Bearbeitung des 3. Aufgabenblattes 5. Dezember 2022

Autoren: Elena Frank, Stefan Vesovic

3-1 (Threads)

- **gegenseitiger Ausschluss**
  Gegenseitiger Ausschluss bedeutet ein bestimmter Zugriff oder Programmteil nicht gleichzeitig von mehreren Threads ausgeführt werden kann, d.h. der Eintritt von einem Thread in den sogenannten "kritischen Abschnitt"den Ausschluss von einem anderen Thread von diesem Abschnitt bedeutet. Der andere Thread muss dann z.B. warten bis der erste Thread den Abschnitt wieder verlassen hat. Eine Möglichkeit zur Umsetzung von gegenseitigem Ausschluss is *Signalisieren.*

- **Signalisierung**
  Signalisierung zwischen Threads ermöglichst es zu Erzwingen dass bestimmte Operationen in einem Thread A stattgefunden haben bevor ein Thread B bestimmte andere Operationen ausführen kann. Umgesetzt wird es durch die Operationen *Signal* und *Wait*: ein Thread B wartet (*Wait*) an einer bestimmten Stelle im Programmablauf auf ein *Signal* von Thread A bevor es sein Programm weiter ausführt. Ein Anwendungsfall für Signale sind z.B. Locks wie sie zum *gegenseitige Ausschluss* genutzt werden: Ein Thread darf erst in einen kritischen Abschnitt eintreten wenn ein Signal zum ünlockërhalten wurde.

- **Synchronisation**
  Signalisierung kann auch genutzt werden um mehrere Threads zu synchronisieren, d.h. zu erzwingen dass alle Threads einen bestimmten Punkt in ihrem Programmablauf erreicht haben, bevor irgendeiner der Threads weiterlaufen darf. Dies passiert indem jeder Thread an dem SSynchronisationspunktërst eine *signal* und dann eine *wait* Operation ausführt. Konkret nutzt es ein sogenanntes *AND-signal* und *AND-wait*, d.h. **alle** Threads wartet auf das Signal von **allen** anderen Threads.

- **Koordination**
  Die Interaktion von Threads kann in Zwei Aspekte unterteilt werden. Einen temporären und einen funktionalen Teil. Dabei beschäftigt sich der temporäre Teil mit der Koordination. Koordination ist sehr wichtig, da sowohl Kommunikation als auch Kooperation darauf angewiesen sind. 2 Möglichkeiten Koordination zu ermöglichen ist Signalisierung und Synchronisation zu implementieren.

- **Kommunikation**
  Bei Kommunikation zwischen Threads werden explizit Daten zwischen ihnen transportiert. Dabei wird von einem data object eine send und eine receive Operation bereitgestellt. Über diese können ein Sender und ein Receiver miteinander Kommunizieren. Da beide zu jedem Zeitpunkt einen call machen können kann es dazu kommen das der Receiver zuerst anfragt und dann gesendet wird oder das der Sender zuerst sendet und dann der Receiver es sich holt. Bei Kommunikation wird zwischen asynchroner und synchroner Kommunikation unterschieden und es wird Koordination zwischen den Threads benötigt um dies zu ermöglichen.

- **Kooperation**
  Kooperation finder über shared areas statt auf welche die Threads zugriff auf geteilte Daten haben. Damit es zu keinen Fehlern kommt muss hier der Zugriff auf diese gemeinsamen Daten koordiniert werden. Kooperation von Threads ist ein gutes Beispiel für einen kritischen Abschnitt, welcher mit Hilfe von gegenseitigem Ausschluss bewältigt werden kann. Dafür können Locks, Monitore oder Semaphoren benutzt werden.

3-2 (Interrupts – System Timer und serielle Schnittstelle)

Git: `https://github.com/elenaf9/WS2022-betriebssysteme`
Branch: `assignment-3`

Interrupts werden von uns mit einem geteilten Handler behandelt, der die nötigen Register sichert und in den System-Mode wechselt (unter Mitnahme des IRQ stacks) um die eigentliche Fehlerbehandlung zu unternehmen. Davor geben wir die Interrupts im Prozessorstatus-register wieder frei.
Im Gegensatz zur Beispielausgabe auf dem übungsblatt ist es bei uns trotzdem nicht möglich dass ein System-timer-interrupt unseren dbug-interrupt handler (im system mode) unterbricht. Grund dafür ist, dass wir wie im `AT91RM9200.pdf`, *22.7.3.4 Interrupt Handlers* (Seite 246f) beschrieben das End of Interrupt Command Register (AIC_EOICR) erst schreiben wenn die Interrupt-Behandlung komplett abgeschlossen ist (also erst nach unserer "Berechnung"bei DBGU interrupts). Solange AIC_EOICR nicht geschrieben wurde können nur Interrupts mit **höherer** Priorität unseren Interrupt-Handler unterbrechen. Da System-Timer und DBGU jedoch beide auf Interrupt-Source 1 liegen und somit die selbe Priorität haben können sie sich nicht gegenseitig Unterbrechen.