# IO-System

## Ziele hier, speziell für Echtzeitsysteme

- Verständnis der Abläufe, um Hardware integrieren zu können („Treiber programmieren“)
- Wie entstehen welche Latenzzeiten: Beurteilung von Aussagen, Verstehen kritischer Punkte

# IO-System Aufgaben

**Applikationssicht:**

**Schnittstelle für den einheitlichen Zugriff auf unterschiedlichste Hardware**

**Hardwaresicht:**

**Umgebung, um einfach Hardware systemkonform in den Kernel zu integrieren**

**Zusätzlich:**

**Realisiert Organisationsstrukturen auf Hintergrundspeicher (Filesysteme)**

# IO-System Schnittstellenfunktionen

- **Zugriffe auf Peripherie abgebildet auf**
  - **Lesen und Schreiben (read und write)**
  - **Konfigurieren (ioctl)**
  - **Öffnen und Schließen (open und close)**

- **Anforderung der Ressource beim Betriebssystem**

  **Gründe für Ablehnung:**
  - **fehlende Zugriffsrechte**
  - **die Ressource ist bereits belegt**

# IO-System Schnittstellenfunktionen

```
#define ZUMACHEN   0x00
#define AUFMACHEN  0x01
```

**Was kann man in diesem Beispiel besser machen?**

```
...
char Tuere;

...
fd=open( "Aufzugtuer", O_RDWR );
```

**Ressource besorgen (allozieren)**

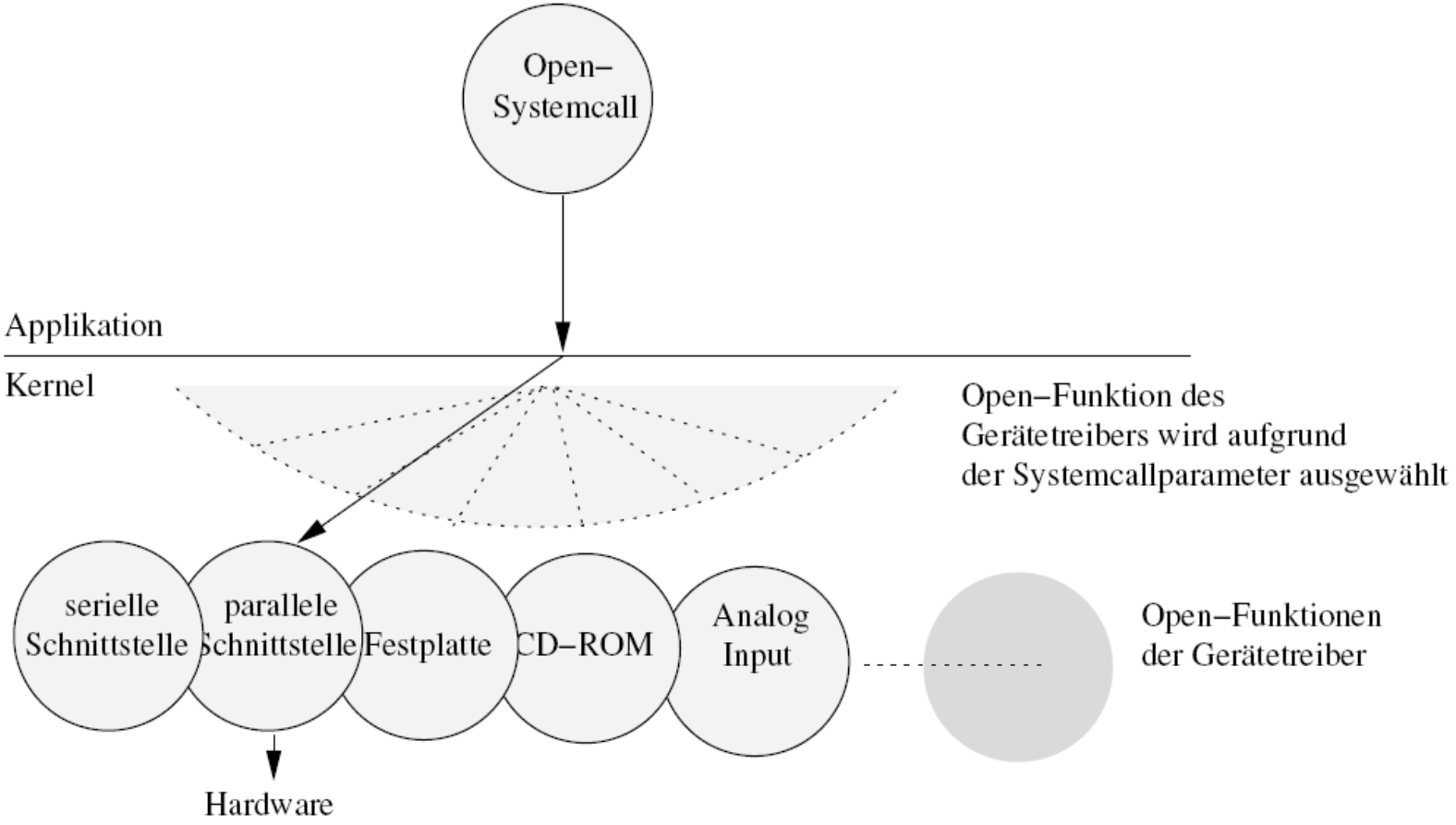
```
Tuere = AUFMACHEN;
write( fd, &Tuere, sizeof(Tuere) );
```

**Ressource (schreibend)verwenden**

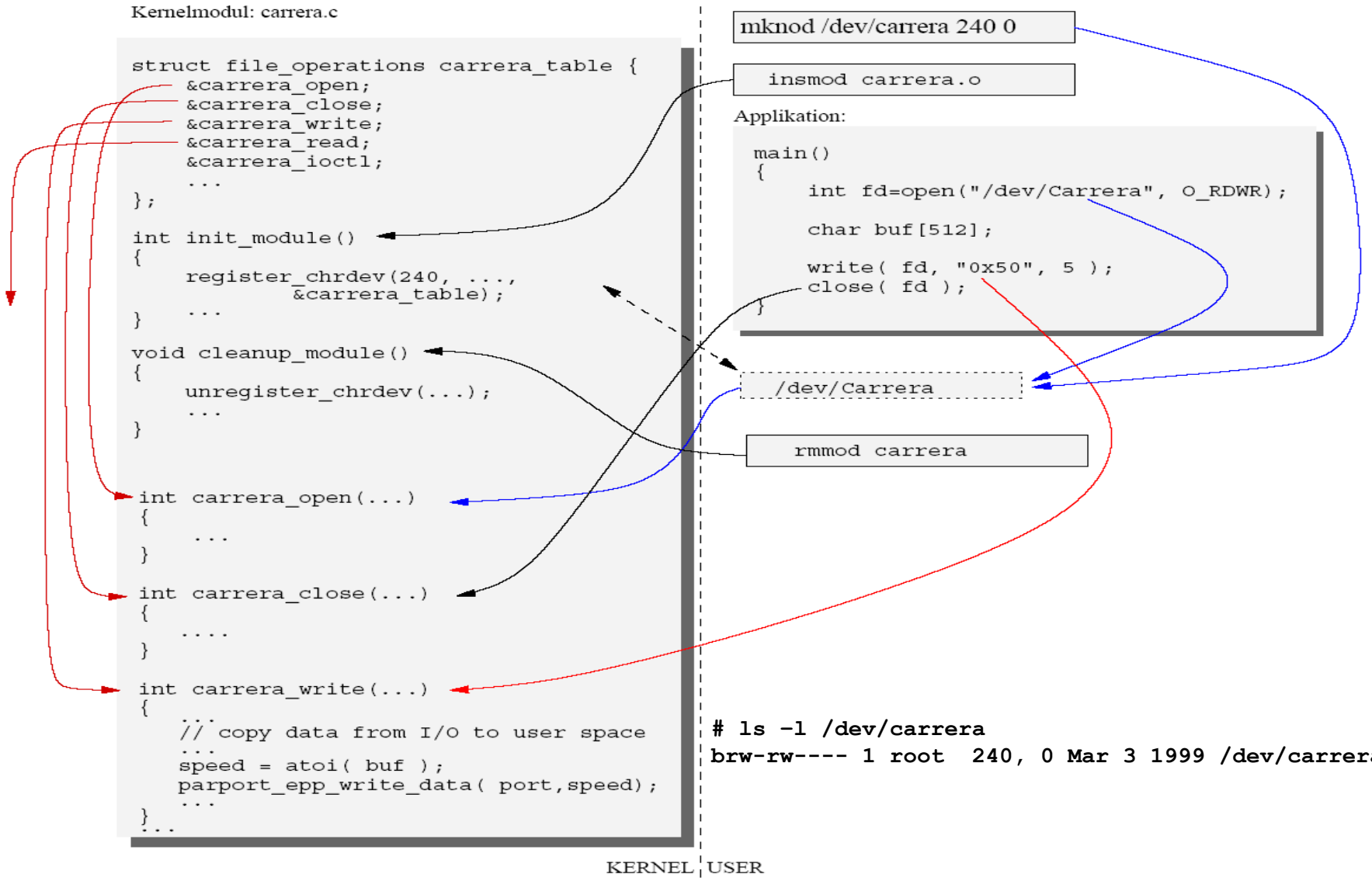
```
// Tuere auf
...
close( fd );
```

**Ressource freigeben**

# IO-System, Schnittstellenfunktionen



# IO-System, Schnittstellenfunktionen




```
# ls -l /dev/carrera
brw-rw---- 1 root  240, 0 Mar 3 1999 /dev/carrera
```

# Echtzeitbetriebssysteme

**IO-System, Schnittstellenfunktionen („*klassisch*“)**
**UNIX,OS9: Geräte werden ins Dateisystem abgebildet , API identisch**

```
int open(const char *pathname, int flags, mode_t mode);
OS9: error_code _os_open(const char *name, u_int32 mode,
path_id *path);
```

**Zugriffswunsch auf ein Gerät beim Betriebssystem anmelden.**
**Gerät wird als Pfad mit Dateinamen (`pathname`) angegeben**
**Zugriffsart: lesend/schreibend, blockierend/nicht blockierend (`flags`)**
**Zugriffsrechte: (`mode`)**
**liefert als Ergebnis *Descriptor* zurück, als Referenz für spätere Zugriffe**

```
int close(int fd)
OS9: error_code _os_close(path_id path);
```

**Gibt die angeforderte Ressource wieder frei**

# Echtzeitbetriebssysteme

## IO-System, Schnittstellenfunktionen („*klassisch*“)

```
ssize_t read(int fd, void *buf, size_t count)
OS9: error_code _os_read( path_id path, void *buffer,
u_int32 *count);
```

**Lesender Zugriff**

```
ssize_t write(int fd, const void *buf, size_t count)
OS9: error_code _os_write(path_id path, void *buffer,
u_int32 *count);
```

**Schreibender Zugriff**

```
int ioctl(int fd, int request, ...)
OS9: error_code _os_ss_xxx(..)/_os_gs_xxx(...)
```

**Konfigurieren des Geräts („IO-Control“), Parameter und Betriebsarten einstellen**

## IO-System Gerätezugriff im Betriebssystem-Kern

**Warum Geräte nicht aus der Applikation heraus ansteuern???**

- **Einheitliche Ressourcenverwaltung (Zugriff, Interrupts, I/O-Bereiche)**
  - **„Einfache“ Interruptbehandlung verzögert andere Interrupts unnötig lange**
- **Kapseln systemkritischer Teile**
  - **Hardwarezugriffe sind sicherheitskritisch, sollten nur innerhalb eines Treibers durchgeführt werden.**
  - **Hardwarezugriffe aus der Applikation heraus erfordert Abbildung der HW-Adressen in den Prozessadressraum**
  - **Programmierfehler können zum Absturz des gesamten Systems führen.**
- **Überführen eines Geräts in einen sicheren Zustand bei Applikationsfehlern**
  - **Treiber nach Applikationsfehler im BS noch vorhanden**
  - **Gerät wird bei Beendigung einer Applikation automatisch freigegeben**
  - **Insbesondere für sicherheitskritische Echtzeitsysteme wesentlich!**

# IO-System

**Gerätezugriff im Betriebssystem-Kern**

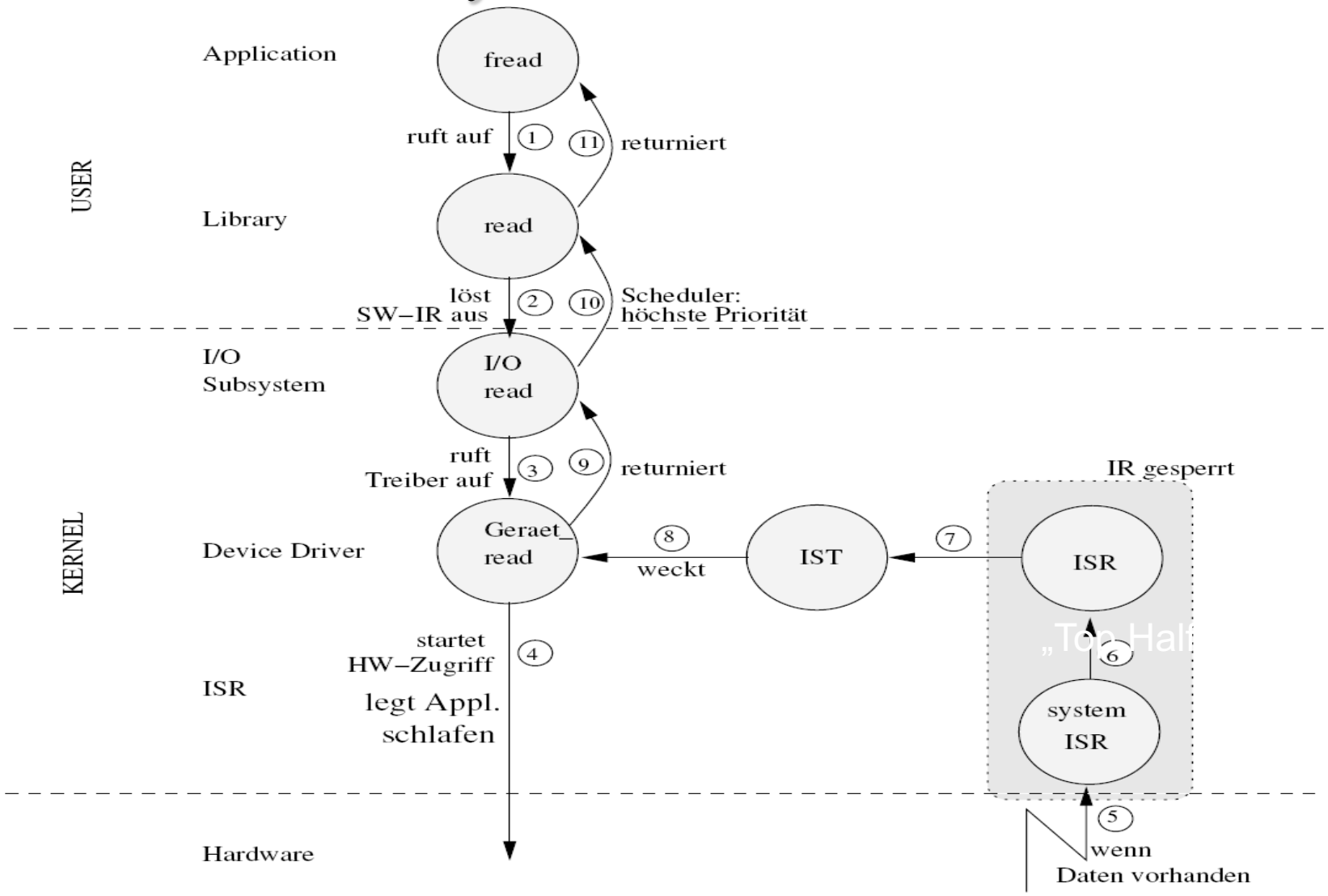
**Klassische Geräte-Schnittstelle wird von Applikationen benutzt**

**Daneben: Treiber verwenden/realisieren Funktionen, die durch den Betriebssystemkern selbst getriggert (aufgerufen) werden**

- **Interrupt Service Routinen (ISRs)**
- **Funktionen in einer Kernel-Queue und**
- **Funktionen in einer Timer-Queue.**

# Ziele: Genaueres Verständnis für unsere Wartezeiten

# Echtzeitbetriebssysteme



# Echtzeitbetriebssysteme

**IO-System, Gerätezugriff im Betriebssystem-Kern**

***Kernel-Queues***

**Ziel:**

- **Verkürzung der Zeit im SysCall (z.B. LynxOS-Kernel-Threads)**
- **Verkürzung der ISR-Zeit, in der weitere Interrupts gesperrt sind**
- **Periodische Vorgänge in Treibern (z.B. Polling)**

**Kernel-Queues: Listen von (Treiber-)Funktionen (Kernel-Queue-Funktionen), die vom Kernel in bestimmten Zuständen abgearbeitet werden**

**Kernel-Queue-Funktionen: Treiber-Routinen, bei `init_module` oder `open` in entsprechende Queue eingehängt**

**Linux-Kernel:**

- **Nach Abarbeitung aller anliegenden Interrupts (IST, „bottom half“, z.B. bei LynxOs auch als Thread realisiert: Höherpriore User-Threads haben Vorrang).**
- **Vor dem Scheduling**
- **Bei jedem Tick der Systemuhr (periodischeVorgänge innerhalb des Treibers, z.B. Pollen eines Gerätes)**

# IO-System

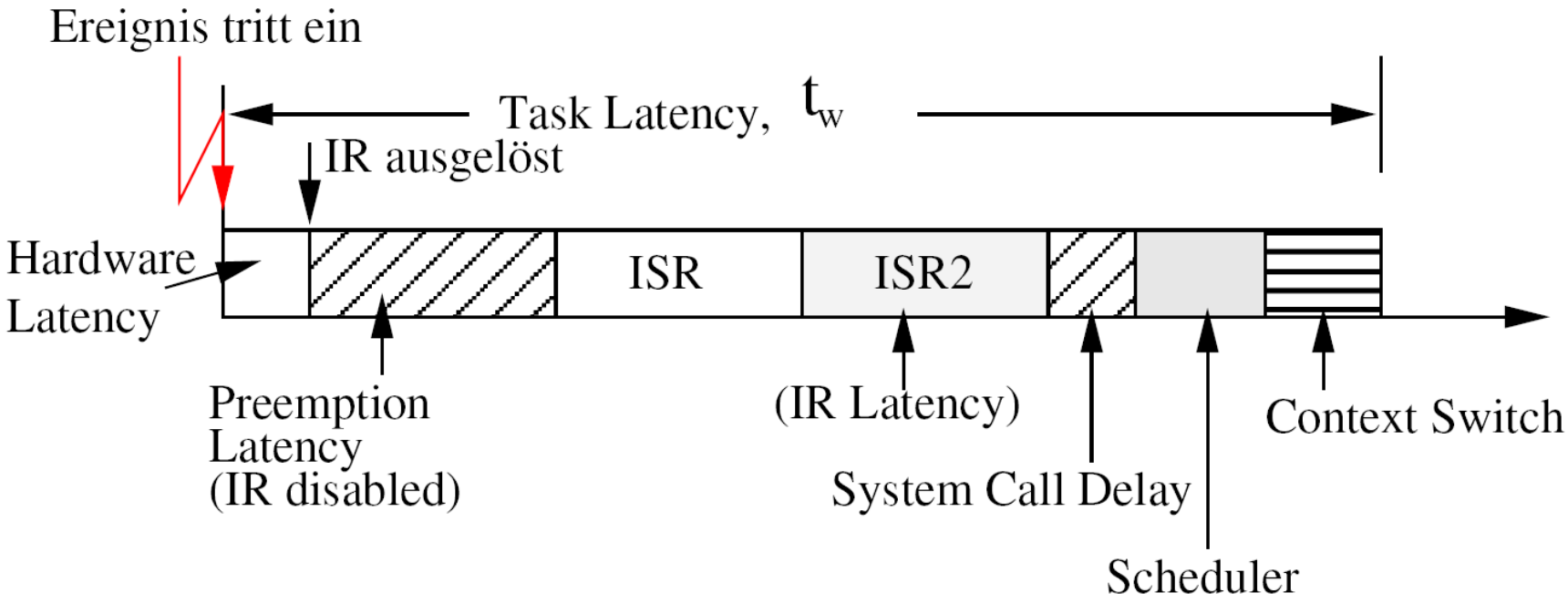
## Gerätezugriff im Betriebssystem-Kern

### *Timer-Queue*

- Liste aus Paaren (`Treiberfunktion, Zeitpunkt`)
- Zum spezifizierten `Zeitpunkt` wird die angegebene `Treiberfunktion` durch den Kernel aufgerufen

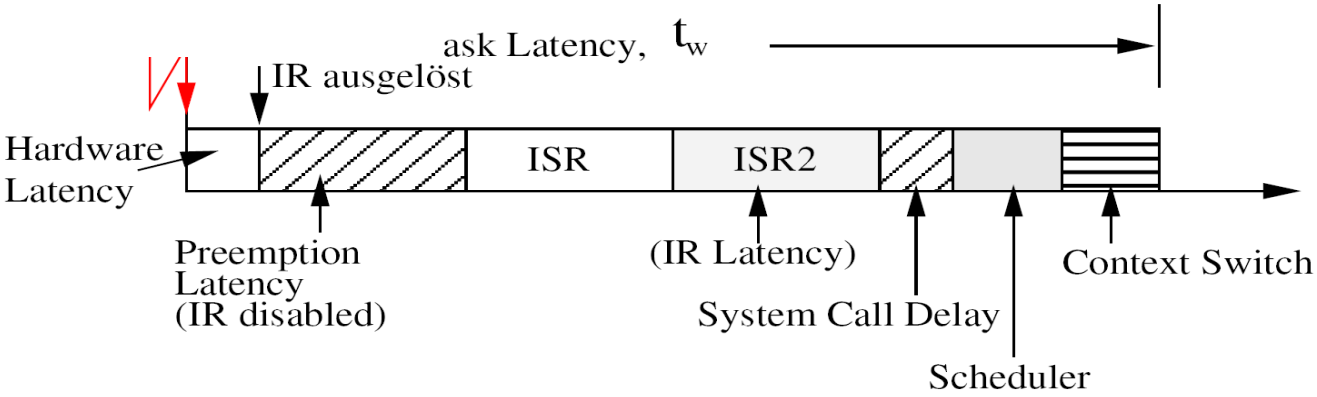
## Latenzzeit

- **Vor langer, langer Zeit: Reaktionszeit $t_R = t_V + t_W$ (Verarbeitungszeit + Wartezeit)**



# Echtzeitbetriebssysteme

## Latenzzeit




1. **Hardware-Latency: Zeit, bis Hardware Ereignis als Interrupt über den Bus dem Prozessor gemeldet hat (einige wenige Gatterlaufzeiten, einige ns).**
2. **Preemption-Delay: BS in kritischem, nicht unterbrechbarem Abschnitt (Interrupts sind gesperrt; sehr hoch bei Standard-BS).**
3. **Verarbeitungszeit der Interrupt-Service-Routine (ISR).**
4. **Interrupt Latency: Verzögerungszeit, weil BS andere Interrupts bearbeitet**
5. **System Call: Interrupt während eines Systemcalls: Systemcall fertig stellen (oder abbrechen). (sehr hoch bei Standard-BS: viele ms)**
6. **Scheduling:**
7. **Context-Switch: Aufwand, um andere Task weiterzubearbeiten**
8. **Weiterbearbeitung höher priorer Tasks (bisherige Wartezeit-Definition)**

# IO-System, Filesysteme

**Abspeichern, Sichern auf Hintergrundspeicher von:**

- **Programmen (BS-Kern, Dienstprogramme und Applikationen)**
- **Konfigurationsinformationen**
- **Daten (z.B. HTML-Seiten)**

**Embedded Systems: Kaum Festplatten, sondern**

- **RAM-Filesysteme (beim Starten aus persistentem Speicher geladen, beim Beenden auf persistenten Speicher geschrieben)**
- **EEPROM**
- **Flash („soldered“, CompactFlash, ...)**

# Echtzeitbetriebssysteme

## IO-System, Filesysteme

**Organisationsstruktur (Filesystem, z.B.** FAT**, VFAT, NTFS v3, Linux** Ext2, Ext3**,Ext4,XFS,OS9) soll:**

- **schnellen Zugriff ermöglichen**
- **wenig Overhead (Speicher) für Verwaltungsinfo benötigen**

**Meist wird Cache verwendet**

**Problem:**

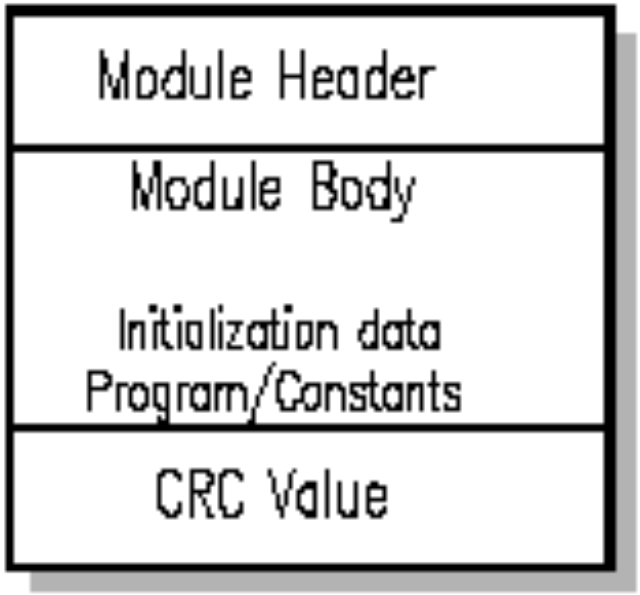
- **Daten temporär inkonsistent**
- **Kein zeitlicher Determinismus bei Zugriff**

**Sync-Mode:**

- **Keine Verwendung von Cache**
- **Explizit synchronisieren**

# IO-System, OS9 Modulsystem I

- Ein OS9-Modul ist ein speicherform-unabhängiges Objekt, das durch CRC gesichert ist. (→RAM, Festplatte,Flash…)
- Beim Booten wird der Speicher nach gültigen Modulen durchsucht
- Alles sind Module -selbst der OS9000-Kernel- :
  - Programme, Unterprogramme,
  - Datenbereiche
  - Devicetreiber, Devicedescriptoren, Filemanager
  - Systemprogramm, Spezialmodule CSD
  - Libraries (Trapmodule)
- OS9 Programm-Module → Inplace-Execution



# IO-System, OS9 Modulsystem II

- **CRC-Wertberechnung: 24-bit CRC über das erste Byte bis zum letzten Byte (ohne CRC) des Moduls**
  **►Module werden nur dann als Module akzeptiert, wenn ein gültiger Modulkopf vorliegt und! der CRC i.O. ist !**
- **Ausführbare-Module müssen (bis auf wenige Ausnahmen) sein: reentrant und position-independent**
- **Alle im ASP befindlichen Module werden im ASP im sog. Modulverzeichnis verwaltet.**

# IO-System, OS9 Modulsystem III

- **Modulkopf (offset Angaben vom Modulstart ab gerechnet):**

```
0x0000h:   u_int16  m_sync,      /* sync bytes ►PDT MPC 555-System: 0xF00D  */
0x0002h:   m_sysrev;             /* system revision check value */
0x0004h:   u_int32 m_size;       /* module size ►Alles!! incl CRC Body*/
0x0008h:   owner_id m_owner;     /* group/user ID */
0x000Ch:   u_int32 m_name;       /* offset to module name ►PDT: Offsetwert 0x0058h*/
0x0010h:   u_int16 m_access;     /* access permissions */
```

- **Die Zugriffsmöglichkeiten auf ein Modul sind an die *Access -Permission* Rechte gebunden.**
  - **m_owner (group/userID) gibt an wem das Modul gehört.**
  - **m_access legt für MP_OWNER_*, MP_GROUP_*,MP_WORLD_* jeweils die EXEC/READ/WRITE Rechte fest.**
  - **Bei Zugriff auf ein Modul (anlinken,ausführen..) wird überprüft ob der Benutzer, identifiziert durch seine group/userID, berechtigt ist oder nicht.**

# IO-System, OS9 Modulsystem IV

**0x0012h: u_int16 m_tylan; /***

**module type : MT_ANY 0 = Not used (wildcard value), MT_PROGRAM , MT_DATA …**

**and language: ML_ANY 0 = Unspecified language (wildcard), ML_OBJECT Machine…**

**0x0014h: u_int16 m_attrev; /* module attributes (first byte) and revision (second byte)**

**Bit 7 The module is re-entrant (sharable by multiple tasks).**

**Bit 6 The module is sticky. A sticky module is not removed from memory until its link count becomes -1 or memory is required for another use.**

**Bit 5 The module is a system-state module.**

**Wenn zwei Module gleichen Namens im Speicher sind, so wird nur das Modul verwendet, dass die höhere Revisionsnummer besitzt. */**

**0x0016h: u_int16 m_edit; /* module edition number ►semantic User-definable */**

**0x0018h: u_int32 m_needs, /* module hardware requirements flags ►not used!*/**

**0x001Ch: m_share, /* offset of shared data in statics */**

**0x0020h: m_symbol, /* offset to symbol table ►reserved*/**

**0x0024h: m_exec, /* offset to execution entry point ►Bedeutung hängt vom Modultyp ab ►Datamodul Offset zum ersten Datenbyte*/**

**….**

# IO-System, OS9 Modulsystem V

**mfree -e** **Display Free System Memory**

**mdir [<opts>] [<modname>]** **Display Modul Directory (➔im Speicher)**

**Beispiel : $ mdir -e**

**Current Module Directory**

| Addr | Size | Owner | Perm | Type | Revs | Ed | # Lnk | Module name |
|---|---|---|---|---|---|---|---|---|
| fff58f28 | 4016 | 1.0 | 0555 | Prog | c001 | 5 | 1 | activ |
| 01050000 | 6320 | 1.0 | 0555 | Prog | c001 | 37 | 1 | attr |
| ffe0aba0 | 14776 | 1.0 | 0555 | Prog | c001 | 208 | 1 | bootptest |
| fff09368 | 6200 | 0.0 | 0555 | Sys | a000 | 18 | 1 | bootsys |
| fff5b1a0 | 3256 | 0.0 | 0555 | Prog | c001 | 10 | 1 | break |
| fff041b0 | 1088 | 0.0 | 0555 | Data | 8000 | 1 | 1 | cnfgdata |
| fff045f0 | 4408 | 0.0 | 0555 | Sys | a000 | 16 | 1 | cnfgfunc |
| fff56340 | 11240 | 1.0 | 0555 | Prog | c001 | 52 | 1 | copy |
| 01014420 | 72184 | 1.0 | 0555 | Subr | c000 | 19 | 4 | csl |
| fff5be58 | 4456 | 1.0 | 0555 | Prog | c001 | 23 | 1 | date |
| 010587a0 | 3840 | 1.0 | 0555 | Prog | 8001 | 7 | 1 | datmod2file |
| 0105a0a0 | 192 | 0.0 | 0333 | Data | 8000 | 1 | 1 | datstartup |
| fff87a90 | 16232 | 1.0 | 0555 | Prog | c001 | 33 | 1 | dcheck |

**►Es gibt voreingestellt zwei Pfade : Datemodulpfad und Executionmodulpfad**

**►Bei versuchtem Programmstart von dem Shell-Prompt aus wird in folgender Reihenfolge gesucht:**

**1. Moduldirectory im Speicher 2. Executionmodulpfad**

**➔Beim programmgesteuerten Laden von Datenmodulen:**

**1. Moduldirectory im Speicher 2. Datenmodulpfad**

# IO-System, OS9 Modulsystem VI

**pd** — **Print Working Directory ►aktuelles Datenmoduldirectory**

**pd -x** — **Print Working Directory ►aktuelles Executionmoduldirectory**

**chd [<path>]** — **Change Current Data Directory**

**chx <path>** — **Change Current Execution Directory**

**load [<opts>] {<file>}** — **Load Modul from File into Memeory**

**►Läd die Datei <file> in den Speicher. <file> wird zunächst im Executiondirectory gesucht und dann entlag der Enviromentvariablen *PATHp PATH***

**►*-d* Option läd aus dem Datenmoduldirectory.**

**►In Abhängigkeit vom *revision*-Level wird ein gleichnamiges Modul im Speicher ggf. überschrieben**

**►Beim Laden werden die *modul-permissions* überprüft owner/group/public**

**unlink {<modulname>]** — **Unlink Memory Modul im Arbeitsspeicher**

**►wenn der Link-Count = 0 (und nicht in *bootlist* or *sticky*) dann wird das betreffende Modul gelöscht**

**Aufgabe: Ein bereits geladenes!! Modul soll erneut vom Massenspeicher geladen werden**

**►mittels mehrfach Aufrufs von *unlink* <*modulname*> zuerst völlig entladen!! ► Link-count auf „0“**

# IO-System, OS9 Filesystem I

- **OS9 hat ein hierarchisches Dateisystem. ►Directory/Subdirectory -Struktur beliebig verschachtelt.**
- **OS9-Dateinamen unterscheiden nicht Groß und Kleinschreibung!! ➔!! Aber mit abgespeichert!!**
- **►Dateinamen können bis zu 43 Zeichen lang sein. Gültige Zeichen:A-Z,a-z,0-9,_, .,$**
- **►*CMDS* und *cmds* sind ein und dasselbe!! Keine zwei Dateien!! *Konvention: Dirnames in Großb.***
- **►Als Programmparameter eines shell-Kommandos findet jedoch keine Konvertierung statt!!**
- **Zu jeder Datei werden Namen, Dateianfangsposition , Dateilänge, Datum der letzten Modifikation abgespeichert**
- **OS9 unterstützt Zugriffsschutzmechanismus für Dateien**
- **Jeder Benutzer hat eine eigene *user/group-ID* , die durch den Login-Prozess (*password-Datei)* festgelegt ist. ➔Group/User 0,0 ist der Super-User und der darf alles.**
- **Jede Datei gehört zunächst dem Ersteller (owners *user/groupID) =owner.*. Änderung der Rechte nur durch *super-user* oder *owner (oder alle mit der gleichen groupID)* mittels *attr Kommando* möglich.**
- **Jede Datei enthält ein Feld zur Beschreibung der Zugriffsrechte für den *owner* und *public***

# IO-System, OS9 Filesystem II

- *$ dir -e*

```
Directory of . 13:32:32
  Owner    Last modified    Attributes      Block      Bytecount  Name
---------  -------------  ----------------  -------    ---------  ----
  0.0      03/10/06 0000   d----swr-swr-swr  7              128   CMDS
  0.0      03/10/06 0000   ------wr------wr  A               70   STARTUP
  0.0      03/10/06 0252   d----swr-swr-swr  5              192   SYS
  0.0      03/10/06 1332   --------------wr  C               41   password
```

***Bitgruppen: do---public-group-owner***

***Position** 1. d=directory 2. o single user file 6/10/14 s=search permission if directory/ e= executable 7/11/15 w=write allowed 8/12/16 r= read allowed*

# IO-System, OS9 Filesystem III

- **jedes benutzbare Geräte hat einen systemweiten eineindeutigen Namen.**
  - ***default disk drive: /dd RAM-Disk: /r0 Pipe-device: / pipe Null-device /nil***
- **Der absolute Zugriffspfad (*pathname*) einer Datei setzt sich aus dem Gerätenamen, dem Directorypfad und dem Dateinamen selbst zusammen. Der Gerätenamen wird vorangestellt:**

  **$ list /dd/CMDS/BOOTOBJS/test.c**
- **relative Zugriffspfadangaben erfolgen ohne vorangestelltem / und beziehen sich immer auf die aktuelle Pfadposition die durch die vorangegangenen *chd*-Kommandos eingestellt wurde►*pd* zeigt diese Position**

  **$ pd**
  **/r0/SYS**
  **$chd ..**
  **$pd**
  **/r0**
  **$chd SYS**