2. Übungszettel

Felix Suhl, Leander Tolksdorf

Aufgabe 1

1)

Echtzeitbetriebssysteme müssen Interrupts immer in einer vorher bekannten Zeit abarbeiten (sie sind "deterministisch"). Dafür müssen Interrupts präemptiv und prioritätsbasiert ausgeführt werden.

Der "Unfall"-Interrupt, der den Airbag auslöst, hat eine viel höhere Priorität als beispielsweise der "Fensterheber-Interrupt", wird also sofort ausgeführt, auch wenn dadurch andere Interrupts unterbrochen werden.

Eine weitere Möglichkeit, die Latenz bei kritischen Interrupts zu verringern wäre, die ISR zu verkürzen, indem bspw. keine Register gespeichert werden. Dadurch gehen zwar Daten verloren, aber der Airbag löst im Zweifelsfall schneller aus.

2)

Bei einem normalen Systemcall wechselt die CPU immer in den Kernel-Mode. Dafür müssen u.A. Register gesichert und wiederhergestellt werden, was relativ aufwändig ist. Für wiederkehrende Operationen (wie z.B. das Abrufen der Uhrzeit) reicht es aber, den entsprechenden Teil des Kernelbereichs auszulesen. Dafür gibt es virtuelle Systemcalls, die im User-Mode laufen, für die also kein Context switch notwendig ist.

Dafür bieten sich aber nur Read-only-Operationen an, da im User-mode nicht in den Kernel-Space geschrieben werden darf. Bzw. das OS nichts machen kann wenn nur gelesen wird.

3)

- mehr Fokus: Bei kooperativem Multitasking kann der CPU seine Kapazitäten auf einzelne Prozesse fokussieren, wohingegen präemptives Multitasking fortlaufend Prozesse unterbricht und andere rechenbereite Prozesse weiterführt.
- früher weniger Eintrittsvarianzprobleme: Da Programme selber entscheiden, wann sie unterbrochen werden, und da es früher keine Mehrkernprozessoren gab, waren Programme eher eintrittsvariant/wiedereintrittsfähig. Beispielsweise indem sie vor der Unterbrechung globale Variabeln wiederherstellen.
- weniger Overhead: Bei präemptivem Multitasking ist der Overhead des Prozess-Wechselns größer, weil kooperatives Multitasking den Prozesswechsel Programmen überlässt. Diese wechseln an bestimmten Stellen und so weniger häufig als ein Hardware-Timer.

4)

Im Process Control Block eines Prozesses müssen stehen:

- Identifikation (PID), ggfs. PIDs der Kindprozesse
- Inhalte der 'general purpose'-CPU-Register
- Stack-Pointer
- Program Counter

- Infos über belegten Primär- und Sekundärspeicher
- Status ('ready', 'running'...)
- Priorität
- Zustände von geöffneten Dateien, Ein-/Ausgabegeräten und anderen relevanten Einheiten

Richtig, aber auch abhängig vom System.

5)

Zum Beispiel, wenn innerhalb eines Programmes verschiedene Aufgaben übernommen werden müssen, diese aber von denselben Daten abhängen. Prozesse zu erstellen ist einerseits teuer. Außerdem teilen Threads eines Prozesses denselben Adressraum, sie können untereinander auf dieselben Daten innerhalb ihres Speicheraddressraums zugreifen und sie manipulieren, ohne dafür kostspielige IPC-Methoden zu nutzen.
Ein weiterer Nachteil bei Prozesses ist die Synchronisierung. Prozesse müssen hierfür meist teure System Calls aufrufen, während Threads Kontrollvariabeln innerhalb des gemeinsamen Heaps beobachten und sich so synchronisieren können.

2/3

Aufgabe 2

Sequential Interrupt Processing

Annahme: Gleichzeitig eintreffende Interrupts werden nach *Highest Priority First* abgearbeitet.

Zeitslot	Interrupt
0	-
1	I0 1/4
2	I0 2/4
3	I0 3/4
4	I0 4/4
5	I2 1/2 Hier müsste I3 laufen (höhere Priorität!!!)
6	I2 2/2
7	I1 1/1
8	I3 1/3
9	I3 2/3
10	I3 3/3
11	I4 1/3
12	I4 2/3
13	I4 3/3
14	I5 1/2
15	I5 2/2
16	I6 1/1
17	-

Ablauf I0 -> I3 -> I5 -> I2 -> I4 -> I1 -> I6

Nested Interrupt Processing

Annahme: Gleichzeitig eintreffende Interrupts werden nach *Highest Priority First*, bei gleicher Priorität nach *First Come First Serve* abgearbeitet.

Zeitslot	Interrupt	Pausierte Interrupts	Pending Interrupts
0	-		
1	I0 **1** 1/4		
2	I2 **2** 1/2	I0 **1** 1/4	I1 **1** 0/1
3	I3 **3** 1/3	I0 **1** 1/4 - I2 **2** 1/2	I1 **1** 0/1
4	I3 **3** 2/3	I0 **1** 1/4 - I2 **2** 1/2	I1 **1** 0/1
5	I3 **3** 3/3	I0 **1** 1/4 - I2 **2** 1/2	I1 **1** 0/1 - I4 **2** 0/3
6	I5 **3** 1/2	I0 **1** 1/4 - I2 **2** 1/2	I1 **1** 0/1 - I4 **2** 0/3
7	I5 **3** 2/2	I0 **1** 1/4 - I2 **2** 1/2	I1 **1** 0/1 - I4 **2** 0/3 - I6 **1** 0/1
8	I2 **2** 2/2	I0 **1** 1/4	I1 **1** 0/1 - I6 **1** 0/1
9	I4 **2** 1/3	I0 **1** 1/4	I1 **1** 0/1 - I6 **1** 0/1
10	I4 **2** 2/3	I0 **1** 1/4	I1 **1** 0/1 - I6 **1** 0/1
11	I4 **2** 3/3	I0 **1** 1/4	I1 **1** 0/1 - I6 **1** 0/1
12	I0 **1** 2/4		I1 **1** 0/1 - I6 **1** 0/1
13	I0 **1** 3/4		I1 **1** 0/1 - I6 **1** 0/1
14	I0 **1** 4/4		I1 **1** 0/1 - I6 **1** 0/1
15	I1 **1** 1/1		I6 **1** 0/1
16	I6 **1** 1/1		

Das Interrupt-Interrupts Diagramm ist korrekt! :)

2/3