceGreaterThanZero) {
13         action_controllerOn();
14         g_flightState = InFlight;
15     }
16 }
17 static void state_transition_inFlight(Event event) {
18     if (event == GroundDistanceZero) {
19         action_controllerOff();
20         g_flightState = Landed;
21     }
22 }
```





- 1 Übernahmeprüfung
- 2 Exkurs: Zustandsautomaten
- 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 BM

## Synchronisation

- $\rightsquigarrow$  Nichtfunktionale Eigenschaft
- Prioritätsumkehr  $\rightsquigarrow$  kontrolliert vs. unkontrolliert

## Synchronisationsprotokolle

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



## Verdrängungssteuerung (NPCS)

- Unterbindet Verdrängung im kritischen Abschnitt
- Blockierungszeit  $\leadsto \max(cs)$
- + **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 höher Prioreren erben)
- Blockierungszeit  $\leadsto \min(n, k) \cdot \max(cs)$
- + Verbessert Verzögerung von Jobs ohne Konflikt
- Transitive Blockierung möglich; Deadlocks möglich



## Prioritätsbergrenzen (Priority Ceiling Protocol)

- Variante der PV mit Prioritätsbergrenzen
- BM-Obergrenze  $\rightsquigarrow \max(p_i)$  aller Jobs die das BM nutzen
- Systemobergrenze  $\rightsquigarrow$  höchstpriorres, belegtes BM (zur *Laufzeit*)
- Betriebsmittelvergabe  $\rightsquigarrow$  BM-Graph (lineare Ordnung)
- Blockierungszeit  $\rightsquigarrow \max(cs)$  (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 Übernahmeprüfung
- 2 Exkurs: Zustandsautomaten
- 3 Zugriffskontrolle
- 4 Zugriffskontrolle in eCos**
- 5 Hinweise zu Aufgabe 7



# 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 +1 höherprior als Thread*

<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 Übernahmeprüfung
- 2 Exkurs: Zustandsautomaten
- 3 Zugriffskontrolle
- 4 Zugriffskontrolle in eCos
- 5 Hinweise zu Aufgabe 7**



## Aufgabensysteme

- 1 3 Aufgaben, 1 Betriebsmittel  $\rightsquigarrow$  Pathfinder-Beispiel
- 2 3 Aufgaben, 3 Betriebsmittel  $\rightsquigarrow$  Transitive Blockierung
- 3 2 Aufgaben, 2 Betriebsmittel  $\rightsquigarrow$  Deadlocks

## Implementierung von 1–3

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

