Betriebsmittelprotokolle

Tobias Klaus Florian Schmaus Peter Wägemann

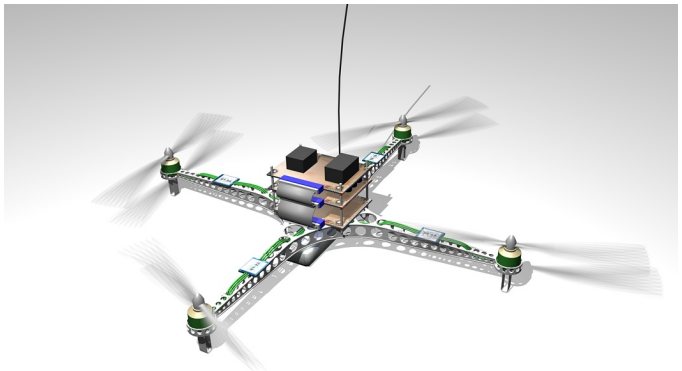
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
Lehrstuhl für Informatik 4 (Verteilte Systeme und Betriebssysteme)
https://www4.cs.fau.de

23. Januar 2017

Wiederholung:
Übernahmeprüfung bei terminbasierter Einplanung

Übersicht

Beispiel-Zustandsautomat



- I4Copter grundsätzlich instabil

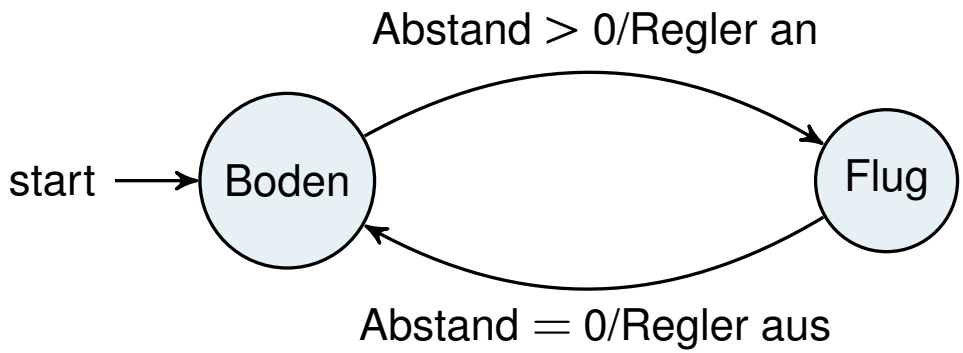
  ⇝ Fluglageregelung zwingend erforderlich

- Im Flug: Regelkreis geschlossen
- Aber: Am Boden Regelkreis offen

  ⇝ Regler darf am Boden nicht laufen
  - Andernfalls Verfälschung des Reglerzustands

⇒ Zustandsmaschine mit zwei Zuständen

Zustandsautomat



Datenstrukturen

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

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

9  static FlightState g_flightState;
```

Ereignisbehandlung

```
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 }
```

Übersicht

Zugriffskontrolle

Konkurrenz und Koordination

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

Zugriffskontrolle

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

Zugriffskontrolle

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

Zugriffskontrolle

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

Zugriffskontrolle

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 Prioren erben)
- Blockierungszeit $\leadsto min(n, k) \cdot max(cs)$
  - \+ Verbessert Verzögerung von Jobs ohne Konflikt
  - \- Transitive Blockierung möglich; Deadlocks möglich

Zugriffskontrolle

Prioritätsobergrenzen (Priority Ceiling Protocol)

- Variante der PV mit Prioritätsobergrenzen
- BM-Obergrenze $\rightsquigarrow$ $max(p_i)$ aller Jobs die das BM nutzen
- Systemobergrenze $\rightsquigarrow$ höchstpriores, belegtes BM (zur *Laufzeit*)
- Betriebsmittelvergabe $\rightsquigarrow$ BM-Graph (lineare Ordnung)
- Blockierungszeit $\rightsquigarrow$ $max(cs)$ (wie NPCS)

\+ Deadlock Avoidance $\rightsquigarrow$ Kein „cyclic wait“

\+ Vermeidet transitive Blockierung

\- à priori Wissen nötig; aufwendig; avoidance blocking

Zugriffskontrolle

Prioritätsobergrenzen (Priority Ceiling Protocol)

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

Stackbasierte Prioritätsobergrenzen

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

Übersicht

Gegenseitiger Ausschluss – eCos-NPCS[1]

Nicht-preemptiver kritischer Abschnitt durch Sperren des Schedulers

Kerneldatenstrukturen durch Sperren des Schedulers geschützt

$\rightsquigarrow$ **Big Kernel Lock** (BKL)

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

[1] `http://ecos.sourceware.org/docs-latest/ref/kernel-schedcontrol.html`

Gegenseitiger Ausschluss – eCos-NPCS[1]

Nicht-preemptiver kritischer Abschnitt durch Sperren des Schedulers

Kerneldatenstrukturen durch Sperren des Schedulers geschützt

$\rightsquigarrow$ **Big Kernel Lock** (BKL)

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

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

[1] `http://ecos.sourceware.org/docs-latest/ref/kernel-schedcontrol.html`

Gegenseitiger Ausschluss – eCos-Mutex[2]

- 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*

[2] http://ecos.sourceware.org/docs-latest/ref/kernel-mutexes.html

Gegenseitiger Ausschluss – eCos-Mutex

- 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);
```

Gegenseitiger Ausschluss – eCos-Mutex

Beispiel

```
1   static cyg_mutex_t s_mutex;

2   void cyg_user_start(void) {
3     // Mutex initialisieren
4     cyg_mutex_init(&s_mutex);

5     // Protokoll auswaehlen
6     cyg_mutex_set_protocol(&s_mutex, CYG_MUTEX_CEILING);

7     // Prioritaetsobergrenze festlegen
8     cyg_mutex_set_ceiling(&s_mutex, 3);

9     // Tasks, Alarme etc.
10  }

11  void task_entry(cyg_addrword_t data) {
12    cyg_mutex_lock(&s_mutex); // auf Freigabe warten
13    // kritischer Abschnitt
14    cyg_mutex_unlock(&s_mutex); // Mutex freigeben
15  }
```

Übersicht

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

Fragen?

Fragen?