Betriebsmittelprotokolle

Florian Franzmann Tobias Klaus Florian Korschin
Florian Schmaus Peter Wägemann

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

20. Januar 2016

Wiederholung: Übernahmeprüfung bei terminbasierter Einplanung

Konkurrenz und Koordination

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

Synchronisation

⇝ Nichtfunktionale Eigenschaft

- Prioritätsumkehr ⇝ kontrolliert vs. unkontrolliert

Synchronisationsprotokolle

- Verdrängungssteuerung
- Prioritätsvererbung
- Prioritätsobergrenzen
- Blockierungszeit ⇝ direkt vs. durch Vererbung

Verdrängungssteuerung (NPCS)

- Unterbindet Verdrängung im kritischen Abschnitt
- Blockierungszeit ⇝ $max(cs)$
- \+ Deadlock Prevention ⇝ 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 ⇝ $min(n, k) \cdot max(cs)$
- \+ Verbessert Verzögerung von Jobs ohne Konflikt
- \- Transitive Blockierung möglich; Deadlocks möglich

Rekapitulation der Vorlesung (Forts.)

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

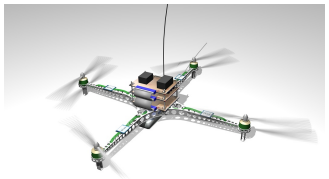
Stackbasierte Prioritätsobergrenzen

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

Übersicht

Beispiel-Zustandsautomat

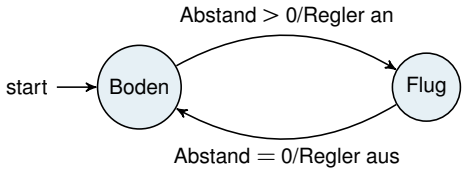
I4Copter



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

$\Rightarrow$ Zustandsmaschine mit zwei Zuständen

Zustandsautomat



Datenstrukturen

```
enum FlightState {
  Landed,
  InFlight
};

enum Event {
  GroundDistanceGreaterThanZero,
  GroundDistanceZero
};

static FlightState g_flightState;
```

Ereignisbehandlung

```
static void state_init(void) {
  calibrateSensors();
  initializeController();

  g_flightState = Landed;
}
```

```
static void event_loop(void) {
  state_init();
  while (true) {
    Event event = waitForEvent();
    state_transition(event);
  }
}
```

Zustandsübergang

```
static void state_transition(Event event) {
  switch (g_flightState) {
    case Landed:
      state_transition_landed(event);
      break;
    case InFlight:
      state_transition_inFlight(event);
      break;
  }
}
static void state_transition_landed(Event event) {
  if (event == GroundDistanceGreaterThanZero) {
    action_controllerOn();
    g_flightState = InFlight;
  }
}
static void state_transition_inFlight(Event event) {
  if (event == GroundDistanceZero) {
    action_controllerOff();
    g_flightState = Landed;
  }
}
```

Übersicht

Gegenseitiger Ausschluss – eCos-NPCS[1]

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 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] ecos.sourceware.org/docs-latest/ref/kernel-schedcontrol.html

Gegenseitiger Ausschluss – eCos-Mutex[2]

Initialisierung

- 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);
```

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

Gegenseitiger Ausschluss – eCos-Mutex

Verwendung

- 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 ⇝ Pathfinder-Beispiel
2. 3 Aufgaben, 3 Betriebsmittel ⇝ Transitive Blockierung
3. 2 Aufgaben, 2 Betriebsmittel ⇝ Deadlocks

Implementierung von 1–3

- `aufgabe_1.c` ⇝ Verdrängungssteuerung
- `aufgabe_2.c` ⇝ Prioritätsvererbung
- `aufgabe_3.c` ⇝ Prioritätsobergrenzen

Aufgabe 7

Aktuelles Problem im I4Copter

- **Problem**: Gemeinsame Nutzung des synchronen SPI Busses per DMA
- **Forderung**:
  - Faden gibt Kontrolle über die CPU während der Übertragung ab!
    ⇝ Kein aktives Warten!
  - Unbeteiligte (niederpriore) Fäden sollen laufen dürfen!
  - Nachrichten sollen mit der Fadenpriorität versendet werden!
  - Kein extra Faden für die Buskoordinierung!
- **Problem**: Klassische Zugriffskontrolle (NPCS, PI, PCP) nicht anwendbar!

Lösungsprämie

Für jede fundierte Lösung (textuell oder codiert) spendieren wir 5€ Guthaben auf unserer Kaffeekasse! ☺