UNIVERSITÄT ZU LÜBECK
STIFTUNGSUNIVERSITÄT
SEIT 2015

Betriebssysteme

Kapitel 4: Eingabe/Ausgabe

Prof. Dr. Stefan Fischer – Institut für Telematik

Motivation

- Für die *Interaktion* mit dem Benutzer hat ein Rechner Ein- und Ausgabegeräte.
  - Tastatur
  - Maus
  - Bildschirm
  - Drucker
  - etc.
- Diese müssen verwaltetet werden und den jeweiligen Prozessen zugeteilt werden.
- Es gibt auch weitere Ein- und Ausgabegeräte zur Kommunikation mit anderen Rechnersystemen z.B. Netzwerkschnittstelle, serielle Schnittstelle, USB.
- Weiterhin ist eine entsprechende *Schnittstelle* vom Betriebssystem bereitzustellen, wie diese Ein- und Ausgabegeräte vom Programmierer angesprochen werden können.

Ziele des Kapitels

- Kommunikation zwischen Ein-/Ausgabegeräten und CPU skizzieren können
- Zugriffsarten (Polling, Interrupt, DMA) erklären können
- Zugriffsarten für Geräte beispielhaft anwenden können
- Ein-/Ausgabe (Input/Output - I/O)-Software-Schichten erklären können

Überblick

- Grundlagen von Ein-/Ausgabe-Hardware und -Software
- Kommunikation zwischen Ein-/Ausgabe-Geräten und der CPU
- Zugriffsarten
  - Polling
  - Interrupt
  - DMA
- I/O-Software-Schichten

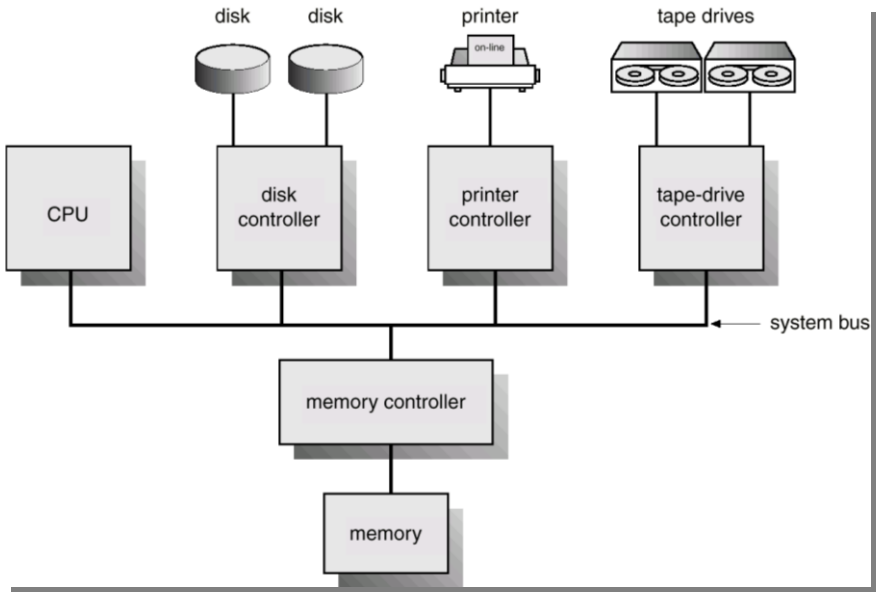
I/O-Hardware – Grundlagen

- I/O-Hardware umfasst alle technischen Komponenten eines Rechensystems, die (Nutzer-)Eingaben erfassen und Daten ausgeben können.
  - Beispiele Input
    - Maus, Tastatur, Touchscreen, Kamera, Mikrofon, Gesten-Controller, etc.
  - Beispiele Output
    - Bildschirm, Soundsystem, Drucker, Force-Feedback, etc.
- In den Anfängen der Rechentechnik hatte das BS die Aufgabe, auch die I/O-Hardware zu verwalten
- Heute sind I/O-Geräte meist selbständig (Blackbox)
- Wichtig für das BS: wie kommen Daten von und zu den Geräten
  - Device Controller
  - Interrupts
  - Memory Mapped I/O

Prinzipieller Aufbau von I/O-Geräten

- I/O–Geräte haben zwei Komponenten:
  - Mechanischer Aufbau
    - Die eigentliche Funktion des Geräts mit Ein- und Ausgabetechnik
  - Elektronik
    - Steuerung der Komponenten
    - Device Controller
- Aufgaben des Controllers
  - Daten der CPU verfügbar machen
  - Konvertierung des seriellen Bitstroms in Datenblöcke
  - Fehlerkorrektur, wenn notwendig

Beispiel:
SCSI Controller



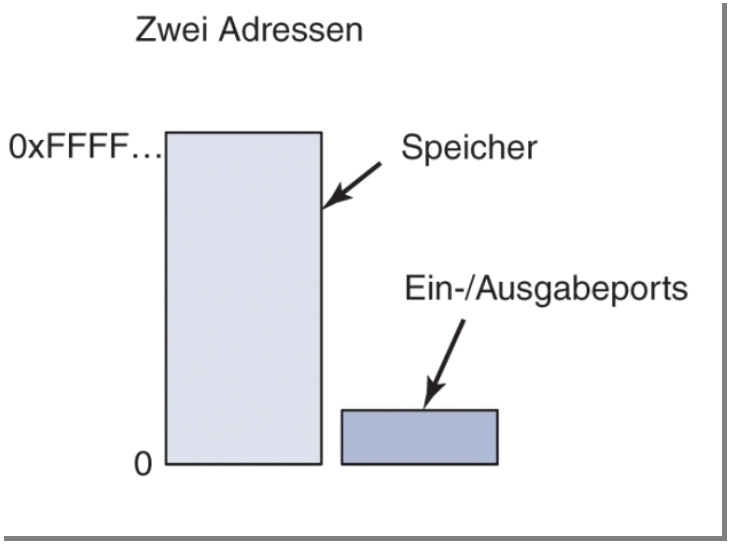
Bildquelle: http://upload.wikimedia.org/wikipedia/commons/1/17/Kontroler_scsi_isa.jpg

Schnittstelle der Device Controller

- Jeder Controller hat einige Steuerungsregister
- CPU kann Befehle mit Parametern in die Register schreiben.
  - Beispiel: Festplatte: Lies Block 10342
- CPU kann Statusinformation aus den Registern lesen.
  - Beispiel: DVD: Datenträger im Laufwerk?
- Zusätzlich haben viele Geräte einen Datenpuffer, der vom BS geschrieben/gelesen werden kann.
  - Beispiel: Videospeicher auf der Grafikkarte

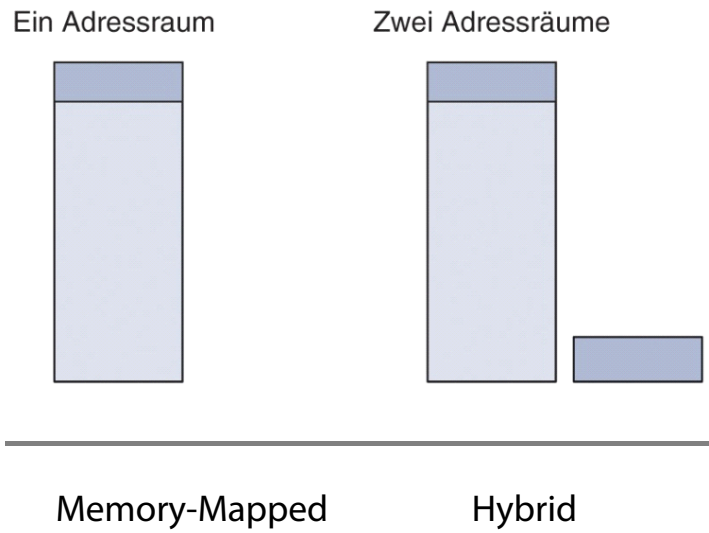
Kommunikation zwischen CPU/BS und Device Controller

- Methode 1: Trennung zwischen I/O-Register und Arbeitsspeicher
  - Der Controller hat speziellen Speicher (I/O ports)
    - Jedes Register hat I/O port number
    - Menger aller Register bildet I/O port space
  - Das BS hat spezielle Instruktionen zum Zugriff
  - Anwendungen haben keinen Zugriff (Schutzfunktion)
  - Datenpuffer wird vom Controller bereit gestellt



Kommunikation zwischen CPU/BS und Device Controller

- Methode 2: *Memory-Mapped*
  - Ein Teil des Hauptspeichers wird für die Geräte reserviert
  - Vorteile:
    - Einheitlicher Zugriff auf Speicher und Geräte
    - Kein Assembler-Code für Registerzugriff notwendig (Treiber in C/C++)
    - Einfacherer Schutz (Benutzerprogramme haben keine Adressen im Speicherbereich)
    - Schneller: Umkopieren von Status-Infos in Hauptspeicher unnötig
  - Nachteil:
    - Caching muss selektiv verhindert werden (Sonst ist Zustand u.U. nicht aktuell)
    - Geräte müssen Zugriff auf Speicher abfangen
- Methode 3: *Hybrid*
  - Pufferspeicher im Hauptspeicher
  - Kontrollregister für I/O-Ports
  - Beispiel: Pentium



Kommunikation zwischen CPU/BS und Device Controller

- Beispiel: Pentium PC-Architektur
  - I/O-Geräte haben keinen Zugriff auf dedizierten (schnellen) Speicherbus
  - PCI-Bridge fängt Zugriffe auf I/O-Port ab und leitet an Geräte weiter
  - Nachteil: Größe des I/O-Port Space muss vorher bekannt sein

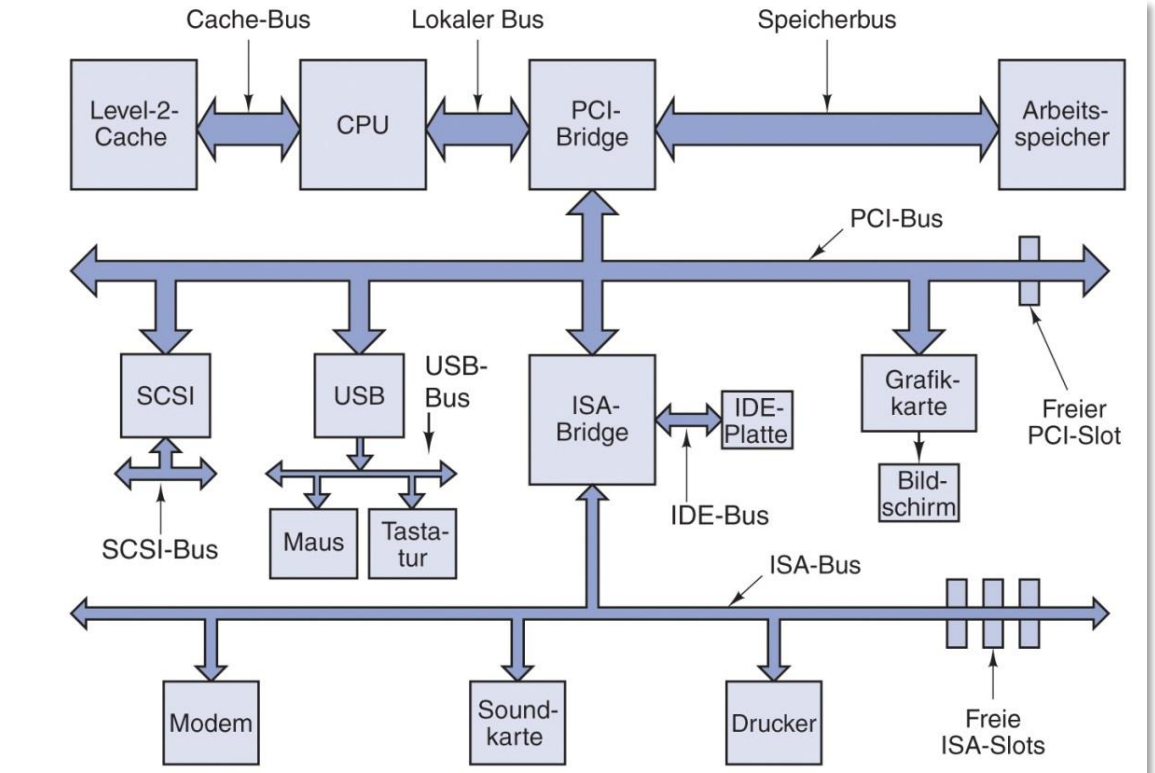


Abbildung 1.12: Der Aufbau eines ausgebauten Pentium-Systems

Zugriff auf Geräte

- Drei verschiedene Verfahren
  - *Polling* – Ständiges Abfragen
  - *Interrupt* - Unterbrechungsroutinen
  - *DMA* - Direct Memory Access

Polling

- Nach dem Absetzen eines Befehls an das Gerät wird der Status ständig abgefragt (busy wait)
- Programmcode im BS:

```
for (int i=0;i<buffer_length;i++){
  while (device_status_register != READY);
  device_data_register = buffer[i];
}
```

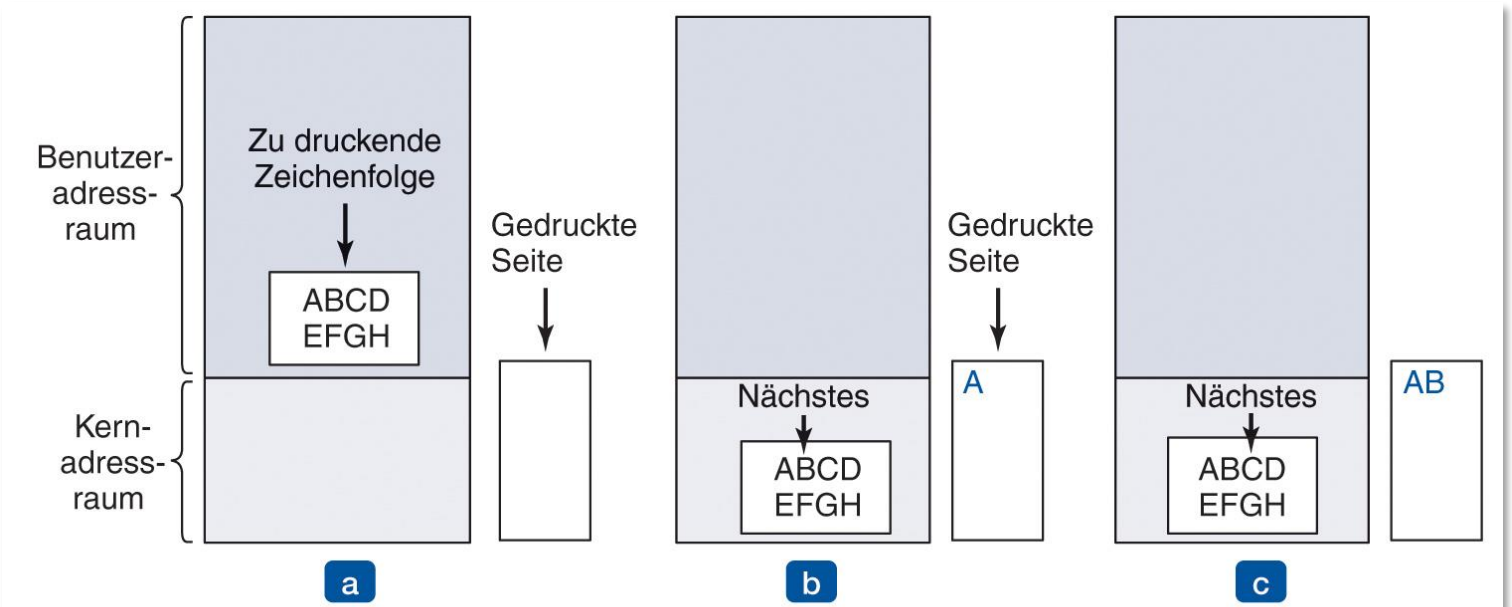


Abbildung 5.7: Die einzelnen Schritte beim Drucken einer Zeichenfolge

Polling

- Alltagsbeispiel:
  - Während der Bearbeitung der Übungsaufgaben für Betriebssysteme bekommst Du Hunger und Du bestellst beim Pizza-Service zwei Pizzen per Telefon (Request), die Du natürlich zusammen essen willst. Der Pizza-Service liefert an Dich aber immer nur 1 Pizza gleichzeitig aus und macht die zweite erst, wenn die erste bei der Auslieferung bezahlt wurde. Außerdem gibt es keine Vorhersage, wann die Pizzen wirklich kommen.
  - 18:25 - 18:30: Bestellung
  - 18:30 – 19:15: Laufe danach ständig zur Tür, um zu schauen, ob schon Pizza angekommen ist.
  - 19:15 – 19:20: Pizzabote ist da. Nehme die 1. Pizza in Empfang bezahle sie.
  - 19:20 – 19:35: Laufe danach weiter ständig zur Tür bis auch die 2. Pizza angekommen ist.
  - 19:35 – 19:40: Pizzabote ist da. Nehme die 2. Pizza in Empfang bezahle sie.
  - 19:40 – 20:00: Essen
  - 20:00 – 21:00: Mache die Übungsaufgaben zu Ende.
  - 21:00: Schalte den Fernseher an.

Interrupts

- Interrupts werden von Geräten verwendet, um das Ende eines Vorgangs anzuzeigen.
- Ablauf:

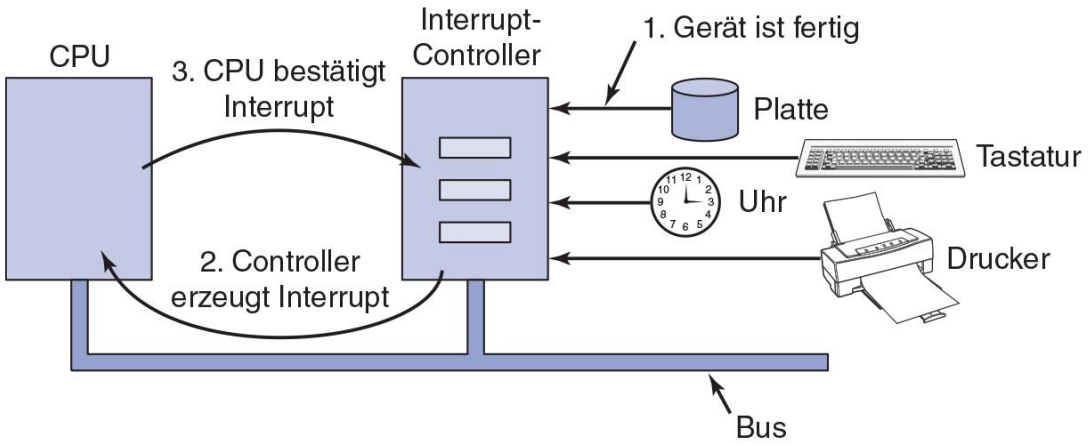


Abbildung 5.5: Ablauf eines Interrupts. Die Verbindung zwischen dem Gerät und dem Controller benutzt statt eigener Leitungen die Interrupt-Leitungen auf dem Bus.

1. Ein Gerät hat Daten zum Abruf.
2. Der Controller sendet einen Interrupt.
3. Die CPU bestätigt den Interrupt, wird unterbrochen und beginnt etwas Neues – abhängig vom auslösenden Gerät wird ein bestimmter Interrupt-Handler aufgerufen und ausgeführt.

– Später macht die CPU an der „alten“ Stelle weiter.

Interrupts

- Alltagsbeispiel (Zweite Version)
  - 18:25 – 18:30: Bestellung
  - 18:30 – 19:15: Da du noch fit bist, mache weiter mit den Übungsaufgaben.
  - 19:15 – 19:20: Türklingel läutet: Pizzabote ist da; nimm die 1. Pizza in Empfang bezahle sie.
  - 19:20 – 19:35: Da du immer noch fit bist, mache die Übungsaufgaben zu Ende.
  - 19:35 – 19:40: Türklingel läutet: Pizzabote ist da; nimm die 2. Pizza in Empfang bezahle sie.
  - 19:40 – 20:00: Essen
  - 20:00: Schalte den Fernseher an und schaue die Tagesschau.

Direct Memory Access (DMA)

- I/O kann mittels DMA deutlich beschleunigt werden, da die CPU weniger belastet ist
- Prinzip:
  - I/O-Aufträge werden an DMA-Controller (eigener Chip) weitergereicht
  - Bis zum Ende des Auftrags kann die CPU andere Dinge erledigen

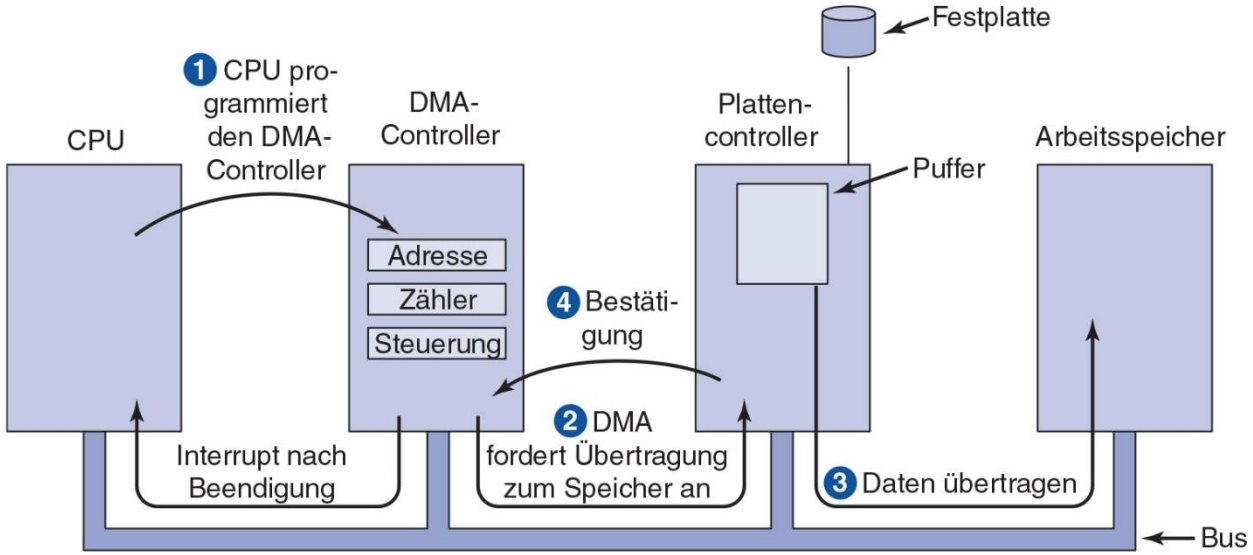


Abbildung 5.4: Ablauf eines DMA-Transfers.

Direct Memory Access (DMA)

- Alltagsbeispiel (Dritte Version)

- 18:25 – 18:30: Sag Deiner Mutter (DMA-Controller) Bescheid, dass sie Pizza bestellen soll.
- 18:30 – 19:30: Da du noch fit bist, mache die Übungsaufgaben zu Ende.
- 19:30 – 19:35: Gehe ins Internet, lies ein paar E-mails und schreibe Deinen Kommilitonen, dass Du schon fertig bist.
- 19:35 – 19:55: Essen
- 19:55: Schalte den Fernseher an und schaue vor der Tagesschau noch die Börsennachrichten.
- 18:30 – 18:35: Deine Mutter bestellt die Pizza. Da sie aber immer zuverlässig bezahlt, bittet sie den Pizzaboten die Pizza in die Ablage zu werfen und erst zu klingeln, wenn die zweite Pizza da ist. Natürlich kann der Pizza-Service wie immer den Betrag vom Konto abbuchen.
- 19:20 : Erste Pizza kommt an und wird eingeworfen.
- 19:35: Türklingel läutet: Deine Mutter nimmt beide Pizzen aus der Ablage und klopft an die Tür.

I/O-Software – Grundlagen

- I/O-Software soll vor allem die Komplexität der Hardware vor dem BS bzw. dem Anwendungsprogrammierer verbergen.
- Wichtige Prinzipien daher:
  - Geräteunabhängigkeit: ein „write“ funktioniert auf lokal angeschlossener Festplatte genauso wie auf das Netzwerk
  - Einheitliche Namensverwendung (alle Geräte sind bspw. über Pfade im Dateisystem erreichbar)
  - Fehlerbehandlung so nah wie möglich an der Quelle (Hardware)

Schichten der I/O-Software

- Ein-/Ausgabe-Software wird üblicherweise in vier Schichten eingeteilt.

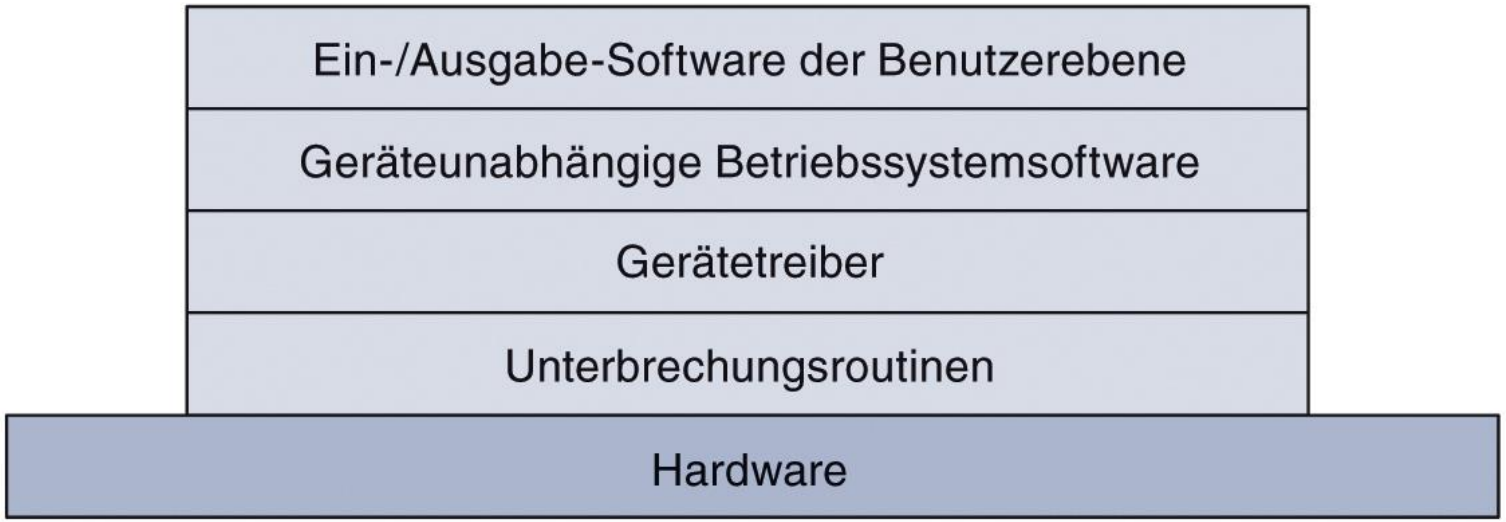


Abbildung 5.11: Schichten des Ein-/Ausgabe-Softwaresystems

Gerätetreiber – Device Driver

- Aufgabe: Verbergen der Komplexität der Registerbelegungen des Controllers
- Jedes Gerät benötigt üblicherweise seinen eigenen Device Driver
- Treiber sind praktisch immer Teil des Kernels
- API für die darüber liegende Schicht: read- und write-Anfragen

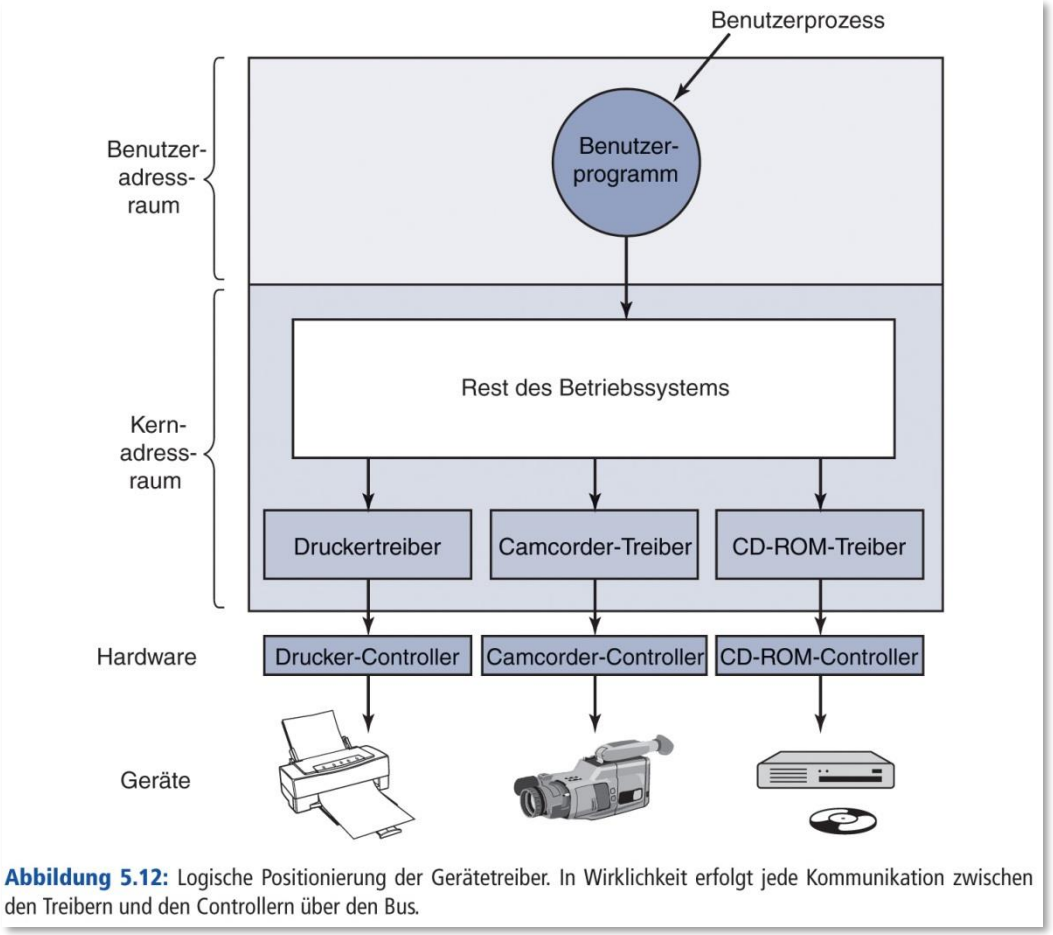


Abbildung 5.12: Logische Positionierung der Gerätetreiber. In Wirklichkeit erfolgt jede Kommunikation zwischen den Treibern und den Controllern über den Bus.

Geräteunabhängige I/O-Software

- Wichtige Aufgaben:
  - Bereitstellen einheitlicher Schnittstellen für Gerätetreiber erleichtert den Einsatz neuer Geräte
  - Puffern von verfügbaren Daten
    - Vorteil: erhöht die Performance (kein Interrupt pro ankommendes Datum, sondern pro Block)
    - Problem: zu langes Liegen von Daten im Puffer, zu viele Kopien zwischen Speicherbereichen (Kernel, User Space, Gerät): schlechte Performance
  - Geräteunabhängige Fehlerbehandlung
  - Spezielle Behandlung dedizierter Geräte (CD-ROM) z.B. durch explizites „open“
  - Bereitstellung einer einheitlichen Blockgröße

Beispiel: einheitliche Schnittstellen

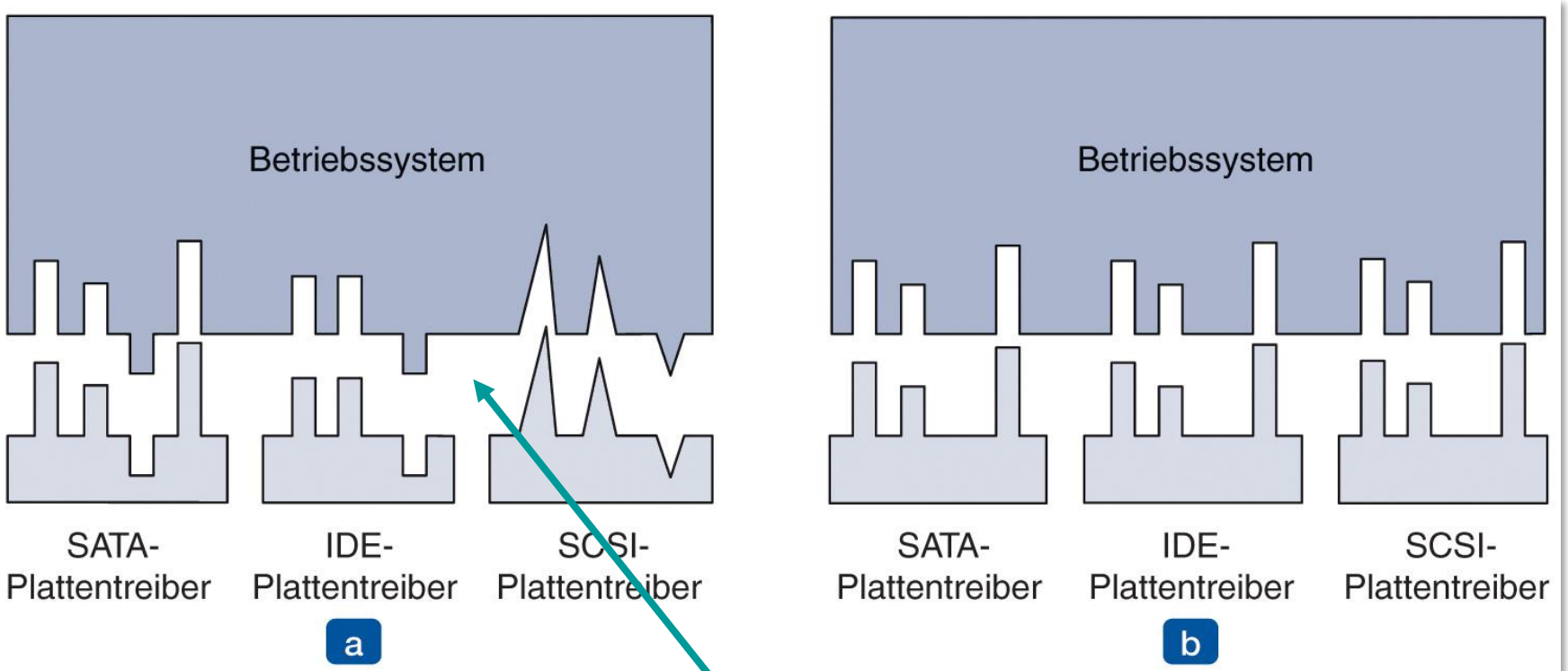


Abbildung 5.14: (a) Ohne eine Standardschnittstelle für Treiber (b) Mit einer Standardschnittstelle für Treiber

Muss für jedes neue Gerät
wieder neu programmiert werden

Buffering als Performance-Fresser

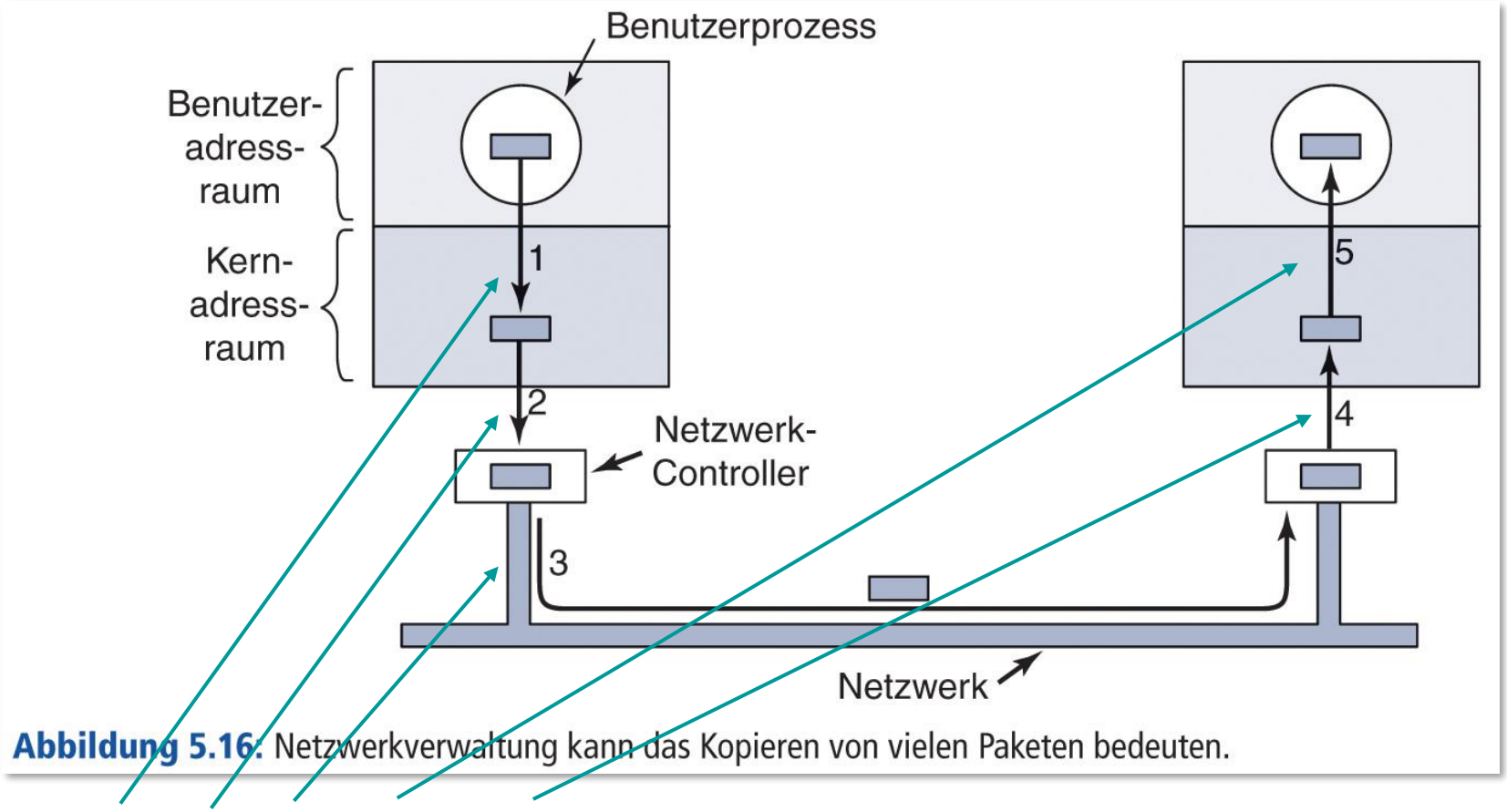


Abbildung 5.16: Netzwerkverwaltung kann das Kopieren von vielen Paketen bedeuten.

Häufiges Kopieren kostet Zeit!

I/O-Software im User Space

- Teil der I/O-Software läuft in User Space:
  - Bibliotheken, die mit den Anwendungsprogrammen gelinkt werden
    - Beispiele:
      - write, read, open, close
      - printf()
    - Zum Teil einfach nur Abbildung auf die entsprechenden Systemaufrufe, zum Teil aber auch Aufgaben wie Formatierung der Daten (printf)
- Spooling-Programme
  - Integration exklusiv benutzbarer Geräte in ein Multiprogrammiersystem
  - Beispiel Drucker:
    - Anwenderprogramme drucken in spezielles spooling directory
    - Hintergrund-Daemons druckt die Spooler-Dateien, da nur er Zugriffsberechtigungen auf den Drucker-Controller besitzt.

Zusammenfassung des Kapitels

- Kommunikation: IO-Register, Memory Mapped, Hybrid
- Zugriff: Polling, Interrupt, DMA
- IO-Schichten (s.u.)

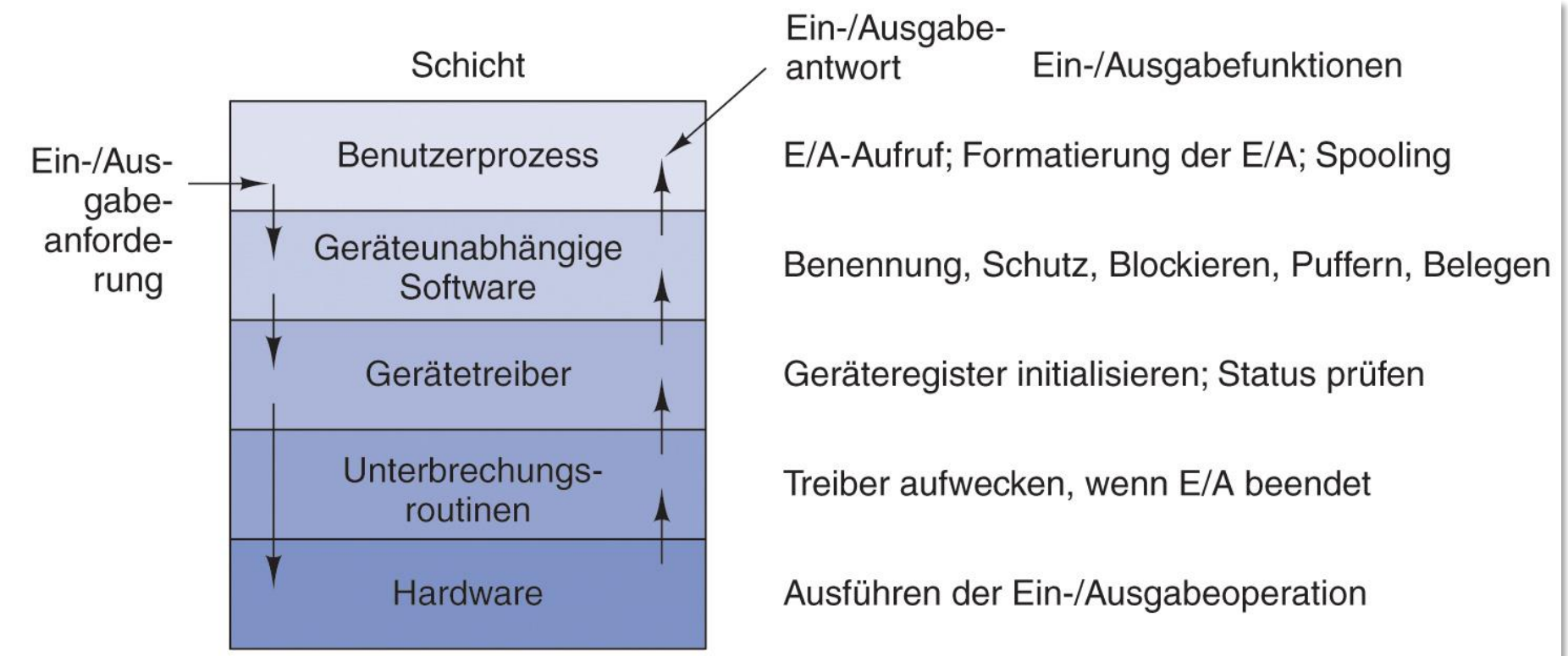


Abbildung 5.17: Schichten des Ein-/Ausgabesystems und die Hauptfunktion jeder Ebene

Kontakt

Prof. Dr. Stefan Fischer
Direktor des Instituts für Telematik

Universität zu Lübeck
Ratzeburger Allee 160
23562 Lübeck
Tel: +49 451 3101 6400
Email: stefan.fischer@uni-luebeck.de