

# Betriebssysteme 1

Foliensatz G, Kap. 7 (Teil 1: SKA 28–34)

Prof. Dr. Hans-Georg Eßer

Sommersemester 2022

v3.0 – 23.06.2021

PS/2-Tastatur: IBM Model M



Bild: [https://commons.wikimedia.org/wiki/File:IBM\\_Model\\_M.png](https://commons.wikimedia.org/wiki/File:IBM_Model_M.png)

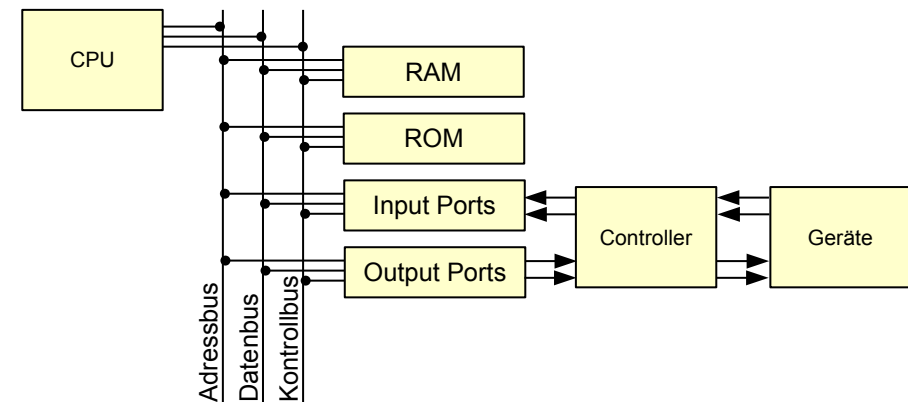
## SKA 28

### 28. Aktives vs. passives Warten

Nennen Sie je ein Beispiel für *aktives Warten* und *passives Warten* und erläutern Sie, warum aktives Warten in der Regel die schlechtere Wahl ist.

Beispiel in beiden Fällen:  
Tastaturtreiber, Polling vs. Interrupt

### • Klassisch: Kommunikation über Ports



## Tastatur am PC (3)

- PS/2-Keyboard-Controller hat zwei solche Ports

```
#define KBD_DATA_PORT      0x60
#define KBD_STATUS_PORT    0x64
```

- Kommunikation über Assembler-Befehle

inb (liest Wert aus Port, speichert in Reg.),  
outb (schreibt Reg.-Inhalt auf Port)

## Tastatur am PC (4)

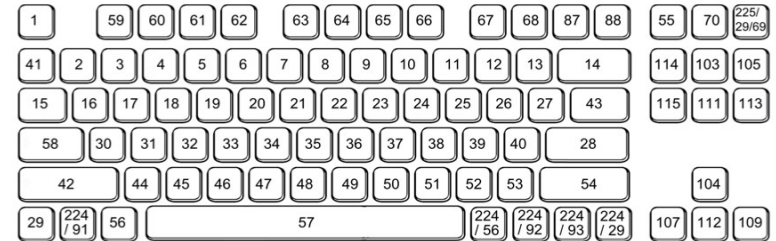
- „Events“ (Tastendruck oder Loslassen) generieren Scancodes
- Beispiel, Taste „A“:
  - Drücken = Scancode 30
  - Loslassen = Scancode 30 + 128 = 158
- Scancodes über Datenport (0x60) des Keyboard-Controllers auslesen:  
char scancode = inb (KBD\_DATA\_PORT);
- Statusport verrät, ob Event vorliegt

## Tastatur am PC (5)

Belegung  
(US-englisch)



Scancodes  
(„key press“)



## Tastatur am PC (6) – Polling = aktives Warten

```
do forever {
    // warte auf Event
    while (inb (KBD_STATUS_PORT) != NEW_EVENT) ;

    // lies Scancode
    scancode = inb (KBD_DATA_PORT);

    if (scancode < 128) {
        // nur keypress event verarbeiten
        ascii = lookup_table (scancode);
        printf ("Zeichen %d eingegeben\n", ascii);
    }
}
```

- Interrupt tritt auf
- Laufender Prozess wird (nach aktuellem Befehl) unterbrochen, BS übernimmt Kontrolle
- BS speichert Daten des Prozesses (wie bei Prozesswechsel → Scheduler)
- BS ruft Interrupt-Handler auf
- Danach (evtl.): Prozess-Fortsetzung (evtl. ein anderer Prozess)

## 29. Temperatursensor

Sie konzipieren für ein Embedded-System, das in einem mobilen, Akku-betriebenen Temperatursensor für den Außenbereich arbeitet, die Steuerungselektronik. Das Gerät soll alle 10 Minuten die Temperatur messen und via WLAN an einen Server funken. Die Hauptplatine besitzt einen Timer-Chip, den Sie auf einen 10-Minuten-Rhythmus programmieren können, und Sie haben herausgefunden, dass die Code-Zeile

```
for (long int i = 0; i < 3627458763L; i++) z = i;
```

auch genau zehn Minuten Zeit benötigt.

Um zwischen zwei Messungen je zehn Minuten zu warten, können Sie also zwischen zwei möglichen Implementierungen wählen. Welche wählen Sie und warum?

## Interrupt-Handler für Tastatur

```
void keyboard_handler() {
    // lies Scancode
    scancode = inb (KBD_DATA_PORT);

    if (scancode < 128) {
        // nur keypress event verarbeiten
        ascii = lookup_table (scancode);
        printf ("Zeichen %d eingegeben\n", ascii);
    }
}
```

- Kein Code mehr, der auf Tastendruck wartet
- Handler muss in Tabelle eingetragen werden, Interrupts aktiviert werden

## SKA 30

## 30. Kernel Mode

Warum laufen Interrupt-Handler im Kernel Mode und nicht im User Mode?

## User Mode vs. Kernel Mode (1)

Anwendungen laufen im nicht-privilegierten User Mode

Beispiel: Intel

- Ring 0 = Betriebssystem (Kernel Mode)  
→ Vollzugriff auf die Hardware
- Ring 3 = Anwendung (User Mode)

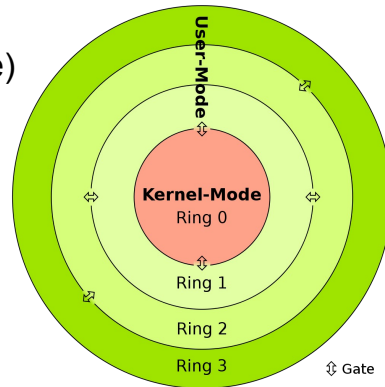
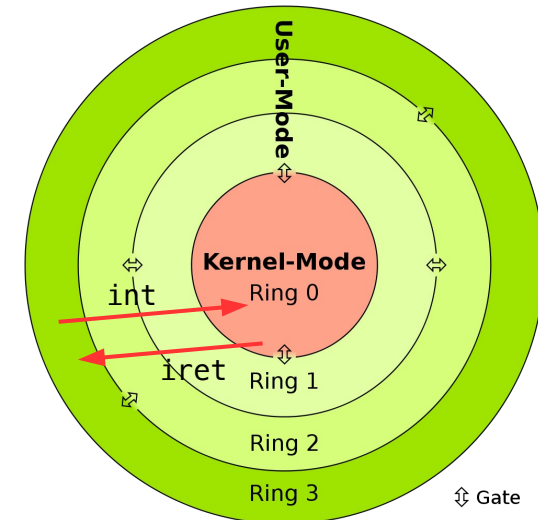


Bild: [http://commons.wikimedia.org/wiki/File:CPU\\_ring\\_scheme.svg](http://commons.wikimedia.org/wiki/File:CPU_ring_scheme.svg)  
(Autor: <http://commons.wikimedia.org/wiki/User:Sven>; GNU Free Documentation License)

## User Mode vs. Kernel Mode (3)



## User Mode vs. Kernel Mode (2)

- Problem:
  - Daten und Code des Betriebssystems sollen vor Zugriff durch Anwendungen geschützt sein,  
**aber:**
  - Anwendungen müssen Betriebssystem-Funktionen nutzen, um z. B. auf Datenträger zuzugreifen.
- System Calls: kontrollierter Übergang vom User Mode in den Kernel Mode (int 0x80, syscall etc.)
- Interrupt-Handler: Hardware-Zugriff braucht auch Kernel-Mode

## SKA 31

### 31. Interrupt Handler vs. Signal Handler

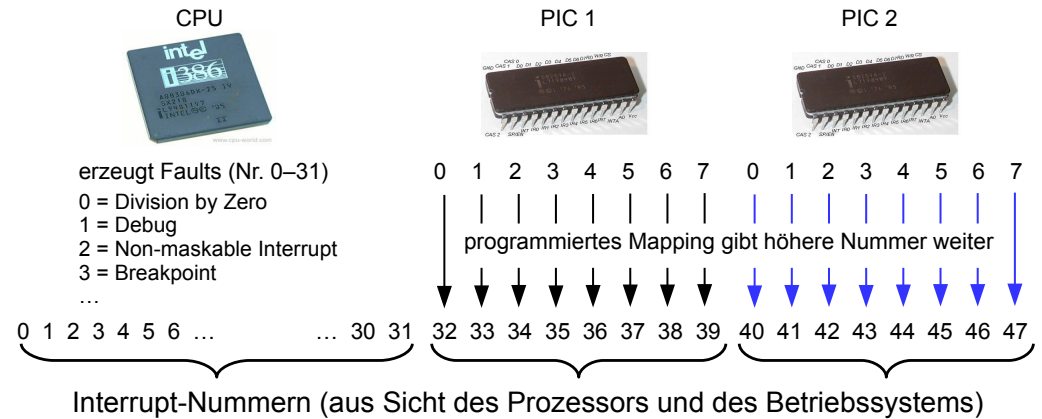
Nennen Sie Gemeinsamkeiten und Unterschiede zwischen den in diesem Kapitel beschriebenen Interrupt Handlern und den in Kapitel 5.5.3.2 beschriebenen Signal Handlern.

## 32. PIC-Kaskade

Warum ist in klassischen PCs beim Kaskadieren von zwei PICs (Programmable Interrupt Controllers) eine Umprogrammierung der an die CPU gemeldeten Interrupt-Nummern notwendig?

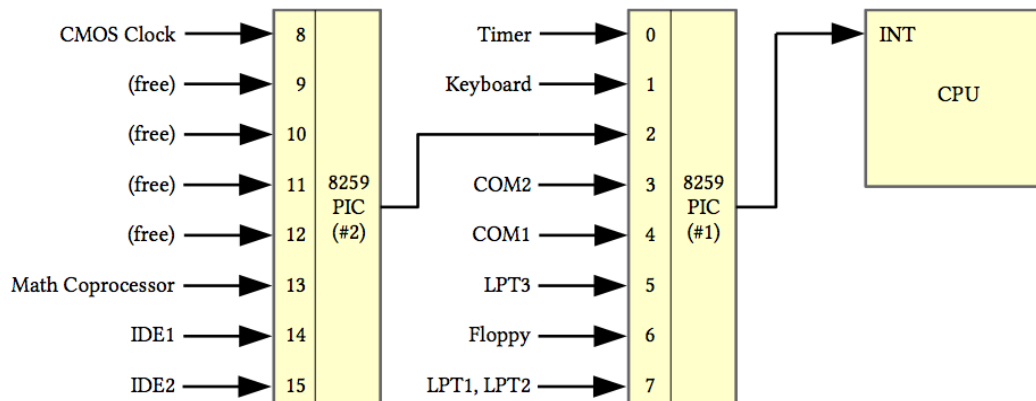
## Intel 8259 ist programmierbar

Bildquellen:  
IDE Controller: <http://www.ebay.de/usr/sm-pc>,  
8259A: <http://www.brokenhorn.com/Resources/OSDevPic.html>,  
I386: <http://www.cpu-world.com/CPUs/80386/Intel-A80386DX-25%20IV.htm>



## PIC-Kaskade (1)

Klassischer PIC (**Programmable Interrupt Controller**) Intel 8259 hat acht Eingänge → Kaskadierung von zwei PICs



## SKA 33: Mehrfach-Interrupts

## 33. Mehrfach-Interrupts

Fassen Sie kurz die Vor- und Nachteile der drei Interrupt-Be-handlungsmodelle *A* (Interrupts gesperrt, während Handler läuft), *B* (jeder Interrupt unterbricht den laufenden Handler) und *C* (Priorisierung der Interrupts) zusammen.

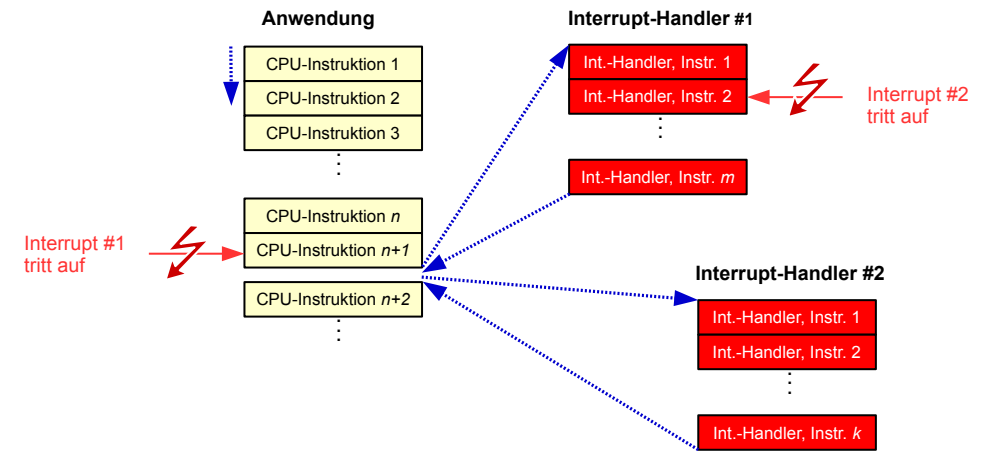
## Mehrfach-Interrupts (1)

### Drei Möglichkeiten

- Während Abarbeitung eines Interrupts alle weiteren ausschließen (DI, disable interrupts)  
→ Interrupt-Warteschlange
- Während Abarbeitung andere Interrupts zulassen
- Interrupt-Prioritäten: Nur Interrupts mit höherer Priorität unterbrechen solche mit niedrigerer

## Mehrfach-Interrupts (3)

### Variante 1



## Mehrfach-Interrupts (2)

### Variante 1

- Alle Interrupts „gleichwertig“, keine Prioritäten
- zu Beginn einer Int.-Routine alle Interrupts abschalten  
→ kein Interrupt unterbricht einen anderen

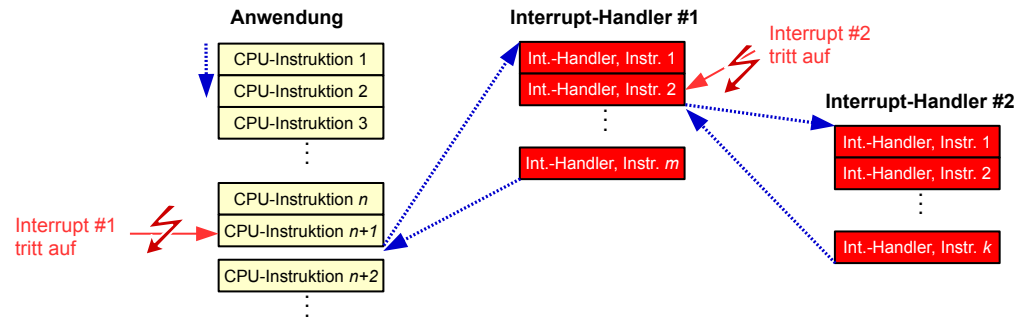
## Mehrfach-Interrupts (4)

### Variante 2

- Interrupt-Handler können unterbrochen werden
- Rücksprung in vorherigen Interrupt-Handler
- Zustand sichern!

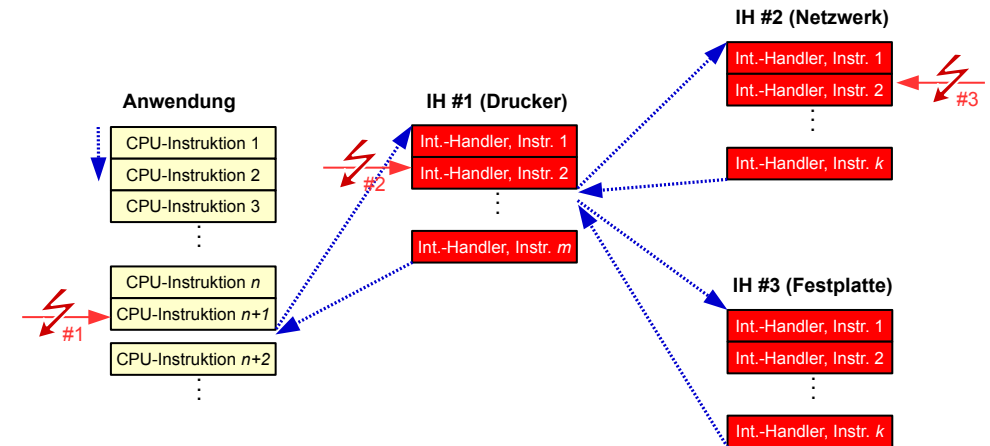
## Mehrfach-Interrupts (5)

### Variante 2



## Mehrfach-Interrupts (7)

### Variante 3



## Mehrfach-Interrupts (6)

### Variante 3

- Interrupts haben Prioritäten, z. B. Netzwerkkarte > Drucker
- Interrupt mit hoher Priorität unterbricht Interrupt mit niedrigerer Priorität

## Mehrfach-Interrupts (8)

- Problem bei gesperrten Interrupts: Behandlung muss schnell erfolgen
- Lösung: Aufteilung des Interrupt-Handlers in zwei Komponenten → **SKA 34**



**34. Top half und bottom half**

Welchen Vorteil bietet die Aufteilung eines Interrupt-Handlers in eine *top half* und eine *bottom half*, wie es viele Betriebssysteme erlauben?

- Interrupt handler
- startet sofort, erledigt zeitkritische Dinge
- bestätigt (der Hardware) den Erhalt des Interrupts, setzt Gerät zurück etc.
- erzeugt bottom half
- Alles andere → bottom half

- längere Berechnungen, die zur Interrupt-Verarbeitung gehören
  - dabei sind Interrupts zugelassen
- Tasklet ist kein Prozess, läuft direkt im Kernel