

| Adr. | IRQ | Prior. | Komponente |
|------|--------|--------|---|
| 08H | IRQ 0 | 16 | Timer (Motherboard) |
| 09H | IRQ 1 | 15 | Tastatur |
| 0AH | IRQ 2 | 14 | IRQ-Kaskadierung - IRQ 9 |
| 0BH | IRQ 3 | 5 | COM # 2 IRQ 3 COM # 2 |
| 0CH | IRQ 4 | 4 | COM # 1 IRQ 4 COM # 1 |
| 0DH | IRQ 5 | 3 | Festplatten-Controller, LPT 2 |
| 0EH | IRQ 6 | 2 | Diskettenlaufwerk-Controller |
| 0FH | IRQ 7 | 1 | LPT 1 |
| 70H | IRQ 8 | 13 | Echtzeituhr (Motherboard) |
| 71H | IRQ 9 | 12 | Kaskade zum IRQ 2, meist frei, reserviert |
| 72H | IRQ 10 | 11 | frei verfügbar |
| 73H | IRQ 11 | 10 | frei verfügbar |
| 74H | IRQ 12 | 9 | frei verfügbar oder PS/2 Mausport |
| 75H | IRQ 13 | 8 | NMU (Coprozessor) ab 486 on Board |
| 76H | IRQ 14 | 7 | Festplatten-Controller IDE Kanal # 1 |
| 77H | IRQ 15 | 6 | Festplatten-Controller IDE Kanal # 2 |
| 78H | IRQ 16 | 5 | Festplatten-Controller IDE Kanal # 3 |
| 79H | IRQ 17 | 4 | Festplatten-Controller IDE Kanal # 4 |

Unterbrechungen

Unterbrechungen (Interrupts)

Auftretende Ereignisse (Mausbewegungen, etc.) sollten jederzeit vom Betriebssystem erkannt und bearbeitet werden können. Auf der CPU^{*)} kann jedoch nur ein Prozess zur Zeit laufen. Wie aber sollen Ereignisse behandelt werden, wenn gar kein Betriebssystemprozess aktiv ist?

Mögliche Lösung: Polling

- CPU unterbricht laufende Prozess regelmäßig und prüft, ob Ereignisse vorliegen.
- Ineffizient, weil während der Prüfphase der Prozess ruht, selbst wenn keine Ereignisse anliegen.

^{*)} Wir gehen der Einfachheit halber von nur einer CPU mit nur einem Kern aus

Unterbrechungen (Interrupts)

Bessere Lösung:

1. für den laufenden Prozess **Unterbrechung anfordern**
(*Interrupt request, IRQ*)
2. in der Folge eine **Betriebssystemfunktion aufrufen**,
um das Ereignis zu behandeln (*Interrupt service routine, ISR* bzw. *Interrupt Handler, IR*)
3. zum vorherigen Prozess **zurückkehren**

Im Vergleich zum Polling wesentlich effizienter, weil nur dann unterbrochen wird, wenn Ereignisse anliegen. Hierfür ist jedoch Unterstützung durch Hardware und erhöhter Verwaltungsaufwand notwendig.

Interrupts – Klassifikation nach Quelle

Hardwareinterrupts

- Ereignisquelle: **Hardware**
- Signalisierung eines Ereignisses von Controller^{*)} an CPU durch speziell dafür vorgesehene **Signalleitung**
- **externes** Ereignis
- nicht reproduzierbar, **asynchron** zum Programmablauf

Softwareinterrupts

- Ereignisquelle: **Software**
- ausgelöst durch aktuell auszuführenden **Befehl**
- **internes** Ereignis
- reproduzierbar, **synchron** zum Programmablauf

^{*)} *Controller* sind Hardwarebausteine, die eine bestimmte Gerätekategorie ansteuern können.
Beispiel: USB Controller für den USB Datenbus, SATA-Controller für SATA Festplatten

Interrupts – Klassifikation nach Quelle

Hardwareinterrupts

■ Beispiele:

- Bewegen der Maus
- Schreiben eines Blocks auf Festplatte abgeschlossen
- Nachricht aus Netzwerk empfangen
- Timer abgelaufen (wichtig für Mehrprozessbetrieb!)

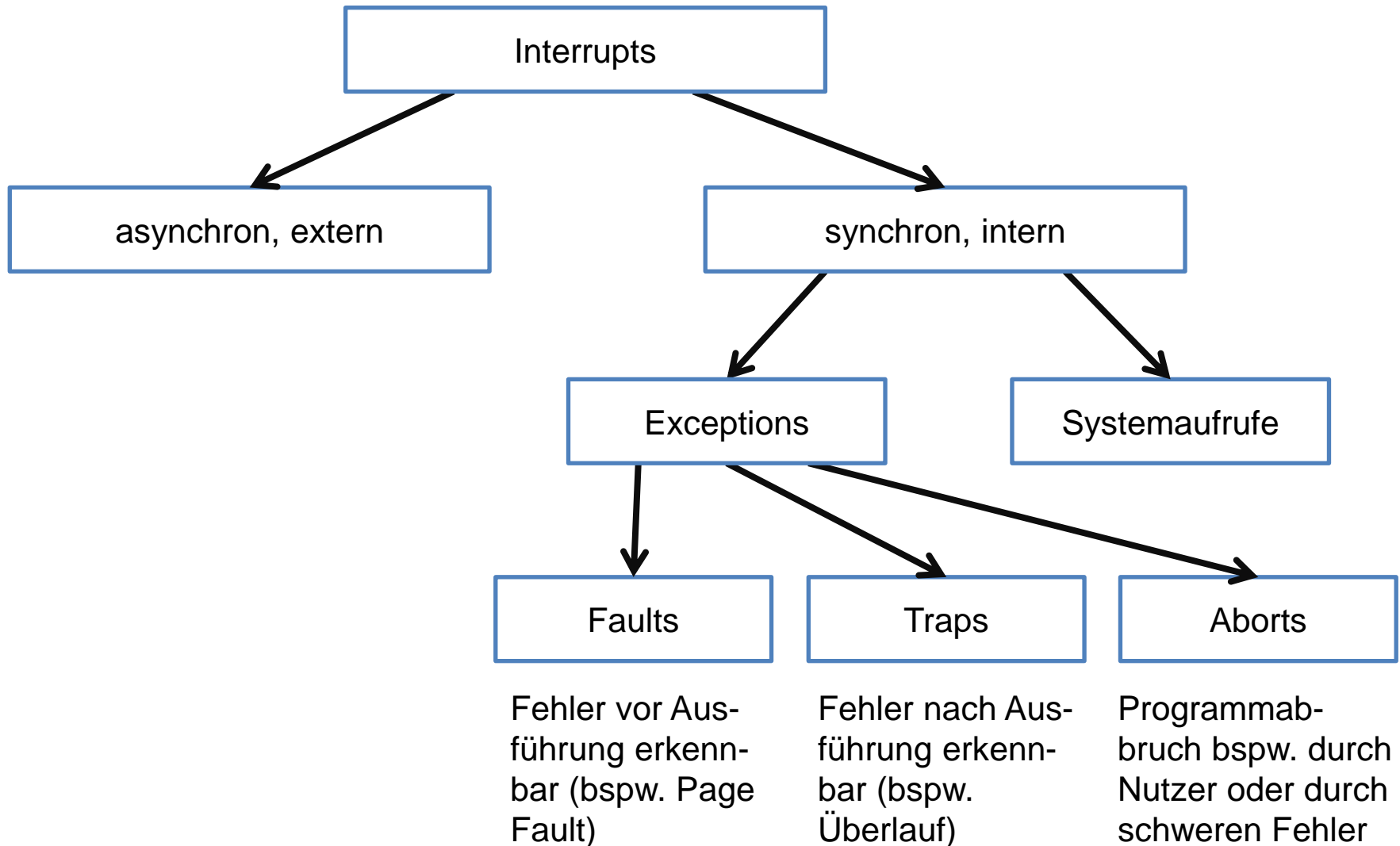
Softwareinterrupts

■ Beispiele:

- Division durch Null
- Unerlaubter Speicherzugriff
- Systemaufrufe ^{*)}

^{*)} Wir klären noch, was Systemaufrufe sind

Interrupts – Etwas feinere Klassifikation



Unterbrechungsbehandlung

Wenn ein Unterbrechungssignal anliegt (bei Hardware-Interrupts) bzw. der aktuell auszuführende Befehl einen Softwareinterrupt auslöst, wird ...

1. der **Zustand** des laufenden Prozesses **gesichert**,
2. in den **Kernel-Mode** gewechselt,
3. bei Bedarf die Behandlung weiterer **Unterbrechungsanforderungen gestoppt** (maskiert),
4. anhand der Quelle des Interrupts die passende **Behandlungsroutine** identifiziert und gestartet,

Unterbrechungsbehandlung

Wenn ein Unterbrechungssignal anliegt (bei Hardware-Interrupts) bzw. der aktuell auszuführende Befehl einen Softwareinterrupt auslöst, wird ...

5. die **Behandlungsroutine** mit einem speziellem Kommando (*return from interrupt, RTI*) **beendet**,
6. die **Sperre** für eventuell geblockte Unterbrechungsanforderungen wieder **aufgehoben** (siehe Punkt 3),
7. im Falle weiterer anliegender Interrupts **zurück zu 3.** gesprungen,
8. der **Zustand** des unterbrochenen Prozesses **wiederhergestellt**,

Unterbrechungsbehandlung

Wenn ein Unterbrechungssignal anliegt (bei Hardware-Interrupts) bzw. der aktuell auszuführende Befehl einen Softwareinterrupt auslöst, wird ...

9. zurück in den **User-Mode** gewechselt,
10. die **Ausführung** des unterbrochenen Prozesses an der Stelle der Unterbrechung **fortgeführt**.

Auswahl der Unterbrechungsbehandlung

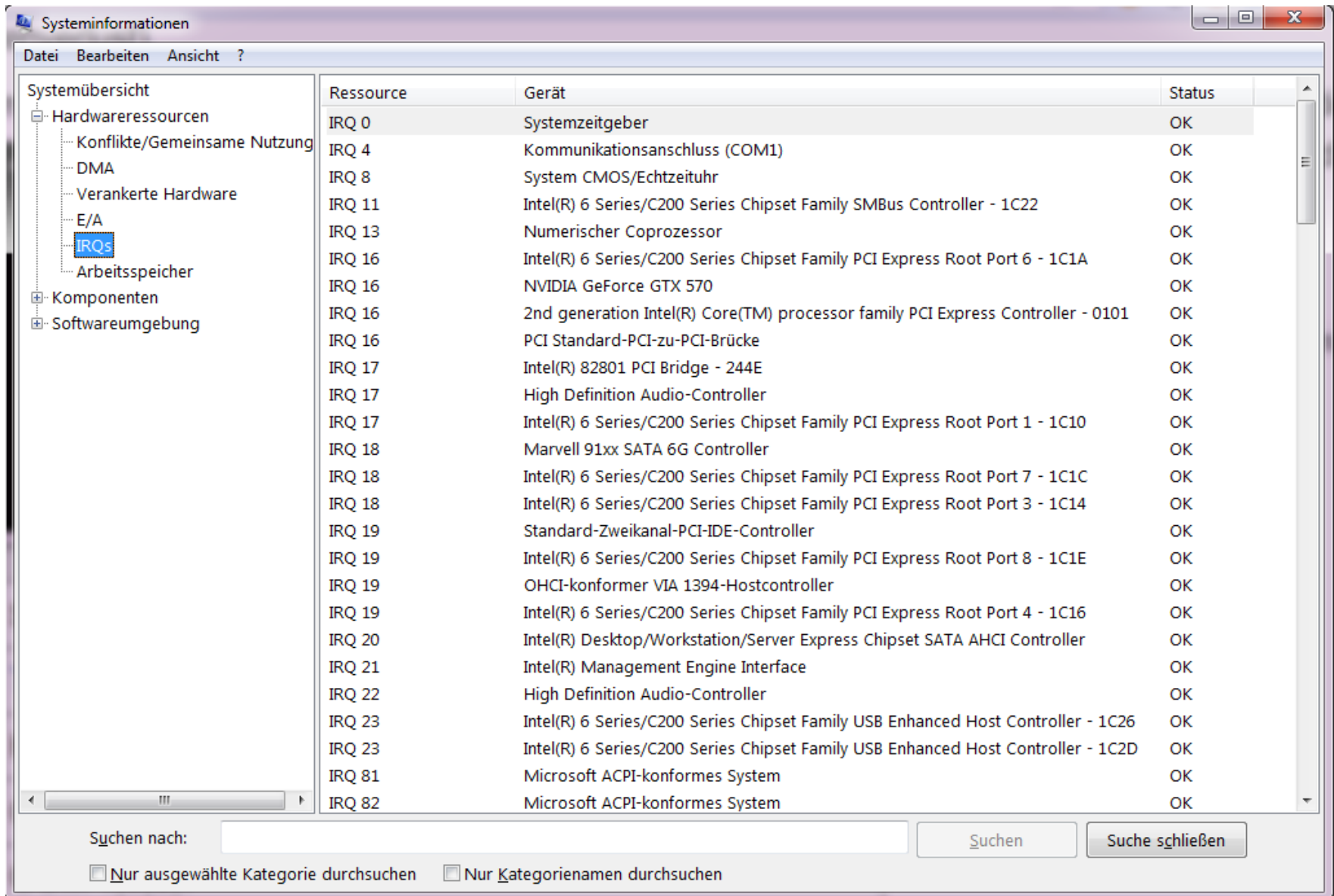
Wenn ein Unterbrechungssignal anliegt (bei Hardware-Interrupts) bzw. der aktuell auszuführende Befehl einen Softwareinterrupt auslöst, wird ...

1. der Zustand des laufenden Prozesses gesichert,
2. bei Bedarf die Behandlung weiterer Unterbrechungsanforderungen gestoppt (maskiert),
- 3. anhand der Quelle des Interrupts die passende Behandlungsroutine identifiziert und gestartet,**
4. die Behandlungsroutine mit einem speziellem Kommando (*return from interrupt, RTI*) beendet,

Auswahl der Unterbrechungsbehandlung

- Bei der Interruptanforderung durch Hardware (*IRQ*) wird eine Quellenkennung (positive Ganzzahl) mitgeliefert.
- Diese identifiziert in der sog. Interruptvektortabelle (*IVT*) die aufzurufende Betriebssystemfunktion, die den Interrupt behandelt.
- Bei Softwareinterrupts wird analog vorgegangen.

Auswahl der Unterbrechungsbehandlung – Beispiel



The screenshot shows the 'Systeminformationen' (System Information) window in Windows. The left-hand navigation pane is expanded to 'Hardwareressourcen' (Hardware Resources), and the 'IRQs' (Interrupt Requests) section is selected. The main pane displays a table of system resources and their assigned IRQs.

| Ressource | Gerät | Status |
|-----------|--|--------|
| IRQ 0 | Systemzeitgeber | OK |
| IRQ 4 | Kommunikationsanschluss (COM1) | OK |
| IRQ 8 | System CMOS/Echtzeituhr | OK |
| IRQ 11 | Intel(R) 6 Series/C200 Series Chipset Family SMBus Controller - 1C22 | OK |
| IRQ 13 | Numerischer Coprozessor | OK |
| IRQ 16 | Intel(R) 6 Series/C200 Series Chipset Family PCI Express Root Port 6 - 1C1A | OK |
| IRQ 16 | NVIDIA GeForce GTX 570 | OK |
| IRQ 16 | 2nd generation Intel(R) Core(TM) processor family PCI Express Controller - 0101 | OK |
| IRQ 16 | PCI Standard-PCI-zu-PCI-Brücke | OK |
| IRQ 17 | Intel(R) 82801 PCI Bridge - 244E | OK |
| IRQ 17 | High Definition Audio-Controller | OK |
| IRQ 17 | Intel(R) 6 Series/C200 Series Chipset Family PCI Express Root Port 1 - 1C10 | OK |
| IRQ 18 | Marvell 91xx SATA 6G Controller | OK |
| IRQ 18 | Intel(R) 6 Series/C200 Series Chipset Family PCI Express Root Port 7 - 1C1C | OK |
| IRQ 18 | Intel(R) 6 Series/C200 Series Chipset Family PCI Express Root Port 3 - 1C14 | OK |
| IRQ 19 | Standard-Zweikanal-PCI-IDE-Controller | OK |
| IRQ 19 | Intel(R) 6 Series/C200 Series Chipset Family PCI Express Root Port 8 - 1C1E | OK |
| IRQ 19 | OHCI-konformer VIA 1394-Hostcontroller | OK |
| IRQ 19 | Intel(R) 6 Series/C200 Series Chipset Family PCI Express Root Port 4 - 1C16 | OK |
| IRQ 20 | Intel(R) Desktop/Workstation/Server Express Chipset SATA AHCI Controller | OK |
| IRQ 21 | Intel(R) Management Engine Interface | OK |
| IRQ 22 | High Definition Audio-Controller | OK |
| IRQ 23 | Intel(R) 6 Series/C200 Series Chipset Family USB Enhanced Host Controller - 1C26 | OK |
| IRQ 23 | Intel(R) 6 Series/C200 Series Chipset Family USB Enhanced Host Controller - 1C2D | OK |
| IRQ 81 | Microsoft ACPI-konformes System | OK |
| IRQ 82 | Microsoft ACPI-konformes System | OK |

Suchen nach:

☐ Nur ausgewählte Kategorie durchsuchen ☐ Nur Kategorienamen durchsuchen

Systemaufrufe bieten Prozessen die Möglichkeit, **Funktionalität des Betriebssystems** in Anspruch zu nehmen. Sie sind damit ein Mittel zur **Kommunikation** zwischen Nutzerprozessen und Betriebssystem.

Systemaufrufe

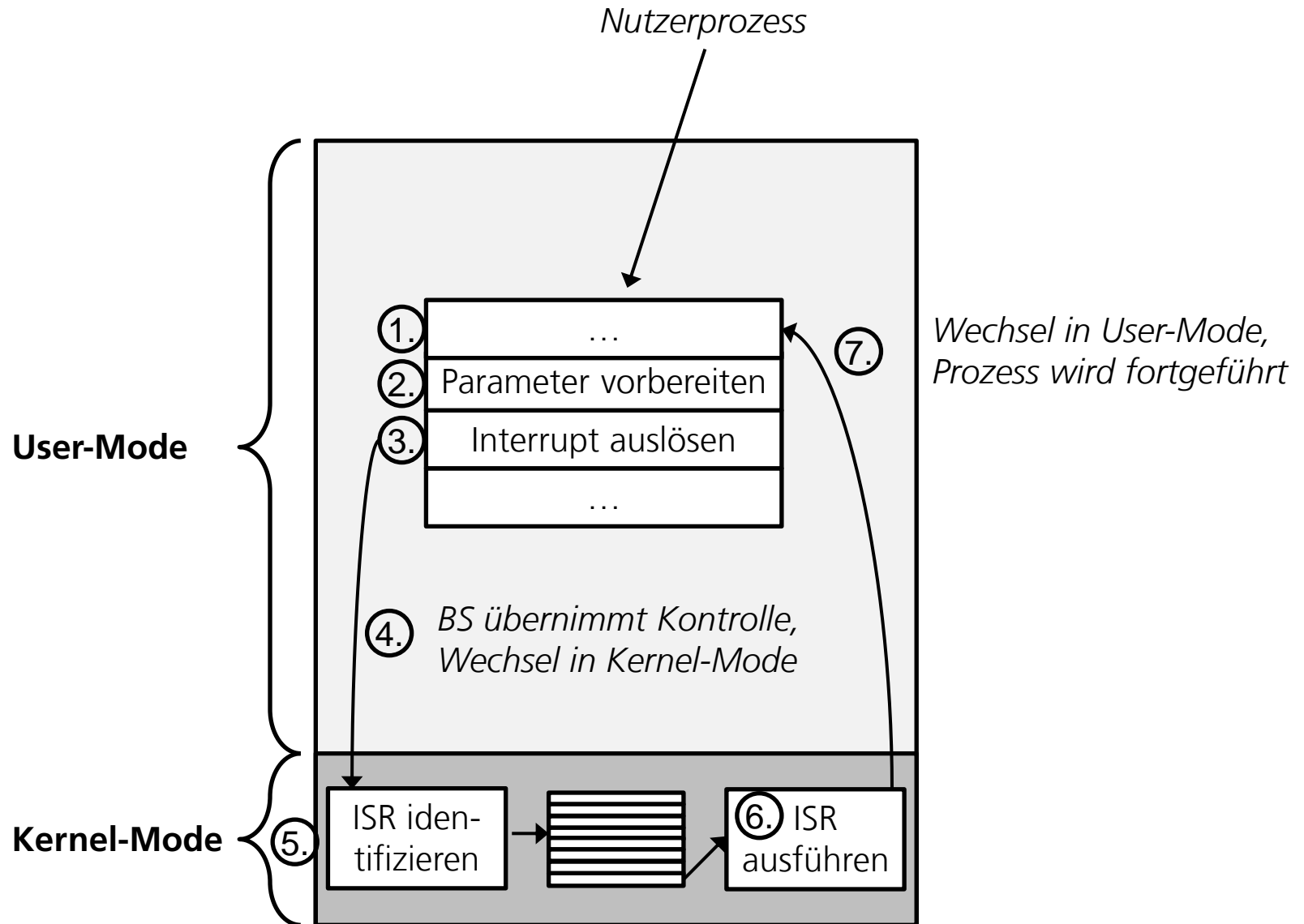
- Nutzerprozesse im User-Mode haben keinen direkten Zugriff auf Hardware, Dateisystem, etc.
- Bei Bedarf (bspw. Lesen aus einer Datei) stellt der Prozess eine entsprechende Anfrage an das Betriebssystem.

Wie aber übergibt der Prozess die Kontrolle an das Betriebssystem? *)

Antwort: Durch einen bewusst herbeigeführten Interrupt (Systemaufruf) wechselt das System in den Kernel-Mode und das Betriebssystem übernimmt die Kontrolle.

*) Zur Erinnerung: Nur der Nutzerprozess ist aktiv!

Systemaufrufe



Systemaufrufe – Beispiel Linux

- Interrupt **80h** ($=128_{10}$) für Systemaufrufe
- **gewünschte Funktion** im Register `eax`,
- eventuelle weitere Parameter in den Registern `ebx`, `ecx` und `edx`

Auswahl von Systemaufrufen in Linux

| eax | Name | Beschreibung | ebx | ecx | edx |
|-----|-----------|------------------------|-------------------------|---|--------------|
| 1 | sys_exit | Prozess beenden | Rückgabewert | - | - |
| 2 | sys_fork | Neuen Prozess erzeugen | Zeiger auf Prozessdaten | - | - |
| 3 | sys_read | Lesen aus Datei | Datei-deskriptor | Zeiger auf Speicherbereich zum Einlesen | Anzahl Bytes |
| 4 | sys_write | Schreiben in Datei | Datei-deskriptor | Zeiger auf Speicherbereich mit Inhalt | Anzahl Bytes |
| 5 | sys_open | Öffnen einer Datei | Zeiger auf Dateiname | Flags | Modus |
| 6 | sys_close | Schließen einer Datei | Datei-deskriptor | - | - |
| ... | | | | | |

Systemaufrufe – Beispiel Linux

- Interrupt **80h** ($=128_{10}$) für Systemaufrufe
- **gewünschte Funktion** im Register `eax`,
- eventuelle weitere Parameter in den Registern `ebx`, `ecx` und `edx`
- **Beispiel:** `eax = 5` für *Datei öffnen*:

```
...  
mov  eax, 5  
mov  ebx, path  
mov  ecx, flags  
mov  edx, mode  
int 80h  
...
```

Systemaufrufe – Höhere Programmiersprachen

- typischerweise kein direkter Systemaufruf in höheren Programmiersprachen (C/C++, Java, etc.)
- stattdessen Nutzung von **Programmierbibliotheken**:
 - C-Standardbibliothek ([glibc](#) in Linux, [msvcrt.dll](#) in Windows)
 - Java I/O package ([java.io.*](#))
- **Beispiel in C:** `#include <stdio.h>`

```
int main(int argc, char *argv[])
{
    FILE *f;
    f = fopen("test.txt", "r");
    return 0;
}
```

Systemaufrufe – Höhere Programmiersprachen

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    FILE *f;
    f = fopen("test.txt", "r");
    return 0;
}
```

ruft in der Unterbibliothek `__libc_open` folgenden Code auf:

```
movl    0x10(%esp,1), %edx
movl    0xc(%esp,1), %ecx
movl    0x8(%esp,1), %ebx
movl    $0x5, %eax
int    $0x80
```

Zusammenfassung

- Interrupts sind ein globales Kommunikationskonzept für unterschiedliche Aufgabenbereiche.
- Sie dienen
 - zur Ereignis- und Fehlerbehandlung im laufenden Betrieb,
 - zum Aufruf von Betriebssystemfunktionen aus dem User-Mode heraus.
- In allen Fällen erhält das Betriebssystem im Kernel-Modus die Kontrolle.
- Hierfür ist Hardwareunterstützung zwingend notwendig.