BETRIEBSSYSTEME
EIN- UND AUSGABE

Prof. Dr. Jörg Mielebacher
mail@mielebacher.de

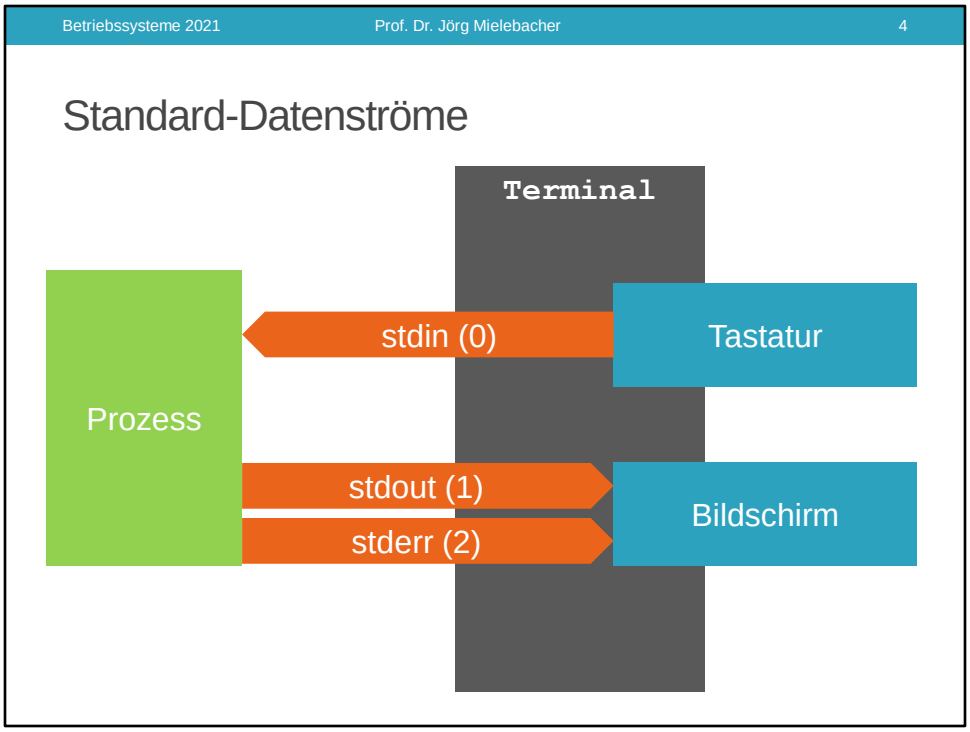
Die folgenden Folien führen in die Verwaltung des Hauptspeichers durch das Betriebssystem ein. Zu jeder Folie sind Notizenseiten erfasst.

Verbesserungsvorschläge und Fehlerhinweise können Sie gerne an die Adresse mail@mielebacher.de senden.

Wiederholung

- Bei der Speichernutzung beobachtet man zeitliche und räumliche Lokalität.
- Die Speicherhierarchie vereint Technologien mit unterschiedlichen Zugriffszeiten und Speichervolumina.
- Aktuelle Betriebssysteme stellen jedem Prozess einen virtuellen Adressraum zur Verfügung und nutzen Paging.
- Die MMU übernimmt mit Unterstützung des TLB das Mapping zwischen virtuellen und realen Adressen durch Seitentabellen, teilweise mit invertierten Seitentabellen.
- Ein Segmentation Fault tritt bei unerlaubten Zugriffen auf virtuelle Adressen auf.
- Ein Page Fault tritt auf, wenn eine Seite noch nicht in den Hauptspeicher geladen ist.
- Es gibt verschiedene Ersetzungsstrategien (z.B. NRU, LRU, FIFO, NFU).

Standard-Datenströme



Beim Stichwort Ein- und Ausgabe kommen einem vermutlich zuerst Tastatur und Bildschirm in den Sinn. Dieser Gedanken ist wiederum eng verbunden mit dem Prinzip der Standard-Datenströme, die von Prozessen genutzt werden können. In Linux und anderen Betriebssystemen gibt es meist drei Standard-Datenströme, die Standardeingabe stdin, die Standardausgabe stdout und die Standardfehlerausgabe stderr. Stdin ist üblicherweise mit der Tastatur verbunden; stdout und stderr sind mit dem Bildschirm verbunden.

Standard-Datenströme in /dev

```
student ~ ls /dev/
agpgart          kmsg          sda2      tty25  tty53      ttyS22  vcsa1
autofs           lightnvm      sda5      tty26  tty54      ttyS23  vcsa2
block            log           sda6      tty27  tty55      ttyS24  vcsa3
bsg              loop0         sg0       tty28  tty56      ttyS25  vcsa4
btrfs-control    loop1         sg1       tty29  tty57      ttyS26  vcsa5
bus              loop2         shm       tty3   tty58      ttyS27  vcsa6
cdrom            loop3         snapshot  tty30  tty59      ttyS28  vcsa7
cdrw             loop4         snd       tty31  tty6       ttyS29  vcsu
char             loop5         sr0       tty32  tty60      ttyS3   vcsu1
console          loop6         stderr    tty33  tty61      ttyS30  vcsu2
core             loop7         stdin     tty34  tty62      ttyS31  vcsu3
cpu_dma_latency  loop-control  stdout    tty35  tty63      ttyS4   vcsu4
cuse             mapper        tty       tty36  tty7       ttyS5   vcsu5
disk             mcelog        tty0      tty37  tty8       ttyS6   vcsu6
dm-0             mem           tty1      tty38  tty9       ttyS7   vcsu7
dm-1             midi          tty10     tty39  ttyprintk  ttyS8   vfio
dm-2             mqueue        tty11     tty4   ttyS0      ttyS9   vga_arbiter
dmmidi           net           tty12     tty40  ttyS1      udmabuf vgmint
dri              null          tty13     tty41  ttyS10     uhid    vhci
dvd              nvram         tty14     tty42  ttyS11     uinput  vhost-net
ecryptfs         port          tty15     tty43  ttyS12     urandom vhost-vsock
fb0              ppp           tty16     tty44  ttyS13     userio  vmci
fd               psaux         tty17     tty45  ttyS14     vcs     vsock
```

Stdin, stdout und stderr sind unter Linux als Dateien in /dev erreichbar. Stdin hat den Dateideskriptor 0, stdout 1 und stderr 2. Sie fügen sich damit in das unter Linux übliche Bild ein, dass alles eine Datei ist.

Beispiel

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    // Read from stdin and write it to stdout and stderr
    string line;
    getline( cin, line );

    cout << "stdout: " << line << endl;
    cerr << "stderr: " << line << endl;

    return 0;
}
```

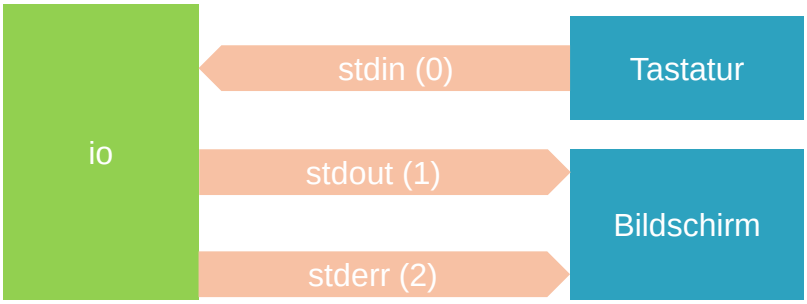
cin	stdin
cout	stdout
cerr	stderr

Das folgende C++-Programm zeigt, wie sich die drei Standard-Datenströme verwenden lassen. C++ stellt hierfür den Eingabestream cin (stdin) sowie die Ausgabestreams cout (stdout) und cerr (stderr) zur Verfügung. Für gewöhnliche Ausgaben verwendet man cout, für Fehlermeldungen kann man cerr verwenden, auch wenn beide i.d.R. auf dem Bildschirm dargestellt werden. Allerdings gibt es Möglichkeiten die Standard-Datenströme umzuleiten, um beispielsweise Fehlermeldungen in einer Datei zu protokollieren – das wollen wir uns nun im Folgenden unter Linux anschauen.

In C liest scanf von stdin, printf schreibt zu stdout. Da stdin, stdout, stderr als Dateizeiger angelegt sind, kann man aber auch z.B. fprintf(stdout, "…") verwenden bzw. damit auf stderr schreiben, also fprintf(stderr, "…").

Beispiel

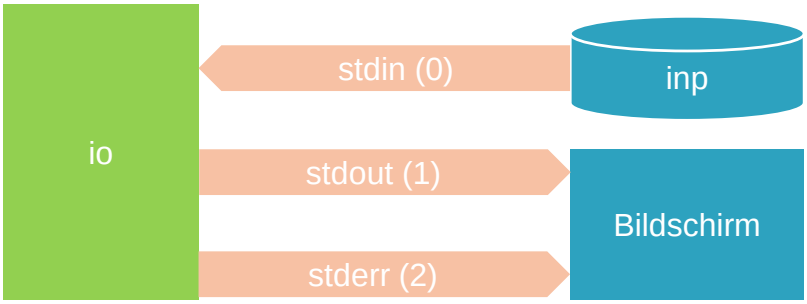
```
student ~ ./io
Zeile von Tastatur
stdout: Zeile von Tastatur
stderr: Zeile von Tastatur
```



Ruft man das Programm io ohne weitere Umleitungen auf, so liest cin (also stdin) eine Zeile von der Tastatur und gibt diese per cout (stdout) und cerr (stderr) auf dem Bildschirm aus.

Beispiel

```
student ~ ./io < inp
stdout: Zeile aus Datei
stderr: Zeile aus Datei
```

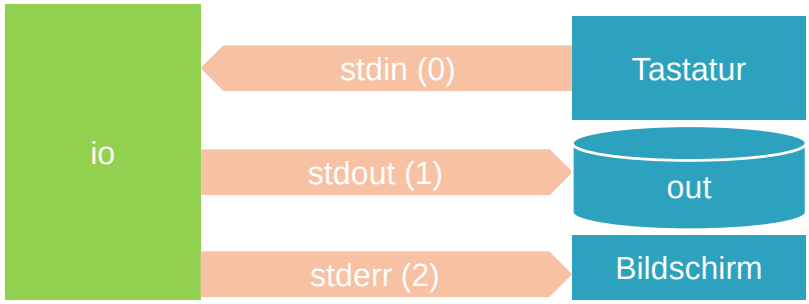


Möchte man die Eingabe aus einer Datei lesen, so kann man hierfür den <-Operator verwenden. stdin erhält in diesem Fall die Eingabe nicht von der Tastatur, sondern aus der Datei inp. Die Ausgabe erfolgt aber weiterhin auf dem Bildschirm.

Diese Art der Umleitung ist praktisch, wenn eine interaktive Nutzung automatisiert werden soll und sich z.B. aus einer Datei speisen soll.

Beispiel

```
student ~ ./io > out
Zeile von Tastatur
stderr: Zeile von Tastatur
student ~ cat out
stdout: Zeile von Tastatur
```



Häufig möchte man die Ausgabe eines Prozesses in einer Datei speichern. Hierfür kann man stdout mit dem >-Operator in eine Datei umleiten. Alles, was sonst über stdout auf dem Bildschirm angezeigt worden wäre, wird jetzt in die Datei (hier: out) geschrieben.

Auf diese Weise könnte man sich beispielsweise mit history > befehle.log die zuletzt in der Shell eingegebenen Befehle in eine Textdatei schreiben lassen.

Beispiel

```
student ~ ./io > out
1. Zeile von Tastatur
stderr: 1. Zeile von Tastatur
student ~ cat out
stdout: 1. Zeile von Tastatur
student ~ ./io >> out
2. Zeile von Tastatur
stderr: 2. Zeile von Tastatur
student ~ cat out
stdout: 1. Zeile von Tastatur
stdout: 2. Zeile von Tastatur
```

> Datei neu schreiben

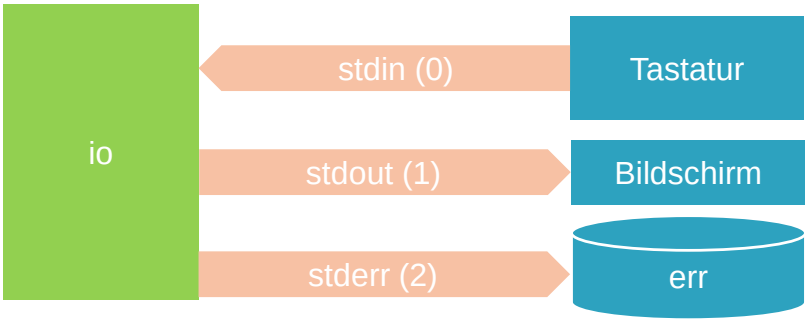
>> An Datei anhängen

In dem hier gezeigten Beispiel sieht man auch, dass der >-Operator jeweils die Datei neu schreibt; ist sie nicht vorhanden, wird sie neu erstellt, ist sie bereits vorhanden, wird der Inhalt gelöscht und sie neu geschrieben (je nach Systemeinstellung muss man dies ggf. mit >! erzwingen).

Möchte man die Ausgaben jedoch an den vorhandenen Dateiinhalt anhängen, so muss man den >>-Operator verwenden. Das kann nützlich sein, wenn man die Ausgaben mehrerer Befehle in eine Datei schreiben möchte.

Beispiel

```
student ~ ./io 2> err
Zeile von Tastatur
stdout: Zeile von Tastatur
student ~ cat err
stderr: Zeile von Tastatur
```

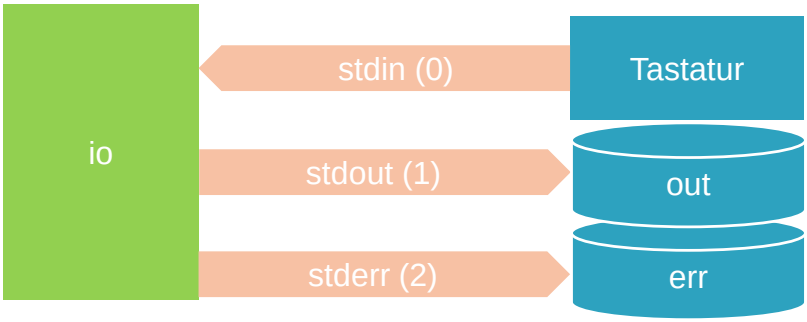


Nutzt man den >-Operator, bezieht er sich auf stdout. Möchte man stderr umleiten, so muss man 2> verwenden. In gleicher Weise könnte man auch 1> für stdout nutzen.

Das Umleiten von stderr ist besonders nützlich, wenn man Batch-Jobs automatisiert (z.B. per Cronjob) ablaufen lässt und die entstehenden Fehlermeldungen in einer Datei protokollieren möchte. Der >>-Operator stünde hierfür in gleicher Weise zur Verfügung.

Beispiel

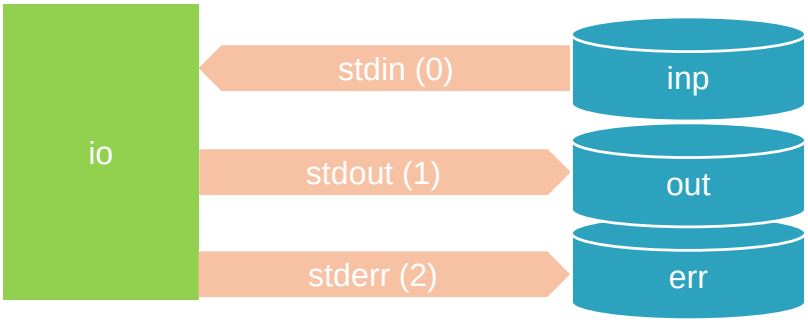
```
student ~ ./io 1> out 2> err
Zeile von Tastatur
student ~ cat out
stdout: Zeile von Tastatur
student ~ cat err
stderr: Zeile von Tastatur
```



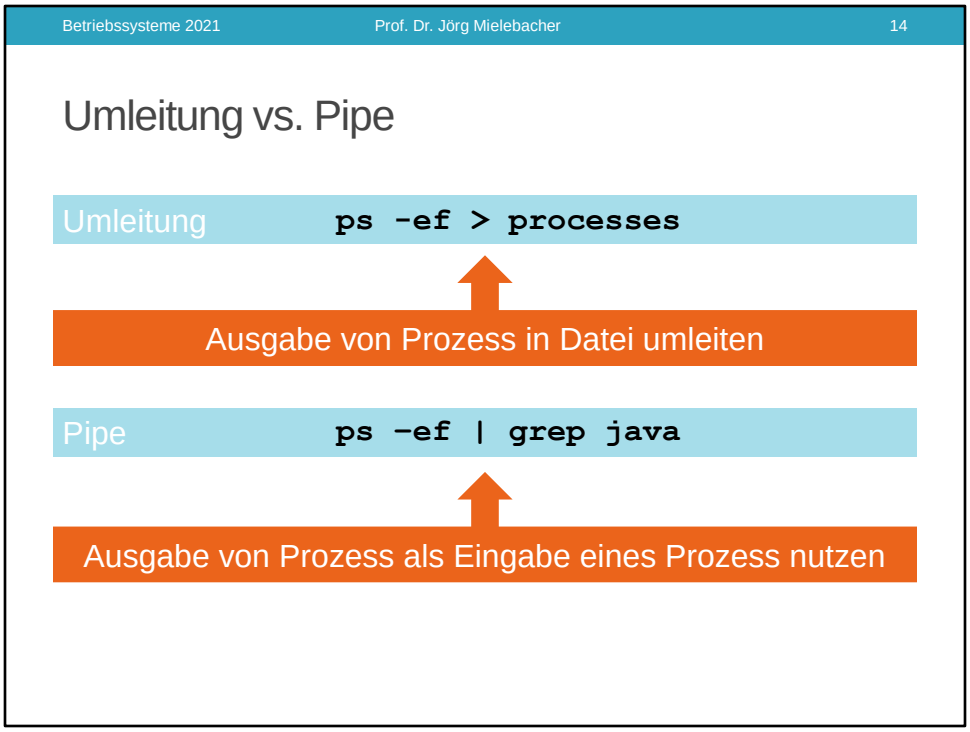
Man kann auch mehrere Umleitungen gemeinsam verwenden; hier werden stdout und stderr jeweils in eine Datei umgeleitet, während stdin weiterhin von der Tastatur liest.

Beispiel

```
student ~ ./io < inp 1> out 2> err
student ~ cat out
stdout: Zeile aus Datei
student ~ cat err
stderr: Zeile aus Datei
```



Natürlich kann man auch alle drei Standard-Datenströme zur gleichen Zeit umleiten. stdin erhält hier die Eingaben aus der Datei inp, während stdout in out umgeleitet wird und stderr in err.



Wenn Sie an die Interprozesskommunikation denken, dürften Ihnen der Gedanke, Ausgaben eines Prozesses zu Eingaben eines anderen zu machen, nicht fremd sein – hierfür nutzt man Pipes. Dennoch verfolgen Stream-Umleitungen (<,>,>>) und Pipes (|) zwei unterschiedliche Ziele:
Bei Umleitungen geht es darum, die Standard-Datenströme auf Dateien zu lenken, d.h. aus Dateien zu lesen oder in sie zu schreiben; man beschreibt also eine Verbindung zwischen Prozess und Datei. Bei den Pipes beschreibt man eine Verbindung zwischen zwei Prozessen, wobei die Ausgabe des einen Prozesses (hier: ps) zur Eingabe des anderen Prozesses (hier: grep) wird.

Aufgabe

- Wie könnte man unter Linux eine Pipe zwischen den Prozessen a und b durch das Umleiten von Standard-Datenströmen ersetzen?
- Was muss man machen, wenn eine Ausgabe unterdrückt werden soll?

Normalerweise könnte man die Ausgabe von a direkt an b weiterreichen, indem man a | b schreibt. Ohne eine Pipe könnte man z.B. erst a > temp eingeben und dann b < temp, oder zusammen a > temp && b < temp, wobei temp eine les- und schreibbare Datei im aktuellen Verzeichnis ist. Vorteil der Pipe: Sie erfordert keine Datei im Dateisystem, sondern nutzt direkt die vom Betriebssystem bereitgestellten Pipes.

Soll eine Ausgabe unterdrückt werden, ist es unter Linux am einfachsten, als Datei /dev/null zu verwenden. Alternativ könnte man in eine Datei schreiben und sie danach direkt löschen, also a > temp && rm –f temp; was aber eher umständlich und unüblich ist.

Und Windows?

```
C:\Users\jm\Documents\Projekte\Vorlesungen\BS-MOS\Material>io
Zeile von Tastatur
stdout: Zeile von Tastatur
stderr: Zeile von Tastatur

C:\Users\jm\Documents\Projekte\Vorlesungen\BS-MOS\Material>io > out
Zeile von Tastatur
stderr: Zeile von Tastatur

C:\Users\jm\Documents\Projekte\Vorlesungen\BS-MOS\Material>io < inp
stdout: Zeile aus Datei
stderr: Zeile aus Datei

C:\Users\jm\Documents\Projekte\Vorlesungen\BS-MOS\Material>io 2>err
Zeile von Tastatur
stdout: Zeile von Tastatur

C:\Users\jm\Documents\Projekte\Vorlesungen\BS-MOS\Material>io < inp 1>out 2>err

C:\Users\jm\Documents\Projekte\Vorlesungen\BS-MOS\Material>
```

Ebenfalls möglich: <, >, >>, |, &&

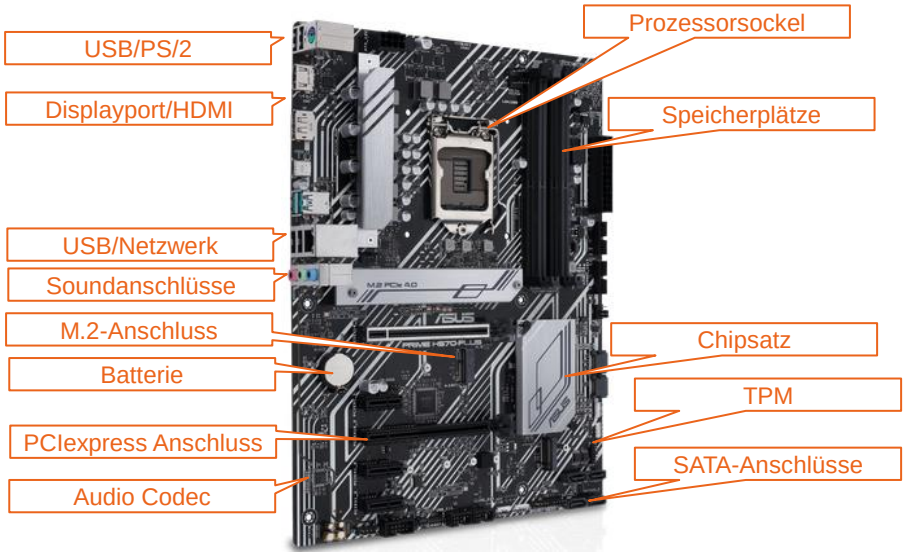
Wie zu sehen ist, funktioniert das Umleiten der Standard-Datenströme in gleicher Weise auch unter Windows, auch die anderen Operatoren sind verfügbar - >> für das Anhängen, | als Pipe und && um zwei Befehle nacheinander auszuführen (unter der Bedingung, dass der erste erfolgreich war).

Soll eine Ausgabe ins Leere laufen, verwendet man unter Windows statt /dev/null das Ziel nul, also z.B. a > nul, um die Ausgaben von a zu unterdrücken (die Fehlermeldungen bleiben sichtbar).

Das Ein- und Ausgabe-Subsystem

Das Ein- und Ausgabe-Subsystem des Betriebssystems ist sein zentraler Bestandteil, i.d.R. umfasst es die meisten Codezeilen. Schätzungen für Linux sprechen von über 60%.

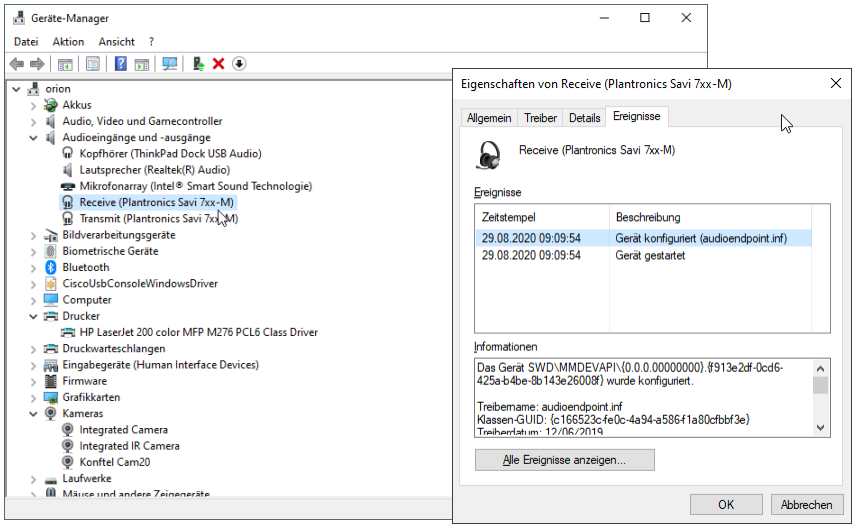
Ein Blick auf das Motherboard



Asus Prime H570 Plus, www.asus.com

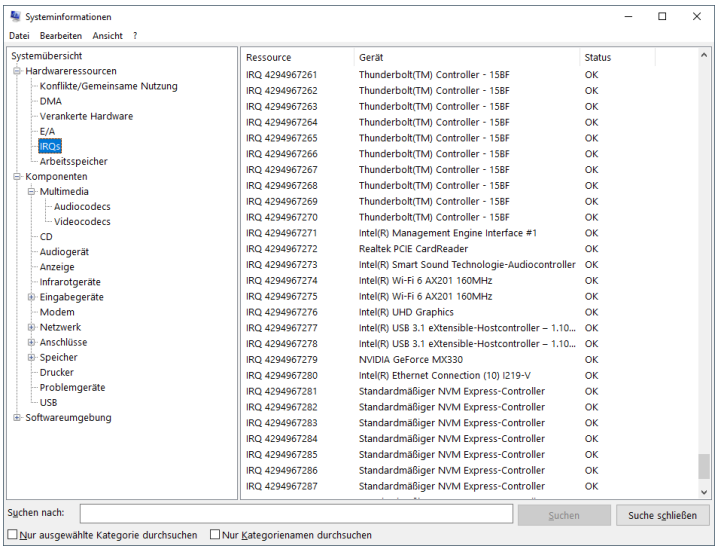
Der Blick auf eine typische Hauptplatine (Motherboard) eines Rechners zeigt, wie vielfältig die daran angeschlossenen Geräte sind: Neben dem Prozessorsockel (nebst leistungsangepasster Kühlung) und den Speicherplätzen befindet sich darauf der zum Prozessor passende Chipsatz, der die verschiedenen Schnittstellen usw. zur Verfügung stellt. Ein sog. Trusted Platform Module (TPM), das Passwörter speichern und Verschlüsselung unterstützen kann, gehört heute ebenfalls zum Standard. Über SATA-Anschlüsse lassen sich Festplatten oder interne, optische Laufwerke anschließen. Für die Soundausgabe sind Audio Codecs vorhanden. Über den PCIexpress-Bus lassen sich Erweiterungen anbinden. Eine Batterie puffert die interne Uhr usw., wenn der Rechner ausgeschaltet ist. Über M.2-Anschlüsse kann man SSD-Festplatten und andere Erweiterungen anschließen. Soundanschlüsse, meist 3.5mm-Klinkenbuchsen, erlauben es, Mikrofone oder Lautsprecher bzw. Kopfhörer anzuschließen. USB-Anschlüsse werden für vielfältige Zwecke (USB-Sticks, Maus, Tastatur, Webcam usw.) verwendet. Für das kabelgebundene Netzwerk ist ein RJ45-Anschluss vorhanden. Außerdem sie für den Anschluss eines Monitors Displayport- und HDMI-Anschluss vorhanden. Für Maus und Tastatur existiert neben USB-Anschlüssen auch noch der veraltete PS/2-Anschluss.

Der Geräte-Manager von Windows



Dass das Betriebssystem die Hardware verwaltet, hattet wir bereits mehrfach festgehalten. Praktisch sehen kann man dies unter Windows z.B. mit dem Geräte-Manager, der die angeschlossenen Geräte nach Kategorien auflistet und es erlaubt, Details z.B. zu verwendeten Gerätetreibern anzuschauen.

Alternative: msinfo32



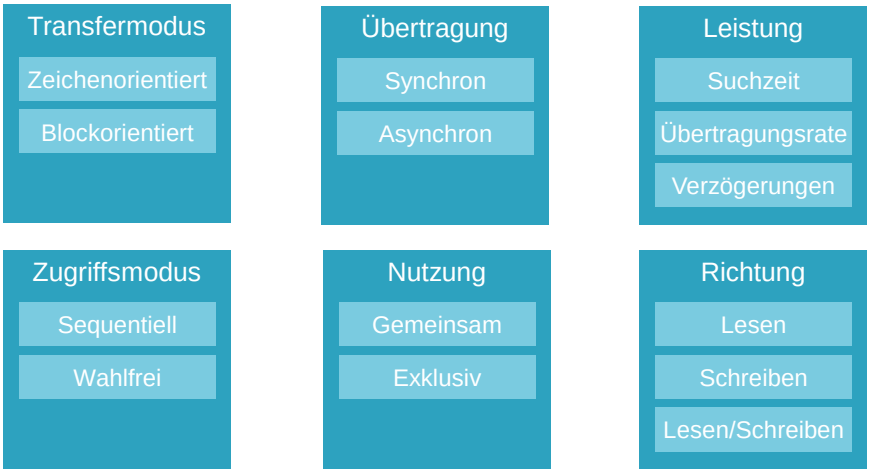
Eine Alternative stellt msinfo32 dar, das weitere EInblicke bietet, z.B. genutzte Speicherbereiche, Interrupts usw.

Geräte-Informationen unter Linux

- Geräte am PCI-Bus: `lspci`
- USB-Geräte: `lsusb`
- Festplatten/Partitionstabellen: `fdisk -l`
- Blockorientierte Geräte: `lsblk`
- Festplattennutzung: `df`
- Inhalte von `/dev` bzw. `/proc`
- u.v.m.

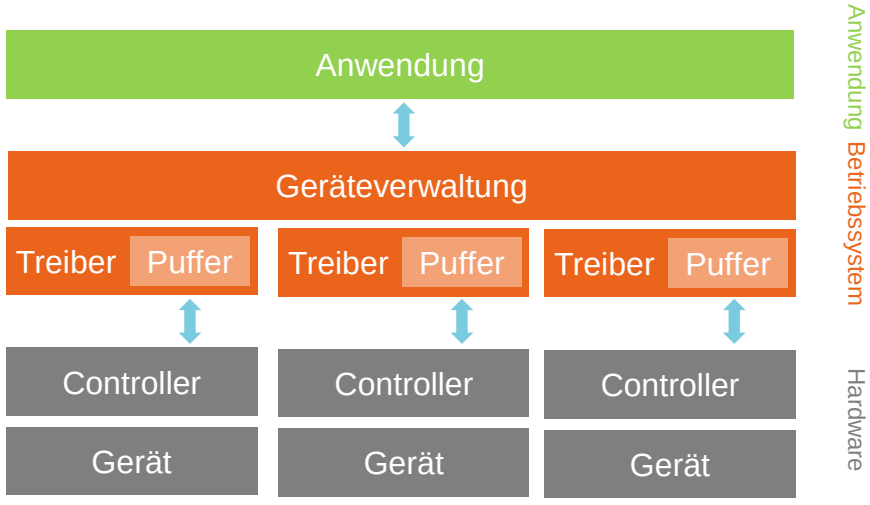
Unter Linux gibt es verschiedene Möglichkeiten, Informationen zu angeschlossenen Geräte zu erhalten. Die am internen PCI-Bus angeschlossenen Geräte (z.B. Grafikkarten usw.) lassen sich mit lspci anzeigen. Für USB-Geräte nutzt man meist lsusb. Informationen zu den Festplatten und Partitionen erhält man mit fdisk –l bzw. allgemein alle blockorientierten Geräte mit lsblk. Informationen zur Belegung der Festplatten liefert df. Außerdem befinden sich in /dev alle Geräte und in /proc zahlreiche Informationen zum Systemzustand und zu den Geräten. Daneben gibt es zahlreiche weitere Möglichkeiten, z.B. das Paket hwinfo, mit dem sich mit einem Befehl alle Hardwareinformationen zusammenstellen lassen.

Unterschiede von Geräte



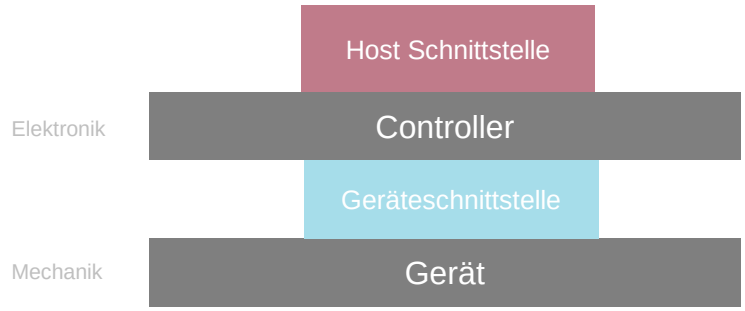
Die Vielfalt der Geräte kann man auch kategorisieren: Es unterscheidet Geräte…
… die Daten zeichenweise (Tastatur) oder in Form von Datenblöcken übertragen (Festplatte),
… die sequentiell (Bandlaufwerk) oder wahlfrei (Festplatte) zugegriffen werden,
… die Daten synchron (Bandlaufwerk) oder asynchron (Tastatur) übertragen,
… die gemeinsam oder exklusiv genutzt werden,
… mit unterschiedlichen Suchzeiten, Übertragungsraten und Verzögerungen,
… die lesend (CD), schreibend (Bildschirm) oder lesen und schreibend (Festplatte) verwendet werden.

Ein- und Ausgabe



Jedes Gerät, das mit dem Computer verbunden ist, besitzt einen Controller, der für die Ansteuerung des Geräts zuständig ist. Beides zusammen bildet die Hardware-Ebene. Das Betriebssystem wiederum nutzt Treiber, um mit dem Controller zu kommunizieren. Diese Treiber nutzen i.d.R. Puffer für die Kommunikation – dazu gleich mehr. Der Treiber wiederum kommuniziert auf der anderen Seite mit der Geräteverwaltung, die ihrerseits den Anwendungen, die im User Mode ablaufen, über Systemaufrufe die Nutzung der Hardware ermöglichen. Die Anwendungen können aber stets nur mit dem Betriebssystem interagieren, nicht direkt mit der Hardware.

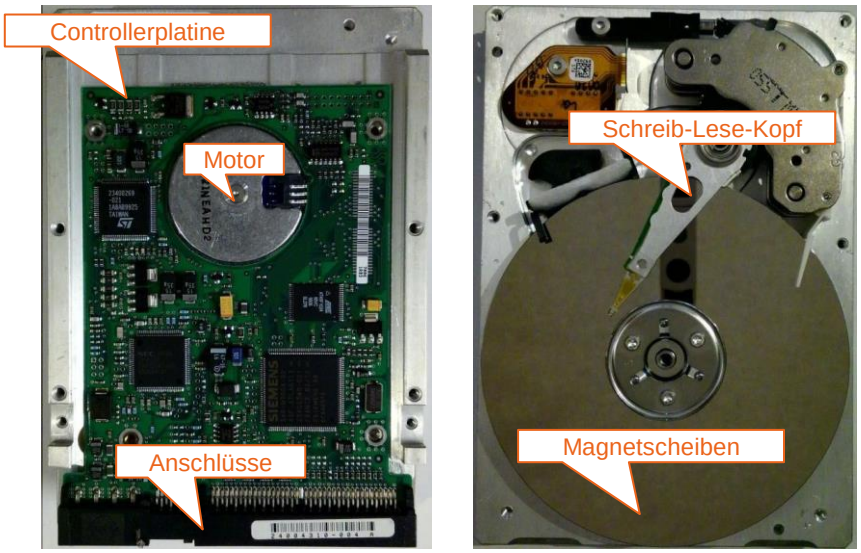
Controller



Typischerweise kann man bei Geräten den eher mechanischen Anteil (z.B. Druckkopf und Walzen) vom elektronischen Anteil unterscheiden. Der Controller entspricht dabei dem elektronischen Teil – er ist über eine Geräteschnittstelle mit dem Gerät verbunden und ist in Richtung der CPU über die Host-Schnittstelle angebunden.

Während über die Host-Schnittstelle vorwiegend abstrakte Befehle an den Controller gelangen, werden über die Geräteschnittstelle die gerätespezifischen Ansteuerungen vorgenommen. So könnte beispielsweise an ein Display die Anweisung gelangen, das Zeichen Z an die Koordinate X|Y zu schreiben. Das An- und Ausschalten der einzelnen Bildpunkte übernimmt dann der Controller.

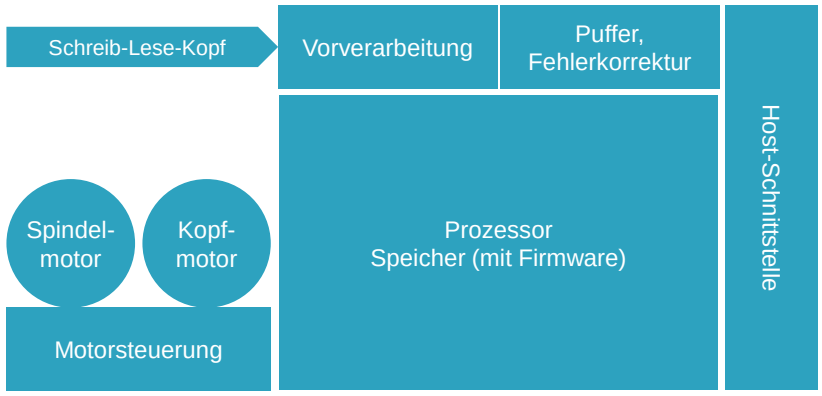
Beispiel: Festplatte



Bei Festplatten ist der Controller heutzutage fest verbaut. Die hier gezeigte 3.5“-Festplatte von Seagate hat eine Größe von 14.2 GB und ist über 20 Jahre alt. Man kann an ihr aber die wichtigsten Baugruppen gut erkennen. Hauptaufgabe ist des Festplatten-Controllers ist es, die Mechanik anzusteuern und die Kommunikation über die Geräteschnittstelle zu sicherzustellen.

Bei den damals üblichen (und auch heute noch verbreiteten) Magnet-Festplatten sind die Daten auf einem Stapel von Scheiben gespeichert, deren Oberfläche magnetisierbar ist. Für das Schreiben und Lesen wird ein Schreib-Lese-Kopf verwendet (genauer: ein Kopf pro Oberfläche), der sich dicht über die Scheibenoberfläche bewegt. Er lässt sich auf einer Kreisbahn bewegen. Der Plattenstapel dreht sich schnell. Durch diese Drehung und die Bewegung des Kopfes lässt sich jeder Punkt auf der Scheibenoberfläche erreichen.

Beispiel: Festplatte

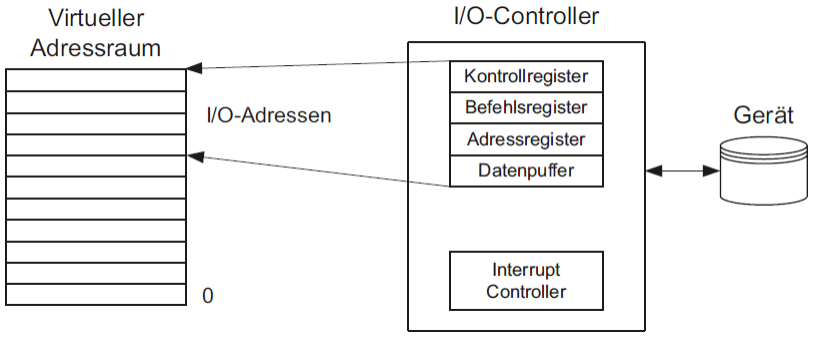


Der eingebaute Festplattencontroller hat die Aufgabe, über die Motorsteuerung den Spindel- und den Kopfmotor so anzusteuern, dass der gewünschte Bereich des Festplatte korrekt gelesen bzw. beschrieben wird. Außerdem überwacht der Controller den Zustand der Festplatte, z.B. Temperatur, Betriebsstunden usw. und speichert diese in den sog. SMART-Registern. Der Prozessor des Controllers nutzt dabei eine meist aktualisierbare Firmware, die im zugehörigen Speicher abgelegt ist.

Der Schreib-Lese-Kopf nutzt einen Vorverstärker und verschiedene Vorverarbeitungstechniken, um die schwachen, von der Magnetisierung hervorgerufenen Signaländerungen zu verstärken. Diese Signale werden Bits bzw. Bytes zugeordnet, wobei auch Fehlerkorrekturcodes verwendet werden, um Fehler direkt zu erkennen und zu eliminieren. Ein Puffer sammelt die zu lesenden Daten. Der Controller steuert außerdem die Host-Schnittstelle.

Typische Befehle, die über die Hostschnittstelle eintreffen, sind z.B. das Lesen oder Schreiben von Daten, das Leeren des Puffers oder das Wechseln in den Standby-Betrieb. Dateien, Verzeichnisse usw. sind für den Controller aber unbekannt – dies ist Sache des Dateisystems und damit des Betriebssystems. Der Controller „sieht“ nur die einzelnen Bits und Bytes, die in einem Bereich der Festplatte (mehr dazu später) gespeichert sind, ohne dass diese einen größeren logischen Kontext repräsentieren.

Kommunikation mit Controllern



Quelle: Mandl: Grundkurs Betriebssysteme

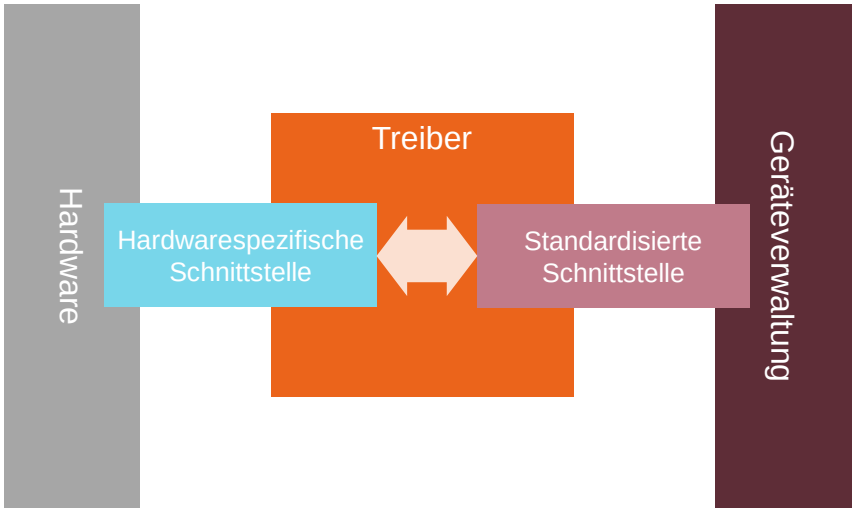
➔ Memory-Mapped-IO

Die Kommunikation mit Controllern erfolgt meist durch sog. Memory-Mapped-IO. Die für die Ansteuerung des Controllers notwendigen Register sowie die Datenpuffer werden dabei in den virtuellen Kernel-Adressraum abgebildet. Jedem Gerät wird dabei (typischerweise durch das Betriebssystem) eine Basisadresse zugewiesen, über die der Controller angesprochen wird.

Der Vorteil der Memory-Mapped-IO liegt darin, dass die Register in Programmen direkt als Variablen gelesen oder geschrieben werden können.

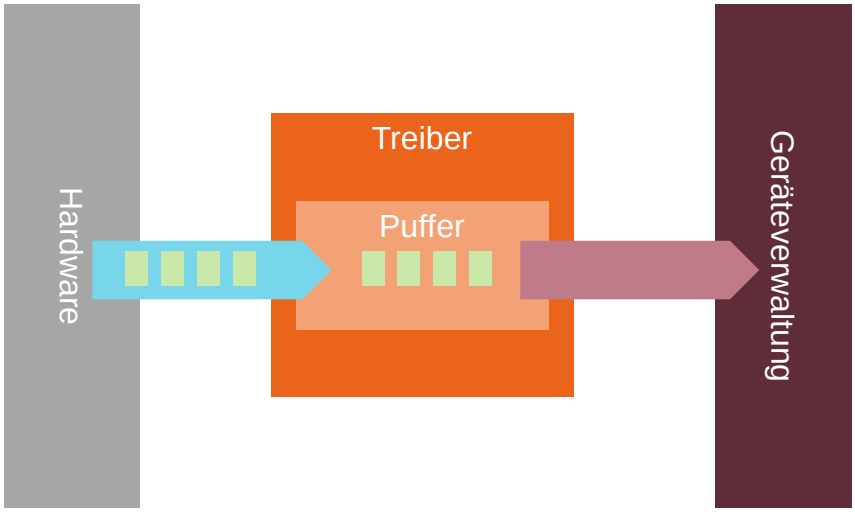
Eine wichtige Abgrenzung vorweg: Memory-Mapped-IO und Direct Memory Access (DMA), das wir später behandeln, sind verschiedene Dinge.

Der Treiber als Dolmetscher



Der Treiber übernimmt in diesem Zusammespiel die Rolle des Dolmetschers, der zwischen der Hardware und der Geräteverwaltung des Betriebssystems übersetzt. Hierfür besitzt er einerseits eine standardisierte Schnittstelle, die ihn mit der Geräteverwaltung kommunizieren lässt sowie andererseits eine hardwarespezifische Schnittstelle, um mit dem Gerätecontroller zu kommunizieren. Er muss also über die standardisierte Schnittstelle eintreffende, generische Befehle, wie z.B. „Starte Gerät“, in konkrete Anweisungen an den Gerätecontroller übersetzen.

Puffer gegen Verarbeitungsunterschiede



Puffer werden bei Ein- und Ausgabeoperationen sehr häufig eingesetzt, vor allem um die unterschiedlichen Geschwindigkeiten zwischen Hard- und Software auszugleichen. So muss der schnellere Partner nicht auf den langsameren warten, sondern kann, wenn die Daten vorliegen, diese schneller abrufen. Oft setzt man hierzu auch doppelte Puffer ein, damit der langsame Partner weiterhin schreiben kann, während der schnellere die Daten abruft.

Auch werden Puffer eingesetzt, wenn z.B. eine Seite nur zeichenweise, die andere aber blockweise arbeitet.

Puffer sind typischerweise als FIFO-Puffer implementiert, meist mit begrenzter Größe. Ein typisches Beispiel ist der Tastaturpuffer.

Caches

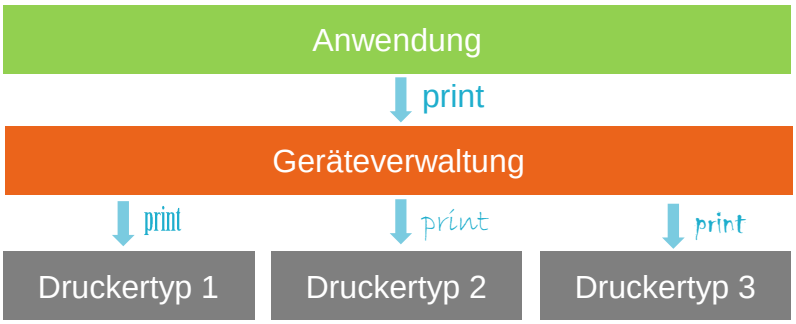
- Zwischenspeicherung von Werten
- Beschleunigung wiederholter Zugriffen
  ➔ zeitliche Lokalität
- Reduktion von Gerätezugriffen

Caches sind ein weiteres Mittel, das von der Hardware oder den Treibern eingesetzt wird. Ein Cache erlaubt schnellere Zugriffe als das Gerät und speichert Daten zwischen, um diese bei wiederholten Zugriffen schneller und mit weniger Gerätezugriffen zur Verfügung stellen zu können. Hierbei nutzt man das aus dem Speichermanagement bekannte Prinzip der zeitlichen Lokalität, d.h. nach einem Zugriff folgt meist noch ein weiterer.

Ziele des Ein- und Ausgabe-Subsystems

Das E/A-Subsystem muss in der Lage sein, die Nutzung einer Vielzahl unterschiedlicher Geräte zu ermöglichen. Die Aufgaben und Ziele, die es hierfür erfüllen muss, sollen auf den folgenden Folien umrissen werden.

Geräteunabhängige Nutzung



Ziel ist es, dass Anwendungen nicht eine konkrete Hardware (Druckermodell XY von Hersteller Z an Anschluss A) ansprechen, sondern ein abstraktes Gerät, z.B. den „Drucker 1“. Hierzu müssen abstrakte Befehle in die für das jeweilige Geräte notwendigen Befehle umgesetzt werden. Notwendig für diese Art der Abstraktion ist einerseits das Bereitstellen geeigneter Systemaufrufe für den User Mode und andererseits die Nutzung der Treiber für die gerätespezifische Kommunikation im Kernel Mode.

Einheitliche Benennung

Das E/A-Subsystem ist üblicherweise so angelegt, dass es Geräte einheitlich benennt. Gut kann man das z.B. unter Linux im /dev-Verzeichnis sehen. Dort beginnen Festplatten immer mit sd, wobei sda für die erste Festplatte, sdb für die zweite Festplatte usw. steht. Die angehängte Nummer entspricht wiederum die Nummer der Partition; sda1 wäre also die erste Partition der ersten Festplatte.

Hardwarenahe Fehlerbehandlung

Entsprechend des Abstraktionszieles versucht man auch, Fehler möglichst hardwarenah zu behandeln. Nur wenn dies nicht gelingt, soll die Anwendungsebene eingebunden werden (z.B. „Papier leer“).

Optimierung der Kommunikation

- Ausgleich von Geschwindigkeitsunterschieden
- Ausgleich von unterschiedlichen Transferansätzen (z.B. zeichen- vs. blockorientiert)
- Optimierung von Lesezugriffen (z.B. durch Caches)

Wie bereits gezeigt, ist der Umgang mit unterschiedlichen Verarbeitungsgeschwindigkeiten ein zentrales Problem des E/A-Subsystems. Um dies auszugleichen können z.B. Puffer eingesetzt werden, diese können auch verwendet werden, um unterschiedliche Transferansätze (z.B. zeichen- vs. blockorientiert, also einzelne vs. mehrere Zeichen) auszugleichen. Außerdem können Lesezugriffe zwischen mehreren Prozesse so optimiert werden, dass die Gesamtgeschwindigkeit steigt. Zusätzlich können Caches helfen, wiederholte Lesezugriffe zu beschleunigen.

Schutz des Gerätezugriffs

- Abstraktion verhindert direkten Hardwarezugriff
- Schutz bei Memory-Mapped-IO
- Sichtbarkeit von Geräten
- Prüfen von Benutzerberechtigungen

Die Abstraktion des Hardwarezugriffs alleine schützt bereits die Hardware vor unerwünschten Zugriffen durch die Anwendungsebene. Die Speicherverwaltung kann den Zugriff auf die betreffenden Seiten bei Memory-Mapped-IO einschränken. Außerdem können Geräte verborgen werden, auf die die Anwendungsebene keine Zugriffe haben soll. Auch ist es im Zusammenspiel mit der Benutzerverwaltung möglich, den Gerätezugriff an Benutzerberechtigungen zu knüpfen.

Anbindung von Geräten

.

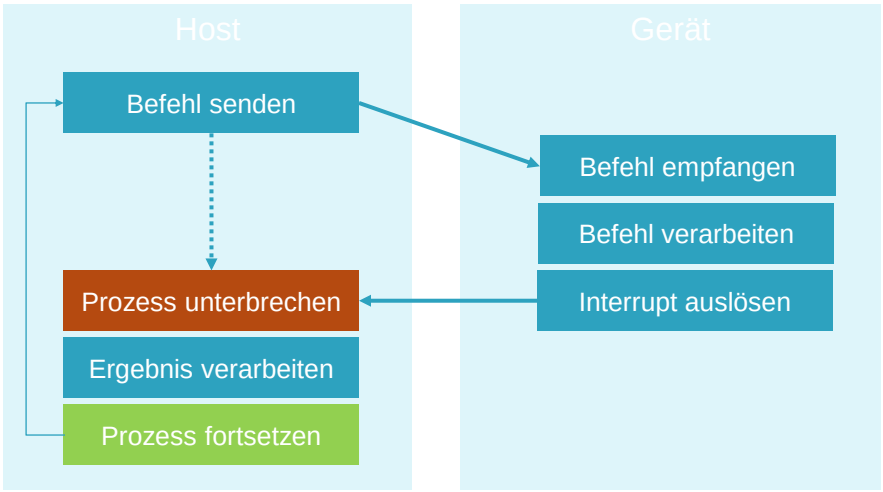
Programmierte Ein-/Ausgabe

- CPU prüft wiederholt, ob das Gerät verfügbar ist
- CPU sendet Befehl an das Gerät
- Gerät signalisiert, dass es beschäftigt ist
- CPU prüft wiederholt, ob der Befehl verarbeitet wurde

➔ Meist ineffizient, da CPU stark beansprucht

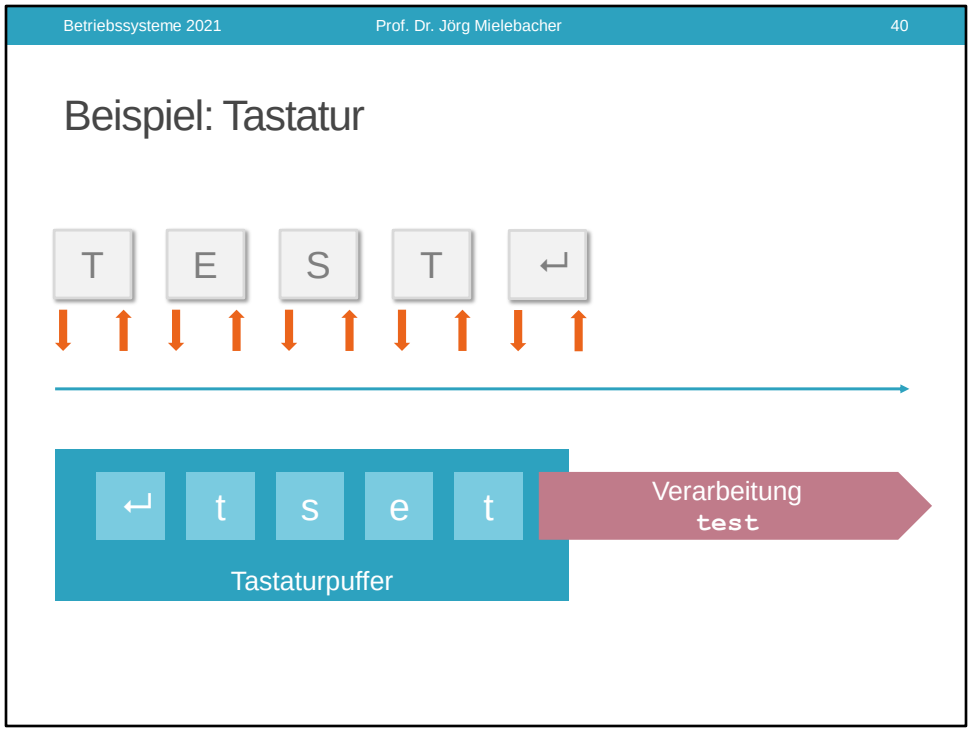
Bei der Kommunikation mit Geräten ist die programmierte Ein-/Ausgabe einer der einfachsten Ansätze. Die CPU muss daher zunächst erkennen, ob das betreffende Gerät verfügbar ist und prüft dies solange wiederholt, bis es verfügbar ist („Polling“). Danach sendet die CPU den Befehl an das Gerät. Das Gerät wiederum signalisiert, dass es nun mit der Verarbeitung beschäftigt ist. Daher muss die CPU nun so lange warten, bis der Befehl verarbeitet wurde. Dieses Warten ist erneut ein aktives Warten, da immer wieder der Status des Geräts abgefragt wird.

Interrupts



Eine spürbare Entlastung der CPU bringt die interruptgesteuerte Ein-/Ausgabe. Die Idee besteht darin, den Befehl an das Gerät zu schicken und danach sofort andere Anweisungen zu verarbeiten – das aktive Warten entfällt, da das Gerät nach dem Verarbeiten des Befehls einen Interrupt auslöst, der den Host dazu veranlasst, den aktuellen Prozess zu unterbrechen, das Ergebnis des Geräts zu verarbeiten und danach den ursprünglichen Prozess wieder fortzusetzen.

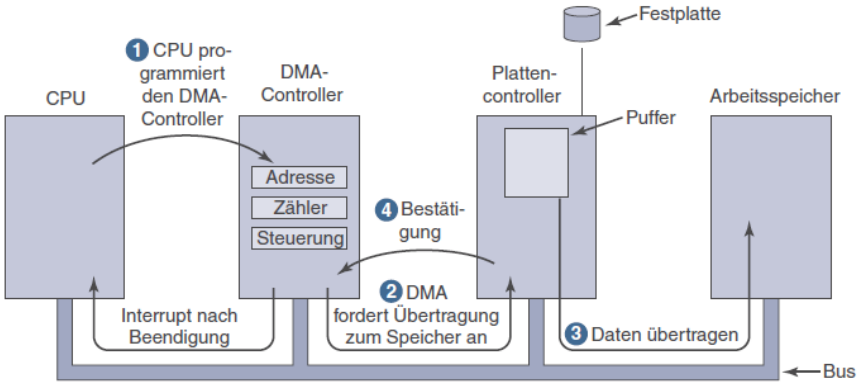
Ein wichtiger Nachteil sind die hierdurch notwendigen Kontextwechsel. Auch wird heutzutage meist zwischen wichtigen (z.B. Fehler) und unwichtigen (z.B. Routine-E/A) Interrupts unterschieden.



Jeder Tastendruck (bzw. das Loslassen) der Tastatur löst einen Interrupt aus. Dieser Interrupt unterbricht die Ausführung des laufenden Prozesses und bewirkt, dass die jeweilige Taste in den sog. Tastaturpuffer gelegt wird. Der auf die Eingabe wartende Prozesse kann dann eingegebenen Daten aus dem Tastaturpuffer entnehmen.

Im gezeigten Beispiel wurde Shift außen vor gelassen. Auch der Druck von Shift ließe sich als Interrupt erfassen.

Direct Memory Access (DMA)



Quelle: Tanenbaum: Moderne Betriebssysteme

Die Idee des Direct Memory Access (DMA) besteht darin, den Prozessor gegenüber den bisherigen beiden Ansätzen noch weiter zu entlasten. Denn bei beiden Ansätzen wurden Daten vom Prozessor in dessen Register geladen und von dort in den Hauptspeicher. Bei DMA können die Geräte unter Vermittlung des DMA-Controllers direkt in den Speicher schreiben. Der DMA-Controller hat dabei Zugriff auf den Systembus.

Das hier gezeigte Beispiel eines Lesezugriffs auf die Festplatte zeigt die wesentlichen Schritte bei DMA: Zunächst weist die CPU den DMA-Controller an, was von welchem Gerät gelesen werden soll. Der DMA-Controller steuert dann eigenständig (in diesem Fall) den Festplattencontroller an und schickt ihm den Lesebefehl mit dem Zusatz, DMA zu nutzen. Der Festplattencontroller liest nun die Daten in seinen Lesepuffer und überträgt sie direkt in den Arbeitsspeicher (d.h. ohne den Umweg über den Prozessor). Anschließend bestätigt er die erfolgreiche Übertragung gegenüber dem DMA-Controller. Dieser wiederum signalisiert der CPU per Interrrupt, dass die Daten vorliegen.

Der Vorteil gegenüber der interruptgesteuerten E/A liegt darin, dass die Daten mit Auslösen des Interupts bereits im Hauptspeicher liegen und nicht erst über die Prozessorregister in den Hauptspeicher geladen werden müssen.

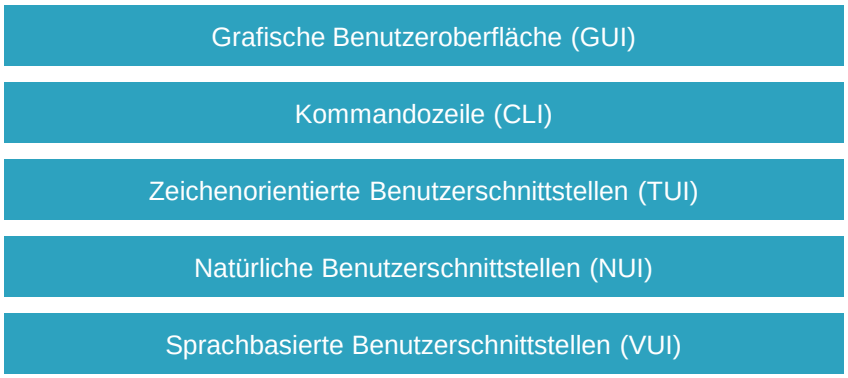
Exkurs: DMA Angriffe

- Normalerweise:
  Kein direkter Speicherzugriff im Usermode
- Aber:
  Direkter Speicherzugriff durch angeschlossene Geräte (DMA) gemäß OHCI 1394 (Thunderbolt usw.) erlaubt
- Folgen:
  Zugriff auf sensible Daten im Speicher (Schlüssel usw.)
- Abhilfe:
  - Anschlüsse physisch sichern
  - Daten im Speicher verschlüsseln
  - DMA Schutz des Betriebssystems

Normalerweise ist im Usermode kein direkter Speicherzugriff möglich; die MMU würde das verhindern. Der Standard OHCI 1394, der für Thunderbolt, Firewire usw. gilt, erlaubt jedoch unter bestimmten Bedingungen den direkten Speicherzugriff durch angeschlossene Geräte (DMA). Würde man nun jemanden dazu bringen, ein manipuliertes Thunderbolt-Gerät anzuschließen, ist es prinzipiell möglich auf sensible Daten im Speicher zuzugreifen, z.B. Schlüssel, Passwörter usw.

Man kann solchen DMA Angriffen entgegebenwirken, indem man Anschlüsse physisch absichert, sensible Daten im Speicher nur verschlüsselt ablegt und ggf. den DMA Schutz des Betriebssystems aktiviert.

Ein- und Ausgabe aus Benutzersicht



Aus Benutzersicht stellen Betriebssysteme heute eine Vielzahl unterschiedlicher Benutzerschnittstellen zur Verfügung. Auch wenn dies nur am Rande (nämlich hinsichtlich der Geräteanbindung, z.B. Monitor) der Geräteverwaltung zuzuordnen ist, ist es doch unmittelbar mit dem Thema Ein- und Ausgabe verbunden.

Während für PCs und Mobilgeräte Grafische Benutzeroberflächen (Graphical User Interface, GUI) im Vordergrund stehen, kommen bei Embedded Systems und Großrechnern vorrangig Kommandozeilen (Command Line Interface, CLI) zum Einsatz. Früher kamen auch bei PCs statt grafischer Oberflächen Zeichenorientierte Benutzerschnittstellen (Text User Interface, TUI) zum Einsatz, bei denen Fenster und ähnliche Elemente durch einzelne ASCII-Zeichen (Linen, Ecken, Flächen usw.) nachgebaut werden. Heute findet man sie noch bei manchen Editoren oder Installationsprogrammen, die im Terminal angezeigt werden. Natürliche Benutzerschnittstellen (Natural User Interface, NUI) nutzen beispielsweise Touchscreens, um die Bewegungen von Fingern und Hand in Form von Gestern zur Steuerung zu verwenden. Sprachbasierte Benutzerschnittstellen (Voice User Interfaces, VUI) nutzen Sprache für die Kommunikation zwischen System und Nutzer; Beispiele sind Assistenzsysteme (z.B. Amazon Echo) oder auch Systeme im Automotive-Bereich.

Weitere Schnittstellen befinden sich in der Entwicklung, z.B. Brain-Computer-Interfaces oder greifbare Benutzerschnittstellen.

Zusammenfassung

- I.d.R. gibt es die Standard-Datenströme stdin (Eingabe), stdout (Ausgabe) und stderr (Fehlerausgabe). Diese lassen sich umleiten.
- Das Ein- und Ausgabe-Subsystem ist ein besonders wichtiger Teil des Betriebssystems und sorgt für Hardwareunabhängigkeit, einheitliche Benennung, effiziente Nutzung, hardwarenahe Fehlerbehandlung sowie Schutz der Geräte.
- Meist nutzt man Memory-Mapped-IO.
- Anwendungen können nur über das Betriebssystem auf die Hardware zugreifen; dafür werden Treiber genutzt, die mit den Gerätecontrollern kommunizieren.
- Puffer wirken gegen Geschwindigkeitsunterschiede.
- Die Anbindung der Geräte erfolgt durch programmierte E/A, Interrupts oder DMA.

Aufgaben

1. Welche drei Standard-Datenströme gibt es üblicherweise und wofür werden sie verwendet?
2. Sie wollen unter Linux den Prozess a starten, seine Eingaben erhalte er aus der Datei ainput. Seine Ausgaben sollen auf dem Bildschirm angezeigt werden, Fehler in die Datei aerrors geschrieben werden. Wie lautet der Aufruf?
3. Worin unterscheiden sich > und |? Wofür setzt man sie ein?
4. Wie und wo werden Geräte unter Linux eingebunden?
5. Welche Ziele verbindet man mit dem E/A-Subsystem?
6. Welche Aufgaben hat ein Gerätetreiber?
7. Wozu setzt man beim Gerätezugriff Puffer ein?
8. Welche Gemeinsamkeiten und Unterschiede haben interruptgesteuerte E/A und DMA?

Aufgaben

9. Was versteht man unter Memory-Mapped-IO?
10. Was geschieht, wenn Sie Tasten auf der Tastatur drücken? Wie ist das Gerät angebunden?
11. Nennen Sie Beispiele für die Abstraktion durch die Geräteverwaltung.
12. Nennen Sie Beispiele für Benutzerschnittstellen.
13. Welche Schichten gibt es bei einer GUI unter Linux?