Betriebsmittelprotokolle

Florian Franzmann Martin Hoffmann Tobias Klaus
Peter Wägemann

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

12. Januar 2015

Manöverkritik

Was gut gelaufen ist

+ Echte Hardware
  ⇝ hat uns riesigen Spass gemacht ☺
+ Engagierte Studenten
  ⇝ da bleiben auch wir gern mal noch nach Feierabend da
+ Trotz Hochhaus-Ausfalls brauchbarer Übungsbetrieb
+ Zusatzübungen zur Entzerrung

Was weniger gut gelaufen ist

- Montags sechs Stunden am Stück war (fast) zu viel für uns
- Zu wenig Zeit um die Aufgaben zu überarbeiten
  ⇝ Teilweise keine funktionierenden Aufgaben zum Ausgabezeitpunkt

Manöverkritik

Zusammenfassung

- Wir entschuldigen uns für alle Unannehmlichkeiten und Verspäteten Folien/Aufgabe
- **Danke**, dass Ihr so . . .
  - konstant dabei geblieben seit!
  - gut mitgearbeitet habt!
  - Geduld mit uns hattet!
- **Wir hoffen** . . .
  - ihr habt viel gelernt und mitgenommen!
  - ihr hattet Spaß!
  - euch für das Thema Echtzeitsysteme und Systemsoftware *angefixt* zu haben und euch am Lehrstuhl mal wieder zu sehen!
- **Uns hat dieses Semester viel Freude bereitet!**

Evaluierung

Jetzt seid ihr am Drücker . . .

- **Evaluierung der Übung**
  - Unabhängig von der Vorlesung
  - Euer Feedback ist uns immens wichtig! ⇝ Lob **und** Kritik
  - Helft uns und euren (nachfolgenden) Kommilitonen/innen
  - **Achtung**: Nur noch bis zum 17.01.2015!
- **In der Vergangenheit . . .**
  . . . evaluierten nur 3 %!

Das sollte doch besser gehen!

Zur Motivation gibt's:
Pro **Prozent** 15 g Knabber- und Naschzeug und bei
$\geq$ **50 %** Kaffee & Tee in der letzten Übung!

Rekapitulation der Vorlesung

Kapitel 7: Zugriffskontrolle

Konkurrenz und Koordination

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

Synchronisation

⇝ nicht-funktionale Eigenschaft

- Prioritätsumkehr ⇝ kontrolliert vs. unkontrolliert

Synchronisationsprotokolle

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

Rekapitulation der Vorlesung (Forts.)

Kapitel 7: Zugriffskontrolle

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.)

Kapitel 7: Zugriffskontrolle

Prioritätsobergrenzen (Priority Ceiling Protocol)

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

Stackbasierte Prioritätsobergrenzen

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

Aufgabe 6

Zugriffskontrolle

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

Gegenseitiger Ausschluss – eCos NPCS[1]

Nicht-preemptiver kritischer Abschnitt durch Sperren des Schedulers

Kerneldatenstrukturen sind durch Sperren des Schedulers geschützt
⇝ **Big Kernel Lock** (BKL)

- Sperre: `void cyg_scheduler_lock();`
  - Sofortiges Anhalten des Schedulings
  - Verzögerung der DSR Ausführungen
  - ISRs werden weiterhin zugestellt!
- Freigabe: `void cyg_scheduler_unlock();`
  - 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]

API – 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

API – 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

```
static cyg_mutex_t s_mutex;

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

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

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

  // Tasks, Alarme etc.
}

void task_entry(cyg_addrword_t data) {
  cyg_mutex_lock(&s_mutex); // auf Freigabe warten

  // kritischer Abschnitt

  cyg_mutex_unlock(&s_mutex); // Mutex freigeben
}
```