Übung 3

3. Theoretische Fragen

0. Worum ging es dieses mal?

- Behandlung von Interrupts
- Programmierung der Clock
- präemptives Scheduling
- Erkennung und Schutz kritischer Abschnitte

1. Was ist präemptives Scheduling und wie kann es realisiert werden?

- CPU wird dem aktiven Prozess gewaltsam entzogen
  - bei uns: Prozesswechsel, wenn Zeitscheibe des Prozesses abgelaufen
- dafür notwendig:
  - Timer, Interrupts
- Probleme können in kritischen Abschnitten auftreten
  - dort darf sich zu jeder Zeit nur ein Prozess befinden
    (Stellen, an denen nebenläufige Aktivitäten auf gemeinsame Daten zugreifen)

2. Wie kann man eine quasiparallele Abarbeitung von Prozessen durch präemptives Scheduling erreichen?

- den Prozessen werden durch Interrupts CPU entzogen
- durch Clock tritt periodisch ein Interrupt auf
  -> Jeder Prozess ist genau t Millisekunden an der Reihe
  -> also ist der erste Prozess eben diese t Millisekunden dran
  -> danach kommt der nächste Prozess dran, der erneut t Millisekunden die CPU besitzt und diese danach wieder abgeben muss
  -> nachdem der letzte Prozess zum ersten Mal die CPU abgibt (wieder an den ersten), haben alle Prozesse quasi den gleichen Stand abgearbeitet, eben die t Millisekunden

3. Was wäre ein passendes Beispiel für einen nebenläufigen Zugriff auf gemeinsame Daten?

- in Costubs: die Readyliste des Schedulers
- wir haben nur die eine Readyliste, die von mehreren Stellen aus manipuliert wird (reschedule i.d.R.)

4. Was ist ein Programmable Interval Timer(PIT)?

- ist ein Timerbaustein auf dem Mainboard
- erzeugt in regelmäßigen Zeitabständen Signale

5. Wie funktioniert der PIT?

- kann mit Hilfe von 4 Ports angesprochen werden
- wir benutzen Zähler 0 -> 0x40
- Steuerregister liegt auf 0x43 (beide Ports für PIT 1, PIT 2 brauchen wir nicht)

- PIT muss über Steuerwort mitgeteilt werden, was man von ihm will (Größe von 8 Bit)
- Bit 0:
  - Zählformat
  - wir brauchen binäre Zählung von 16 Bit -> 0
- Bit 1 - 3:
  - Modus 0 bis 5
  - wir brauchen Modus 2 -> 010
- Bit 4 - 5:
  - Lesen / Schreiben
  - wir brauchen niederwertiges, dann höherwertiges Bit -> 11
- Bit 6 - 7:
  - Zählerauswahl
  - wir nutzen Zähler 0 -> 00
- daraus ergibt sich unser Steuerwort: 0011 0100

- Modus bestimmt, wie Zähler arbeitet und ob externe Ereignisse ausgelöst werden (durchOUTx Leitung)
- Bsp.:
  - Modus 0
    - es wird vom angegebenen Startwert bis 0 heruntergezählt (838 ns pro Schritt)
    - setzt am Ende OUTx Leitung auf 1
  - Modus 2
    - zur Erzeugung von periodischen Impulsen -> das was wir brauchen
    - es wird vom angegebenen Startwert bis 0 heruntergezählt
    - dann kurzer Impuls auf OUTx ausgegeben
    - Zähler dann wieder automatisch mit Startwert initialisiert und es geht von vorne los

6. Wie kann man diesen konfigurieren (Konfigurationssequenz)?

- unsere Konfigurationssequenz ist in PIT::interval(int us)
- dort wird die Intervalldauer, die durch den PIT realisiert wird, initialisiert
- Umrechnung von us, was Mikrosekunden sind in Nanosekunden
  - müssen wir machen, da 1 Schritt = 838 Nanosekunden
- um den Startwert zu erhalten, rechnen wir nun die Eingabe in Nanosekunden / Schrittdauer in Nanosekunden
  -> Startwert, der auf die 2 Bytes des DATA_PORT geschrieben wird (mit Hilfe von Bitverschiebung)

7. Was ist ein Interrupt?

- asynchrone Unterbrechung
- werden von Geräten ausgelöst
- sind ein Mechanismus, um von Hardware aus den Aufruf von Programmen zu verlassen
    - Brücke zwischen Hard- und Software
- Geräte können somit mit ihren Steuerprogrammen (Gerätetreibern) kommunizieren
    - um Anfänge/Beendigung/Fehler von Geräteoperationen anzuzeigen
    - um Datentransfers durch den Treiber zu initiieren / Unterstützung durch den Treiber anfordern
- können prinzipiell jederzeit auftreten
    - parallel zur CPU agierende Geräte sind Verursacher
- sind nicht vorhersagbar
    - Interrupts müssen für die unterbrochenen Programme transparent sein
      (darf vom Programm nicht wahrgenommen werden -> Ausgabe bleibt trotz Interrupts unverändert)

8. Wie kann die CPU Interrupts unterscheiden?

- jeder Interrupt hat eine eindeutige Nummer
- diese wird in Vektortabelle gespeichert -> über Indizierung Zugriff möglich

9. Wie werden Interrupts behandelt?

1. Kontextwechsel
    - unterbrochener Prozess => Behandlungsroutine
    - Wechsel zu Systemmodus / Kernel-Stack
    - internen Prozesszustand sichern
    - Rücksprungadresse sichern
    - Statusregister sichern
2. Behandlung der Unterbrechung
    - Aufruf der Behandlungsroutine
3. Kontextwechsel
    - Behandlungsroutine => unterbrochener Prozess
    - Statusregister wiederherstellen
    - Rücksprungadresse wiederherstellen
    - internen Prozesszustand wiederherstellen
    - Wechsel zu Nutzermodus / User-Stack

- Phase 1 und Phase 3 werden von der Hardware durchgeführt
- Phase 2 wird von Software durchgeführt

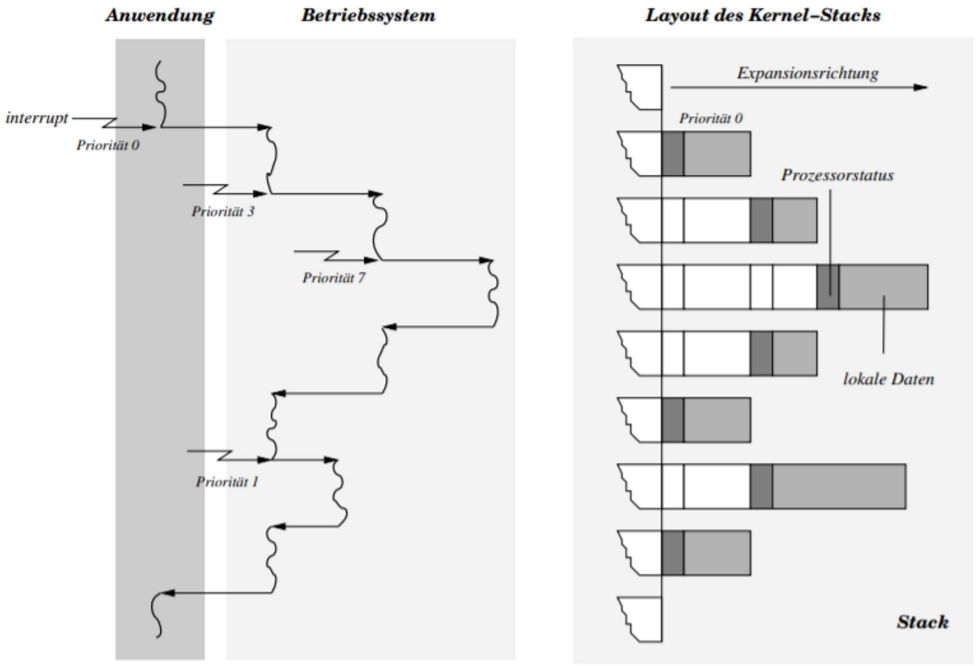
- in Costubs haben wir nur den Systemmodus -> keine Unterscheidung hier zwischen Systemmodus und Nutzermodus nötig
  -> es muss nicht zum Kernel-Stack gewechselt werden (da wir schon darauf arbeiten)

10. Können neue Interrupts während einer Interrupsbehandlung auftreten?

- ja

11. Wenn ja, werden diese dann umgehend behandelt?

- wenn während einer Interruptbehandlung ein neuer Interrupt mit einer höheren Priorität auftritt -> direkter Wechsel zu diesem Interrupt (mit höherer Priorität)



12. Wenn nein, werden die blockierten Interrupts durch das Gerät später nochmal ausgelöst?

13. Zwei Prozesse A und B sollen in einer Endlosschleife immer wieder ihren Namen ausgeben. Diese Ausgabe dauert 10 Takte. Wenn Prozess A eine Zeitscheibe von 10 Takten und Prozess B eine Zeitscheibe von 100 Takten zugewiesen wird, welche Ausgaben sind zu erwarten?

- Ausgabe des Namens dauert 10 Takte
- Erwartete Ausgabe:
  A, B, B, B, B, B, B, B, B, B, A, B, B,
- da die Zeitscheibe von A nur 10 Takte umfasst, wird diese nach einmaliger Namensausgabe aufgebraucht sein
- dann wird zum Prozess B gewechselt
- dessen Zeitscheibe umfasst 100 Takte, wodurch die Namensausgabe hier 10 mal stattfindet
- danach wird wieder zum Prozess A gewechselt und es beginnt von vorne

(Nicht mehr relevant für Abnahme)
Berechnungen zum PIC
intervall = 10 000 Mikrosekunden = 10 Millisekunden = 0,01 Sekunden = 10 000 000 Nanosekunden
TimeBase = 838 Nanosekunden = 0,838 Mikrosekunden

1 Herz = 1 / s
1 MHz = 1 000 000 / s

interruptcount = 10 000 000 Nanosekunden * 838 Nanosekunden = 8 380 000 000 Nanosekunden
= 8380000 Mikrosekunden = 8380 MilliSekunden = 8,38 Sekunden

838 Nansekunden pro Schritt;

20 Millisekunden = 20 000 000 Nanosekunden

Startwert = 20
Schritt = 0,1 Sekunde

Takt = Startwert * Schritt

-> Wir brauchen 20 Schritte -> 2 Sekunden

Takt = Startwert * Schritt

20 000 000 Nanosekunden = Startwert * 838 Nanosekunden

Startwert = 20 000 000 / 838 = 23866

Interrupt alle 20 Millisekunden = 0,02 Sekunden

20 Millisekunden * 50 = 1 000 Millisekunden = 1 Sekunden

Clock clock(3000); -> 3000 Mikrosekunden

nanoSekundenEingabe = 3000 * 1000 = 3000000

interruptCount = 3000000 / 838 = 3579,9523 = Startwert = 3580

3580 * 838 = 3000040