6.4 Fallstudie: PC-Bussysteme (9)

- **Plug & Play**:
  - beim Booten liest BIOS den 256-Byte **Konfigurationsblock** jeder PCI-Karte mit:
    - Code für **Hersteller** und **Gerätenummer**
    - Code für **Gerätetyp**
    - **Zeitparameter**
  - BIOS konfiguriert automatisch
    - **Startadressen** für E/A-Register (im Speicher- oder E/A-Raum) und Erweiterungs-ROM
    - **Interrupt-Leitungen**
  - BIOS löst ggf. alle Konflikte auf
  - keine manuelle Konfiguration der PCI-Karten erforderlich!

31	16	15	0	
Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Revision ID	08h
BIST	Header Type	Latency Timer	Cache Line Size	0Ch
Base Address Registers				10h–24h
Cardbus CIS Pointer				28h
Subsystem ID		Subsystem Vendor ID		2Ch
Expansion ROM Base Address				30h
Reserved			Capabilities Pointer	34h
Reserved				38h
Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line	3Ch

6.5 Fallstudie: USB

- **Universal Serial Bus**, spezifiziert vom USB Implementers Forum (`www.usb.org`)
- Ziel: preiswerter, einheitlicher und einfacher Anschluß diverser E/A-Geräte
- **serieller, asynchroner Peripherie-Bus**
  - 4-adriges Kabel: V_{cc} (Stromversorgung 5V, max. 0.5A), GND, D+, D– (Pegel 3.3V)
  - unterschiedliche Stecker für Host (USB-A) und E/A-Gerät (USB-B)
- **USB1.1** (1995)
  - Transferraten: 1.5 MBit/s (*low speed*) oder 12 MBit/s (*full speed*)
- **USB2.0** (2001)
  - weitere Transferrate: 480 MBit/s (*high speed*)

USB-A USB-B

Beschriftung:

6.5 Fallstudie: USB (2)

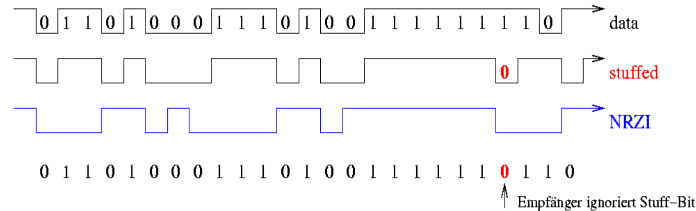
- **hierarchischer Aufbau** eines USB-Bussystems:
  - ausschließlich Punkt-zu-Punkt Verbindungen
  - **USB Host** (auch **Root Hub**, mit 2 bis 4 USB Anschlüssen) ist einziger Master, fragt alle USB Geräte durch *Polling* ab
  - **USB Hub** (Verstärker, ggf. mit Anpassung der Transferrate) verteilt Signale auf mehrere USB-Anschlüsse und ermöglicht den Aufbau eines pyramidenartigen Bussystems
  - maximal **7 physikalische Ebenen** in Pyramide, logisch jedoch eine Ebene
  - insgesamt maximal **127 Buskomponenten**
  - Länge eines Kabels max. **5m** ⇒ insgesamt max. **35m** bei 7 Ebenen (USB-A Stecker stets zum Host, USB-B Stecker stets zum E/A-Gerät gerichtet)
  - **Autokonfiguration**: E/A-Geräte identifizieren sich selbst beim Host und erhalten eine Adresse zwischen 1 und 127 (Host hat Adresse 0)
  - Geräteanschluss im laufendem Betrieb möglich (*Hot Plugging*)
- Kopplung zweier USB Hosts ist nicht möglich!

6.5 Fallstudie: USB (3)

- Busprotokoll gestattet vier verschiedene Übertragungsarten:
  1) **Kontroll-Transfer**: Initialisierung und Konfiguration eines Gerätes durch USB Host
  2) **Interrupt-Transfer**: USB Host fragt alle E/A-Geräte ab, ob Interrupts angefordert wurden
  3) **Bulk-Transfer**: Senden langer Datenströme (nur bei *full / high speed*, falls ausreichende Bandbreite verfügbar)
  4) **Isochroner Transfer**: Übertragung von Daten mit einer garantierten Bandbreite (d.h. in Echtzeit), z.B. für Sprach- oder Videodaten (nur bei *full / high speed*)
- jeder Transfer wird vom USB Host initiiert!
- periodische Transfers (Interrupt-/Isochroner Transfer) dürfen nicht mehr als 80-90% der Busbandbreite verwenden!
- Hin- und Rückrichtung über die gleichen Leitungen!

6.5 Fallstudie: USB (4)

- **NRZI-Kodierung** (*Non Return to Zero Inverted*):
  - Wechsel des Leitungspegels **nur bei Übertragung eines Null-Bit**
  - **Bit-Stuffing**: Einfügen eines Null-Bits nach jeweils 6 Eins-Bits
  - Beispiel: NRZI-Kodierung der drei Bytes 68_{16}, $E9_{16}$ und FE_{16}



- **Differentielle Signale**
  - zur Übertragung des NRZI-Signals über verdrilltes Kabelpaar D+, D–:
    - Sender: (D+) – (D–) > 1V (Eins-Bit) bzw. < –1V (Null-Bit)
    - Empfänger: (D+) – (D–) > 0.2V (Eins-Bit) bzw. < –0.2V (Null-Bit)

6.5 Fallstudie: USB (5)

- **paketorientierte Übertragung**:
  - Einteilung in Zeitabschnitte von **1 ms** Dauer, auch als *Frame* bezeichnet (⇒ 12000 Bit/Frame im *High Speed* Modus)
  - Adressierung der Endgeräte durch **7 Adress-Bits** (für 127 Geräte) und **4 EP-Bits** (für 16 verschiedene Endpunkte je Gerät, z.B. EP0 = *Control*, EP1 = *Bulk*, EP3 = *Interrupt*)
  - Kommunikation zwischen Host und Endpunkten von E/A-Geräten über logische Kanäle (*Pipes*), die einen Teil der Busbandbreite belegen
  - jede Datenübertragung innerhalb eines Frames besteht aus **drei Paketen**, wobei jedes Paket mit einer 8-Bit Typkennung beginnt:
    1) **Paket** mit **Richtung** und **Zieladresse** (11 Bit + 5 Bit Prüfsumme)
    2) **Datenpaket** (variable Länge + 16 Bit Prüfsumme)
    3) **Bestätigungspaket** (Handshaking) des Empfängers

6.5 Fallstudie: USB (6)

- Vergleich von USB1.1 und **USB 2.0**:

	low speed	*full speed*	*high speed*
Bit-Tranferrate	1.5 MBit/s	12 MBit/s	**480 MBit/s**
max. Bulk-Datenpaketgröße	—	64 Byte	**512 Byte**
max. Transferrate	16 KByte/s	1.2 MByte/s	**54 MByte/s**

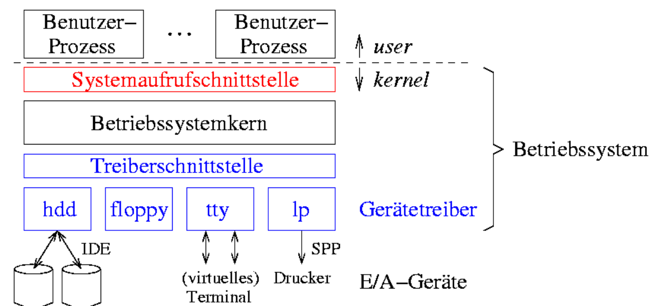
- USB stellt heute die Standardschnittstelle für weit über 50% aller (mittelschnellen) Peripheriegeräte dar
- **Firewire** (IEEE1394) ist sehr ähnlich zu USB:
  - entwickelt und lizenziert von Apple
  - unterstützt mehrere Busmaster (d.h. auch die direkte Kommunikation zwischen zwei E/A-Geräten ist möglich)
  - bis zu 400 MBit/s
  - erfordert aufwendigere Logik auf Host- und Peripherieseite

7 Gerätetreiber

- ein **Gerätetreiber** stellt eine Softwareschicht zwischen dem Betriebssystemkern und dem E/A-Gerät dar:
  - ein Gerätetreiber ist ein **Softwaremodul**, das geräteabhängigen Code zur Steuerung von E/A-Geräten eines Typs enthält
    (⇒ Betriebssystemkern bleibt unabhängig von E/A-Geräten!)
  - es sind mehrere Gerätetreiber erforderlich (z.B. für Festplatten, RS232, EPP, USB), die zum Kern hinzu gebunden werden
  - einheitliche **Treiberschnittstelle** zum Betriebssystem (Implementierungsdetails bleiben verborgen, z.B. Adressen und Inhalt der E/A-Register)
  - Betriebssystem bietet eine einheitliche (d.h. geräteunabhängige) **System-aufrufschnittstelle** zum Benutzerprozess
  - Beispiel:
    Der Unix-Systemaufruf `read(fd,buf,n)` liest `n` Bytes von einem beliebigen E/A-Gerät `fd` in einen Puffer `buf`, wobei das ausführende Programm keine Kenntnis von der Art des E/A-Gerätes haben muss.

7 Gerätetreiber (2)

- Schichten eines Betriebssystems zwischen Benutzerprozess und E/A-Gerät (vereinfacht):



- Benutzerprozess und Gerätetreiber arbeiten in unterschiedlichen Speicherbereichen (*user / kernel space*)

7 Gerätetreiber (3)

- Aufgaben eines Gerätetreibers (Auswahl):
  - **Initialisierung** und **Überwachung** der E/A-Gerätes durch geeignete Programmierung der E/A-Register
  - Bereitstellen einer Schnittstelle zur Annahme **abstrakter** Anfragen an ein E/A-Gerät und Umsetzen der abstrakten Anfrage in eine **konkrete** geräteabhängige Form
  - Übernahme/Übergabe und Pufferung von Daten
  - **Zuteilung** von E/A-Geräten an Benutzerprozesse (zur exklusiven oder gemeinsamen Nutzung)
  - Implementierung von Warteschlangen für E/A-Geräte
  - Verwaltung von **Zugriffsrechten**
  - Behandlung von **Unterbrechungsanforderungen**
  - Behandlung von **Fehlermeldungen** des E/A-Gerätes
  - Wahl von Parametern/Strategien zur optimalen Nutzung eines Gerätes

7 Gerätetreiber (4)

- Betriebsarten eines Gerätetreibers:

1) **Polling**:
- CPU wartet aktiv, bis E/A-Gerät bereit ist
- nach Ausführung jedes Teiltransfers vom/zum E/A-Gerät wartet CPU aktiv, bis der Transfer beendet ist

2) **Interrupt**:
- wenn E/A-Gerät noch nicht bereit ist, schläft Prozess (→ Prozesswechsel durch Betriebssystem)
- E/A-Gerät kann bei Eintritt der Bereitschaft durch Senden einer Unterbrechungsanforderung den schlafenden Prozess aktivieren
- nach Initiierung jedes Teiltransfers vom/zum E/A-Gerät schläft Prozess (→ Prozesswechsel durch Betriebssystem), bis der Transfer beendet ist.
- Vorteil: keine Blockierung anderer Prozesse, höhere Auslastung der CPU
- Nachteil: höherer Aufwand (Interruptroutinen, Sicherung der Register, ...)

7 Gerätetreiber (5)

- Arten der Ein-/Ausgabe:

1) **synchron**:
- Systemaufruf zur Ein-/Ausgabe terminiert erst, wenn die E/A-Operation vollständig abgeschlossen ist
- bei Interrupt-Betrieb kann ggf. zwischenzeitlicher Prozesswechsel durch Betriebssystem erfolgen

2) **asynchron**:
- Systemaufruf initiiert lediglich die Ein-/Ausgabe und gibt Kontrolle an den aufrufenden Prozess zurück
- sinnvoll vor allem bei Ausgabeoperationen!
- durch zusätzlichen Systemaufruf kann sich Benutzerprozess nachträglich mit Ende der E/A-Operation synchronisieren
- Vorteil: Benutzerprozess kann CPU während der E/A-Operation nutzen

7.1 Gerätetreiber unter Linux

- E/A-Geräte werden als **Spezialdateien** repräsentiert:
  - Öffnen der Spezialdatei ermöglicht Zugriff auf E/A-Gerät, implementiert durch Gerätetreiber
  - Geräte können wie Dateien gelesen und geschrieben werden
- zwei Gerätearten:
  - ein **zeichenorientiertes** Gerät (*char device*) gestattet den sequentiellen Zugriff auf einzelne Bytes oder auf einen Bytestrom (Beispiele: Tastatur, Maus, RS232-Schnittstelle)
  - ein **blockorientiertes** Gerät (*block device*) gestattet den wahlfreien Zugriff auf Block fester Größe (z.B. 1KByte) oder Vielfaches davon (Beispiele: Festplatte, Floppy-Disk, CD-ROM)
- jedes Gerät wird eindeutig beschrieben durch
  - Geräteart: `c` = *char device*, `b` = *block device*
  - **Major-Nummer**: eindeutiger Index für jeden Gerätetreiber
  - **Minor-Nummer**: Auswahl eines Gerätes innerhalb eines Gerätetreibers

7.1 Gerätetreiber unter Linux (2)

- Geräte in einem Linux-System (Auszug aus `/dev`):

```
crw-------   1 alfred  audio   14,  4  /dev/audio
lrwxrwxrwx   1 root    root         3  /dev/cdrom -> hdd
crw-------   1 root    root     5,  1  /dev/console
brw-------   1 alfred  disk     2,  0  /dev/fd0
brw-rw----   1 root    disk     3,  0  /dev/hda          (Harddisk)
brw-rw----   1 root    disk     3,  1  /dev/hda1
brw-rw----   1 root    disk     3,  2  /dev/hda2
brw-------   1 alfred  disk    22, 64  /dev/hdd
crw-rw----   1 root    lp       6,  0  /dev/lp0                (SPP)
lrwxrwxrwx   1 root    root        10  /dev/mouse -> /dev/psaux
crw-rw----   1 root    root    10,  1  /dev/psaux            (PS/2)
crw-rw----   1 root    disk     9,  0  /dev/st0        (SCSI Tape)
crw-rw-rw-   1 root    root     5,  0  /dev/tty         (Terminal)
crw--w--w-   1 alfred  tty      4,  0  /dev/tty0
crw-rw----   1 root    tty      4,  1  /dev/tty1
crw-rw----   1 root    uucp     4, 64  /dev/ttyS0          (RS232)
crw-rw----   1 root    uucp     4, 65  /dev/ttyS1
drwxr-xr-x   2 root    root      4096  /dev/usb
crw-rw----   1 root    lp     180,  1  /dev/usb/lp1
crw-------   1 alfred  root   180, 48  /dev/usb/scanner0
```


7.1 Gerätetreiber unter Linux (3)

- Hinzufügen eines zusätzlichen Gerätes im Dateisystem durch privilegierten `mknod`-Befehl

  Beispiel: `mknod /dev/xx1 c 211 1`
  - erzeugt zeichenorientiertes Gerät mit Namen `xx1`
  - gesteuert durch den Gerätetreiber mit Index **211** (**Major-Nummer**)
  - repräsentiert Gerät `1` dieses Treibers (**Minor-Nummer**)
- Gerätetreiber können entweder **statisch** (d.h. beim Übersetzen des Betriebssystemkerns) oder **dynamisch** (d.h. zur Laufzeit) als **Modul** dem Betriebssystem hinzugefügt werden
- dynamisches Laden und Entladen von Modulen erfolgt mit den privilegierten Befehlen `insmod` und `rmod`

  Beispiel: `insmod xx.o` bzw. `rmod xx.o`
  - lädt/entlädt den Treiber `xx` zum/vom Linux Kern
  - erzeugt/entfernt einen entsprechenden Eintrag in `/proc/modules`

7.1 Gerätetreiber unter Linux (4)

- Befehle `insmod` oder `rmod` bewirken den Aufruf der Routinen `init_module` bzw. `cleanup_module` des Gerätetreibers

  Beispiel: Auszug aus Quelltext `xx.c` eines Gerätetreibers `xx`

```
init_module () {
  register_chrdev(211, "xx", &xx_fops)
}
cleanup_module () {
  unregister_chrdev(211, "xx");
}
```

  - Funktion `register_chrdev(211,...)` registriert den Treiber mit Major-Nummer `211` für ein zeichenorientiertes Gerät beim Kern

    (⇒ diese Nummer darf zuvor noch nicht vergeben worden sein; alternativ kann Funktion bei Angabe von **0** auch die nächste freie Major-Nummer zurückliefern)
  - Gerätetreiber bekommt Namen `"xx"` (z.B. für System-Fehlermeldungen)
  - `xx_fops` ist eine Struktur vom Typ `file_operations`
  - Funktion `unregister_chrdev(...)` entfernt Treibereintrag aus Kern

7.1 Gerätetreiber unter Linux (5)

- die Struktur **file_operations** enthält Funktionszeiger auf alle Funktionen des Treibers für ein **zeichenorientiertes** Gerät (Struktur ist definiert in **/usr/include/linux/fs.h**)

  Beispiel: Definition eines Funktionszeigerfeldes für Gerätetreiber **xx** in der Datei **xx.c**:

```
static struct file_operations xx_fops = {
  NULL,                 /* Zeiger für Treiberfunktion llseek */
  &xx_read,             /* Zeiger für Treiberfunktion read */
  &xx_write,            /* Zeiger für Treiberfunktion write */
  NULL,                 /* Zeiger für Treiberfunktion readdir */
  NULL,                 /* Zeiger für Treiberfunktion poll */
  &xx_ioctl,            /* Zeiger für Treiberfunktion ioctl */
  NULL,                 /* Zeiger für Treiberfunktion mmap */
  &xx_open,             /* Zeiger für Treiberfunktion open */
  NULL,                 /* Zeiger für Treiberfunktion flush */
  &xx_release,          /* Zeiger für Treiberfunktion release */
  ...
};
```


7.1 Gerätetreiber unter Linux (6)

- im Quelltext des Gerätetreiber sind die Treiberfunktionen aus dem Funktionszeigerfeld zu implementieren

  Beispiel: einige Treiberfunktionen für Treiber **xx** in der Datei **xx.c**
  - **xx_read(...)** Lesen eines Byte-Stroms
  - **xx_write(...)** Schreiben eines Byte-Stroms
  - **xx_open(...)** Öffnen des E/A-Gerätes
  - **xx_release(...)** Freigeben des E/A-Gerätes
  - **xx_ioctl(...)** Absetzen gerätespezifischer Befehle und Einstellen von geräteabhängigen Parametern
  - **xx_poll(...)** Abfragen des Zustandes eines Gerätes
  - **xx_mmap(...)** Abbilden von Gerätespeicher in den Adressraum des Prozessors

7.1 Gerätetreiber unter Linux (7)

- Benutzerprozess kann **korrespondierende Systemfunktionen** aufrufen; die Zuordnung zu den Treiberfunktionen erfolgt durch das Betriebssystem
  - **int open(char* pathname, int openflag, int mode)**
    Öffnen des E/A-Gerätes und Rückgabe des *File Descriptors* **fd**
  - **int read(int fd, void* buf, int count)**
    Lesen von **count** Bytes vom E/A-Gerät **fd** in den Puffer **buf**
  - **int write(int fd, void* buf, int count)**
    Schreiben von **count** Bytes aus Puffer **buf** zum E/A-Gerät **fd**
  - **int close(int fd)**
    Schließen des E/A-Gerätes **fd** und Freigabe für andere Prozesse
  - **int ioctl(int fd, int request, ...)**
    Kontrolle des E/A-Gerätes **fd** durch eine geräteabhängige Anforderung **request**; alle weiteren Parameter sind ebenso von Anforderung und E/A-Gerät abhängig!

7.1 Gerätetreiber unter Linux (8)

- Beispiel: Steuerung eines Gerätes über Leitung 0 des Ports SPP (in C unter Verwendung von **/dev/lp**):

```
#include <stdio.h>
#include <linux/lp.h>
#define OFF 0x00
#define ON 0x01
char data;
fd = open("/dev/lp", O_WRONLY);
if (fd == -1) {
  fprintf(stderr, "Error: cannot open device lp!");
  exit(1);
  }
ioctl(fd, LPRESET);
data = ON; write(fd,&data,1);
sleep(5);
data = OFF; write(fd,&data,1);
close(fd);
```

7.1 Gerätetreiber unter Linux (9)

- Implementierung eines Treibers für **blockorientiertes** Gerät **yy** ist sehr aufwendig
- einige Unterschiede:
  - Hinzufügen eines Treibers erfolgt durch Funktion `register_blkdev`, Entfernen des Treibers aus Kern durch `unregister_blkdev`
  - der Aufruf von `register_blkdev` erfordert als Parameter einen Zeiger auf eine Struktur (definiert in `/usr/include/linux/fs.h`)

```
struct block_device_operations {
  int (*open)(...);
  int (*release)(...);
  int (*ioctl)(...);
  int (*check_media_change)(...);
  int (*revalidate)(...);
};
```

  - Installation eines Interrupt-Handlers `yy_handler` für Interrupt-Nr. `irq` über Systemaufruf

```
int request_irq (int irq, void (*yy_handler)(...),...);
```


7.1 Gerätetreiber unter Linux (10)

- der blockorientierte Gerätetreiber besitzt im Ggs. zum zeichenorientierten Gerätetreiber keine direkten E/A-Funktionen
- Ein-/Ausgabe erfolgt über eine Warteschlange `*queue` mit E/A-Aufträgen, die vom Gerätetreiber initialisiert werden muss:

```
#include <linux/blkdev.h>
blk_init_queue(request_queue_t *queue,
               request_fn_proc *request);
```

- die `request`-Funktion des Block-Gerätetreibers wird vom Betriebssystem aufgerufen, wenn es den Transfer eines Blockes vom/zum E/A-Gerät anfordert

  z.B. aufgrund eines zugehörigen Systemaufrufs eines Benutzerprozesses an ein blockorientiertes Gerät: `write(fd,buf,8192)`
- die `request`-Funktion setzt sämtliche Parameter des Transfers (z.B. Richtung, Pufferadresse, Blockgröße, Anzahl Blöcke, ...)

8 Lernziele

- **Konzepte der Ein-/Ausgabe**:
  - *Busy Waiting*, *Polling* und *Interrupts*
  - Techniken der Datenübertragung (*open loop*, *closed loop*, *fully interlocked*)
  - Adressierung und Programmierung von E/A-Bausteinen
  - DMA
  - synchrone/asynchrone Ein-/Ausgabe
  - serieller Datentransfer (z.B. Datenkodierung bei RS232 und USB)
  - E/A-Systemarchitektur eines heutigen PC
  - Aufgaben und prinzipielle Arbeitsweise eines Gerätetreibers
- **Konzepte von Bussystemen**:
  - Arten und Architektur von Bussystemen
  - Busprotokolle auf synchronem/asynchronem Bus
  - Verfahren der Busarbitrierung
  - wichtige Eigenschaften heute eingesetzter Bussysteme (PCI, USB)