AUFGABE 2: ANTWORTZEIT

In den ersten beiden Teilaufgaben dieser Aufgabe sollen Sie nützliche Funktionen für den Umgang mit Zeit in einem Rechensystem implementieren, die wir im weiteren Verlauf der Übung immer wieder verwenden werden. Die letzten Teilaufgaben sollen Ihnen dann ein erstes Gefühl für die Probleme beim Zusammenspiel von periodischen und nicht-periodischen Ereignissen in einem Echtzeitsystem vermitteln.

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 Zeitmessung mit der libEZS:

Um die zeitlichen Abläufe im System messen zu können, muss zunächst die `libEZS` erweitert werden. Daher beziehen sich die meisten Dateinamen in dieser Aufgabe auf Dateien im Unterverzeichnis `libEZS`. Einen hardwareunabhängigen Zähler haben wir bereits vorgegeben. Sie können auf diesen mittels der Funktion `ezs_counter_get()` zugreifen und den aktuellen Wert in *Ticks* auslesen. Lesen Sie die Dokumentation der von uns bereitgestellten Funktionen!

☞ `libEZS/...`

☞ `include/ezs_counter.h`

☞ `make doc`

Aufgabe 1

Implementieren Sie die Funktionen `ezs_watch_start()` und `ezs_watch_stop()` in `libEZS/src/ezs_stopwatch.c`. Beachten Sie hierbei auch die vorgegebene Signatur und Kommentare in `stopwatch.h`. Was bedeuten die angegebenen Datentypen für Ihre zukünftigen Messungen?

Antwort:

Hinweise: Damit mehrere Zeitmessungen parallel und über Funktionsgrenzen hinweg erfolgen können, erhalten beide Funktionen einen Zeiger auf den Zustand der Messung durch den Aufrufer. Die Zustandsvariable, die diesen Zustand abbildet, muss daher später in der Anwendung global[1] angelegt werden. Die Funktion `ezs_watch_stop()` liefert das Ergebnis der Messung in Form von Zeitgeber-Ticks zurück. Die Dauer eines solchen Ticks ist prinzipiell hardwareabhängig und wird wie in der Tafelübung besprochen von einer Struktur aus Dividend und Divisor beschrieben (`ezs_counter_get_resolution()`).

1.2 WCET-Simulation:

Um in dieser und den folgenden Aufgaben möglichst reale Anwendungssysteme betrachten zu können, müssen wir den Rechenzeitaufwand komplexer Anwendungen simulieren können. Hierfür soll die Funktion `ezs_simulate_wcet()` genutzt werden. In der von uns vorgegebenen Implementierung nutzt sie den EZS-Zeitgeber um aktiv zu warten, bis die vorgegebene Anzahl Ticks vergangen ist.

☞ ezs_stopwatch.c

Aufgabe 2

Nutzen Sie den in der Tafelübung präsentierten Algorithmus um die Funktion so zu erweitern, dass sie auch mit *Unterbrechungen* umgehen kann. Ignorieren Sie hierfür zunächst den zweiten Parameter `jitter`. Weshalb ist das Beachten der Unterbrechungen notwendig?

Antwort:

Aufgabe 3

Stellen Sie sicher, dass Ihre Funktion nie *mehr* Zeit verbraucht als der übergebene Zeitwert vorschreibt. Woraus ergibt sich diese Anforderung?

Antwort:

[1] Gültigkeit von Variablen in C, Folie 37ff: `https://www4.cs.fau.de/Lehre/SS15/V_SP1/Vorlesung/Folien/SP1-02-A4.pdf`

Aufgabe 4

Stellen Sie mit Hilfe unserer Testfälle sicher, dass Ihre Implementierungen von `ezs_watch_start()`, `ezs_watch_stop()` und `ezs_simulate_wcet()` korrekt funktionieren.

☞ `make sanity-test`

Aufgabe 5

Erweitern Sie `ezs_simulate_wcet()` nun so, dass ein zufällig gewählter Anteil der angefragten Zeit *nicht* verbraucht wird. Die maximale Größe dieses zufälligen Anteils ist über den zweiten Parameter `jitter` der Funktion wählbar. Dieser gibt an, welcher Anteil in Prozent der angefragten Zeit maximal übersprungen werden soll. Ein Wert von 20 für `jitter` bedeutet beispielsweise, dass die tatsächliche Laufzeit von `ezs_simulate_wcet()` zwischen 80% und 100% der angegebenen Laufzeit schwanken soll.

☞ `rand()%100`

1.3 Signalerzeugung:

Die Wiedergabe (Rekonstruktion) von kontinuierlichen Signalen, beispielsweise von Musikstücken, ist eine typische Aufgabe eines Echtzeitsystems. Hierbei gilt das Nyquist-Shannon-Abtasttheorem[2], wonach für eine korrekte Wiedergabe eines zeitdiskreten Signals die Abtastfrequenz f_{sample} so zu wählen ist, dass sie größer $2 \cdot f_{\text{max}}$ (Maximalfrequenz des Signals) ist. Bei der Wiedergabe digitalisierter Musikstücke ist diese Abtastfrequenz (engl. sampling rate) von entscheidender Bedeutung für die verzerrungsfreie Rekonstruktion analoger Signale. Der Einhaltung des Abtasttheorems fällt also eine zentrale Bedeutung bei der Implementierung eines solchen Echtzeitsystems zu.

Aufgabe 6

Verwenden Sie, ähnlich wie in der vorangegangenen Aufgabe, die Funktion `sinf()` um diesmal ein überlagertes Sinus-Signal zu erzeugen. Dieses soll eine Komponente mit einer Frequenz von 2 Hz und eine mit 13 Hz enthalten. Geben Sie den errechneten Signalverlauf auf dem Digital-Analog-Umsetzer so aus, dass das erwünschte analoge Signal korrekt rekonstruiert wird. *Mit welcher minimalen Rate müssen Sie die Abtastwerte wiedergeben?*

[2] `https://de.wikipedia.org/wiki/Abtasttheorem`

Antwort:

Aufgabe 7

Verwenden Sie Ihre `ezs_simulate_wcet()` um zusätzlich zu Ihrer mit `ezs_delay_us()` eingebrachten Verzögerung Rechenlast in der Größenordnung der Periode zu simulieren. Achten Sie darauf, diese Zeit korrekt in die benötigte Anzahl Ticks umzurechnen. Nutzen Sie den Parameter `jitter`, um die tatsächliche zusätzliche Arbeitslast zwischen 10% und 100% schwanken zu lassen. *Wie wirkt sich die so eingebrachte Rechenlast auf die Periodizität Ihrer Aufgabe aus? Betrachten Sie auch die Fouriertransformierte des so wiedergegebenen Signals am Oszilloskop.*

☞ `ms_to_ezs_ticks()`

Antwort:

1.4 Antwortzeit:

In dieser Aufgabe setzen wir für die Erzeugung von Ereignissen die serielle Schnittstelle ein. Diese teilt durch das Auslösen eines Interrupts mit, wenn ein Zeichen eingelesen wurde. Wir haben den entsprechenden Interrupt bereits in der Vorgabe für Sie aufgesetzt und eine rudimentäre Interruptbehandlung implementiert.

Cutecom bietet die Möglichkeit, optional nach jedem übertragenen Zeichen eine Verzögerung einzufügen. Es empfiehlt sich für die folgenden Aufgaben, Tests sowohl mit einer hohen Verzögerung (10ms) und ohne Verzögerung (0ms) durchzuführen. Für beide Modi stehen für die „Send File“ Funktion ebenfalls geeignet dimensionierte Testdateien (`./test/input-{small,large}`) bereit.

☞ `/proj/i4ezs/tools/cutecom`

Aufgabe 8

Wieso ist die von uns bereitgestellte Interruptbehandlung für ein komplexes Echtzeitsystem ungeeignet?

Antwort:

Implementieren Sie unter Nutzung der in der Übung vorgestellten Konzepte *DSR* und *Faden* eine bessere Interruptbehandlung, die die eingelesenen Zeichen mit Hilfe der Funktion `ezs_printf()` auf der seriellen Schnittstelle im Faden-Kontext ausgibt. Vergeben Sie die Prioritäten für diese neue Aufgaben so, dass die Signalerzeugung aus der vorangegangenen Teilaufgabe eine niedrigere Priorität (größere Zahl) erhält.

Aufgabe 9

Verwenden Sie die zuvor implementierte Zeitmessung per Systemzeitgeber, um die Antwortzeit zwischen dem Auftreten des Interrupts und der Ausgabe zu ermitteln. *Wo beginnt die Messung? Wo ist sie zu Ende?*

☞ `ezs_watch_start()`, `ezs_watch_stop()`

Antwort:

Geben Sie die Ergebnisse Ihrer Messung in Nanosekunden wiederum über die serielle Schnittstelle aus. Testen Sie zunächst mit einzelnen Zeichen und danach mit einfachen Zeichenketten (bspw. Alphabet) und unterschiedlichen Char-Delays. Überprüfen Sie auch die Vollständigkeit Ihrer Übertragung.

☞ ezs_print_measurement

☞ No Line end

Aufgabe 10

Nutzen Sie die cutecom-Funktion „Send File“, um die serielle Schnittstelle auszulasten. Führen Sie diese Messung mit unterschiedlichen Char-Delays aus. Wieso ist dies wichtig? Speichern Sie die in beiden Fällen gemessenen Werte nach der Messung über die serielle Schnittstelle auf Ihrem Desktop-PC ab. *Wie verhält sich das Sinussignal während der Übertragung? Wie verhalten sich die Antwortzeiten?*
Hinweis: Als Hilfestellung bieten wir ein Script zur Auswertung von seriellen Mitschnitten an: `./scripts/aggregate.pl path/to/cutecom.log`. *Hinweis: In seltenen Fällen überfordert eine große Datenmenge mit geringem Char-Delay das Board, so dass es nicht mehr auf Eingaben reagiert. In diesem Falle das Board kurz vom Rechner trennen.*

☞ Log-To:

Antwort:

Aufgabe 11

Vertauschen Sie nun die Prioritäten so, dass die Wiedergabe des Sinus-Signals die höchste Priorität(niedrigste Zahl) erhält. *Wie verhält es sich jetzt mit der Antwortzeit? Wodurch kommt der Unterschied zur vorherigen Messung zustande?* Betrachten Sie auch die Minima und Maxima.

Antwort:

Aufgabe 12

Sehen Sie handwerkliche bzw. methodische Schwächen im Messaufbau? Wie könnten diese behoben werden? Wie äußern sich diese in Ihren Messwerten? Welche Parameter begünstigen dieses Fehlverhalten?

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.

Aufgabe 13

Durch die Nutzung der Funktionen `ezs_watch_start()` und `ezs_watch_stop()` bringen Sie zusätzlichen Mehraufwand ein, der im System ohne Messung nicht vorhanden ist. Dadurch wird Ihre Messung verfälscht. *Entwerfen und Implementieren Sie einen Messaufbau, mit Sie diesen Mehraufwand ermitteln können. Wie hoch ist der durch die Messung induzierte Fehler?*

Antwort:

Aufgabe 14

Wie kann dieser Fehler durch entsprechende Programmierung weiter vermindert werden? Setzen Sie mindestens zwei Varianten um und bewerten Sie den Einfluss.

Antwort:

Hinweise

- Bearbeitung: Gruppe mit je drei Teilnehmern.
- Abgabefrist: 14.11.2019 ☞ `make submit`
- Fragen bitte an `i4ezs@lists.cs.fau.de`