

Anwendungsstrukturierung in eCos

Vertiefung

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>

10. November 2014

1 Antwortzeitmessung

2 Ablaufsteuerung – Alarme

3 Zeitsteuerung mit Time-Triggered eCos

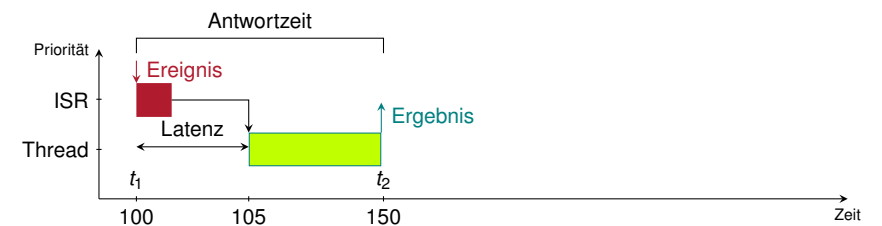
Rekapitulation der Vorlesung

Periodische Echtzeitsysteme

Diskussion

Zeitmessung

Antwortzeit

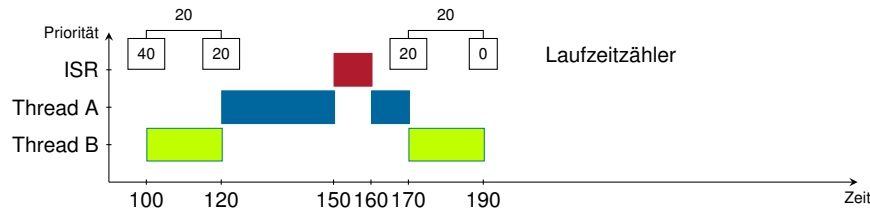


Stoppuhr

- Punkte auf der Zeitachse t_1 und $t_2 \rightsquigarrow$ Ereignis und Ergebnis
- Antwortzeit ist $\Delta t = t_2 - t_1$ (Beispiel: $150 - 100 = 50$ Zählerticks)

Zeitmessung

Rechenzeitverbrauch



Rechenzeitsimulation

- Verbrauchte *Laufzeit* eines Threads
- Vorgegebene Zeit aktiv warten \leadsto Laufzeit verbrauchen

Umsetzung

- Funktion die aktiv t_{wcet} wartet \leadsto Schleife auf Zählerwert
- HW-Zähler läuft bei Unterbrechungen weiter! \leadsto lokaler Zähler
- Dekrement bei jeder Änderung! Beispiel: Sprung von 120 \rightarrow 170

libEZS-WCET-Simulator *ezs_stopwatch.c/.h*

Zur WCET-Simulation soll folgende Funktion implementiert werden:

```
void ezs_lose_time(cyg_uint32 wcet, cyg_uint32 random);
```

Parameter:

- 1.: Gewünschte WCET in 10 μ s-Ticks
- 2.: Maximum des zufällig zu subtrahierenden WCET-Anteils

Implementierung muss internen Zähler verwalten

- \leadsto Bei jeder Änderung des Systemzählers angepasst
- Jeder Thread besitzt einen eigenen Stack!
- \leadsto *keine* globale Zustandsvariable notwendig

Abfragebetrieb

Hinweis

Auflösung der Zähler in Pikosekunden:

\rightarrow `ezs_counter_resolution_ps()`

Alarme und periodische Aktivierung

eCos-Zähler Abstraktion

eCos-*Alarme* basieren auf eCos-*Zählern* (Counter¹)

- Anwendung erzeugt Zähler für bestimmtes Ereignis
 - Zeitgeberunterbrechung (\rightarrow DSR)
 - externes Ereignis (Taster, etc.)
- \rightarrow Zähler wird „von Hand“ inkrementiert

```
void cyg_counter_tick(cyg_handle_t counter)
```

- Alternativ: eCos-interne Uhr als Zähler (\rightarrow Aufgabe 3)
- eCos verwaltet Zählerstand intern
- Aktiviert ggf. Alarm bei Erreichen eines Zählerstandes

¹ecos.sourceware.org/docs-latest/ref/kernel-counters.html

Alarme und periodische Aktivierung – 1

Anlegen eines Alarms

eCos-*Alarm*² führt Aktion bei Erreichen eines Zählerstandes aus

Anlegen:

```
void cyg_alarm_create(cyg_handle_t counter,
                     cyg_alarm_t* alarmfn,
                     cyg_addrword_t data,
                     cyg_handle_t* handle,
                     cyg_alarm* alarm);
```

- counter zugeordneter Zähler
- alarmfn Alarmbehandlung (Funktionspointer)
- data Parameter für Alarmbehandlung
- handle Alarm Handle (vgl. Threaderzeugung)
- alarm Speicher für Alarmobjekt (vgl. Threaderzeugung)

²ecos.sourceware.org/docs-latest/ref/kernel-alarms.html

Alarme und periodische Aktivierung – 2

eCos-Alarm

eCos-*Alarm*³ führt Aktion bei Erreichen eines Zählerstandes aus

2 Alarminitialisierung

```
void cyg_alarm_initialize(cyg_handle_t alarm,
cyg_tick_count_t trigger, cyg_tick_count_t interval);
```

- alarm Alarmhandle
- trigger *Absolute* Zählerticks bis zur *ersten* Aktivierung
 - ~ Nutze `cyg_current_time()` + x
 - ~ Festlegung der *Phase*
- interval Zählerintervall für folgende *periodische* Aktivierungen

3 Alarm freischalten

```
void cyg_alarm_enable(cyg_handle_t alarm)
```

³ecos.sourceware.org/docs-latest/ref/kernel-alarms.html

Alarme und periodische Aktivierung

eCos-Uhr

eCos-*Uhren* (Clocks⁴) sind spezialisierte *Zähler*

- Basierend auf Zeitgeberunterbrechung
- Festgelegte Zeitauflösung beim Erstellen

```
void cyg_clock_create(cyg_resolution_t resolution,
cyg_handle_t* handle, cyg_clock* clock);
```

- Uhrenzähler wird „von Hand“ inkrementiert

1 Handle auf Uhr-internen Zähler holen

```
void cyg_clock_to_counter(cyg_handle_t clock,
cyg_handle_t* counter)
```

2 Inkrementieren

```
void cyg_counter_tick(cyg_handle_t counter)
```

- Alternativ: Handle auf Scheduler Uhr (→ Aufgabe3)

```
cyg_handle_t cyg_real_time_clock(void);
```

⁴ecos.sourceware.org/docs-latest/ref/kernel-clocks.html

tt-eCos

eCos ist eigentlich *ereignisgesteuert*

→ Studienarbeit: Time-Triggered eCos:

- Zeitgesteuerte Ausführung von Tasks in Ablauftabellen.
- Terminüberwachung mit Ausnahmebehandlung
- Angelehnt on OSEKtime (Automobilstandard)

Ausführliche Dokumentation

→ Ausarbeitung der Studienarbeit von Michael Lang:

www.opus.ub.uni-erlangen.de/opus/volltexte/2008/1015/pdf/sa_michael_lang.pdf

tt-eCos Taskkonstruktion

Ablauf Tabellen und Tasks werden statisch (global) angelegt:

- 1 Definition der Ablauf Tabellen unter Angabe der maximalen Ereignisseinträge. (Makro!)

```
tt_DispatcherTable (string <Tabellenname>,
tt_uint32 <Eintragsanzahl>)
```

- 2 Definition der Task(s) und Implementierung des Task-Programms. (Run-to-completion!)

```
tt_Task ( string <Task-Name>) { .. Programm .. }
```

- 3 Definition des Idletasks und optionaler Hook-Routinen.

```
tt_IdleTask {.. Programm .. }
```

tt-eCos Initialisierung

Initialisierung zur Laufzeit (in `cyg_user_start()`):

- 1 Initialisierung der Tasks unter Angabe ihrer Terminüberwachungsmethode.

```
tt_InitTask (tt_tasktype <Task-Name>,
            tt_deadlinemethod <Terminmethode>);
```

- 2 Initialisierung der Ablauftabelle(n).

```
tt_InitDispatcherTable( tt_dispatcherdatatype <Tabellenname>)
```

- 3 Definition der Ereignisse der einzelnen Tabellen.

```
tt_bool tt_DispatcherTableEntry(
    tt_dispatcherdatatype <Tabellenname>,
    tt_ticktype <Zeitpunkt>,
    tt_action <Ereignis>,
    tt_tasktype <Task-ID> )
```

- 4 Starten des Betriebssystems.

```
void tt_startos( tt_dispatcherdatatype <Anfangstabelle> )
```

Einlastung

- Einlastung erfolgt präemptiv
- Neue Aufgabe unterbricht Ausführung laufender Aufgabe
- Anschliessend Fortsetzung der unterbrochenen Aufgabe
- ~ Terminüberprüfung möglich

Terminüberwachung

Für jeden Thread mittels `tt_deadlinemethod` konfigurierbar:

- `TT_STRINGENT` Strikte Terminüberprüfung *direkt* nach Ablauf des Termins
- `TT_NONSTRINGENT` Nicht-Strikte Terminüberprüfung zu einem späteren Zeitpunkt (Terminverletzung möglich)

Einplanung von Taskstart oder Terminüberwachung (`tt_action`):

- `TT_START_TASK` Task Einplanung
- `TT_DEADLINE` Terminüberprüfung

```
tt_bool tt_DispatcherTableEntry(
    tt_dispatcherdatatype <Tabellenname>,
    tt_ticktype <Zeitpunkt>,
    tt_action <Ereignis>,
    tt_tasktype <Task-ID>)
```

Fragen

Fragen?