**Institut für Informatik**
**Arbeitsgruppe Software Engineering**
Barry Linnert

**Betriebssysteme WiSe 2017/18· Übungsblatt 3**
Bearbeitungszeit: 23.11.2017 – 07.12.2017 um 09:59 Uhr

Bereiten Sie Ihre **Lösungen** grundsätzlich so vor, dass Sie diese in der Übung Ihren Kommilitonen in geeigneter Form **zeigen** und **diskutieren** können. Geben Sie bitte stets Ihre verwendeten **Quellen** an.

Die Abgabe erfolgt elektronisch über das KVV.

## Aufgabe 3-1 (Threads) – 6 Punkte

Grenzen Sie die folgenden Begriffe auf Grund der Vorlesung gegeneinander ab:

- gegenseitiger Ausschluss
- Signalisierung
- Synchronisation
- Koordination
- Kommunikation
- Kooperation

Gehen Sie bei Ihrer Diskussion insbesondere auf die Teilnehmer und ihre Beziehung zueinander ein.

## Aufgabe 3-2 (Interrupts – System Timer und serielle Schnittstelle) – 15 Punkte

Mit dem Abfangen von Ausnahmen ist unser „Betriebssystem“ noch kein wirkliches Betriebssystem, sondern eher ein Bare-Metal-Programm mit ein wenig Programmier-Komfort. Um diesen Umstand auszuräumen, unterstützen wir in dieser Aufgabe auch Hardware-Interrupts. Hardware-Interrupts bieten später die einzige Möglichkeit für das Betriebssystem die Kontrolle über den Prozessor wiedererlangen zu können, wenn mal ein Prozess in einer Endlosschleife hängen sollte. Außerdem geben uns Hardware-Interrupts die Möglichkeit, zeitnah auf Hardware zu reagieren, auch wenn der Prozessor gerade eigentlich mit etwas anderem beschäftigt ist.

Speziell in dieser Aufgabe stellen wir das Lesen (und optional das Schreiben) von der seriellen Schnittstelle (bei uns DBGU) von Polling auf Interrupt-getriebene Behandlung um. Interrupt-getriebenes Lesen verringert die Chance, dass wir ein Zeichen verpassen, weil wir es nicht rechtzeitig abgeholt haben. (Interruptgetriebenes Schreiben würde zudem ein Fortsetzen der Programmausführung ermöglichen, ohne dass auf die Fertigstellung der Übertragung des vorherigen Zeichens gewartet werden muss.)

Damit sichergestellt ist, dass uns ein Hardware-Interrupt ereilt (auch wenn der Benutzer keine Taste drückt), setzen wir außerdem proaktiv den System Timer (ST) ein, um so periodisch Interrupts zu erzeugen und unserem Betriebssystem eine garantierte Eingriffsmöglichkeit zu geben.

Um dies zu erreichen, sind folgende Teile zu erledigen (nicht unbedingt in dieser Reihenfolge):

1. Initialisierung des System Timers so, dass er regelmäßig Interrupts auslöst. Machen Sie dabei die Häufigkeit der Interrupts (im Quellcode) konfigurierbar.
2. Aktivierung des entsprechenden Interrupts an der DBGU.
3. Konfigurierung aller Komponenten, durch die ein Interrupt-Signal durchgeht (AIC und ARM-Kern selbst), sodass Sie einen Interrupt auch tatsächlich bemerkt.
4. Entwickelung eines Interrupt-Handlers, der den Interrupt an allen notwendigen Komponenten bestätigt und anschließend die eigentliche Programmausführung fortsetzt.
5. Umstellung des Lesens von der DBGU auf Interrupt-Betrieb: bei einem DBGU-Interrupt wird das Zeichen gelesen und zwischengespeichert; auf Nachfrage der Anwendung wird das Zeichen der Anwendung zur Verfügung gestellt. Legen Sie fest, wie Sie mit dem Fall umgehen wollen, dass ein weiteres Zeichen ankommt, obwohl die Anwendung das vorherige noch nicht abgeholt hat.

Außerdem sollen Sie wieder die Funktionalität Ihres Codes unter Beweis stellen:

6. Schreiben Sie eine „Anwendung“, die immer abwechselnd auf einen Tastendruck wartet und anschließend eine „Berechnung“ durchführt. Die Berechnung soll aus der wiederholten (aber nicht endlosen) Ausgabe des empfangenden Zeichens (mit kleinen Pausen) bestehen.
7. Machen Sie zudem eine Ausgabe, wenn ein Timer-Interrupt festgestellt wird: ein Ausrufezeichen gefolgt von einem Zeilenumbruch.

Damit ergibt sich eine Ausgabe, die der folgenden ähneln sollte (je nach Häufigkeit der Timer-Interrupts, sowie Anzahl der Wiederholungen und Länge der Pausen bei der „Berechnung“):

```
!
!
!
AAAAAAAAAAAAAAAAAAAAAAAA!
AAAAAAAAAAAAAAAAAAAAAAAA!
AAAAAAAAAAAAAAA!
!
!
BBBBBBBBBBBBBBBBBBBBBBB!
BBBBBBBBBBBBBBBBBBBBBBB!
BBBBBBBBBBBBBBBCCCCCCCCCCC!
CCCCCCCCCCCCCCCCCCCCCCCCC!
CCCCCCCCCCCCCCCCCCCCCCCCC!
CCCC!
!
!
```

**Fehlersuche**

Einige der gängigen Probleme sind im Folgenden zusammengefasst.

Kein Interrupt:

- ST, DBGU oder AIC falsch konfiguriert?
- Interrupts in CPRS noch maskiert?
- Interrupt-Handler falsch installiert?

Kein weiterer Interrupt:

- Interrupt an ST, DBGU oder AIC nicht richtig zurückgesetzt?
- Falscher Rücksprung aus Handler?

Sofort wieder ein Interrupt:

- Interrupt an ST, DBGU oder AIC nicht richtig zurückgesetzt?

Register im Anwendungsprogramm korrumpiert:

- Falscher Offset beim Rücksprung?
- Nicht alle Register ordentlich gesichert und wiederhergestellt?

Data Abort:

- Sprung an falsche Adresse?
- Falscher Offset beim Rücksprung?
- Nicht alle Register ordentlich gesichert und wiederhergestellt?
- IRQ während CPU im IRQ-Modus?

Neben diesen Fehlern werdet Sie evtl. feststellen, dass Interrupts auftreten, für die Sie keine Quelle ermitteln können – sog. spurious interrupts. Solange das nicht andauernd geschieht, ist das in Ordnung. Der umgekehrte Fall, dass es mehrere Auslöser für einen Interrupt gibt, ist aufgrund der geteilten Interrupt-Leitung auch normal. Falls Ihr Interrupt-Handler recht lange braucht, werden Sie das häufiger bemerken.

Es empfiehlt sich, die interne Peripherie am AIC als level-sensitive zu belassen und nicht als edge-triggered zu konfigurieren.

Der Rücksprung aus der Ausnahmebehandlung kann (bis auf LR-Offset) unter relativ kontrollierten Bedingungen am SWI geübt werden. Dazu sollten Sie allerdings den Supervisor-Modus verlassen, da Sie sonst evtl. das Link Register und das SPSR korrumpieren: bei einem SWI wird in den Supervisor- Modus gewechselt und LR_svc und SPSR_svc überschrieben.

Es wird ein Stück Assembler-Code zur Verfügung gestellt, welches hoffentlich recht verlässlich nicht gesicherte Register und falsche Rücksprung-Offsets entdeckt.