AUFGABE 4: SIMPLE SCOPE

In den vorangegangenen Übungsaufgaben haben Sie bereits periodische Aufgaben kennengelernt. Bislang erfolgte deren Implementierung durch relative Verzögerung der Fäden. In dieser Aufgabe geht es nun um die ordentliche Umsetzung eines periodischen Aufgabensystems mit den in der Übung vorgestellten Funktionen des eCos-Betriebssystems.
Im Verlauf dieser Übung werden Sie das Gerüst für ein einfaches Oszilloskop mit folgendem Funktionsumfang implementieren:

- Abtasten zweier Signale
- Berechnung der Leistungsdichtespektrums[1] für ein Signal
- Darstellung von Zeit- und Frequenzbereich auf einem (simulierten) Display

Die Anforderungen sollen durch das folgende System periodischer Aufgaben umgesetzt werden:

Aufgabe	**Bezeichnung**	**Periode** / ms	**WCET** / ms	**Priorität**
T_1	Abtastung Signal 1	10	2	_
T_2	Abtastung Signal 2	20	2	_
T_3	Analyse	20	4	_
T_4	Darstellung	40	6	_

Die Aufgaben verfügen über implizite Termine zu Beginn ihrer nächsten Periode. Neben dem Ihnen bereits bekannten (ereignisgesteuerten) eCos verwenden Sie in dieser Übungsaufgabe zusätzlich die zeitgesteuerte Variante tt-eCos. Nach dem Entpacken befinden sich diese Varianten jeweils in den Unterordnern `event-triggered` beziehungsweise `time-triggered`. Beide Varianten unterscheiden sich hinsichtlich der API lediglich in der Umsetzung der Fäden. Den Link zur jeweiligen Funktionsreferenz finden Sie in den allgemeinen Hinweisen.

Auch in dieser Aufgabe benötigen Sie die Funktion `ezs_simulate_wcet`, sowie jeweils eine passende Umrechnung zwischen Realzeit und `{cyg,ezs,tt}_ticks`. Sie können diese mittels der bereitgestellten Tests testen. Verwenden Sie wo möglich Ihre Implementierung aus den vorangegangenen Aufgaben.

☞ make sanity-test

Achten Sie bei der Bearbeitung der Übungsaufgabe darauf, dass Ihre Messungen auch bei der Abgabe noch nachvollziehbar sind!

[1] Betragsquadrat der kurzzeit Fourier-Transformation, (*engl. short-time Fourier-transform, STFT*)

1 Aufgabenstellung

Nach dem erfolgreichen Aufsetzen des `build`*-Verzeichnisses können Sie sich mittels* `make doc` *eine Übersicht über alle in* `libEZS` *bereitgestellten Funktionen inklusive deren Dokumentation erzeugen.*

Denken Sie daran, Ihren Quellcode nach vollständiger Bearbeitung noch vor dem Beginn der Rechnerübung abzugeben. Rufen Sie hierzu in Ihrem `build`*-Verzeichnis* `make submit` *auf.*

1.1 Planbarkeitsanalyse:

Zunächst ist die grundsätzliche Frage der Planbarkeit des periodischen Aufgabensystems zu klären.

Aufgabe 1

Vergeben Sie hierfür statische Prioritäten für die Aufgaben gemäß des aus der Vorlesung bekannten *Ratenmonotonen Algorithmus* (RM).

Aufgabe 2

Überprüfen Sie die generelle Planbarkeit des Systems mittels des antwortzeitbasierten Verfahrens.

Antwort:

Sie werden in diesem Aufgabenblatt zunächst nur die Aufgaben-*Struktur* betrachten. Implementieren Sie hierfür das gegebene Aufgabensystem und simulieren Sie die Ausführungszeiten der Fäden mit Hilfe von `ezs_simulate_wcet()` so, dass diese zwischen 0% und 100% der angegebenen WECT schwanken. Eine Implementierung der tatsächlichen Aufgabenfunktionalität ist nicht erforderlich.

Mit dem bereitgestellten *Software-Tracing* können Sie das zeitliche Verhalten des Systems und speziell die Perioden von T_2 und T_3 bestimmen. Sichern Sie alle relevanten Tracing-Dateien für die Abgabe. Die der letzten Ausführung befindet sich unter `build/tracetmp/tracefile`. Standardmäßig werden 256

☞ `make trace`

Einlastungsereignisse aufgezeichnet.

Hinweis: Der Tracer stellt Prioritätsebenen dar und erlaubt somit keine Unterscheidung von Aufgaben auf einer Prioritätsebene. Weiten Sie bei Bedarf den Prioritätenraum der eingesetzten Planungsalgorithmen auf um Aufgaben unterscheidbar zu machen. Ferner nutzt der tracer zur Kommunikation der Tracewerte die serielle Schnittstelle. **Schließen Sie** deshalb unbedingt während des Tracings andere serielle Konsolenclienten wie **cutecom**.

1.2 Ereignisgesteuerte Umsetzung (eCos):

In der ereignisgesteuerten Umsetzung sollen die Aufgaben nach dem in der Vorlesung vorgestellten *Rate-Monotonic-Algorithmus* eingeplant werden. Verwenden Sie für die periodische Aktivierung der Fäden die von eCos bereitgestellten *Alarme*. Der Parameter `trigger` der Funktion `cyg_alarm_initialize()` soll hierbei zunächst auf `cyg_current_time()` + 1 gesetzt werden.

☞ `.../event-triggered`

Aufgabe 3

Was beobachten Sie hinsichtlich der zeitlichen Parameter der einzelnen Aufgaben? Welche Auswirkungen sehen Sie hinsichtlich der Abtastung von Signal 2?

Antwort:

Aus den gewonnenen Daten lässt sich ein verbesserter Ablaufplan, wie er in der Vorlesung vorgestellt wurde, bestimmen, der die Beeinflussung der einzelnen Fäden untereinander vermindert oder sogar unterbindet.

Aufgabe 4

Passen Sie den Parameter `trigger` der Alarme entsprechend an und wiederholen Sie die Messung. Welches Vorgehen haben Sie gewählt und welches Vorwissen war dafür nötig? Was beobachten Sie?

☞ getrennte Aufzeichnung!

Antwort:

1.3 Taktgesteuerte Umsetzung (tt-eCos):

Im nächsten Schritt soll das Aufgabensystem in die zeitgesteuerte Variante tt-eCos integriert werden. ☞ .../time-triggered

Lesen Sie die folgenden Hinweise, bevor Sie mit der Bearbeitung der zeitgesteuerten Umsetzung anfangen:

- Es ist leider nicht möglich, zwei Aufgaben zum selben Zeitpunkt einzuplanen. Dies betrifft auch Deadlineüberprüfungen. Sie dürfen deswegen davon ausgehen, dass $D_i = p_i - 1\,\text{ms}$.
- Achten Sie auf eine ausreichend groß angelegte Tabelle

  ⇒ `tt_DispatcherTable(table1, XXX);`

  Für jede Ereignisbehandlung (d. h. die eigentlich Behandlung *und* die Deadlineüberprüfung) muss Platz in der Ablauftabelle sein.
- Die Deadlineüberprüfung kann auch schon *vor* dem ersten Start geplant werden, sie wird dann erst in der nächsten Hyperperiode aktiv.
- Für die taktgesteuerte Variante tt-eCos werden automatisch generierte fortlaufende Identifikationsnummern zur Darstellung der einzelnen Fäden im Tracer herangezogen.

Stellen Sie einen geeigneten Ablaufplan auf und planen Sie für jede Aufgabe eine Deadlineüberprüfung ein! Sorgen Sie von Anfang an dafür, dass sich die Fäden nicht überlappen.

Aufgabe 5

Führen Sie wiederum eine Messung der Perioden von T_2 und T_3 durch und zeichnen Sie diese auf. Was beobachten Sie?

Antwort:

Aufgabe 6

Welche Vor- beziehungsweise Nachteile sehen Sie gegenüber der ereignisgesteuerten Lösung?

Antwort:

Aufgabe 7

Was würde hinsichtlich des Aufwands für Analyse und Planung passieren, wenn Sie z. B. 41 ms als Periode für die Ausgabe und 7 ms für die Abtastung wählen?

Antwort:

Aufgabe 8

Wie sollten die Perioden von Aufgaben also in der Praxis ausgewählt werden?

Antwort:

2 Erweiterte Aufgabe

Die Erweiterten Übungsaufgaben sind nur für Teilnehmer verpflichtend, die das 7,5-ECTS-Modul belegen. Wir werden Sie natürlich auch dann bei der Bearbeitung unterstützen, wenn Sie diese Teilaufgaben freiwillig bearbeiten.

2.1 Ablaufplanung mit dynamischen Prioritäten

Bisher haben Sie ausschließlich mit statischen Prioritäten gearbeitet. In der erweiterten Aufgabe werden Sie nun eine einfache Variante eines EDF-Schedulers implementieren und somit dynamische Prioritätsvergabe verwenden. Im Unterordner `ext-edf` finden Sie bereits das Gerüst eines Schedulers.

Um eine Anpassung der eCos-Codebasis zu vermeiden, werden Sie Ihren EDF-Scheduler als Scheduler auf Benutzerebene implementieren. Als weitere Vereinfachung nehmen wir ein kooperatives Aufgabenmodell an, das heißt Aufgaben beenden sich selbstständig durch einen Aufruf von `ezs_thread_suspend()`. Diese Funktion, ebenso wie ihr Pendant `ezs_thread_resume()`, kapselt den entsprechenden Systemaufruf, um so eine einfache Möglichkeit zu haben, dem EDF-Scheduler Zustandsänderungen (bspw. das Beenden eines Threads) mitzuteilen.

Sie dürfen davon ausgehen, dass jeder Fanden zu einem bestimmten Zeitpunkt terminiert, also nicht endlos läuft. Es genügt weiterhin Terminverletzungen zu einem geeigneten Zeitpunkt zu erkennen. Eine Überprüfung exakt zum Termin ist nicht nötig.

Für die EDF-Variante werden auf den Adressen der `cyg_thread`-Objekte basierende Identifikationsnummern zur Darstellung der einzelnen Fäden im Tracer herangezogen.

Aufgabe 9

Erstellen Sie zunächst das Aufgabensystem inklusive der nötigen Alarme. Machen Sie sich hierfür auch mit der Funktionsweise von `ezs_{get,set}_deadline()` vertraut. Nutzen Sie auch hier `ezs_simulate_wcet()`, um die Ausführungszeiten der Aufgaben zu simulieren.

Aufgabe 10

Implementieren Sie nun die Scheduler-Funktionalität entsprechend der in der Tafelübung vorgestellten Vorgehensweise. Diese gliedert sich grob in drei Teile:

- Die Anpassung an zeitliches Fortschreiten des Systems, das heißt, das Vergehen von Ticks (`ezs_edf_pending_ticks()`, `ezs_edf_do_tick()`).
- Die Verwaltung der Aufgabe in einer sortierten Liste (`ezs_edf_insert()`, `ezs_edf_update_priorities()`).
- Die Erweiterung der `resume`- und `suspend` Funktionen, um die nötigen Anpassungen an den Aufgabenverwaltungsstrukturen bei Einlastungsereignissen auszuführen.

Fertigen Sie eine Aufzeichnung ihrer Ausführung an.

Aufgabe 11

In der Vorlesung haben Sie Anforderungen an das Aufgabensystem zur Anwendbarkeit der Antwortzeitanalyse zur Feststellung der Planbarkeit bei ratenmonotoner Einplanung kennengelernt. Inwiefern lässt sich diese Analyse auf die Einplanung mittels EDF übertragen? Muss eine neue Analyse durchgeführt werden?

Antwort:

Hinweise

- Bearbeitung: Gruppe mit je drei Teilnehmern.
- Abgabefrist: 10.06.2024 (23:59) ☞ `make submit`
- Fragen bitte an `i4ezs@lists.cs.fau.de`