ulm university universität ulm

Grundlagen der Betriebssysteme

[CS2100]

Sommersemester 2014

Heiko Falk

Institut für Eingebettete Systeme/Echtzeitsysteme
Ingenieurwissenschaften und Informatik
Universität Ulm

ulm university universität ulm

Kapitel 9

Ein-/Ausgabe und Gerätetreiber

Inhalte der Vorlesung

Inhalte des Kapitels (1)

9. Ein-/Ausgabe und Gerätetreiber

- Geräteaufbau *(exemplarisch)*
  - Tastatur
  - Maus
- Treiberschnittstelle und Treiberimplementierung
- Fallstudie: UNIX/Linux
  - Repräsentation von Geräten
  - Treibergerüst unter Linux
  - Interne Treiberschnittstelle
- Fallstudie: Windows I/O-System
  - Aufbau und Funktionsweise
  - *Plug-and-Play*
  - *Power Management*
- ...

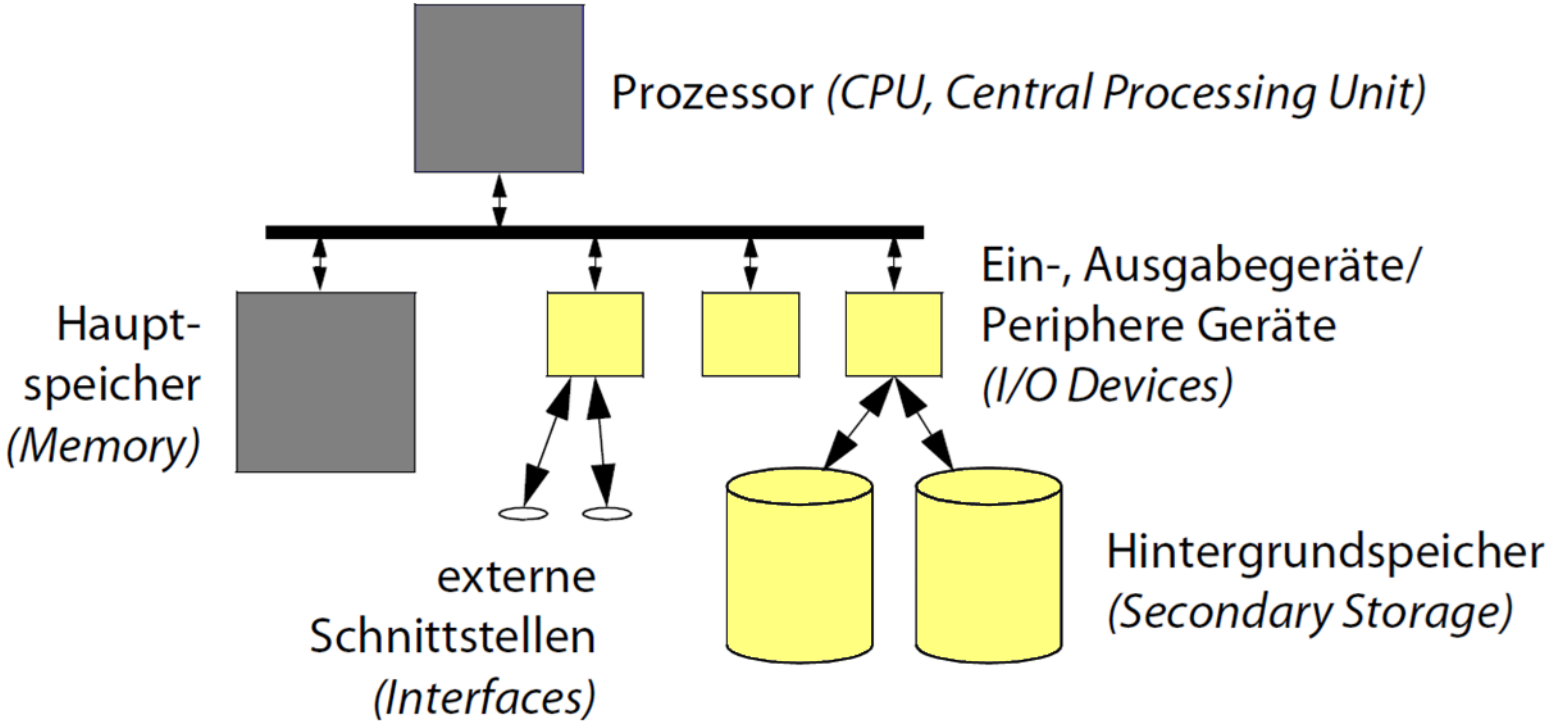
Inhalte des Kapitels (2)

9. Ein-/Ausgabe und Gerätetreiber

- ...
- Festplattentreiber
  - *Interrupt*-basiert
  - DMA-basiert
  - Disk-*Scheduling*
    - *First-Come, First Serve (FCFS)*
    - *Shortest Seek Time First (SSTF)*
    - SCAN
- Treiber für weitere Geräte
  - Serielle Schnittstellen
  - Bildschirm
  - Netzwerk

Einordnung

Betroffene physikalische Ressourcen



Geräteaufbau

Übliche Ein-/Ausgabegeräte

- Speichermedien (z.B. Festplatten, Disketten, CD-ROM, DVD-ROM)
  - ☞ Festplatten schon in Kapitel 5 (Filesysteme) betrachtet
    Disketten, CDs und DVDs sehr ähnlich zu Festplatten, werden daher hier nicht weiter vorgestellt
- Externe I/O-Schnittstellen (z.B. RS232, Parallelport, USB)
  - ☞ Siehe Vorlesung „Grundlagen der Rechnerarchitektur“
- Netzwerkkarten
  - ☞ Siehe Vorlesung „Grundlagen der Rechnernetze“
- *Human Interface Devices (HIDs)*
  - Tastatur
  - Maus

Tastatur (1)

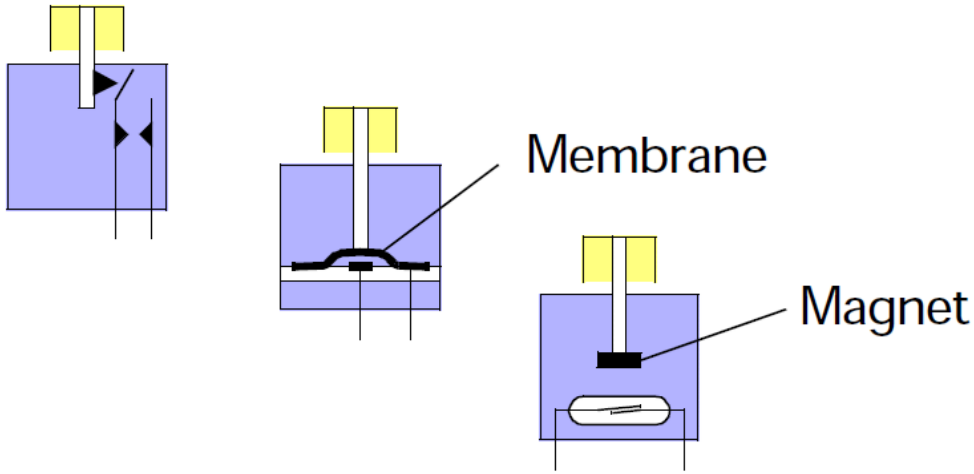
1 Schalter pro Taste

Mechanische Schalter

– Klassischer Schalter

– Membranschalter

– Reed-Relais-Schalter



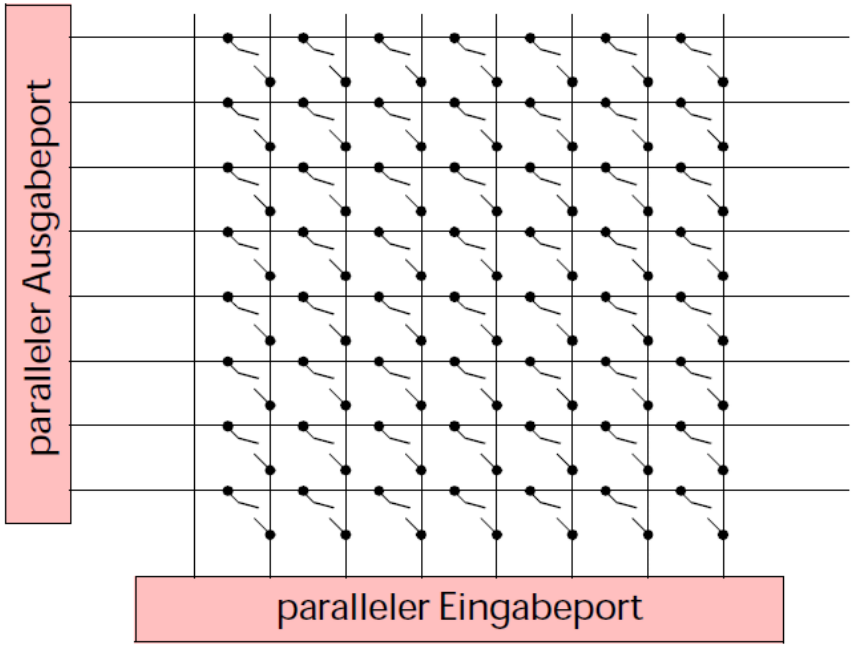
Andere Verfahren

– Z.B. Hall-Effekt-Schalter
  – Näherung eines Magneten induziert Strom in einen Halbleiter
  – Verschleißlos aber teuer

Tastatur (2)

Tastenerkennung

- Matrixanordnung der Tasten
  - Nacheinander wird an jede Spalte ein „*high*“ Pegel angelegt, danach werden jeweils die Zeilen getestet
  - Tastendruck stellt elektrische Verbindung zwischen Spalte und Zeile her
  - Erkennung der genauen Position durch Mikrocontroller



Tastatur (3)

Tastenerkennung (fortges.)

- Umsetzung der erkannten Taste in Tastencode *(Scancode)*
- Jeder Taste sind zwei Codes zugeordnet
  - Einer für die Betätigung der Taste (MAKE-Code)
  - Einer für das Loslassen der Taste (BREAK-Code)

Problem: Entprellung

- Mechanische Kontakte prellen, d.h. Kontakt schließt nicht sofort dauerhaft
- Software: kurzzeitiges Öffnen des Kontakts wird ignoriert
- Hardware: elektrische Schaltung

Maus (1)

Mechanische Maus

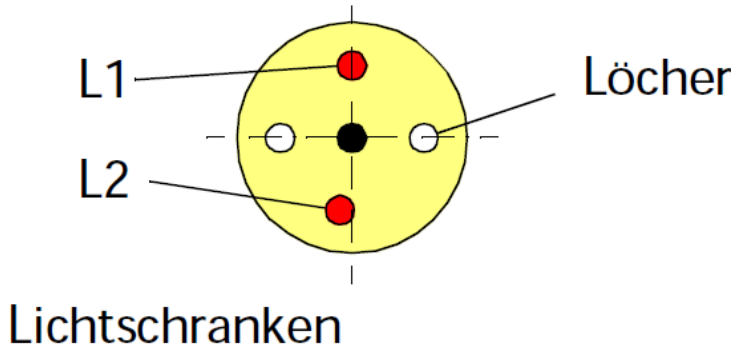
- Gummibeschichtete Stahlkugel
- Abgreifräder in X- und Y-Richtung
- Lochscheiben mit zwei Lichtschranken (über LED) pro Rad
- Lichtsignale interpretierbar
  - Entfernung über Anzahl der Lichtblitze
  - Richtung über Phasenverschiebung der zwei Lichtblitze pro Rad

    Drehung nach links:

    Lichtblitz bei L2, dann L1

    Drehung nach rechts:

    Lichtblitz bei L1, dann L2



Maus (2)

Optische Maus

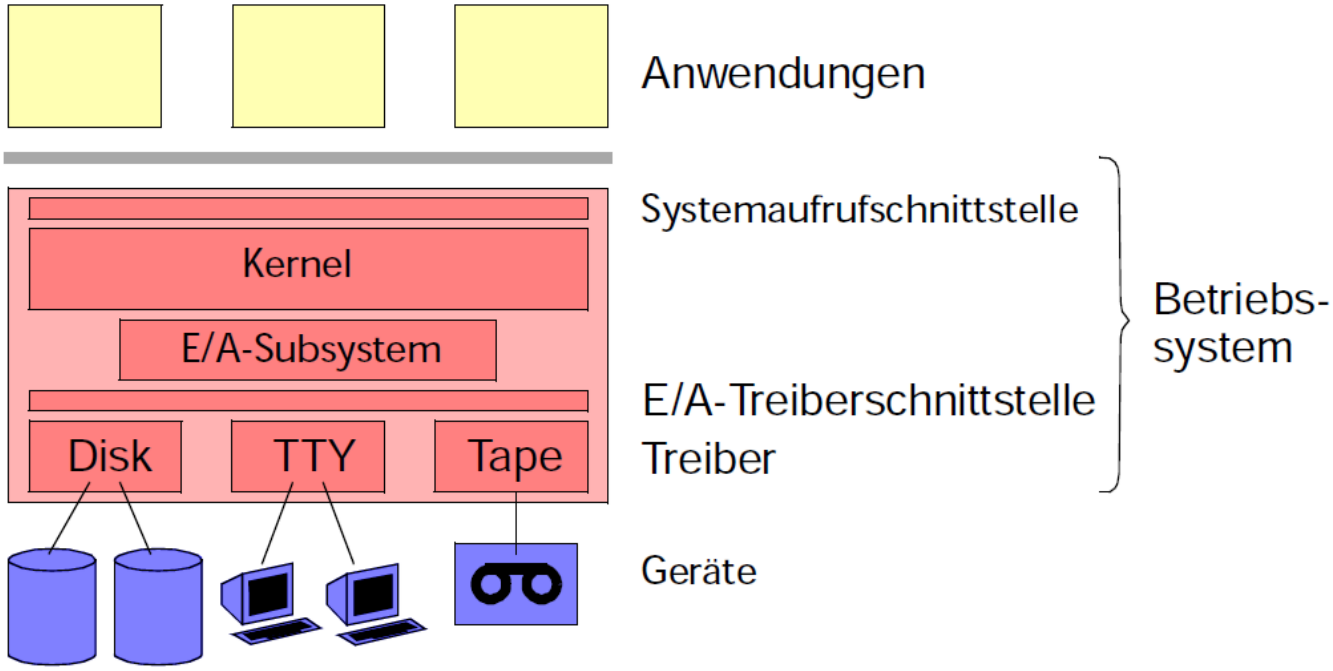
– Lichtquelle und optische Sensoren erkennen Strukturveränderungen auf der Unterlage

Codierung

– Mausbewegungen, Richtungen und Tastendrücke
– Codierung in RS-232 Daten (typischerweise 1.200 Bit/s, 7N1)
– Codierung in USB-Pakete

Geräteverwaltung

Schichtung der Systemsoftware bis zum Gerät



Treiberschnittstelle

Ziel

- Entlastung des Betriebssystemkerns von notwendigen Spezialbehandlungen von einzelnen Geräten
- Gleiche Software-Schnittstelle für alle Geräte einer Art
  - Geräteunabhängige Schnittstelle

Treiber

- Software-Modul für spezielle Geräte
  - Z.B. für einen bestimmten Festplattenkontroller
- Einheitliche Behandlung von Treibern durch das Betriebssystem

Aufgabe der Treiber

- Implementierung der Treiberschnittstelle für bestimmtes Gerät
  - Programmierung der I/O-Register
  - Behandlung der *Interrupts*

Treiberimplementierung (1)

Aufgaben des Treibers

- Verwaltung von Zugriffsrechten
  - Wer darf was?
- Optimale Nutzung der Peripherie
  - Strategische Entscheidungen innerhalb des Treibers
  - Z.B. optimaler Plattendurchsatz
- Zuteilung von Ressourcen an Benutzerprozesse
- Auflösung von Zugriffskonflikten
- Implementierung über *Polling*, *Interrupts* oder I/O-Operationen

Mehrere Benutzerprozesse

- *Scheduler* schaltet zwischen Prozessen um *(☞ Kapitel 4)*
- Langsame Auftragsbearbeitung durch das Gerät
  - Z.B. Latenz- und Positionierzeit einer Festplatte

Treiberimplementierung (2)

***Polling*-Betrieb**

- Prozessor ist einzig aktive Instanz bei einer I/O-Transaktion
- Prozessor fragt I/O-Baustein ab, ob Teilaktion beendet
  - Z.B. ob Daten vorliegen
  - Aktives Warten
- Problem
  - Prozessor ständig aktiv
  - Vergeudung von Rechenzeit
  - Implizites Blockieren anderer Prozesse
- Heute nur sinnvoll bei äußerst kurzen Wartezeiten

Treiberimplementierung (3)

***Interrupt*-Betrieb**

- Gerät kann durch Unterbrechungsauslösung aktiv werden
  - Z.B. Gerät meldet Beendigung eines Teilauftrags
- Problem
  - Komplexes Programmiermodell
  - Zusatzaufwand durch Unterbrechungsbehandlung (Registersicherung etc.)
- Vorteil
  - Prozessor kann nach Anstoßen eines I/O-Teilauftrags weitere Befehle ausführen

Treiberimplementierung (4)

I/O-Operationen durch Anwendungsprozess

- Synchrone Operation
  - Anwendungsprozess weiß, dass I/O-Operation nach Systemaufruf abgeschlossen
- Asynchrone Operation
  - Anwendungsprozess stößt I/O-Operation an
  - Evtl. nachträgliche Synchronisierung mit Ende der Operation
- Realisierung der Synchronisierung mit dem I/O-Ende durch Blockierung des Prozesses
  - Schlechte Alternative: aktives Warten

Treiberimplementierung (5)

Typische Abwicklung mehrerer I/O-Aufträge

- Interne Warteschlange für Aufträge
- Blockierung des Benutzerprozesses (bei synchroner I/O-Operation)
- Strategische Abarbeitung der Aufträge
- Freigabe blockierter Prozesse mit beendeten I/O-Aufträgen (bei synchroner I/O-Operation)

- Nutzung von *Interrupt*-Behandlung zur Fortführung von I/O-Aufträgen
- Falls Treiber inaktiv, Umschaltung auf lauffähige Prozesse

- Synchronisierung bei asynchronen I/O-Operationen
  - Abwicklung wie synchrone I/O-Operation
  - Blockierung, falls Auftrag noch nicht beendet

Roter Faden

9. Ein-/Ausgabe und Gerätetreiber

- Geräteaufbau *(exemplarisch)*
  - Tastatur
  - Maus
- Treiberschnittstelle und Treiberimplementierung
- Fallstudie: UNIX/Linux
- Fallstudie: Windows I/O-System
- Festplattentreiber
- Treiber für weitere Geräte

Fallstudie: Gerätetreiber unter UNIX/Linux (1)

Peripherie-Geräte werden durch Spezialdateien repräsentiert

- Geräte können wie Dateien mit Lese- und Schreiboperationen angesprochen werden
- Öffnen der Spezialdateien schafft eine Verbindung zum Gerät, die durch einen Treiber hergestellt wird
- Direkter Durchgriff vom Anwender auf den Treiber

Zwei Gerätetypen

- Blockorientierte Geräte
- Zeichenorientierte Geräte

Fallstudie: Gerätetreiber unter UNIX/Linux (2)

Blockorientierte Spezialdateien

– Plattenlaufwerke, Bandlaufwerke, *Floppy Disks*, CD-ROMs

Zeichenorientierte Spezialdateien

– Serielle Schnittstellen, Drucker, Audiokanäle, etc.
– Blockorientierte Geräte haben meist auch eine zusätzliche zeichenorientierte Repräsentation

Eindeutige Beschreibung der Geräte durch ein Tupel (Gerätetyp, *Major Number*, *Minor Number*)

– Gerätetyp: *Block Device*, *Character Device*
– *Major Number:* Auswahlnummer für einen Treiber
– *Minor Number:* Auswahl eines Gerätes innerhalb eines Treibers

Fallstudie: Gerätetreiber unter UNIX/Linux (3)

Beispiel eines Verzeichnislistings von /dev (Ausschnitt)

```
crw-------   root root     5, 1 Jan 11 08:05 console
lrwxrwxrwx   root root        3 Jan 11 14:09 dvd1 -> sr0
lrwxrwxrwx   root root        3 Jan 11 14:09 dvdrw1 -> sr0
crw-rw----   root video   29, 0 Jan 11 08:04 fb0
brw-rw----   root disk     8, 0 Jan 11 08:05 sda
brw-rw----   root disk     8, 1 Jan 11 08:05 sda1
brw-rw----   root disk     8, 2 Jan 11 08:05 sda2
brw-rw----   root disk     8, 3 Jan 11 08:05 sda3
brw-rw----   root disk     8, 4 Jan 11 08:05 sda4
brw-rw----+  root cdrom   11, 0 Jan 11 14:09 sr0
crw-rw-rw-   root tty      5, 0 Jan 11 08:04 tty
```

Zugriffsrechte

c: *character device*
b: *block device*
l: *symbolic link*

Eigentümer

Major + *Minor Number*

Erstellungszeitpunkt der Spezialdatei

Name der Spezialdatei

Fallstudie: Treibergerüst unter Linux (1)

```
static struct file_operations fops;
  // ... Initialisierung von fops (Funktionszeiger)

static int __init mod_init( void ) {
  if ( register_chrdev( 240, ”DummyDriver”, &fops ) == 0 )
    return 0;      // Treiber erfolgreich angemeldet
  return –EIO;     // Anmeldung beim Kernel fehlgeschlagen
}

static void __exit mod_exit( void ) {
  unregister_chrdev( 240, ”DummyDriver” );
}

module_init( mod_init );
module_exit( mod_exit );
```

Registrierung für das *char device* mit der *Major Number* 240.

mod_init und **mod_exit** werden beim Laden bzw. Entladen des Treibers in den/aus dem Kernel ausgeführt.

Fallstudie: Treibergerüst unter Linux (2)

```
static int dummy_open( struct inode *geraete_datei,
                       struct file *instanz ) {
  printk( ”driver_open called\n” ); return 0;
}

static int dummy_close( struct inode *geraete_datei,
                        struct file *instanz ) {
  printk( ”driver_close called\n” ); return 0;
}
```

Die Treiber-Operationen entsprechen den normalen Datei-Operationen

In diesem Beispiel machen **open** und **close** nur Debugging-Ausgaben.

Fallstudie: Treibergerüst unter Linux (3)

```
static char hello_world[] = ”Hello World\n”;

static ssize_t dummy_read( struct file *instanz, char *user,
                           size_t count, loff_t *offset ) {
  int not_copied, to_copy;

  to_copy = strlen( hello_world ) + 1;
  if ( to_copy > count ) to_copy = count;
  not_copied = copy_to_user( user, hello_world, to_copy );
  return to_copy – not_copied;
}

static struct file_operations fops = {
  .open = dummy_open,
  .release = dummy_close,
  .read = dummy_read
};
```

Mit **copy_to_user** und **copy_from_user** kann man Daten zwischen Kern- und Benutzer-Adressraum austauschen.

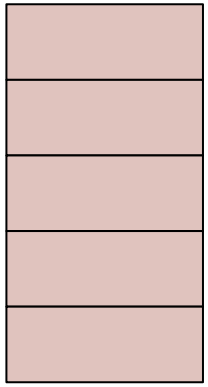
Es gibt noch wesentlich mehr Operationen, sie sind jedoch größtenteils optional.

[Siehe auch: http://www.faqs.org/docs/kernel/x571.html]

Fallstudie: Gerätetreiber unter BSD-UNIX (1)

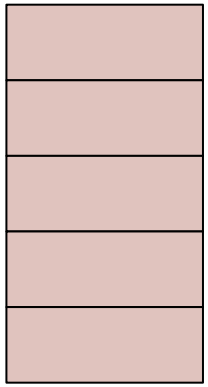
Interne Treiberschnittstelle

– Struktur von Funktionszeigern pro Treiber *(Major Number)*



```
d_open
d_close
d_strategy
d_size
d_xhalt
...
```

`struct bdevsw`
für blockorientierte Geräte



```
d_open
d_close
d_write
d_read
d_ioctl
...
```

`struct cdevsw`
für zeichenorientierte Geräte

[Z.B. http://fxr.watson.org/fxr/source/sys/conf.h?v=NETBSD;im=3#L69]

Fallstudie: Gerätetreiber unter BSD-UNIX (2)

Funktionen eines *Block device*-Treibers

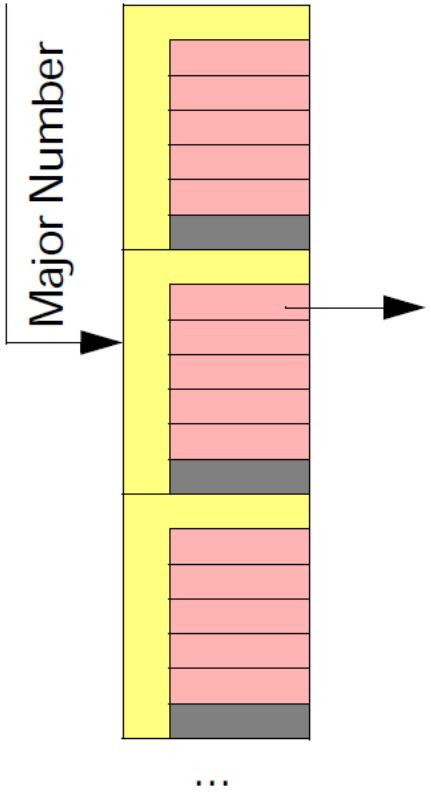
- **`d_open`**: Öffnen des Gerätes
- **`d_close`**: Schließen des Gerätes
- **`d_strategy`**: Abgeben von Lese- und Schreibaufträgen auf Blockbasis
- **`d_size`**: Ermitteln der Gerätegröße (z.B. Partitions- oder Plattengröße)
- **`d_xhalt`**: Abschalten des Gerätes
- u.a.

Funktionen eines *Character device*-Treibers

- **`d_open`**, **`d_close`**: Öffnen und Schließen des Gerätes
- **`d_read`**, **`d_write`**: Lesen und Schreiben von Zeichen
- **`d_ioctl`**: Generische Kontrolloperation *(☞ siehe RS-232 Treiber)*
- u.a.

Fallstudie: Gerätetreiber unter BSD-UNIX (3)

Felder für den Aufruf von Treibern (`bdevsw[]` und `cdevsw[]`)



`struct bdevsw bdevsw[]`

Funktion des Treibers

- *Major Number* bestimmt Element des Feldes
- Zeiger innerhalb des Feldelementes bestimmt Eintrittspunkt in die entsprechende Treiberfunktion
- *Minor Number* wird beim Aufruf als Parameter übergeben

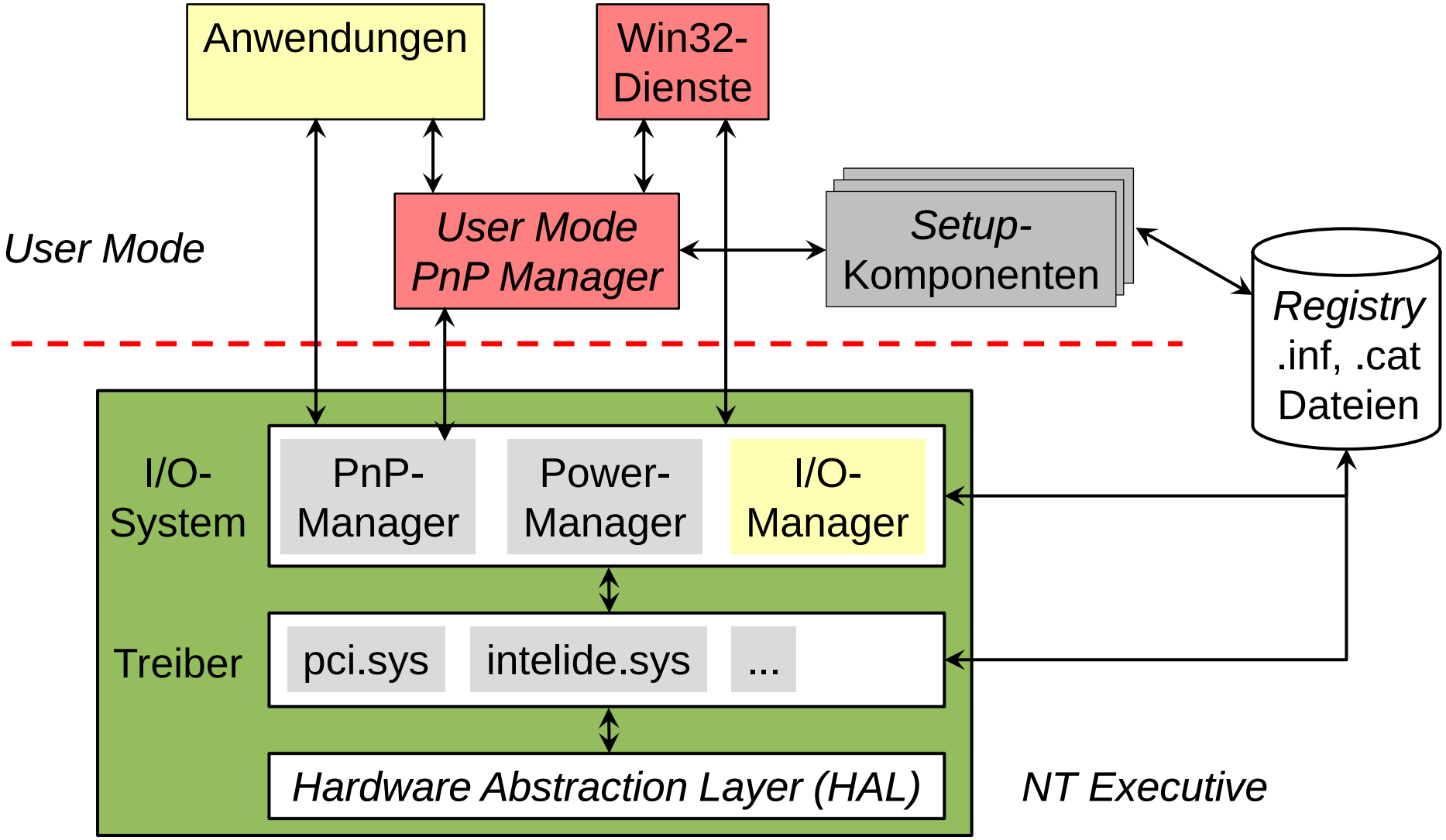
– Nachempfindung objektbasierter Aufrufe

Roter Faden

9. Ein-/Ausgabe und Gerätetreiber

- Geräteaufbau *(exemplarisch)*
- Treiberschnittstelle und Treiberimplementierung
- Fallstudie: UNIX/Linux
  - Repräsentation von Geräten
  - Treibergerüst unter Linux
  - Interne Treiberschnittstelle
- Fallstudie: Windows I/O-System
- Festplattentreiber
- Treiber für weitere Geräte

Fallstudie: Windows I/O-System (1)



Fallstudie: Windows I/O-System (2)

I/O-Manager

– Zentrale Komponente des Windows I/O-Systems
– Führt mit Hilfe von Treibern I/O-Aufgaben durch

Plug-and-Play (PnP)

– PnP-Manager erkennt neue Geräte
– Fragt mit Hilfe des *User-Mode* Teils nach einem Treiber

Registry

– Zentrale hierarchische Konfigurationsdatenbank von Windows
– Neue Gerätetreiber bestehen aus einer Kollektion von Files (Binärcodes der Treiber, Meta-Information über den Treiber in .inf-Datei (z.B. Geräteklasse, welche Dateien sind wohin zu kopieren), und einer kryptographischen Signatur über alle Dateien in .cat-Datei

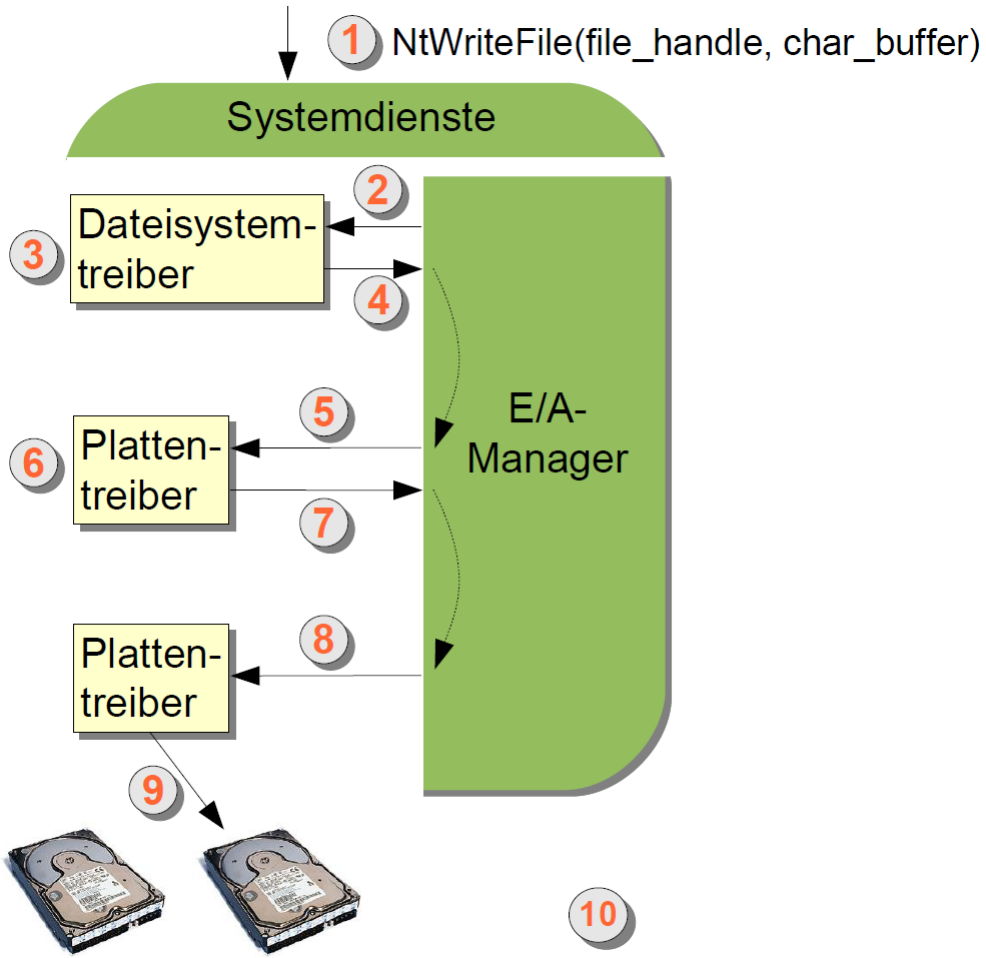
Fallstudie: Windows I/O-System (3)

Das I/O-System steuert einen Treiber mit Hilfe der ...

- Initialisierungsroutine / Entladeroutine
  - Wird nach/vor dem Laden/Entladen des Treibers ausgeführt
- Routine zum Hinzufügen von Geräten
  - PnP-Manager hat ein neues Gerät für den Treiber
- „Verteilerroutinen“
  - Öffnen, Schließen, Lesen, Schreiben und gerätespezifische Operationen
- *Interrupt Service Routine (ISR)*
  - Wird von der zentralen *Interrupt*-Verteilungsroutine aufgerufen
- I/O-Komplettierungs- und Abbruchroutine
  - Liefern Informationen über den Ausgang weitergeleiteter I/O-Aufträge

Fallstudie: Windows I/O-System (4)

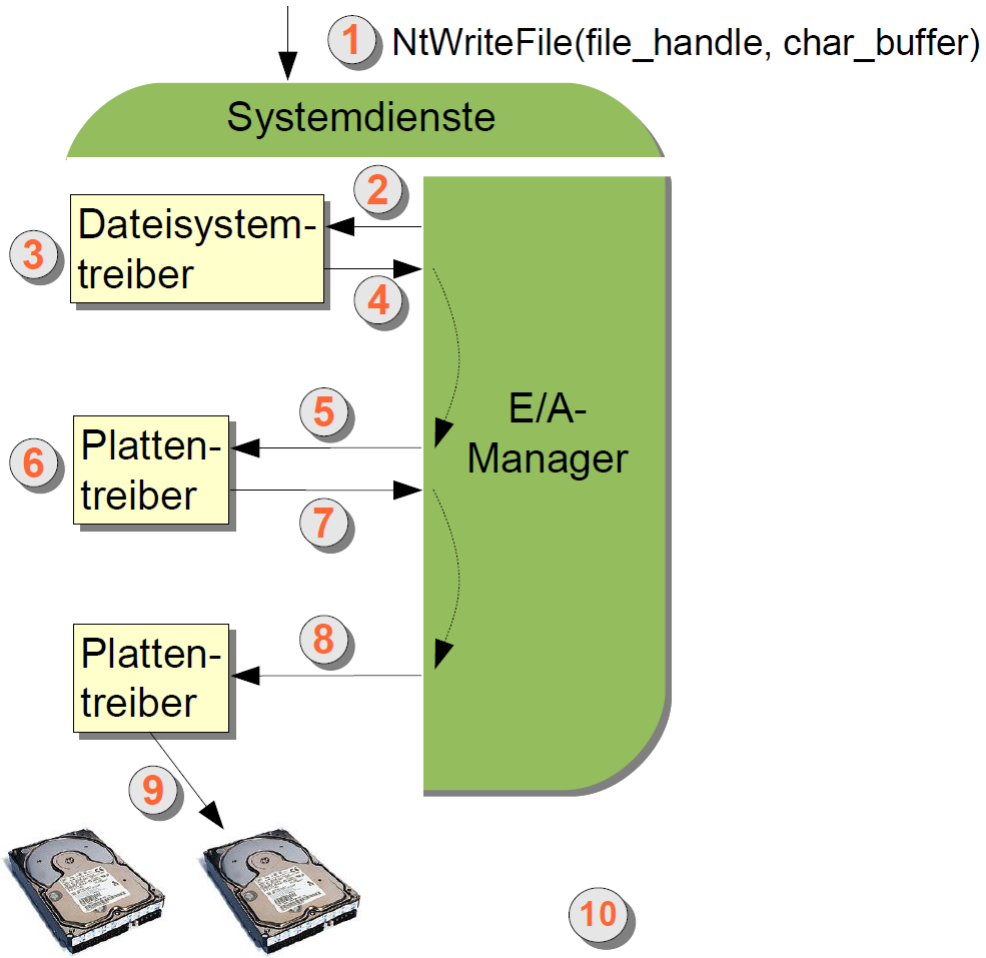
Typischer I/O-Ablauf



1. Über das Dateiobjekt wird das Filesystem und der Treiber gefunden
2. Daten an bestimmten Byte-Offset in Datei schreiben
3. Position auf Datenträger berechnen
4. I/O-Auftrag weitergeben

Fallstudie: Windows I/O-System (5)

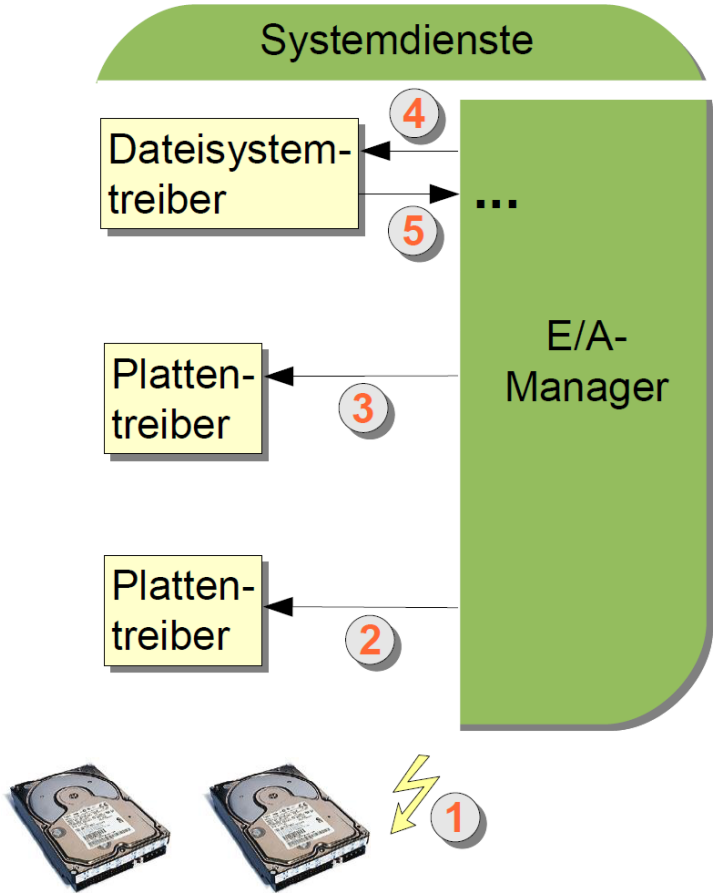
Typischer I/O-Ablauf *(Fortsetzung)*



5. Daten an bestimmten Byte-Offset auf Datenträger schreiben
6. Position in Plattennummer und Offset umrechnen
7. I/O-Auftrag weitergeben
8. Daten an bestimmten Byte-Offset auf Platte 2 schreiben
9. Physikalischen Block berechnen und Operation initiieren
10. Rückkehr zum Anwendungsprozess

Fallstudie: Windows I/O-System (6)

Typischer I/O-Ablauf *(Fortsetzung, nachdem die Platte fertig geworden ist)*



1. Plattencontroller signalisiert per *Interrupt* den Abschluss der Operation
2. Aufruf der ISR
3. Aufruf der Komplettierungsroutine
4. Aufruf der Komplettierungsroutine
5. Weiterer (Teil-)Auftrag an den Datenträgertreiber

Fallstudie: Windows *Plug-and-Play* (1)

Hardware-Erkennung

- PnP-kompatible Geräte werden vom Hersteller bei der Produktion mit eindeutigen Geräte-Kennzahlen *(Device Identification Strings)* versehen
  - *Device ID:* durch den Hersteller vergebene Kennzahl; ein Gerät hat stets genau eine *Device ID*
  - *Hardware IDs:* eine oder mehrere durch den Hersteller vergebene Kennzahlen; per Definition ist die *Device ID* stets die erste Kennzahl in der Liste der *Hardware IDs*
  - *Compatible IDs:* zeigen an, dass ein Gerät kompatibel mit einem anderen ist und dessen Treiber verwenden kann; ein Gerät kann *Compatible IDs* haben, muss aber nicht

Fallstudie: Windows *Plug-and-Play* (2)

Ablauf im Fall von USB-Geräten

- Anschluss eines USB-Geräts, Rechner stellt Änderung am USB-Bus fest
- Der Rechner fragt die *Device Identification Strings* des Geräts ab, das neue Gerät liefert diese über standardisierte USB-Nachrichten zurück
- Windows sucht nach einem zum Gerät passenden Treiber
  - Die in der *Registry* hinterlegten .inf-Dateien werden nach IDs durchsucht, die am besten zu den IDs des neuen Geräts passen
    - Stimmt die ID in .inf-Datei exakt mit *Device ID* überein, hat man den exakt passenden Treiber gefunden
    - Stimmt die .inf-ID nur mit einer *Hardware ID* überein, hat Windows nur einen „brauchbaren“ Treiber gefunden

    ☞Angabe der *Hardware IDs* in Folge absteigender „Brauchbarkeit“

    - Ansonsten Abgleich mit den *Compatible IDs* des Geräts

Fallstudie: Windows *Power Management* (1)

Unterstützung des *Advanced Configuration and Power Interface (ACPI)*

– ACPI versetzt komplettes Rechensystem und Geräte in energiesparende Zustände

☞ System (*Mainboard* und BIOS) und Geräte müssen ACPI unterstützen

– ACPI-Standard definiert sechs System- und vier Gerätezustände
– Der Windows Power-Manager verwaltet Systemzustände und versetzt einen Rechner in ACPI-Modi S0 – S5
– Power-Manager schickt Nachrichten an Gerätetreiber, um angeschlossene Geräte in *low-power* Zustände zu versetzen
– Gerätetreiber konvertiert den Systemzustand des Power-Managers in Gerätezustand D0 – D3; Treiber sichert aktuellen Zustand des Geräts bzw. stellt diesen wieder her und legt Gerät „schlafen“

Fallstudie: Windows *Power Management* (2)

ACPI-Systemzustände

- S0 *(Working)*: Maschine ist voll verfügbar
- S1 *(Standby)*: Kontext von CPU und Speichern bleibt erhalten; CPU ist angehalten und führt keine Befehle aus; CPU und Speicher stehen noch unter Strom; Wiederanlaufen in ≈ 2s
- S2 *(Standby)*: weniger Stromaufnahme als S1; Kontext von CPU und insbes. *Caches* geht verloren und muss durch Betriebssystem gesichert werden; *Caches* sind abgeschaltet; Wiederanlaufen in > 2s
- S3 *(Suspend-to-RAM, Standby, Sleep)*: nur noch Hauptspeicher erhält Strom; CPU, *Caches* und (Teile des) Mainboard sind ausgeschaltet
- S4 *(Suspend-to-Disk, Hibernation)*: System ist vollständig abgeschaltet; Speicherinhalte und Kontexte sind auf Platte gesichert
- S5 *(Off)*

Fallstudie: Windows *Power Management* (3)

ACPI-Gerätezustände

- D0 *(Fully-On)*: Gerät ist vollständig aktiv und einsatzbereit
- D1: Verhalten des Geräts herstellerabhängig; reduzierte Stromaufnahme und schneller Übergang nach D0; Treiber kann auf Gerät nicht zugreifen
- D2: weniger Stromaufnahme als D1, aber längere Zeit bis D0 wieder erreicht ist; Geräteverhalten ebenfalls herstellerabhängig; Treiber kann auf Gerät nicht zugreifen
- D3 *(Off)*: Gerät ist komplett von der Stromversorgung abgetrennt; Betriebssystem muss Gerät neu initialisieren, wenn es wieder hochgefahren wird

- Für viele Klassen von Geräten sind D1 und D2 nicht definiert

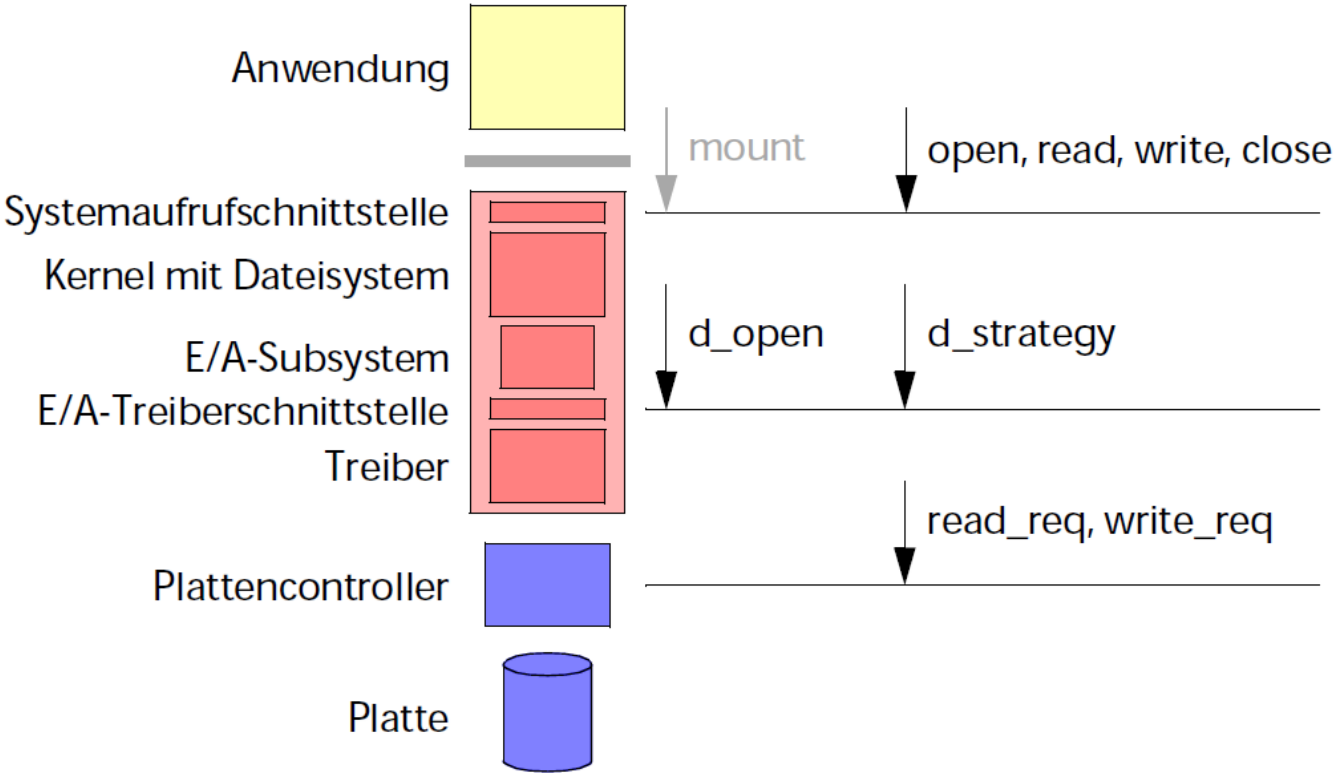
Roter Faden

9. Ein-/Ausgabe und Gerätetreiber

- Geräteaufbau *(exemplarisch)*
- Treiberschnittstelle und Treiberimplementierung
- Fallstudie: UNIX/Linux
- Fallstudie: Windows I/O-System
  - Aufbau und Funktionsweise
  - *Plug-and-Play*
  - *Power Management*
- Festplattentreiber
- Treiber für weitere Geräte

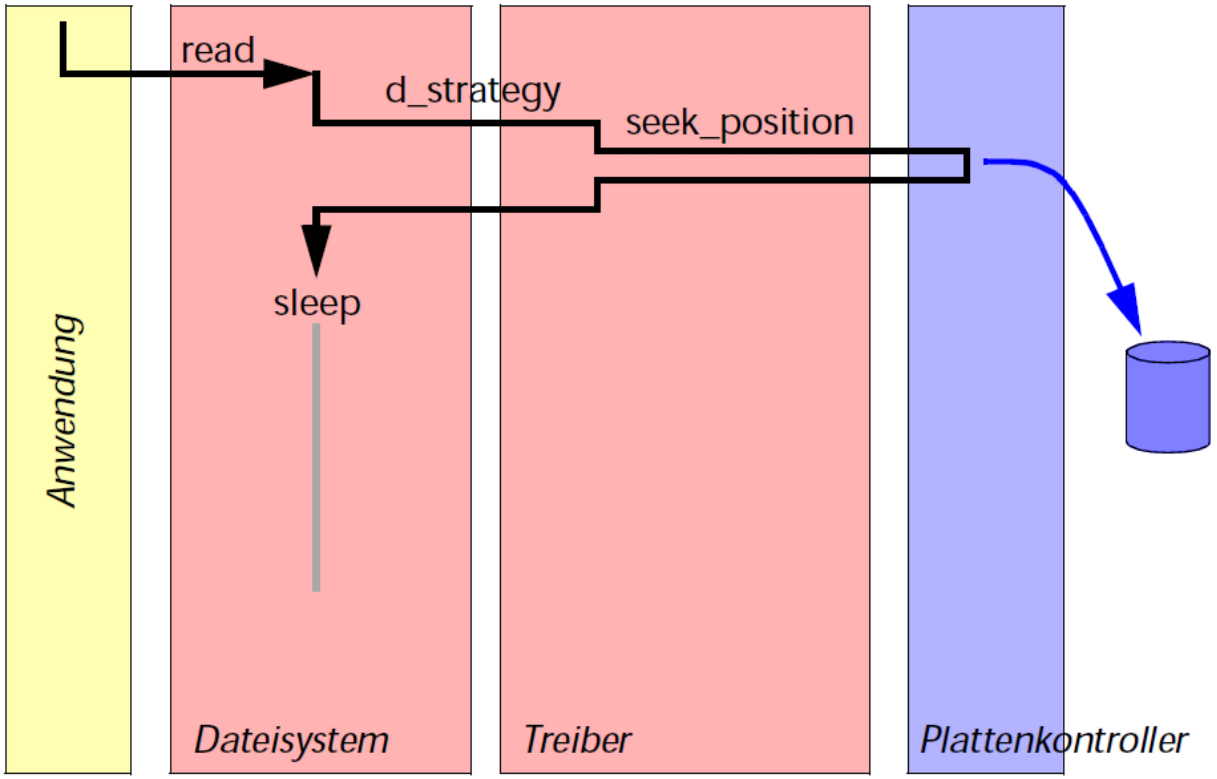
Plattentreiber

Software und Hardware zwischen Anwender und Platte



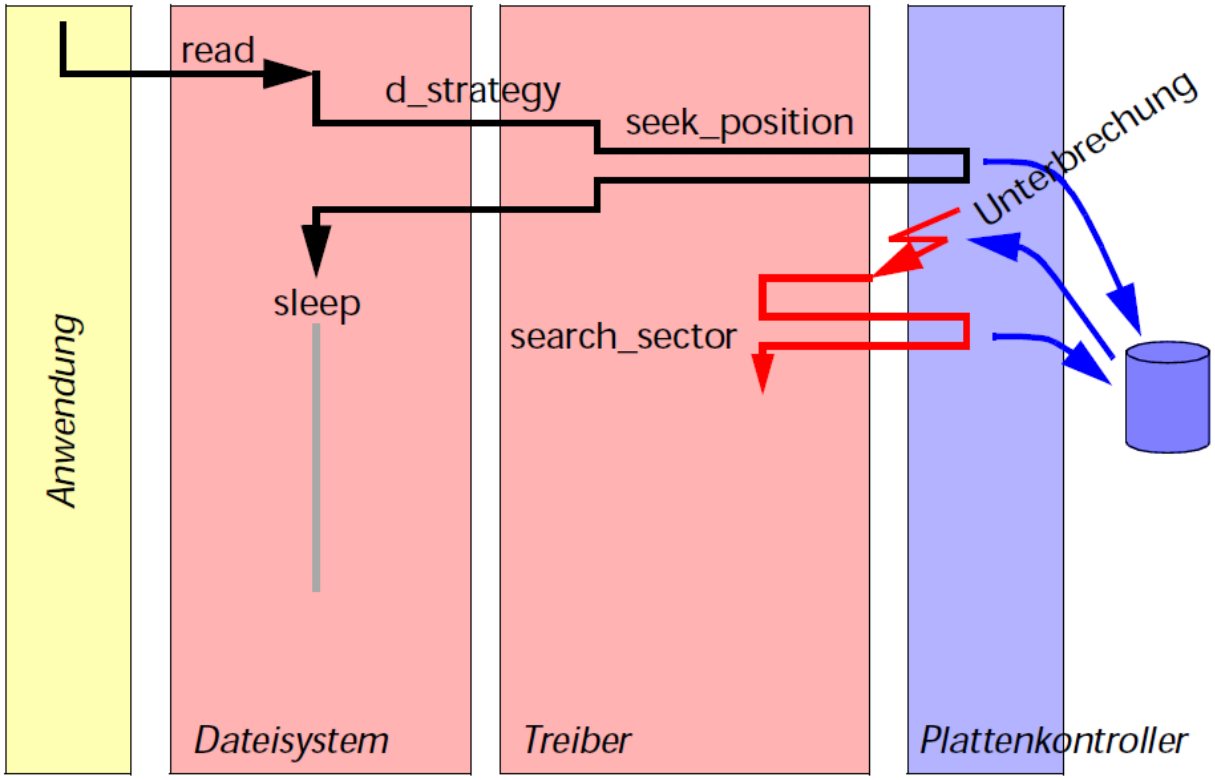
Einfacher Plattentreiber (1)

Ablauf eines Leseaufrufs (synchron)



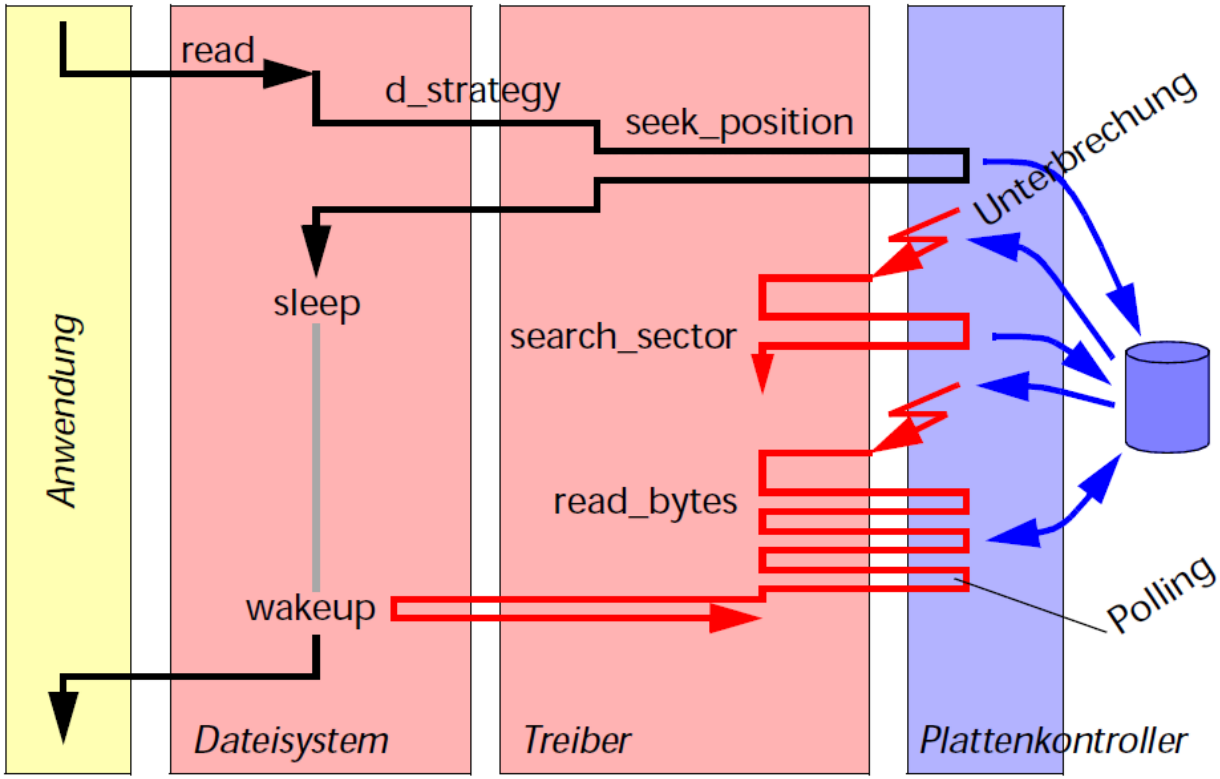
Einfacher Plattentreiber (2)

Ablauf eines Leseaufrufs (synchron)



Einfacher Plattentreiber (3)

Ablauf eines Leseaufrufs (synchron)

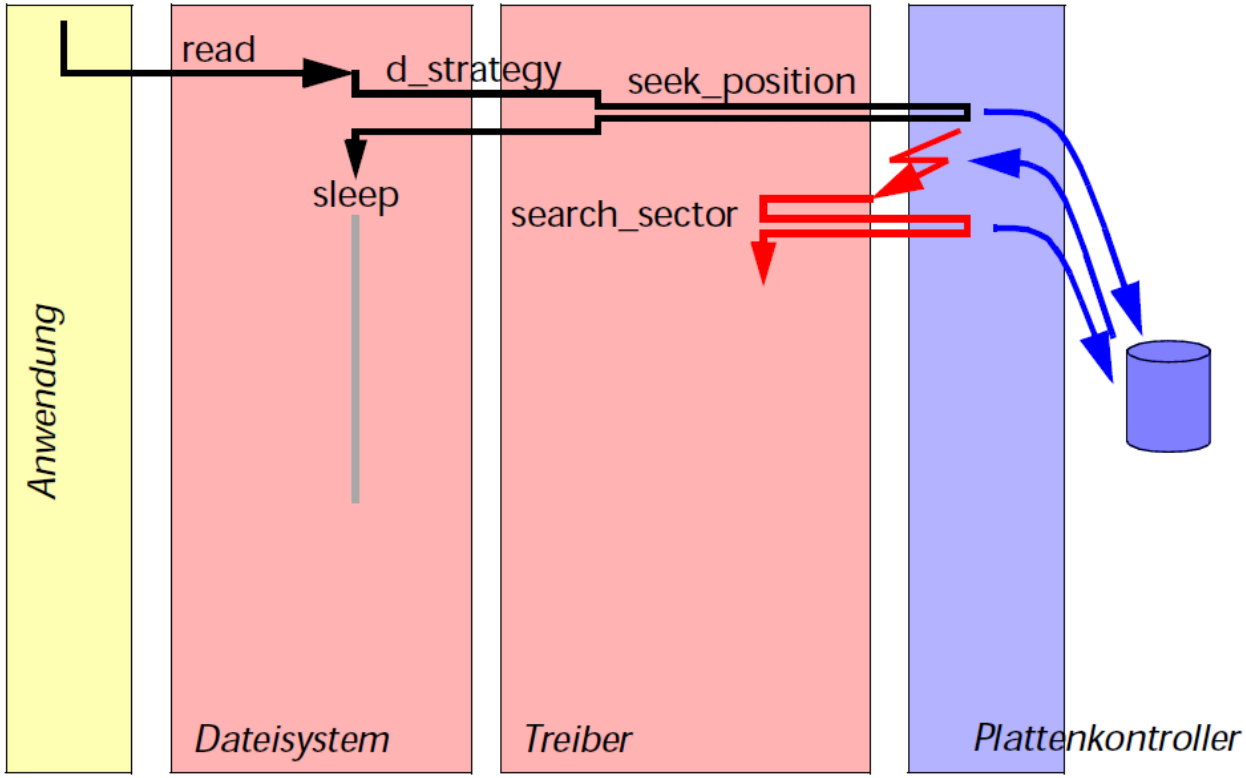


Einfacher Plattentreiber (4)

- Anwendung führt **`read()`** Systemaufruf aus
- Dateisystem prüft, ob entsprechender Block im Speicher vorhanden
- Falls Block nicht vorhanden: Speicherplatz wird bereitgestellt und **`d_strategy()`** im entsprechenden Treiber aufgerufen
- Ausführung von **`d_strategy()`** stößt Plattenpositionierung an
- Anwendung blockiert sich im Kernel, System kann andere Prozesse laufen lassen
- Plattencontroller meldet sich bei erfolgter Positionierung per *Interrupt*
- *Interrupt-Handler* im Treiber stößt Sektorsuche an
- Nach erneuter Unterbrechung bei gefundenem Sektor werden Daten mittels *Polling* eingelesen
- Anwendungsprozess wird wieder aufgeweckt, d.h. in den Zustand „bereit“ überführt

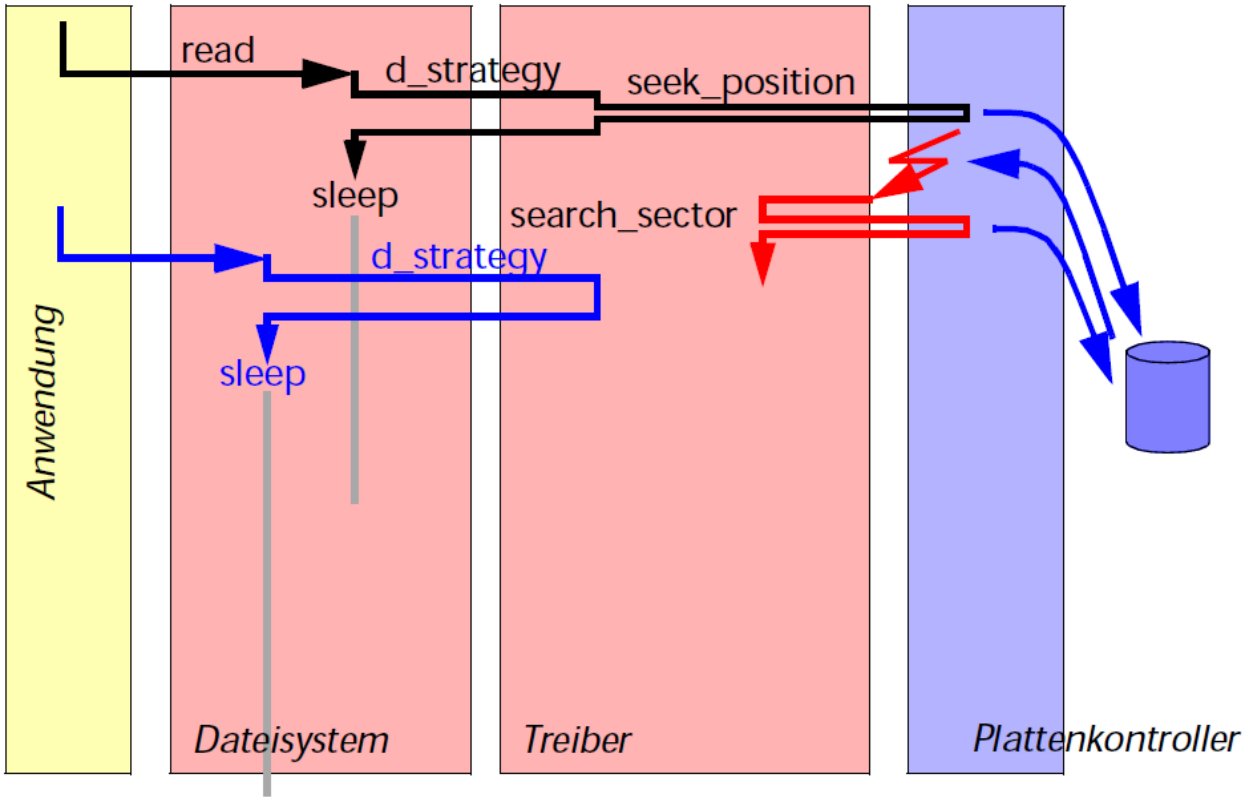
Einfacher Plattentreiber (5)

Ablauf mehrerer synchroner Leseaufrufe



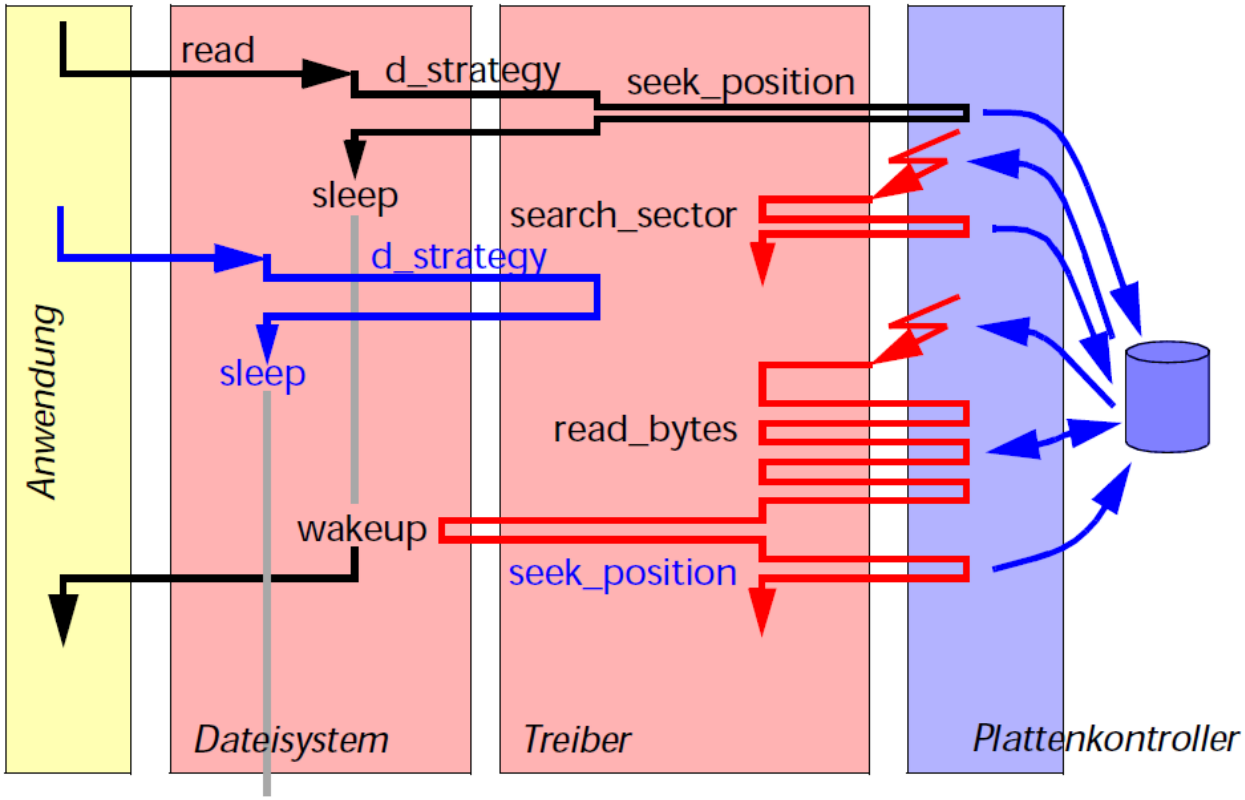
Einfacher Plattentreiber (6)

Ablauf mehrerer synchroner Leseaufrufe



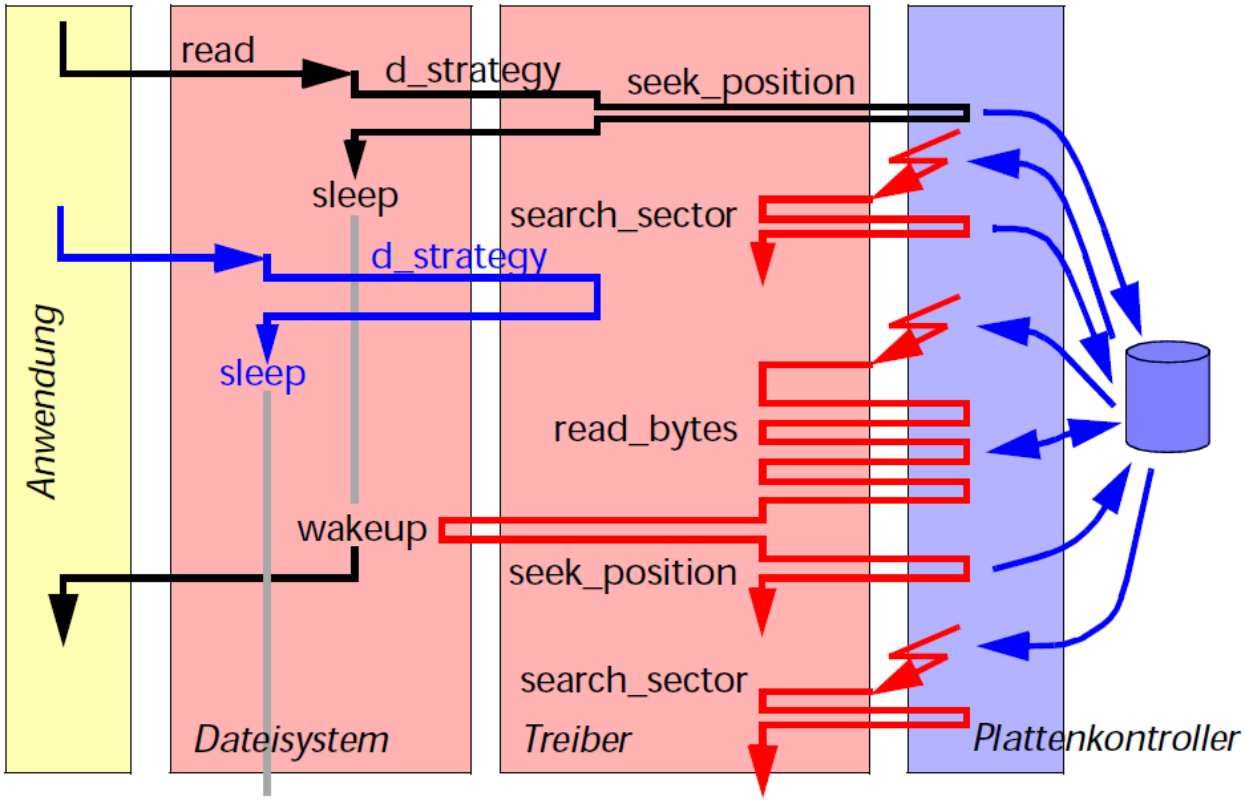
Einfacher Plattentreiber (7)

Ablauf mehrerer synchroner Leseaufrufe



Einfacher Plattentreiber (8)

Ablauf mehrerer synchroner Leseaufrufe



Einfacher Plattentreiber (9)

Unterbrechungsbehandlung ist auch für weitere Aufträge zuständig

- Zugehörige Prozesse sind alle blockiert
- Unterbrechungsbehandlung am Schluss eines Auftrags
  - Beenden des Auftrags
  - Deblockieren des zugehörigen Prozesses
  - Auswählen und Aufsetzen eines Folgeauftrags (falls vorhanden)

Direct Memory Access, DMA (1)

Bislang: Datentransport zwischen Speicher und I/O-Bausteinen

- Nur der Prozessor hat die volle Kontrolle über den Bus (legt Adressen an, erzeugt Kontrollsignale, ...)
- Andere Einheiten dürfen nur auf Anforderung des Prozessors tätig werden
- Prozessor liest aus Speicher und übergibt Daten an I/O-Baustein oder liest Daten vom I/O-Baustein und schreibt in Speicher
  - Üblicherweise relativ viele Daten zu übertragen (z.B. Plattentransfer)
  - Ständige Wartezeiten durch relativ langsame I/O-Geräte

☞ ***Nur der Prozessor hat bislang Funktion des Bus-Masters.***

☞ ***Programmed Input/Output (PIO)***

Direct Memory Access, DMA (2)

Beim DMA werden die Datentransporte *direkt* zwischen Gerätesteuerung und Speicher durchgeführt, *ohne* Beteiligung des Prozessors.

- Spezielle Hardware tritt als Aktor auf dem Bus auf
  - Liest Daten und übergibt sie an I/O-Baustein, oder umgekehrt
  - CPU ist währenddessen frei (lediglich Bus teilw. durch DMA belegt)
- Programmierung des DMA-Systems
  - Länge, Speicheradresse, I/O-Adresse von Ein-/Ausgaberegister
  - Unterbrechung zur Signalisierung des Transferendes
  - Unterbrechung zur Signalisierung von Fehlern
  - ☞ Aus Sicht des Programmierers ist DMA die einfachste Technik.

☞ ***DMA macht es erforderlich, dass Gerätesteuerungen Bus-Master werden können.***

Direct Memory Access, DMA (3)

Arbeitsmodi

- *Burst*-Modus: DMA-Baustein übernimmt Systembus für die Dauer des vollständigen Transfers
  - I/O-Baustein benötigt internen Pufferspeicher
  - Vorteil: hohe Transferrate möglich
  - Nachteil: CPU für lange Zeit am Speicher-/Buszugriff gehindert
- *Cycle Stealing:* Mischen von DMA- und CPU-Buszyklen
  - Fester Anteil an Buszyklen (z.B. jeder zweite) wird vom DMA-Baustein der CPU „gestohlen“
  - DMA kann zusätzlich alle von der CPU nicht benötigten Buszyklen nutzen (z.B. während der Bearbeitung von Maschineninstruktionen, die nur CPU-interne Ressourcen benötigen)
  - Vorteil: CPU nur geringfügig behindert
  - Nachteil: geringere Transferrate

Treiber mit DMA (1)

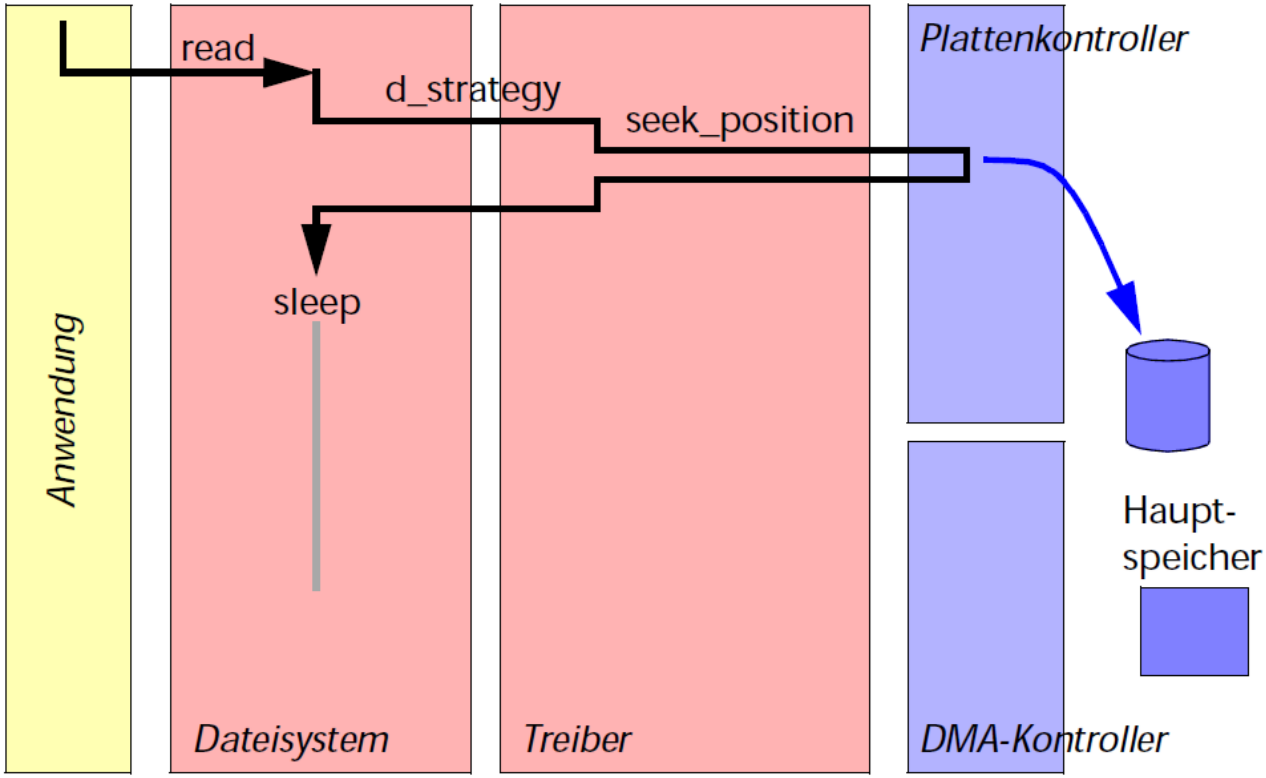
DMA erlaubt Einlesen und Schreiben ohne Prozessorbeteiligung

- DMA-Controller erhält vom Treiber verschiedene Parameter
  - Die Hauptspeicheradresse zum Abspeichern bzw. Auslesen eines Plattenblocks
  - Die Adresse des Plattencontrollers zum Abholen bzw. Abgeben der Daten
  - Die Länge der zu transferierenden Daten
- DMA-Controller löst bei Fertigstellung eine Unterbrechung aus

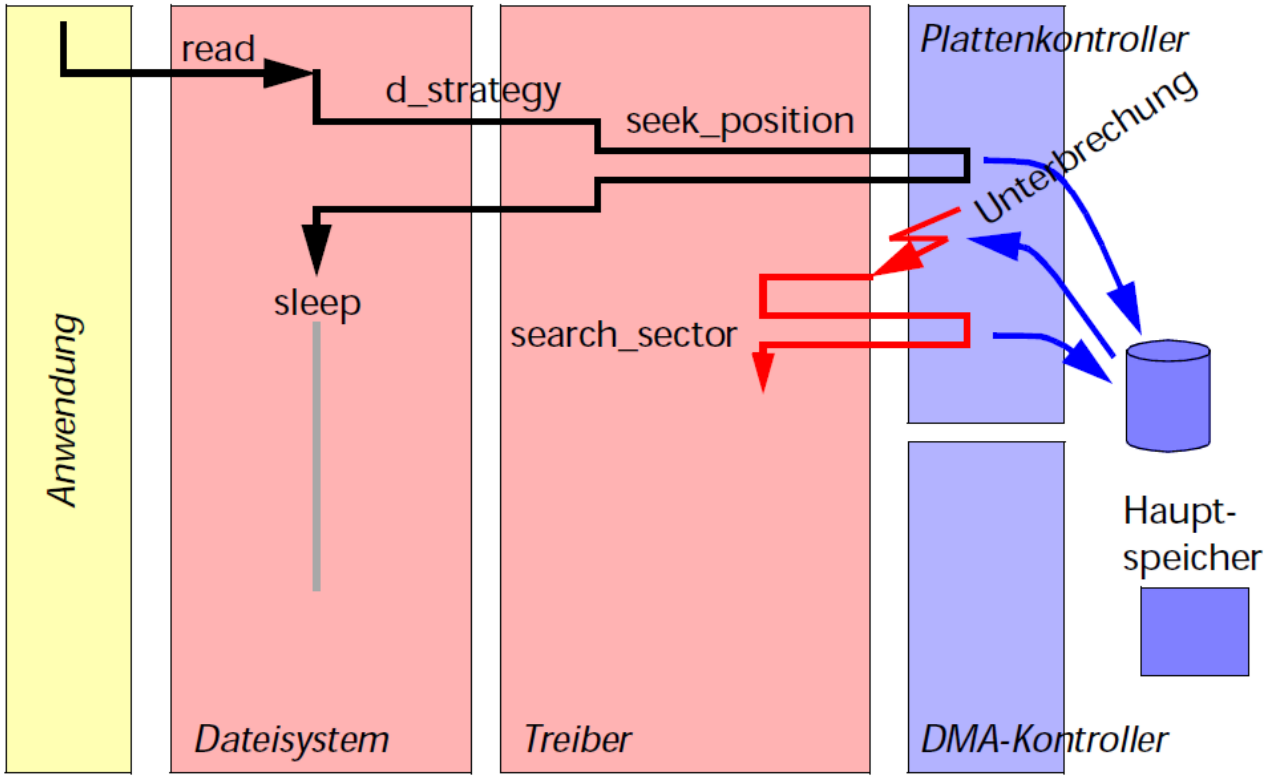
Vorteile

- Prozessor muss Zeichen eines Plattenblocks nicht selbst abnehmen (kein *Polling* sondern *Interrupt*)
- Plattentransferzeit kann zum Ablauf anderer Prozesse genutzt werden

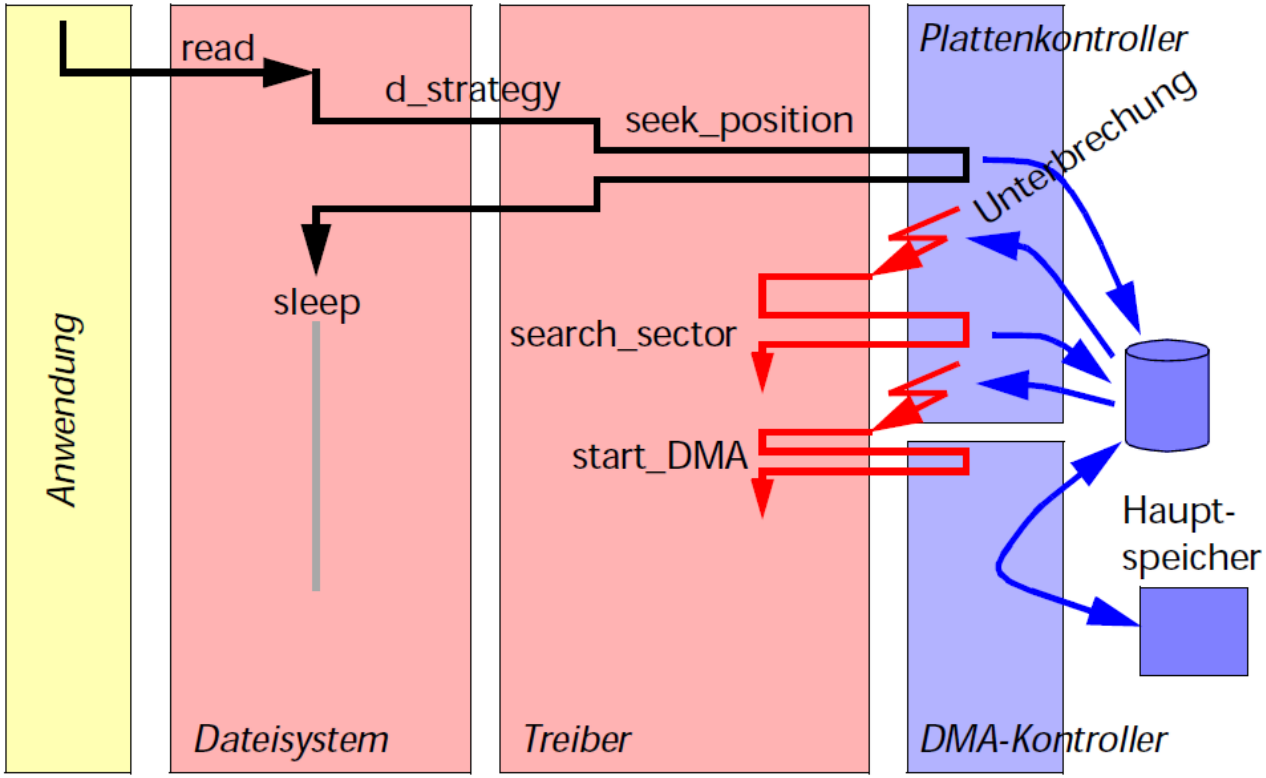
Treiber mit DMA (2)



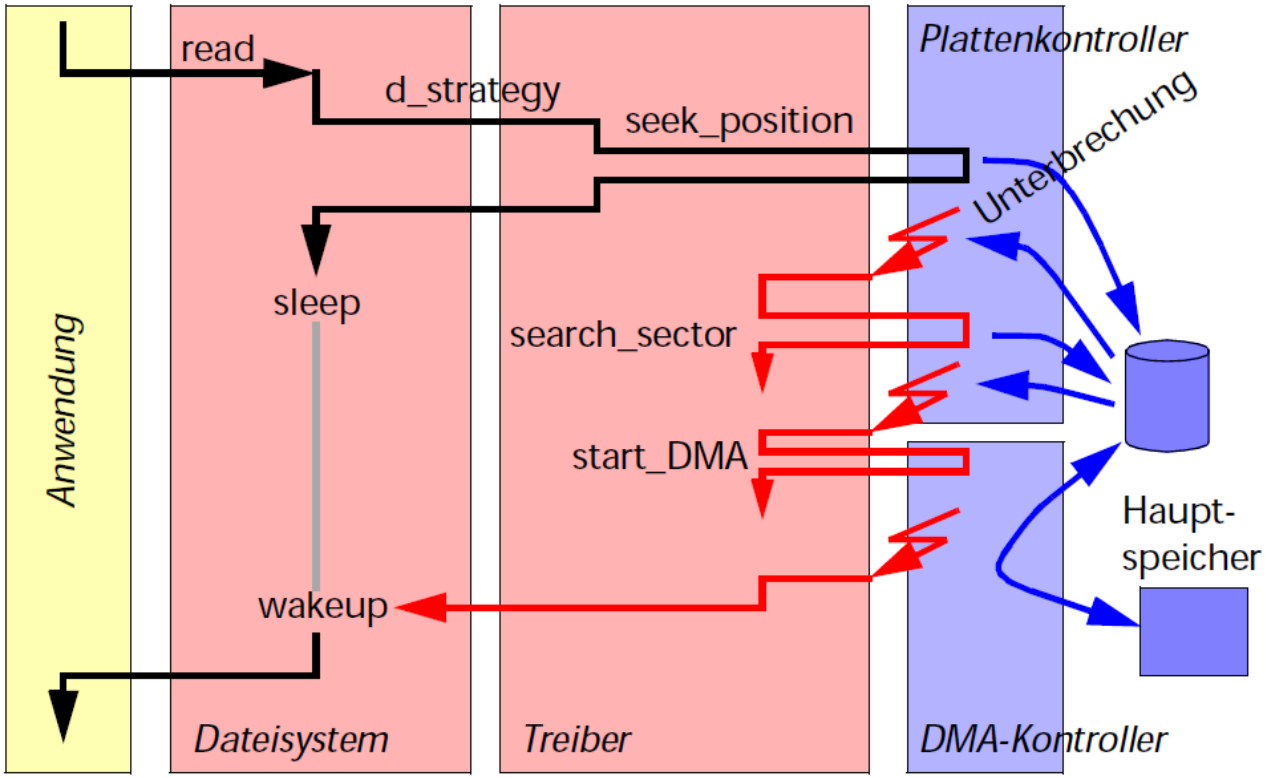
Treiber mit DMA (3)



Treiber mit DMA (4)



Treiber mit DMA (5)



Treiber mit DMA (6)

Große Systeme mit mehreren DMA-Kanälen und vielen Platten

– Es muss ein freier DMA-Kanal gesucht werden bzw. auf einen freien gewartet werden, bevor der Auftrag ausgeführt werden kann
– Anforderung kann parallel zur Plattenpositionierung erfolgen

***Mainframe*-Systeme**

– Steuereinheit fasst mehrere Platten zu einem Gerät zusammen
– Mehrere Steuereinheiten hängen an einem Kanal zum Hauptspeicher
– Zum Zugriff auf die eigentliche Platte muss erst die Steuereinheit und dann der Kanal belegt werden (Teilwegbelegung)

DMA und *Caching*

– Heutige Prozessoren arbeiten mit Datencaches
– DMA läuft am *Cache* vorbei: Betriebssystem muss vor dem Aufsetzen von DMA-Transfers *Caches* zurückschreiben und invalidieren

Disk-*Scheduling*

Plattentreiber hat in der Regel mehrere Aufträge in der Warteschlange

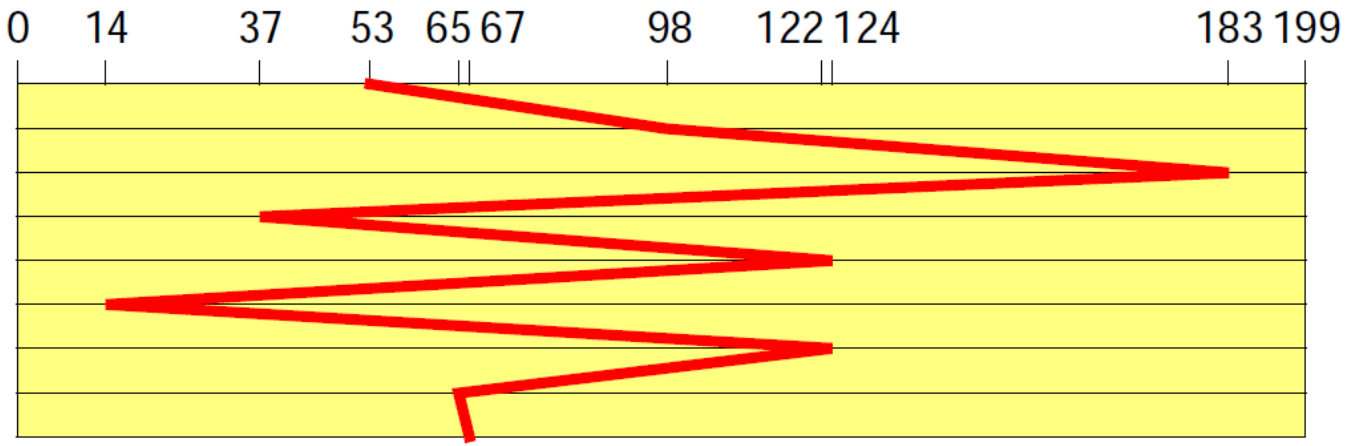
- Warteschlange wird z.B. in UNIX durch Aufruf der Funktion **`d_strategy()`** gefüllt
- Eine bestimmte Ordnung der Ausführung kann Effizienz steigern
- Zusammensetzung der Bearbeitungszeit eines Auftrags
  - Positionierzeit: abhängig von der aktuellen Stellung des Plattenarms
  - Latenzzeit: Zeit, bis der Magnetkopf den Sektor bestreicht
  - Übertragungszeit: Zeit zur Übertragung der eigentlichen Daten

Ansatzpunkt: Positionierzeit

First-Come, First Serve Scheduling (FCFS)

Bearbeitung gemäß Ankunft des Auftrags

- Referenzfolge (Folge von Zylindernummern):
  98, 183, 37, 122, 14, 124, 65, 67
- Aktueller Zylinder: 53

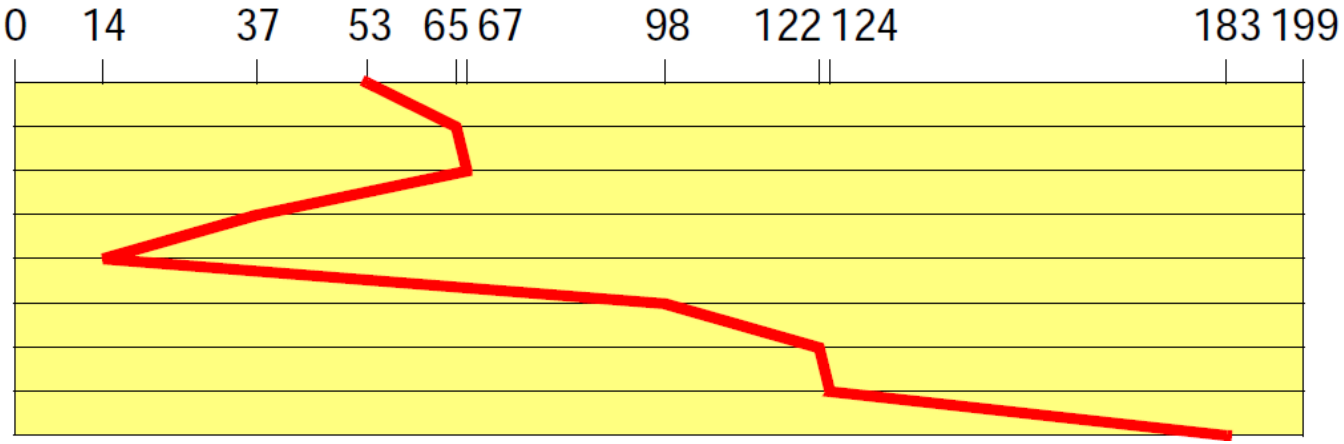


- Gesamtzahl der Spurwechsel: 640
- Weite Bewegungen des Schwenkarms; mittlere Bearbeitungsdauer lang

Shortest Seek Time First Scheduling (SSTF)

Es wird der Auftrag mit der kürzesten Positionierzeit vorgezogen

- Eine *Tie Break* Heuristik entscheidet, wenn zwei Aufträge die gleiche kürzeste Positionierzeit haben
- Gleiche Referenzfolge
  (Annahme: Positionierzeit proportional zum Zylinderabstand)



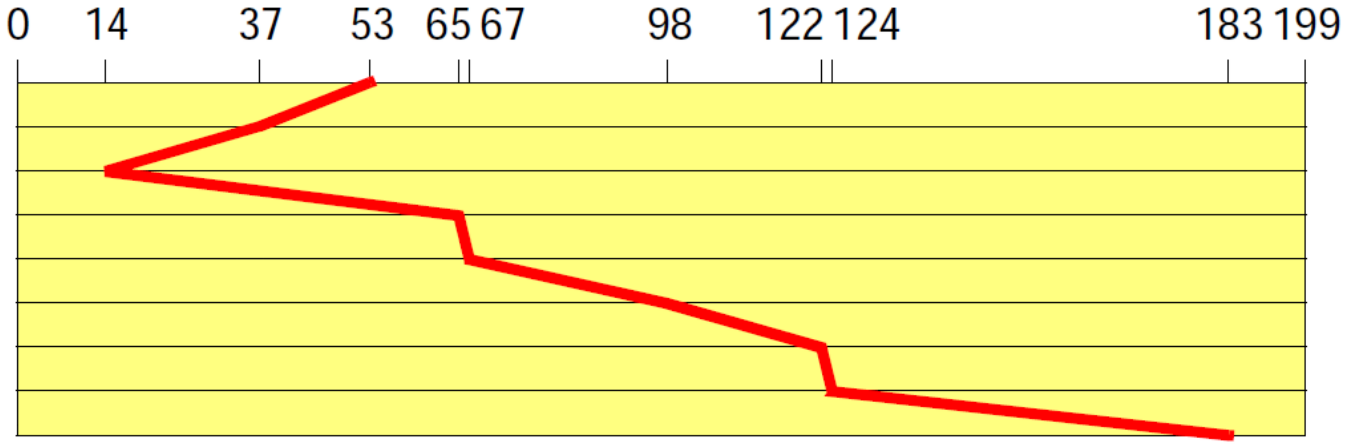
- Gesamtzahl der Spurwechsel: 236
- SSTF kann zur Aushungerung führen, noch nicht optimal

SCAN Scheduling

Bewegung des Plattenarms in eine Richtung, bis keine Aufträge in dieser Richtung mehr vorhanden sind (Fahrstuhlstrategie)

– Gleiche Referenzfolge

(Annahme: bisherige Kopfbewegung in Richtung 0)



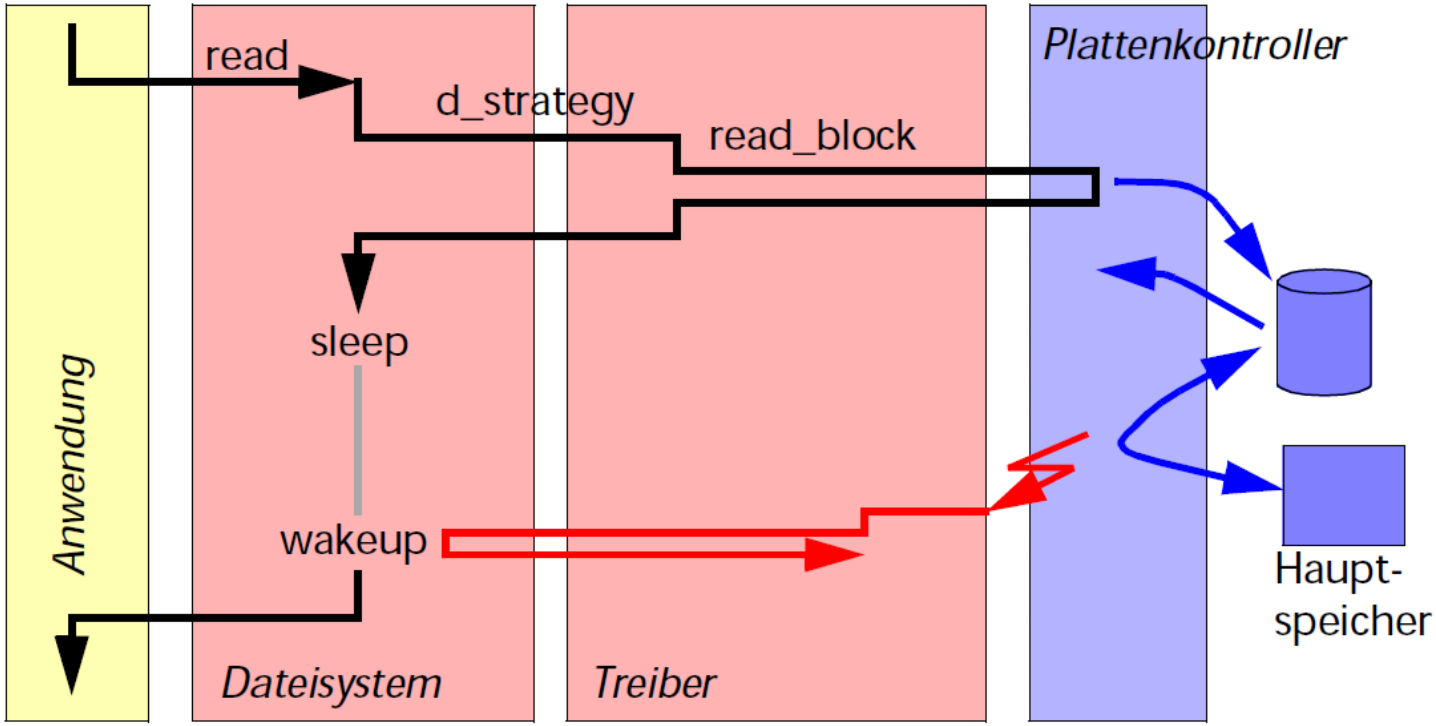
– Gesamtzahl der Spurwechsel: 208

– Neue Aufträge werden miterledigt ohne zusätzliche Positionierzeit und ohne Aushungerung

– Variante *C-SCAN (Circular SCAN)*: Bewegung in nur eine Richtung

Treiber für Busmaster-Controller

Heutige Festplattencontroller können Bus-Master sein



– Controller transportiert Daten selbständig in den Hauptspeicher bzw. aus dem Hauptspeicher heraus

Roter Faden

9. Ein-/Ausgabe und Gerätetreiber

- Geräteaufbau *(exemplarisch)*
- Treiberschnittstelle und Treiberimplementierung
- Fallstudie: UNIX/Linux
- Fallstudie: Windows I/O-System
- Festplattentreiber
  - *Interrupt*-basiert
  - DMA-basiert
  - Disk-*Scheduling*
    - *First-Come, First Serve (FCFS)*
    - *Shortest Seek Time First (SSTF)*
    - SCAN
- Treiber für weitere Geräte

Ähnliche Treiber

CD-ROM

– Wird wie Platte behandelt (eigener Treiber)
– Nicht beschreibbar
– Spezielle Treiber möglich für Audio-Tracks

Floppy Disk

– Wird im Prinzip wie Platte behandelt (eigener Treiber)

Treiber für serielle Schnittstellen

Treiber

- Zeichenorientiertes Gerät
- Zeichenorientierte Spezialdateien (z.B. **`/dev/tty`** unter UNIX)
- Vom Prinzip her ähnlich dem Plattentreiber
  - Lediglich zeichenorientierte statt blockorientierte Übertragung

Parametereinstellung

- Neben **`read()`** und **`write()`** dritter Systemaufruf **`ioctl()`** *(I/O Control)*
- Treiberabhängige Funktionsparameter zum Einstellen von Geräteparametern
  - Für serielle Schnittstellen festgelegte Einstellungen
  - Treiber übernimmt Editierfunktionen sowie Softwareflusskontrolle

I/O Controls zum Einstellen gerätespezifischer Parameter

Spezielle Geräteeigenschaften werden über `ioctl` angesprochen

```
IOCTL(2)                  Linux Programmer's Manual                 IOCTL(2)

NAME
       ioctl - control device

SYNOPSIS
       #include <sys/ioctl.h>

       int ioctl(int d, int request, ...);
```

Schnittstelle generisch und Semantik gerätespezifisch

```
CONFORMING TO
       No single standard. Arguments, returns, and semantics of
       ioctl() vary according to the device driver in question (the
       call is used as a catch-all for operations that don't cleanly
       fit the UNIX stream I/O model). The ioctl() function call
       appeared in Version 7 of AT&T UNIX.
```

Bildschirmtreiber

Bildspeicher

- Spezieller Speicher, der den Bildschirminhalt repräsentiert
- Zeichenorientiert
- Pixelorientiert

Aufgaben des Treibers

- Bereitstellen von Grafikprimitiven (z.B. Ausgabe von Text, Zeichnen von Rechtecken, etc.)
- Ansprechen von Grafikprozessoren (schnelle Verschiebeoperationen, komplexe Zeichenoperationen, 3D Rendering, Texturen)
- Einblenden des Bildspeichers in Anwendungsprogramme (z.B. X11-Server)

Netzwerktreiber (1)

Beispiel: Ethernet

- Schneller serieller Bus mit CSMA/CD
  (Carrier Sense Multiple Access / Collision Detect)
  Zu deutsch: jeder sendet, wenn er meint, der Träger sei frei, und zieht sich zurück, wenn es eine Kollision gibt.
- Spezieller Netzwerkchip
  - Implementiert unterstes Kommunikationsprotokoll
  - Erkennt eintreffende Pakete

Netzwerktreiber

- Wird von höheren Protokollschichten innerhalb des Betriebssystems angesprochen, z.B. von der IP-Schicht

Netzwerktreiber (2)

Senden

– Treiber übergibt Netzwerkchip eine Datenstruktur mit den notwendigen Informationen: Sendeadresse, Adresse und Länge von Datenpuffern
– Netzwerkchip löst *Interrupt* bei erfolgtem Senden aus

Empfangen

– Treiber übergibt dem Netzwerkchip eine Datenstruktur mit Adressen freier Arbeitspuffer
– Erkennt der Netzwerkchip ein Paket (für die eigene MAC-Adresse), füllt er das Paket in einen freien Puffer
– Der Puffer wird in eine Liste von empfangenen Paketen eingehängt, ein *Interrupt* wird ausgelöst
– Treiber kann die empfangenen Pakete aushängen

Netzwerktreiber (3)

Übertragung der Daten erfolgt durch DMA bzw. Busmastering

– Evtl. direkt durch den Netzwerkchip

Intelligente und nicht-intelligente Netzwerkhardware

– Intelligente Hardware: kann evtl. auch höhere Protokolle, Filterung etc.

– Nicht-intelligente Hardware: benötigt mehr Unterstützung durch den Treiber (Prozessor)

Zusammenfassung (1)

Treiberschnittstelle und Treiberimplementierung

- Geräteunabhängige Schicht im Betriebssystem, die gleiche Behandlung für alle Geräte einer Art erlaubt
- Implementierungen über *Polling*, *Interrupts* oder DMA
- Synchrone und asynchrone I/O-Operationen

Fallstudie: UNIX/Linux

- Unterscheidung zwischen Block- und Zeichenorientierten Geräten
- Gerätерepräsentation durch *Major* und *Minor Number* und über Spezialdateien unter **`/dev`**
- Datenstrukturen (Felder und C-**`structs`**) verweisen auf Funktionen im Code von Treibern, die für I/O mit einem Gerät aufgerufen werden können

Zusammenfassung (2)

Fallstudie: Windows I/O-System

- Windows-Kern unterscheidet zwischen I/O-System, Gerätetreibern und einer Hardware-Abstraktionsebene (HAL)
- Anwendungen und Systemdienste kommunizieren mit dem I/O-System bestehend aus I/O-Manager, PnP-Manager und Power-Manager (u.a.)
- I/O-System reicht EA-Anforderungen an Treiber weiter, enge Kopplung mit zentraler Windows-*Registry*
- *Plug-and-Play*
  - Betriebssystem erkennt Geräte anhand von *Hardware IDs* und sucht in *Registry* nach bestem passenden Treiber
- Power Management
  - Unterstützung von ACPI System- und Gerätezuständen durch Hardware und Betriebssystem, insbes. um Zustand von abgeschalteten Geräten zu sichern und restaurieren

Zusammenfassung (3)

Festplattentreiber

- *Interrupt*-basierte Implementierung: Sämtliche mechanischen Vorgänge der Platte müssen per *Interrupt* bestätigt werden; Datenübertragung beim Lesen erfolgt über *Polling*
- DMA-basierte Implementierung: DMA-Controller tritt als Bus-Master auf; Daten können zwischen Platte und Speicher ohne CPU-Beteiligung (insbes. ohne *Polling*) übertragen werden
- Weitere Variante: Festplatten-Controller tritt als Bus-Master auf; Platte kann Inhalte direkt vom/zum Hauptspeicher übertragen
- Disk-*Scheduling* wichtig, um Positionierzeiten der langsamen Festplatten-Mechanik über mehrere I/O-Anforderungen hinweg zu minimieren