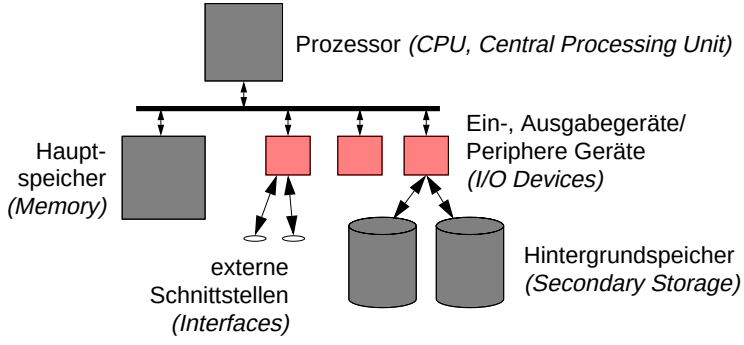
G Ein-, Ausgabe

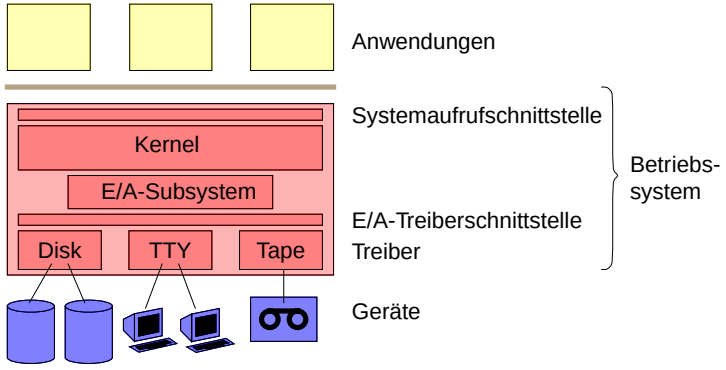
G Ein- und Ausgabe

- Einordnung



1 Gerätezugang und Treiber

- Schichtung der Systemsoftware bis zum Gerät



Nach Vahalia, 1996

1.1 Gerätерepräsentation in UNIX

- Periphere Geräte werden als 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
- Blockorientierte Spezialdateien
  - Plattenlaufwerke, Bandlaufwerke, Floppy Disks, CD-ROMs
- Zeichenorientierte Spezialdateien
  - Serielle Schnittstellen, Drucker, Audiokanäle etc.
  - blockorientierte Geräte haben meist auch eine zusätzliche zeichenorientierte Repräsentation

1.1 Geräterepräsentation in UNIX (2)

- 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

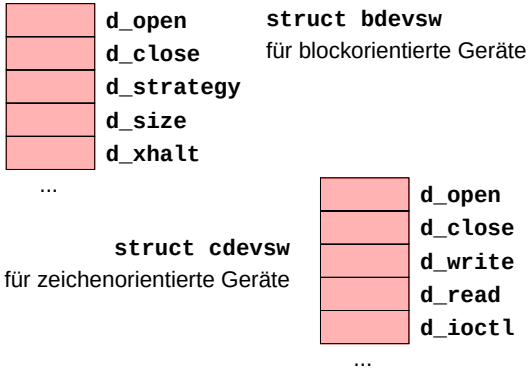
1.1 Geräterepräsentation in UNIX (3)

- Beispiel eines Kataloglisting von **`/dev`** (Ausschnitt)

```
crw-------   1 fzhauck  108,  0 Oct 16  1996 audio
crw-------   1 fzhauck  108,128 Oct 16  1996 audioctl
crw-rw-rw-   1 root      21,  0 May  3  1996 conslog
brw-rw-rw-   1 root      36,  2 Oct 16  1996 fd0
crw-------   1 fzhauck   17,  0 Oct 16  1996 mouse
crw-rw-rw-   1 root      13,  2 Jan 13 09:09 null
crw-rw-rw-   1 root      36,  2 Jul  2  1997 rfd0
crw-r-----   1 root      32,  0 Oct 16  1996 rsd3a
crw-r-----   1 root      32,  1 Oct 16  1996 rsd3b
crw-r-----   1 root      32,  2 Oct 16  1996 rsd3c
brw-r-----   1 root      32,  0 Oct 16  1996 sd3a
brw-r-----   1 root      32,  1 Oct 16  1996 sd3b
brw-r-----   1 root      32,  2 Oct 16  1996 sd3c
crw-rw-rw-   1 root      22,  0 Sep 19 09:11 tty
crw-rw-rw-   1 root      29,  0 Oct 16  1996 ttya
crw-rw-rw-   1 root      29,  1 Oct 16  1996 ttyb
```


1.1 Geräterepräsentation in UNIX (4)

- Interne Treiberschnittstelle
  - Vektor von Funktionszeigern pro Treiber (Major Number):

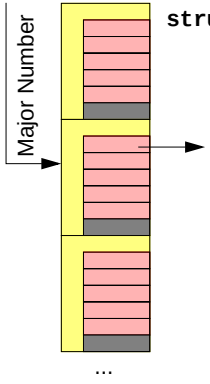


1.1 Geräterepräsentation in UNIX (5)

- 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
  - u.a.

1.1 Gerätepräsentation in UNIX (6)

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

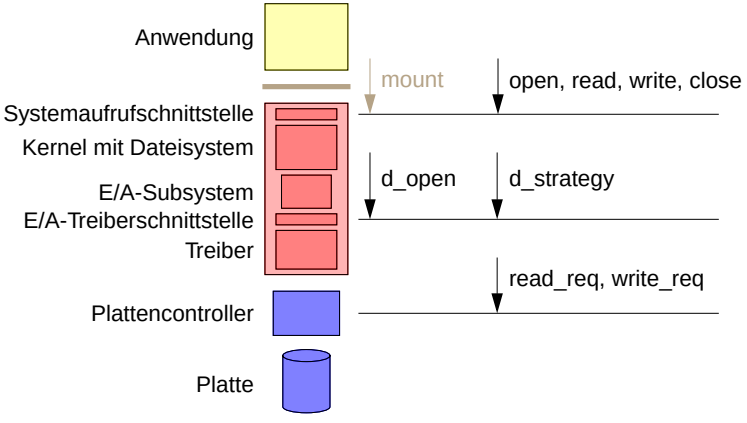


Funktion des Treibers

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

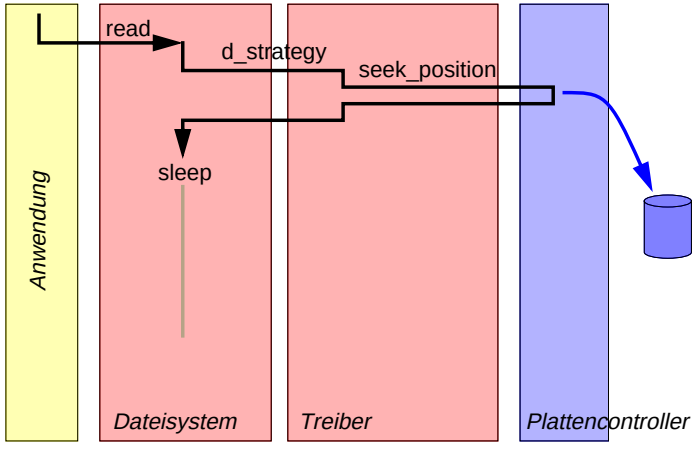
2 Plattentreiber

- Software und Hardware zwischen Anwender und Platte



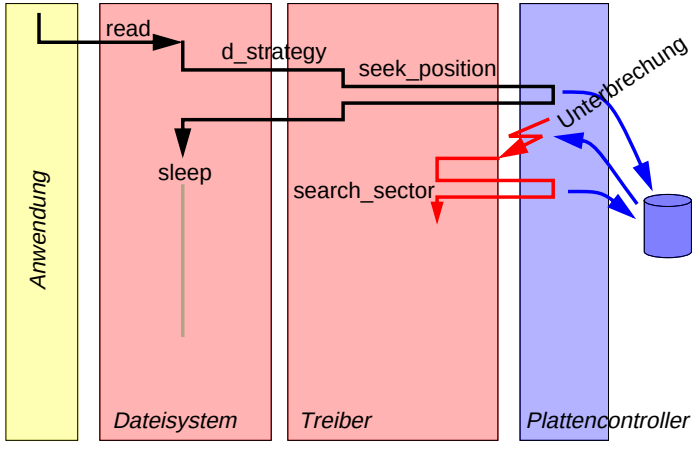
2.1 Einfacher Treiber

- Ablauf eines Leseaufrufs



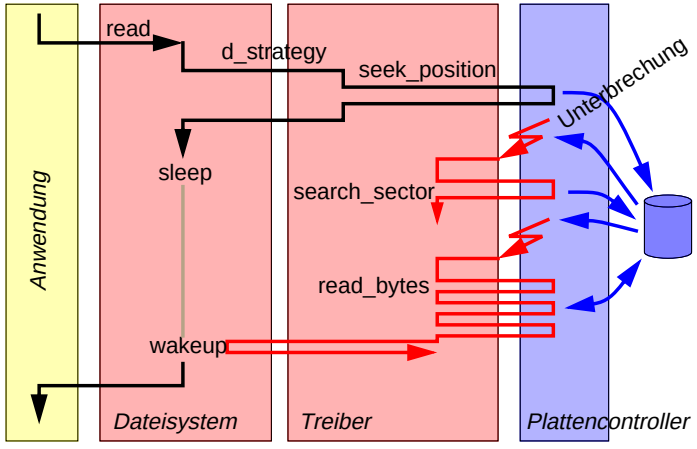
2.1 Einfacher Treiber

- Ablauf eines Leseaufrufs



2.1 Einfacher Treiber

- Ablauf eines Leseaufrufs

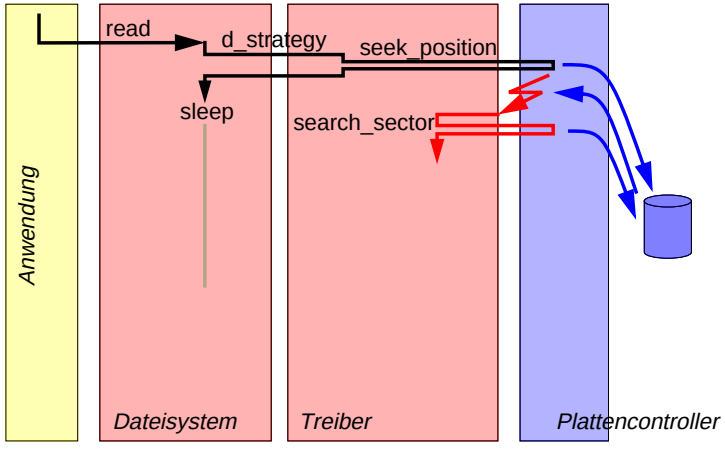


2.1 Einfacher Plattentreiber (2)

- Anwendung führt **`read()`** Systemaufruf aus.
- Dateisystem prüft, ob entsprechender Block im Speicher vorhanden.
- Falls der Block nicht vorhanden ist, wird ein Speicherplatz bereitgestellt und **`d_strategy`** im entsprechenden Treiber aufgerufen.
- Die Ausführung von **`d_strategy`** stößt Plattenpositionierung an.
- Die Anwendung blockiert sich im Kern. System kann andere Prozesse ablaufen lassen.
- Plattencontroller meldet sich bei erfolgter Positionierung durch eine Unterbrechung.
- Unterbrechungsbehandlung stößt Sektorsuche an.
- In erneuter Unterbrechung nach gefundenem Sektor werden die Daten im Pollingbetrieb eingelesen.
- Schließlich wird der Anwendungsprozess wieder aufgeweckt (in den Zustand bereit überführt).

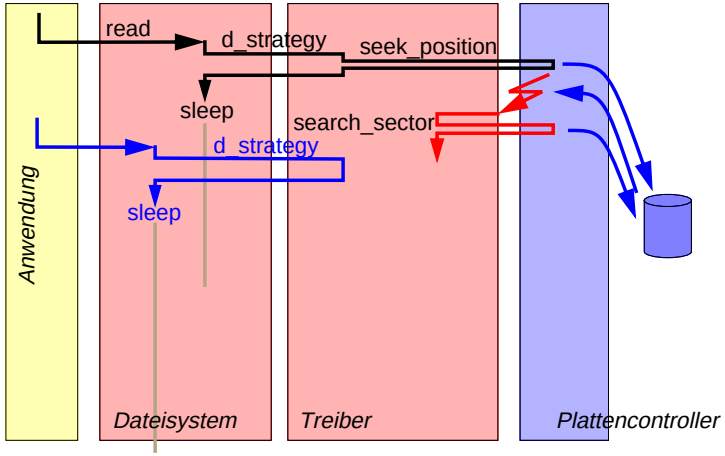
2.1 Einfacher Plattentreiber (3)

- Ablauf mehrerer Leseaufrufe



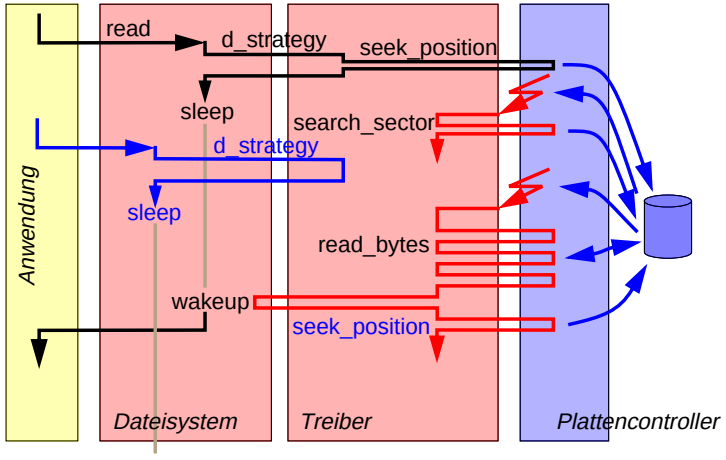
2.1 Einfacher Plattentreiber (3)

- Ablauf mehrerer Leseaufrufe



2.1 Einfacher Plattentreiber (3)

- Ablauf mehrerer Leseaufrufe

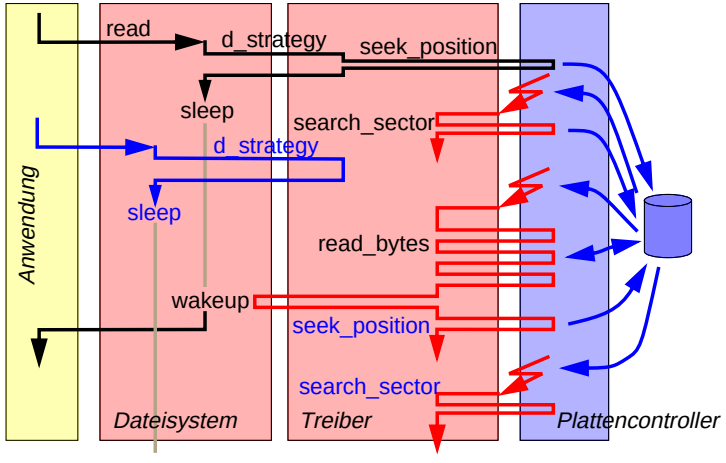


2.1 Einfacher Plattentreiber

- Unterbrechungsbehandlung ist auch für weitere Aufträge zuständig
  - Ist der Auftrag abgeschlossen, muss die Unterbrechungsbehandlung den nächsten Auftrag auswählen und aufsetzen, da der zugehörige Prozess bereits blockiert ist.
  - Die Unterbrechungen laufender Aufträge sorgen für die Abwicklung der folgenden Aufträge.

2.1 Einfacher Plattentreiber (3)

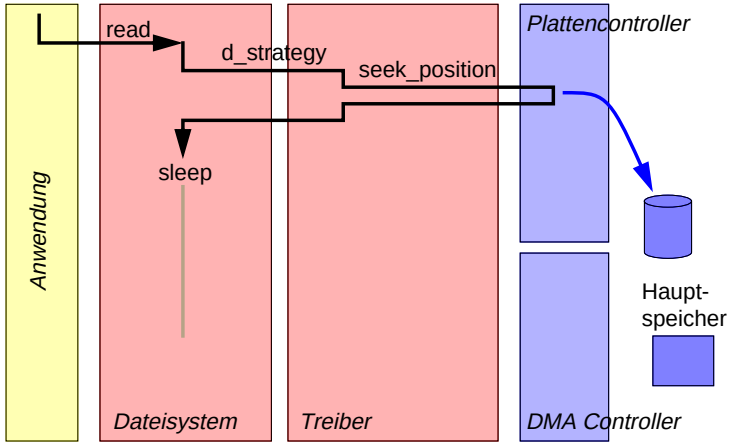
- Ablauf mehrerer Leseaufrufe



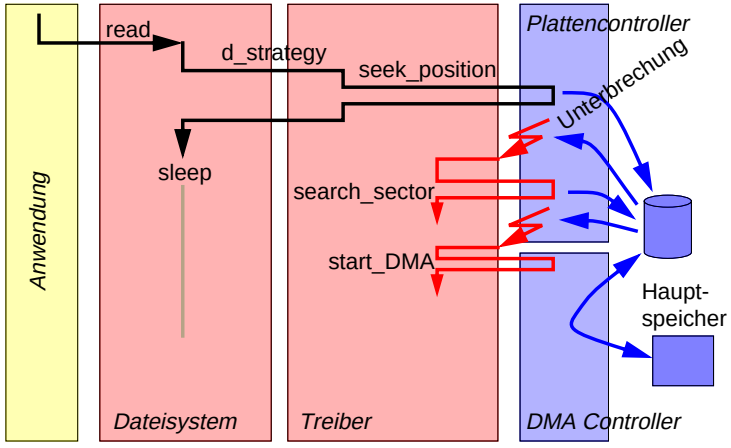
2.2 Treiber mit DMA

- DMA *(Direct Memory Access)* erlaubt Einlesen und Schreiben ohne Prozessorbeteiligung
  - DMA Controller erhält 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

2.2 Treiber mit DMA (2)

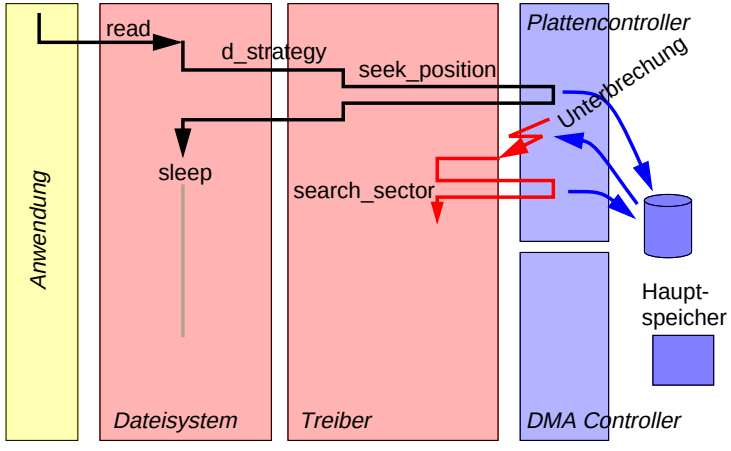


2.2 Treiber mit DMA (2)



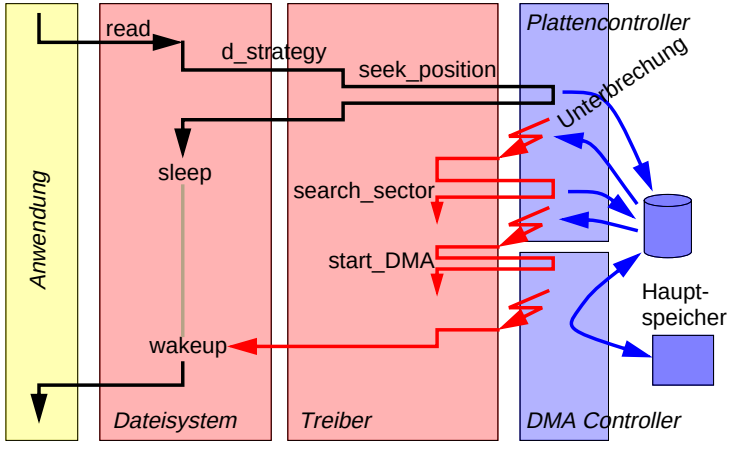
Systemprogrammierung I
G – 16
© 1997-2003, F. J. Hauck, W. Schröder-Preikschat, Inf 4, FAU Erlangen-Nürnberg[G-InOut.fm, 2004-01-20 16.58]
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2.2 Treiber mit DMA (2)



Systemprogrammierung I
G – 16
© 1997-2003, F. J. Hauck, W. Schröder-Preikschat, Inf 4, FAU Erlangen-Nürnberg[G-InOut.fm, 2004-01-20 16.58]
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2.2 Treiber mit DMA (2)



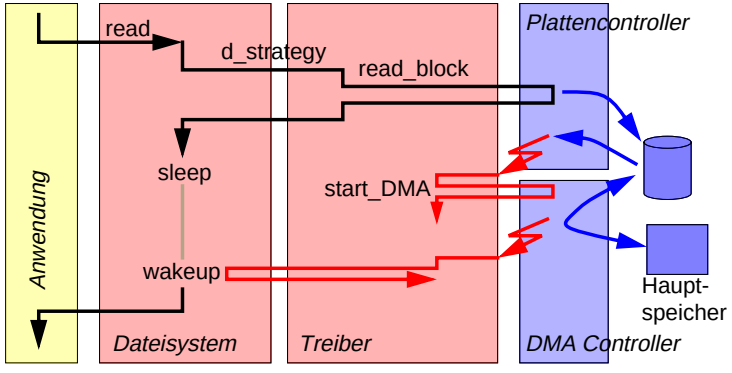
Systemprogrammierung I
G – 16
© 1997-2003, F. J. Hauck, W. Schröder-Preikschat, Inf 4, FAU Erlangen-Nürnberg[G-InOut.fm, 2004-01-20 16.58]
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2.2 Treiber mit DMA (3)

- Große Systeme mit mehreren DMA-Kanälen und vielen Platten
  - es muss ein freier DMA-Kanal gesucht werden und evtl. 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

2.3 Treiber für intelligente Platte

- Intelligente Platten besitzen eigenen Prozessor für
  - das Umsortieren von Aufträgen (interne Plattenstrategie)
  - eigene Bad block-Erkennung, etc.

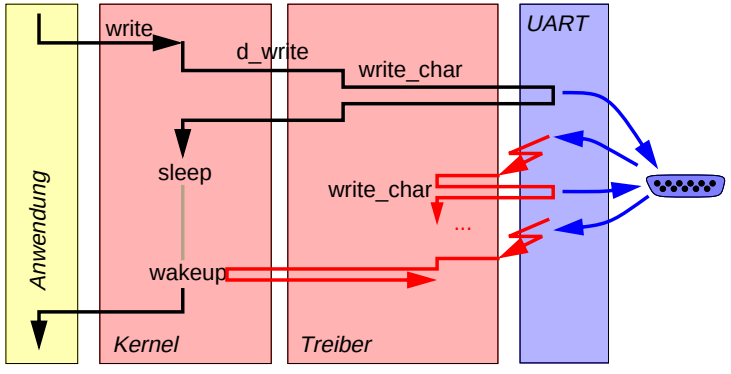


3 Treiber für serielle Schnittstellen

- Einsatz serieller Schnittstellen (z.B. RS-232)
  - Terminals
  - Drucker
  - Modems
- Datenübertragung
  - zeichenweise seriell (z.B. Startbit, Datenbits, Stopbits)
  - getaktet in bestimmter Geschwindigkeit (Bitrate, z.B. 38.400 Bit/s), im Vergleich zu Platten relativ langsam
  - Flusskontrolle (d.h. Empfänger kann Datenfluss bremsen)
  - bidirektional
- Treiber
  - zeichenorientiertes Gerät
  - vom Prinzip her ähnlich dem Plattentreiber

3.1 TTY-Treiber

- TTY-Treiber (*Teletype*, Fernschreiber) und der Ablauf eines Schreibaufrufs



  - UART = Universal Asynchronous Receiver / Transmitter

3.1 TTY-Treiber (2)

- Enger Zusammenhang zwischen Ein- und Ausgabe
  - Echofunktion (getippte Zeichen werden angezeigt)
    - eingelesene Zeichen werden gleich wieder ausgegeben
  - Flusskontrolle (bestimmtes Zeichen in der Eingabe hält Ausgabe an: ^S)
    - wird ^S eingelesen wird Ausgabe angehalten bis ^Q eingelesen wird
- Zeilenorientierte Treiber
  - Anwendung will Zeichen zeilenweise, z.B. Shell
  - Treiber blockiert Prozess bis Zeilenende erkannt
  - Treiber erlaubt das Editieren der Zeile (Backspace, etc.)
- Signale
  - bestimmte Zeichen lösen Signale an korrespondierende Prozesse aus

3.2 TTY-Treiber in UNIX

- Konfigurierbar
  - Repräsentation einer seriellen Schnittstellen als zeichenorientiertes Gerät
  - durch Aufruf von ioctl kann Treiber konfiguriert werden

```
int ioctl( int fildes, int request, /* arg */ );
```

  - Kommando zum Lesen der Konfiguration: Übergabe einer Strukturadresse

```
struct termios t;
ioctl( fd, TCGETS, &t );
```

  - Kommando zum Schreiben einer Konfiguration:

```
ioctl( fd, TCSETS, &t );
```

  - Struktur enthält Bitfelder für verschiedene Einstellungen
  - Bitmasken sind als Makros verfügbar
  - näheres: „**`man termios`**“ und „**`man ioctl`**“

3.3 Einstellung der physikalischen Parameter

- Bitrate einer seriellen Schnittstelle
  - **`B2400`** 2400 Bit/s
  - **`B4800`** 4800 Bit/s
  - **`B9600`** 9600 Bit/s
  - **`B19200`** 19200 Bit/s
  - **`B38400`** 38400 Bit/s
  - **`B57600`** 57600 Bit/s
- Zeichengröße, Parität, Stopbits
  - **`CS7`** 7 Bits
  - **`CSTOPB`** zwei Stoppbits sonst eins
  - **`PARENB`** Parität einschalten
  - **`CRTSCTS`** Hardware-basierte Flusskontrolle einschalten

3.4 Einstellung der Ein-, Ausgabeverarbeitung

- Festlegen der Zeichen mit Sonderbedeutung
  - Erase-Character: löscht letztes Zeichen (Backspace)
  - Kill-Character: löscht ganze Zeile (**`^K`**)
- Eingabeverarbeitung
  - **`ICRNL`** CR-Zeichen wird als NL-Zeichen gelesen
  - **`ICANON`** kanonische Eingabeverarbeitung (Zeileneditierung)
  - **`IXON`** erlaube Flusskontrolle mit **`^Q`** und **`^S`**
- Ausgabeverarbeitung
  - **`ECHO`** schaltet Echofunktion ein
  - **`ECHOE`** Echo von Backspace als Backspace, Leerzeichen, Backspace
  - **`ONLCR`** NL-Zeichen wird als CR, NL ausgegeben

3.5 Signalauslösung und Jobkontrolle

- Signalauslösung
  - **ISIG**: Schaltet Signale ein
  - **INTR**-Zeichen: sendet **SIGINT**-Signal (**^C**)
  - **QUIT**-Zeichen: sendet **SIGQUIT**-Signal (**^|**)
- Signal wird an ganze Prozessgruppe geschickt
  - alle Prozesse der Gruppe empfangen Signal
  - Beispiel: `cat /etc/passwd | grep Mueller | sort`
  - alle Prozesse erhalten **SIGINT** bei **^C**
- Prozessgruppe
  - Prozessgruppen-ID wird wie eine Prozess-ID (PID) bezeichnet
  - Prozess mit gleicher PID und Prozessgruppen-ID ist Gruppenführer
  - Shell sorgt dafür, dass im Beispiel **cat**, **grep** und **sort** in der gleichen Prozessgruppe sind (**sort** wird Gruppenführer)

3.5 Signalauslösung und Jobkontrolle (2)

- Vordergrund- und Hintergrundprozesse
  - Hintergrundprozesse erhalten keine Signale.
  - Bei Shells mit Jobkontrolle kann zwischen Vorder- und Hintergrundprozessen umgeschaltet werden.
- Sessions
  - Shell öffnet eine Session, die mehrere Prozessgruppen enthalten kann (spezieller sytemabhängiger Systemaufruf).
  - Shell wird Sessionführer.
  - Shell erzeugt Prozesse und Prozessgruppen.
  - Ein TTY wird Controlling-Terminal für alle Prozessgruppen der Session.
  - Unterbrechen der Terminalverbindung (**SIGHUP**) wird dem Sessionführer zugestellt.

3.5 Signalauslösung und Jobkontrolle (3)

- Vordergrundprozess
  - Eine Prozessgruppe der Session kann zur Vordergrundprozessgruppe gemacht werden.
  - **SIGINT** und **SIGQUIT** sowie die Eingabe vom Terminal werden nur der Vordergrundprozessgruppe zugestellt.
- Hintergrundprozesse
  - Alle Hintergrundprozesse bekommen keine Eingabe vom Terminal und werden gestoppt, wenn sie lesen wollen (Shell wird benachrichtigt).
- Jobkontrolle
  - Shell kann zwischen Vorder- und Hintergrundprozessgruppen umschalten
  - Benutzer kann Vordergrundprozesse stoppen und gelangt zur Shell zurück

3.5 Signalzustellung und Jobkontrolle (4)

- Beispiel: Stoppen und wiederaufnehmen eines Vordergrundprozesses

```
prompt> cc -o test.c
^Z
Suspended
prompt> jobs
[1] Suspended cc -o test.c
prompt> fg %1
```

  - Realisiert mit einem Signal namens **SIGTSTP**, das die Prozessgruppe stoppt
  - Shell bekommt dies mit über ein **waitpid()**
  - Shellkommando **fg** sendet einn Signal **SIGCONT** und die Prozesse fahren fort

3.5 Signalzustellung und Jobkontrolle (5)

- Beispiel: Stoppen eines Vordergrundprozesses, Umwandlung in einen Hintergrundprozess

```
prompt> cc -o test.c
^Z
Suspended
prompt> bg
[1] Running cc -o test.c
prompt>
```

  - Wie auf vorheriger Folie, aber:
    Shell schaltet die Prozessgruppe in den Hintergrund und wartet nicht mehr auf deren Beendigung.

3.5 Signalzustellung und Jobkontrolle (6)

- Beispiel: Starten eines Hintergrundprozesses und Umwandlung in einen Vordergrundprozess

```
prompt> cc -o test.c &
prompt> jobs
[1] Running cc -o test.c
prompt> fg %1
```

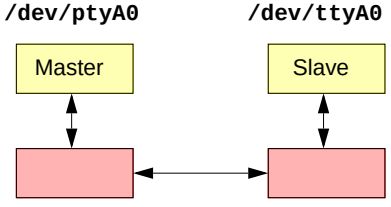
  - Shell startet eine Hintergrundprozessgruppe und nimmt Kommandos entgegen
  - **fg** Kommando schaltet die Hintergrundgruppe in eine Vordergrundprozessgruppe um und wartet auf deren Beendigung mit **waitpid()**

3.6 Pseudo-Terminals

- Pseudo-TTY-Treiber *(PTTY)*
  - keine echte serielle Schnittstelle vorhanden
  - Shell und andere Prozesse benötigen aber ein TTY für
    - Flusskontrolle,
    - Echofunktion,
    - Job-Kontrolle etc.
  - fungiert als gewohnte Schnittstelle von Anwendungsprozessen
  - Einsatz beispielsweise bei einem Fenstersystem (xterm-Programm)
    - xterm-Programm bedient die Masterseite eines PTTY
    - Shell und Anwendungsprogramme sehen xterm-Fenster wie ein TTY (Slaveseite)

3.6 Pseudo-Terminals (2)

- Master- und Slaveseite sehen wie ein normales TTY-Device aus



  - Slaveseite besitzt Modul zur Flusskontrolle, Eingabeeditierung, Signalzustellung, Flusskontrolle etc.

3.7 Warten auf mehrere Ereignisse

- Bisher: Lese- oder Schreibaufrufe blockieren
  - Was tun beim Lesen von mehreren Quellen?
- Alternative 1: nichtblockierende Ein-, Ausgabe
  - **`O_NDELAY`** beim **`open()`**
  - Pollingbetrieb: Prozess muss immer wieder **`read()`** aufrufen, bis etwas vorliegt

3.7 Warten auf mehrere Ereignisse (2)

- Alternative 2: Blockieren an mehreren Filedeskriptoren
  - Systemaufruf:

```
int select( int nfds, fd_set *readfds, fd_set *writefds,
            fd_set *errorfds, struct timeval *timeout);
```

  - **`nfds`** legt fest, bis zu welchem Filedeskriptor **`select`** wirken soll.
  - **`XXXfds`** sind Filedeskriptoren, auf die gewartet werden soll:
    - **`readfds`** — bis etwas zum Lesen vorhanden ist
    - **`writefds`** — bis man schreiben kann
    - **`errorfds`** — bis ein Fehler aufgetreten ist
  - Timeout legt fest, wann der Aufruf spätestens deblockiert.
  - Makros zum Erzeugen der Filedeskriptormengen
  - Ergebnis: in den Filedeskriptormengen sind nur noch die Filedeskriptoren vorhanden, die zur Deblockade führten

4 Bildschirmtreiber

- Bildspeicher
  - zeichenorientiert
  - pixelorientiert
- Aufgaben des Treibers
  - Bereitstellen von Graphikprimitiven (z.B. Ausgabe von Text, Zeichnen von Rechtecken, etc.)
  - Ansprechen von Graphikprozessoren (schnelle Verschiebeoperationen, komplexe Zeichenoperationen, 3D Rendering, Textures)
  - Einblenden des Bildspeichers in Anwendungsprogramme (z.B. X11-Server)
- Bildspeicher
  - spezieller Speicher, der den Bildschirminhalt repräsentiert
  - Dual ported RAM (Videochip und Prozessor können gleichzeitig zugreifen)

5 Netzwerktreiber

- Beispiel: Ethernet
  - schneller serieller Bus mit CSMA/CD
    (Carrier sense media access / Collision detect)
    zu deutsch: es wird dann gesendet, wenn nicht gerade jemand anderes sendet; Kollisionen werden erkannt und aufgelöst
  - spezieller Netzwerkchip
    - implementiert unterstes Kommunikationsprotokoll
    - erkennt eintreffende Pakete
- Netzwerktreiber
  - wird von höheren Protokollen innerhalb des Betriebssystems angesprochen, z.B. von der IP-Schicht

5 Netzwerktreiber (2)

- Senden
  - Treiber übergibt dem Netzwerkchip eine Datenstruktur mit den notwendigen Informationen: Sendeadresse, Adresse und Länge von Datenpuffern
  - Netzwerkchip löst Unterbrechung bei erfolgtem Senden aus
- Empfangen
  - Treiber übergibt dem Netzwerkchip eine Datenstruktur mit Adressen von freien Arbeitspuffern
  - erkennt der Netzwerkchip ein Paket (für die eigene Adresse), füllt er das Paket in einen freien Puffer
  - der Puffer wird in eine Liste von empfangenen Paketen eingehängt und eine Unterbrechung ausgelöst
  - Treiber kann die empfangenen Pakete aushängen

5 Netzwerktreiber (3)

- Übertragung der Daten erfolgt durch DMA
  - 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)

6 Andere Geräte

- Uhr
  - Hardwareuhren (z.B. DCF 77, GPS Empfänger)
  - Systemuhr fast immer in Software (wird mit Hardwareuhren synchronisiert)
  - UNIX: **`getitimer, setitimer`**
    - vier Intervalltimer pro Prozess: Signal **`SIGALRM`** nach Ablauf
    - Ablauf konfigurierbar:
      Realzeit, Virtuelle Zeit, Virtuelle Zeit (einschl. Systemzeit des Prozesses)
- Bandlaufwerk
  - zeichenorientiertes Gerät
  - Spuloperationen durch **`d_ioctl`** realisiert

6 Andere Geräte (2)

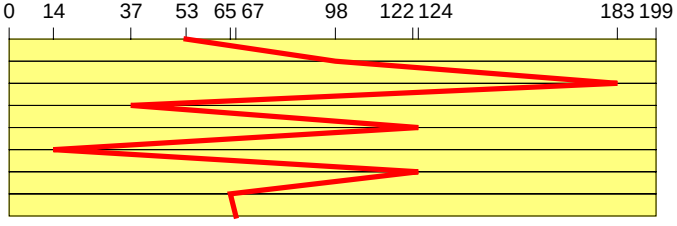
- CD-ROM
  - wird wie Platte behandelt (eigener Treiber)
  - nicht beschreibbar
  - spezielle Treiber für Audio-Tracks möglich
- Maus und Tastatur
  - meist über serielle Schnittstellen und bestimmtes Protokoll implementiert
- Floppy-Disk
  - wird im Prinzip wie Platte behandelt (eigener Treiber)
  - spezielle Dateisysteme zur Realisierung von FAT-Dateisystemen unter UNIX

7 Disk-Scheduling

- Plattentreiber hat in der Regel mehrere Aufträge in seiner 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

7.1 FCFS-Scheduling

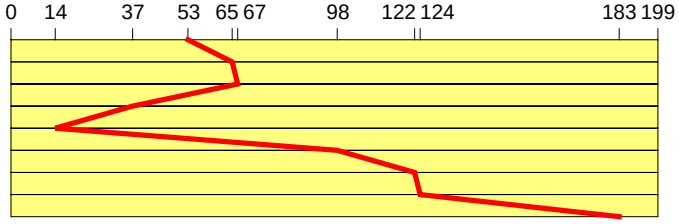
- Bearbeitung gemäß Ankunft des Auftrags
  - Referenzfolge (Folge von Zylindernummern):
    98, 183, 37, 122, 14, 124, 65, 67
  - Aktueller Zylinder: 53



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

7.2 SSTF-Scheduling

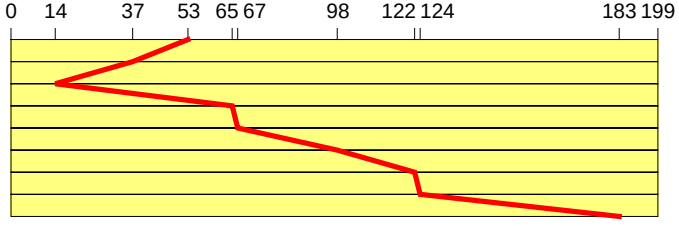
- Es wird der Auftrag mit der kürzesten Positionierzeit vorgezogen *(Shortest Seek Time First)*
  - Gleiche Referenzfolge
    (Annahme: Positionierzeit proportional zum Zylinderabstand)



  - Gesamtzahl von Spurwechseln: 236
  - ähnlich wie SJF kann auch SSTF zur Aushungerung führen
  - noch nicht optimal

7.3 SCAN-Scheduling

- Bewegung des Plattenarm in eine Richtung bis keine Aufträge mehr vorhanden sind (Fahrstuhlstrategie)
  - Gleiche Referenzfolge (Annahme: bisherige Kopfbewegung Richtung 0)



  - Gesamtzahl der Spurwechsel: 208
  - Neue Aufträge werden miterledigt ohne zusätzliche Positionierzeit und ohne mögliche Aushungerung
  - Variante C-SCAN *(Circular SCAN)*: Bewegung nur in eine Richtung