Treiberkonzepte unter UNIX

Anke Steuernagel
FH Worms
steuernagel@fh-worms.de

Mai 1997

Vortrag: Treiberkonzepte unter UNIX

Einleitung

Gerätetreiber: Software um die vielfältige Hardware in UNIX-Systemen (Ein- und Ausgabegeräte und Dateisysteme) anzusprechen.

Sie sind also Bindeglieder zwischen Hardware-Komponenten und Software im Laufzeitsystem, also den Anwenderprogrammen.

Alle Zugriffe eines Benutzerprozesses auf ein angeschlossenes Gerät erfolgen über eine solche Gerätedatei.

Ein Gerätetreiber steuert dabei *viele* physikalische Geräte eines gegebenen Typs (Bsp.: Terminal).

Außerdem werden sog. *,,Softwaregeräte"* unterstützt, die mit keinem physikalischen Gerät verbunden sind (Bsp.: physikalischer Speicher, Befehl *ps*).

(+) Schutzfunktion!

Treiberschnittstellen

Ein UNIX-System verfügt über 2 Gerätetypen:

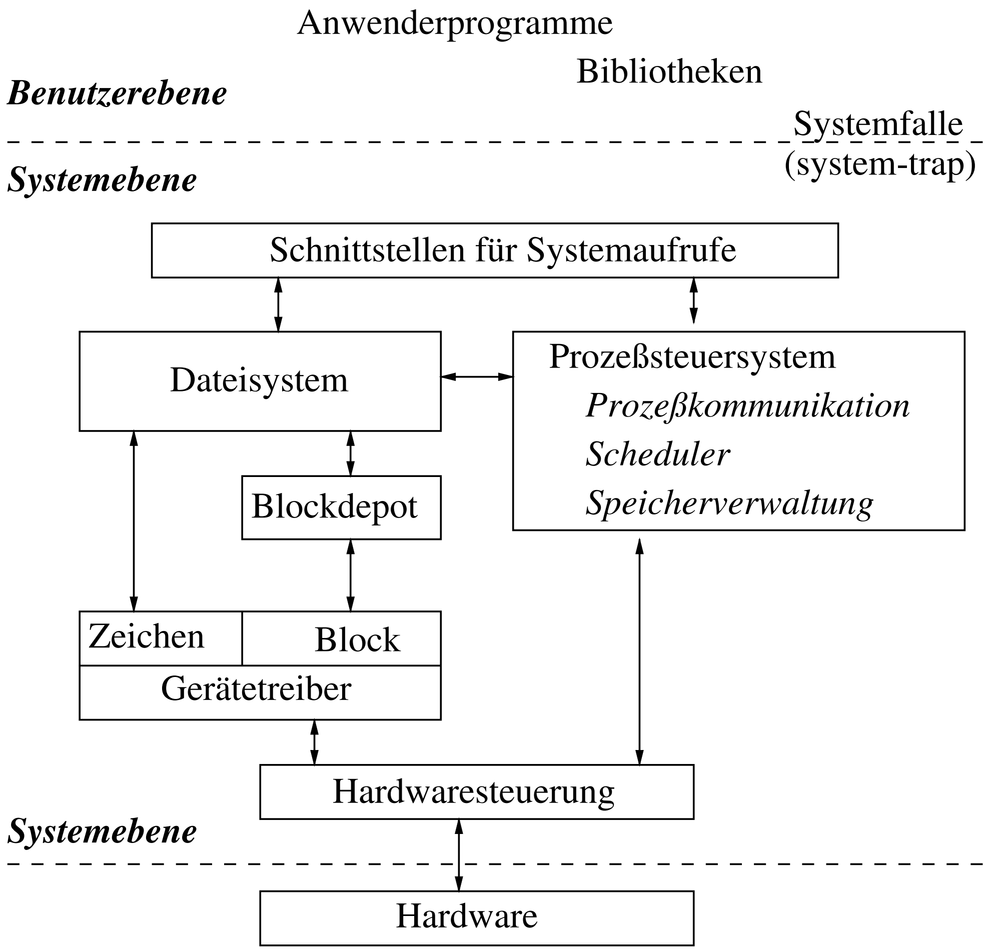
- *Blockgeräte* (Direktzugriffsgeräte, wahlfreier Zugriff, beliebige Blöcke werden gelesen und geschrieben), wie Platten, Disketten oder Magnetbänder, und
- *Zeichengeräte* (Rohdatengeräte, meist nur sequentiell, ungepufferter Zugriff), wie etwa Soundkarten, Scanner, Drucker, Mäuse, Terminals und auch Netzwerke

Blockgeräte können zusätzlich noch eine Schnittstelle für Zeichengeräte haben, die hauptsächlich zur Steuerung des eigentlichen Gerätes benutzt werden (Bsp.: `mkfs` oder `fsck`).

Die Benutzerschnittstelle für Geräte läuft über das Dateisystem:

- Name wie Dateiname
- Zugriff wie auf Datei (Inode und Katalogeintrag)
- Unterscheidung durch in Inode gespeicherten Dateityp: Block (b) oder Zeichen (c)
- Systemaufrufe wie für gewöhnliche Dateien: *open*, *close*, *read* oder *write*
- Steuerschnittstelle für Zeichengeräte: *ioctl*

Erinnerung: Prinzipdarstellung des Systemkerns



Systemkonfiguration

Die Gerätekonfiguration kann auf 3 Ebenen bestimmt werden:

1. Konfigurationsdaten werden fest in Dateien kodiert (dann beim Erstellen des Systemcodes kompiliert und mit eingelinkt).
2. Konfigurationsdaten werden später bei bereits laufendem System angegeben. Der Kern berichtigt dann dynamisch die internen Konfigurationstabellen.
3. An Geräten, die sich selbst zu erkennen geben, kann der Kern feststellen, welche Geräte im System installiert sind. Er liest dabei, um sich selbst zu konfigurieren, die Stellung von Schaltern in der Hardware.

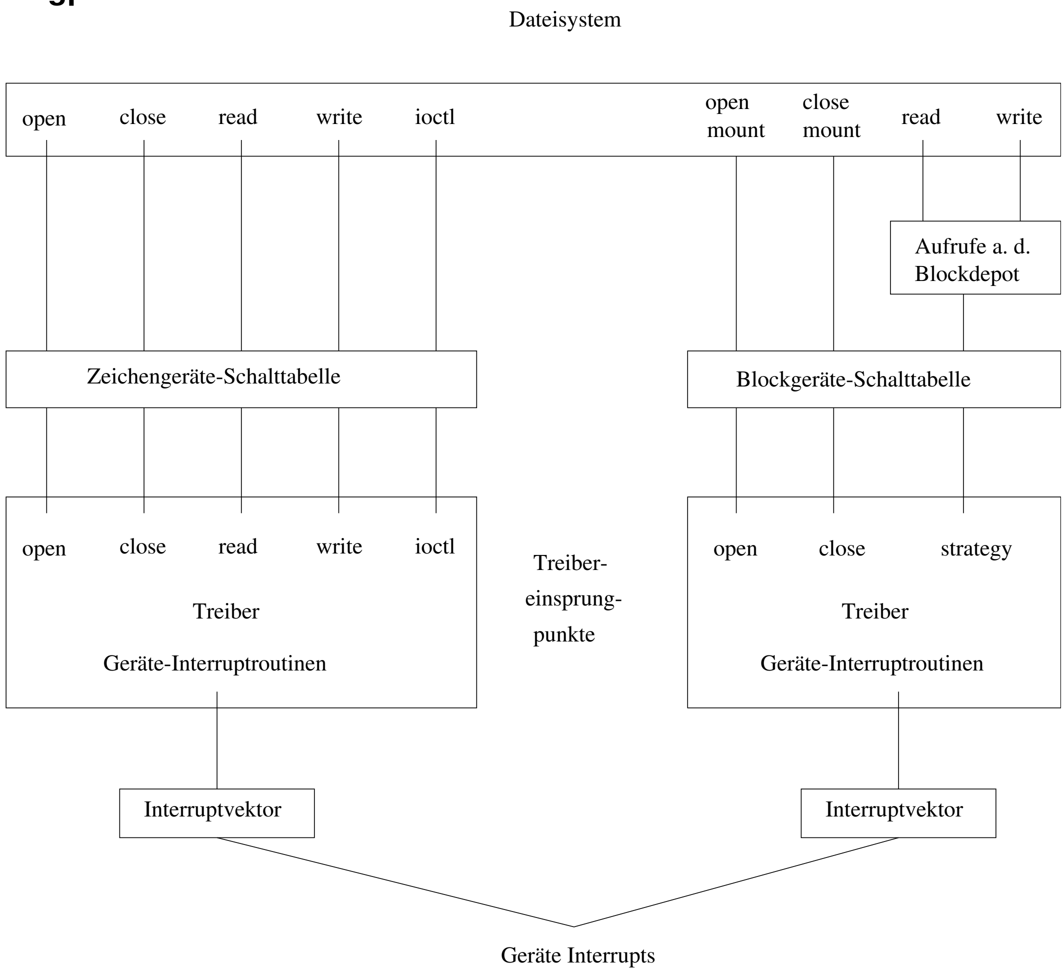
In allen Fällen erzeugt oder vervollständigt die Konfigurationsprozedur *Tabellen*, die Bestandteil des Kerns sind.

Schalttabellen

Die Schnittstelle des Kerns zu den Gerätetreibern wird durch die *Blockgeräte-* und die *Zeichengeräte-Schalttabelle* beschrieben.

Für jeden Gerätetyp bestehen Einträge in der Tabelle, die den Kern bei Systemaufrufen zu den passenden Treibern führen.

Einsprungpunkte in Gerätetreiber:



Erzeugen der Gerätedatei

Der Systemverwalter richtet Gerätespezialdateien mit dem Befehl *mknod* ein und gibt dazu den Dateityp (Block oder Zeichen) und Major- sowie Minor-Gerätenummer an. Der Befehl *mknod* gibt dann zum Erzeugen der Gerätedatei den Systemaufruf *mknod* aus.

Beispiel: `mknod /dev/tty13 c 2 13`

Major-Gerätenummer: Gerätetyp mit entspr. Eintrag in der Block- oder Zeichengeräte-Schalttabelle
Minor-Gerätenummer: Nummer des Gerätes

Beispiel: Geräte unter Linux

```
crw-rw--w-   1 root     sys        14,   4 Apr 25  1995 /dev/audio
crw--w--w-   2 root     tty         4,   0 Apr  4 12:43 /dev/console
brw-rw-r--   1 root     disk        2,   0 Jan  1  1980 /dev/fd0
brw-r--r--   1 root     disk        2,   1 Jan  1  1980 /dev/fd1
brw-rw----   1 root     disk        3,   0 Sep  7  1994 /dev/hda
brw-rw----   1 root     disk        3,  64 Sep  7  1994 /dev/hdb
crw-rw----   1 root     daemon      6,   0 Jan  1  1980 /dev/lp0
crw-r-----   1 root     kmem        1,   1 Jan  1  1980 /dev/mem
crw-rw-rw-   1 root     mem         1,   3 Jan  1  1980 /dev/null
brw-r-----   1 root     disk        1,   1 Jan  1  1980 /dev/ram
crw-------   1 root     tty         4,   0 Feb  8  1996 /dev/tty0
crw-r--r--   1 root     root        4,  64 Apr  5 17:15 /dev/ttyS0
crw-rw-rw-   1 root     mem         1,   5 Nov 19  1995 /dev/zero
```

Dateien:
`/usr/src/linux-2.0.18/include/linux/major.h`
`/usr/src/linux-2.0.18/Documentation/devices.txt`

Beispiel

Eröffnet der Prozeß die Block-Spezialdatei „/dev/dsk1” mit einer Major-Gerätenummer 0, so ruft der Kern die Routine *gdopen* im Eintrag 0 der Blockgeräte-Schalttabelle auf.

Liest ein Prozeß die Zeichen-Spezialdatei „/dev/mem” mit einer Major-Gerätenummer 3, so ruft der Kern die Routine *mmread* im Eintrag 3 der Zeichengeräte-Schalttabelle auf.

Blockgeräte-Schalttabelle			
Eintrag	open	close	strategy
0	gdopen	gdclose	gdstrategy
1	gtopen	gtclose	gtstrategy

Zeichengeräte-Schalttabelle					
Eintrag	open	close	read	write	ioctl
0	conopen	conclose	conread	conwrite	conioctl
1	dzbopen	dzbclose	dzbread	dzbwrite	dzbioctl
2	syopen	nulldev	syread	sywrite	syioctl
3	nulldev	nulldev	mmread	mmwrite	nodev
4	gdopen	gdclose	gdread	gdwrite	nodev
5	gtopen	gtclose	gtread	gtwrite	nodev

Beispiel für Block- und Zeichengeräte-Schalttabellen

Systemaufrufe und Treiberschnittstellen

Wichtiger Unterschied zwischen Systemaufrufen auf Geräte und solchen auf gewöhnliche Dateien: Die Inode einer Spezialdatei ist während der Ausführung des Treibers durch den Kern *nicht gesperrt*.

Der Gerätetreiber interpretiert die Parameter des Systemaufrufs entsprechend dem jeweiligen Gerät.

Der Treiber unterhält Datenstrukturen, die den Zustand jedes Gerätes, das er bedient, beschreiben. Treiberfunktionen und Interruptroutinen laufen entsprechend dem Status des Treibers und der gewünschten Aktion ab.

Open

Zum Eröffnen eines Gerätes folgt der Kern der gleichen Prozedur wie zum Eröffnen gewöhnlicher Dateien, ordnet also eine Speicherinode zu, erhöht ihren Referenzzähler und weist einen Dateitabelleneintrag sowie einen Benutzer-Filedeskriptor zu.

Der Kern gibt den Benutzer-Filedeskriptor an den aufrufenden Prozeß zurück, so daß das Eröffnen eines Gerätes wirklich wie das Öffnen einer gewöhnlichen Datei aussieht.

Er ruft jedoch vor Rückkehr in den Benutzermodus die gerätespezifische *open*-Prozedur auf.

Diese Prozedur stellt eine Verbindung zwischen aufrufendem Prozeß und eröffnetem Gerät her und initialisiert private Datenstrukturen des Treibers (z. B. Baudrate eines Terminals).

Algorithmus zum Eröffnen eines Gerätes

```
algorithm open     /* for device drivers */
input:  pathname, openmode
output: file descriptor
{
   convert pathname to inode, increment inode reference count,
   allocate entry in the file table, user file descriptor, as
      in open regular file;

   get major, minor number from inode;

   save context (algorithm setjmp) in case of long jump from driver;

   if (block device)
   {
      use major number as index to block device switch table;
      call driver open procedure for index:
         pass minor number, open modes;
   }
   else
   {
      use major number as index to character device switch table;
      call driver open procedure for index:
         pass minor number, open modes;
   }

   if (open fails in driver)
      decrement file table, inode counts;
}
```

Close

Ein Prozeß trennt mit *close* seine Hardwareverbindung zu einem eröffneten Gerät. Allerdings ruft der Kern die *close*-Prozedur nur für den letzten *close*-Aufruf auf ein Gerät auf, also nur dann, wenn keine anderen Prozesse mehr das Gerät geöffnet halten.

Der Gerätetreiber weiß nie, wieviel Prozesse das Gerät noch verwenden. Treiber können sich, wenn sie nicht sorgfältig kodiert sind, leicht selbst matt setzen (z. B. undefinierte Kombination von open und close).

Desweiteren werden durch die *close*-Prozedur Datenstrukturen des Treibers sowie Hardware neu initialisiert, damit der Kern das Gerät später wieder eröffnen kann.

Algorithmus zum Schließen eines Gerätes

```
algorithm close    /* for devices */
input:  file descriptor
output: none
{
   do regular close algorithm;
   if (file table reference count not 0)
      goto finish;
   if (there is another open file and its major, minor numbers are
       same as device being closed)
      goto finish;    /* not last close after all */
   if (character device)
   {
      use major number to index into character device switch table;
      call driver close routine: parameter minor number;
   }
   if (block device)
   {
      if (device mounted)
         goto finish;
      write device blocks in buffer cache to device;
      use major number to index into block device switch table;
      call driver close routine: parameter minor number;
      invalidate device blocks still in buffer cache;
   }
   finish: release inode;
}
```

Read und Write

Die *read*- und *write*-Aufrufe des Kerns auf ein Gerät ähneln denen für eine gewöhnliche Datei. Greift ein Prozeß mit *read* oder *write* auf ein Zeichengerät zu, so ruft des Kern die *read*- bzw. *write*-Prozedur des Gerätetreibers aus.

Obwohl es wichtige Fälle gibt, in denen der Kern Daten zwischen Adreßraum des Benutzers und Gerät transportiert, puffern Gerätetreiber die Daten intern. Beispielsweise verwenden Terminaltreiber *C-Listen* zum Puffern von Daten.

Strategie-Schnittstelle und Ioctl

Der Kern verwendet die *Strategieschnittstelle*, um Daten zwischen Blockdepot und einem Gerät zu transportieren.
Greifen die Algorithmen des Blockdepots (z. B. *bread* oder *bwrite*) auf die Platte zu, rufen sie die durch die Major-Gerätenummer bestimmte Strategieprozedur auf.
Die Strategieprozedur verwendet Minor-Gerätenummer und Blocknummer aus dem Pufferheader, um herauszufinden, wo die Daten sich auf dem Gerät befinden, sowie die Pufferadresse, um festzulegen, wohin die Daten übertragen werden sollen.

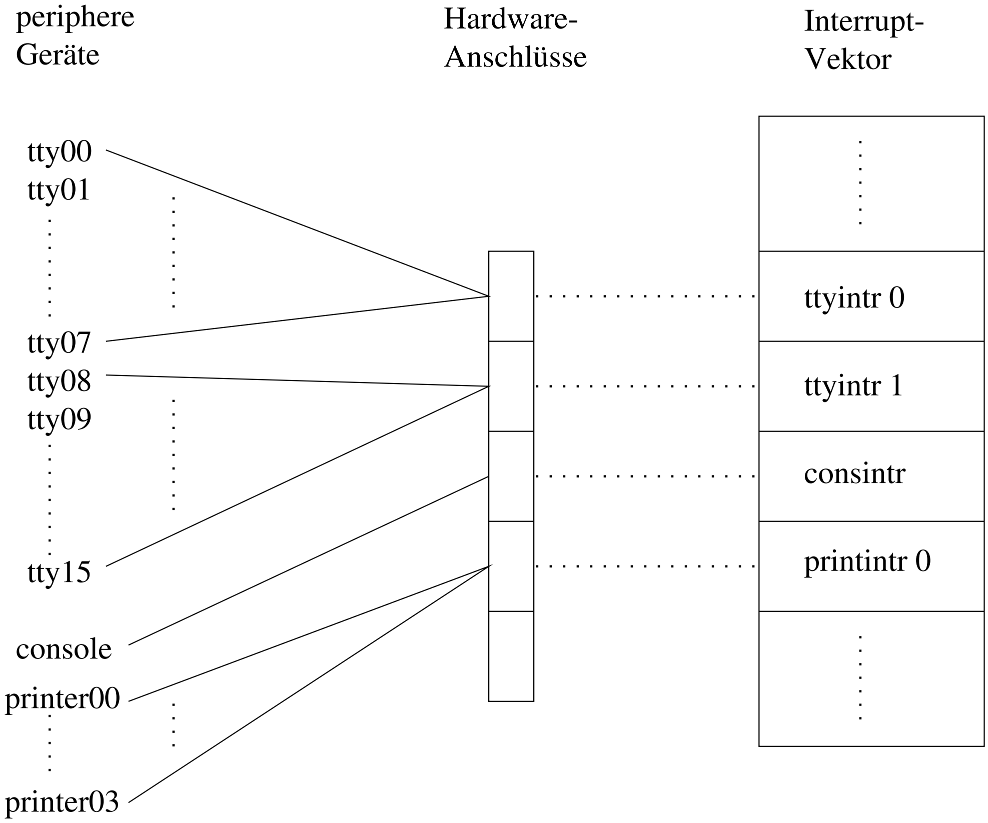
Der Systemaufruf *ioctl* bietet einen allgemeinen und allumfassenden Einsprungpunkt für *gerätespezifische* Befehle und gestattet einem Prozeß das Setzen von Hardwareoptionen für das Gerät sowie von Softwareoptionen für den Treiber.

Syntax: `ioctl(fd, command, arg);`

`fd:` Filedeskriptor; `command:` Aktionsanforderung;
`arg:` Parameter für command (z. B. Geräteeinstellungen wie die Baudrate eines Terminals)

Interruptroutinen

Bsp.: Geräteinterrupts



Die von der Interruptroutine verwendete Gerätenummer identifiziert die Hardware-Einheit. Die Minor-Gerätenummer in der Gerätedatei dagegen identifiziert ein Gerät für den Kern.

Der Treiber setzt dann die Minor-Gerätenummer in Bezug zur Gerätenummer der Hardware.

Plattentreiber

Der Plattentreiber setzt die Adresse des Dateisystems, bestehend aus logischer Gerätenummer und Blocknummer, in einen bestimmten Sektor auf der Platte um.
Der Treiber kann diese Adresse auf 2 Arten erhalten: Entweder verwendet die Strategieschnittstelle einen Puffer aus dem Blockdepot, wobei der Pufferheader Geräte- und Blocknummer enthält, oder die *read*- und *write*-Prozeduren haben die logische Gerätenummer als Parameter übergeben erhalten.

Der Plattentreiber muß den Datendurchsatz maximieren, um bestmöglichen Systemdurchsatz zu erzielen.
Dienstprogramme können zum direkten Zugriff auf Plattendaten entweder die Zeichen- oder die Blockschnittstelle benutzen (Bsp.: `mkfs` und `fsck`).
Vorteil der Zeichenschnittstelle: Geschwindigkeit und kein zusätzliches Kopieren der Daten zwischen Adreßraum und Blockdepot.

Beispiel:

```
ls -l /dev/dsk15 /dev/rdsk15
br-------- 2 root root 0, 21 Feb 12 15:40 /dev/dsk15
crw-rw---- 2 root root 7, 21 Mar  7 09:29 /dev/rdsk15
```

Terminaltreiber und Zeilendisziplin

Terminaltreiber:
Steuerung der Datenübertragung von und zu Terminals
(interne Schnittstelle zu Zeilendisziplin-Modulen)

Kanonischer Modus: Zeilendisziplin konvertiert die eingegebenen und auszugebenden Folgen von Rohdaten in ein bestimmtes Format.

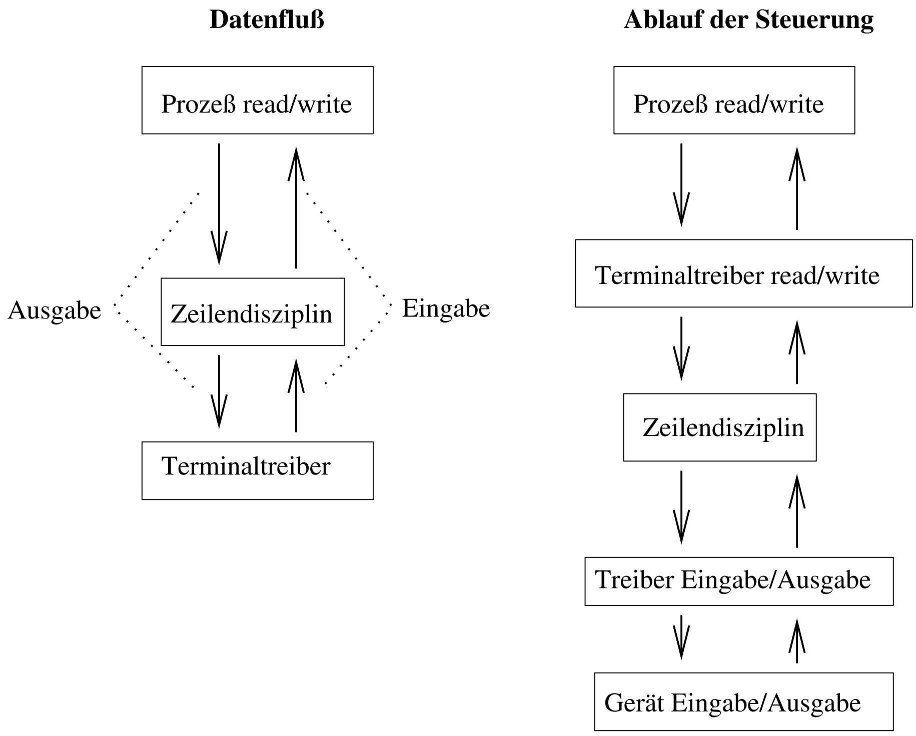
Rohdatenmodus: Zeilendisziplin reicht die Daten zwischen Prozeß und Terminal ohne Veränderung weiter.

Funktionen der Zeilendisziplin:

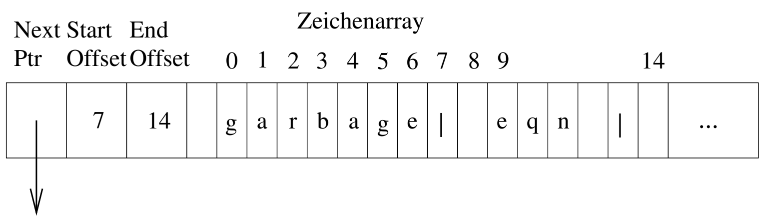
- Zerlegen von Eingabestrings in Zeilen;
- Verarbeiten von Löschzeichen;
- Verarbeiten von ,,Kill”-Zeichen (Zeile löschen);
- Echo der empfangenen Zeichen auf das Terminal;
- Expandieren der Ausgabe (z. B. Tabulator in Leerzeichen);
- Erzeugen von Signalen an Prozesse (Terminalblockade, Leitungszusammenbrüche)
- Zulassen eines Rohdatenmodus (Nichtauswertung von Lösch, Kill oder Wagenrücklauf, auch: Verwendung von asynchronen Terminals);

Zeilendisziplin und C-Listen

Aufruffolge und Datenfluß durch die Zeilendisziplin



Die Zeilendisziplin bearbeitet Daten in *C-Listen*. Eine C-Liste (character list) ist eine verkettete Liste variabler Länge aus *C-Blöcken*:



Der Terminaltreiber im kanonischen Modus

Die Datenstrukturen für Terminaltreiber sind mit 3 C-Listen verbunden:

- eine zum Speichern der „Roh"-Daten,
- eine zum Speichern der aufbereiteten Eingabedaten und
- eine zum Speichern der auszugebenden Daten.

Algorithmus zum Schreiben von Daten auf ein Terminal

```
algorithm terminal_write
{
   while (more data to be copied from user space)
   {
      if (tty flooded with output data)
      {
         start write operation to hardware with data
                       on output clist;
         sleep (event: tty can accept more data);
         continue;     /* back to while loop */
      }
      copy cblock size of data from user space to output clist:
                       line diszipline converts tab characters, etc;
   }

   start write operation to hardware with data on output clist;
}
```

Algorithmus zum Lesen von einem Terminal

```
algorithm terminal_read
{
   if (no data on canonical clist)
   {
      while (no data on raw clist)
      {
         if (tty opened with no delay option)
            return;
         if (tty in raw mode based on timer and timer not active)
            arrange for timer wakeup (callout table);
         sleep (event: data arrives from terminal);
      }

      /* there is data on raw clist */
      if (tty in raw mode)
         copy all data from raw clist to canonical clist;
      else    /* tty is in canonical mode */
      {
         while (characters on raw clist)
         {
            copy one character at a time from raw clist to
                  canonical clist: do erase, kill processing;
            if (char is carriage return or end-of-file)
               break;    /* out of while loop */
         }
      }
   }

   while (characters on canonical clist and read count not satisfied)
      copy from cblocks on canonical clist to user address space;
}
```

Der Terminaltreiber im Rohdatenmodus

Prozesse können das Terminal auch in den *Rohdatenmodus* versetzen.

- ⇒ direkte Weiterleitung der Zeichen durch die Zeilendisziplin
- ⇒ keine Bearbeitung der Eingabe

Der Kern muß hier jedoch wissen, wann er einen *read*-Aufruf des Benutzers zu beenden hat, denn auch das Wagenrücklaufzeichen wird als normale Eingabe behandelt.

Kriterien:

- Mindestzahl von Zeichen oder
- festgelegte Zeitspanne

Beide Verfahren werden über *ioctl* gesetzt.

Wichtig bei bildschirmorientierten Anwendungen wie etwa dem Editor *vi* (Kommandos enden nicht mit Wagenrücklauf).

Terminaldetails

Polling auf ein Terminal: read-Aufruf an Gerät, um festzustellen, ob dort Daten anliegen. Nachteil: Verschwendung von Rechnerleistung.

Steuerterminal: steuert Prozesse, die der Benutzer vom Terminal initialisiert. Eröffnender Prozeß ist der Steuerprozeß, typischerweise die Login-Shell. ⇒ wichtige Rolle bei der Behandlung von Signalen (z. B. Ausschalten des Terminals).

Indirekte Terminaltreiber: ,,dringende" Nachrichten der Shell können ans Terminal gesendet werden, selbst wenn Standard- und Fehler-Ausgabe auf andere Dateien umgelenkt sind. Ermittlung des Steuerterminals aus dem Dateinamen ,,/dev/tty" (Kernelimplementierung).

Einloggen

Der *init*-Prozeß läuft in einer Endlosschleife, erzeugt *getty*-Prozesse („get terminal" oder „get tty") und führt Buch darüber, welcher *getty*-Prozeß welches Terminal eröffnet.

Algorithmus zum Einloggen

```
algorithm login   /* procedure for logging in */
{
   getty process executes;
   set process group (setpgrp system call);
   open tty line;  /* sleeps until opened */
   if (open successful)
   {
      exec login program;
      prompt for user name;
      turn off echo, prompt for password;
      if (successful)   /* matches password in /etc/passwd */
      {
         put tty in canonical mode (ioctl);
         exec shell;
      }
      else
         count login attempts, try again up to a point;
   }
}
```

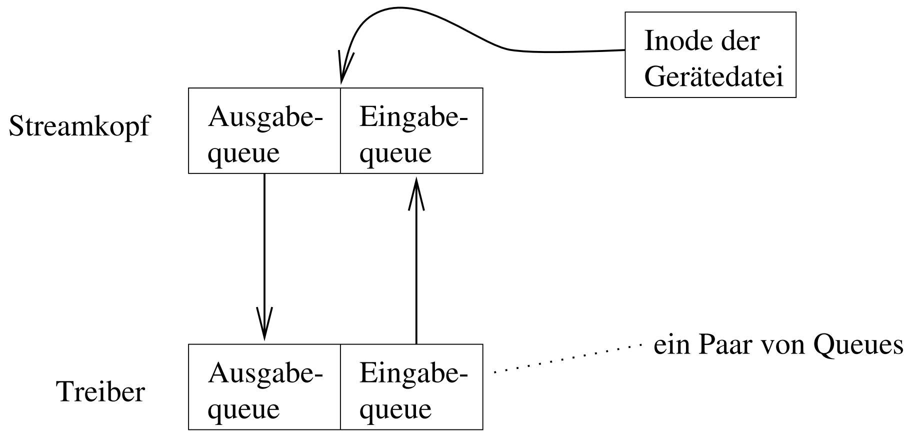
Streams

Stream: Vollduplexverbindung zwischen einem Prozeß und einem Gerätetreiber

Ein Stream besteht aus einem Satz linear verketteter Paare von Queues, wobei von jedem Paar ein Element für die Eingabe und ein Element für die Ausgabe bestimmt ist.

Über eine wohldefinierte Schnittstelle tauschen benachbarte Queues untereinander Nachrichten aus.

Ein Stream nach Open:



Ein Stream kann Module für Zeilendisziplin und Protokolle zur Bearbeitung der auf dem Weg befindlichen Daten enthalten.

Zusammenfassung I

- Geräte sind entweder *Block*- oder *Zeichengeräte.*
- Schnittstelle zwischen Gerät und System ist die jeweilige Schalttabelle.
- Die *Blockgeräte-Schalttabelle* besteht aus Einsprungpunkten für *open*, *close* und Strategieprozeduren.
- Die *Strategieprozedur* steuert den Datentransport vom und zum Blockgerät.
- Die *Zeichengeräte-Schalttabelle* besteht aus Einsprungpunkten für *open-*, *close-*, *read-*, *write-* und *ioctl*-Prozeduren.
- Der Systemaufruf *ioctl* benutzt die *ioctl*-Schnittstelle der Zeichengeräte, die den Austausch von Steuerinformationen zwischen Prozessen und Geräten erlaubt.
- Bei Empfang eines Geräte*interrupts* ruft der Kern in Abhängigkeit von Informationen in der Interruptvektortabelle und von durch die unterbrechende Hardware gelieferten Parametern die Routine für Geräteinterrupts auf.

Zusammenfassung II

- *Plattentreiber* wandeln die vom Dateisystem verwendeten logischen Blocknummern in Positionen auf der physikalischen Platte um.
- Die *Blockschnittstelle* gestattet dem Kern das Puffern von Daten.
- Die *Rohdatenschnittstelle* dagegen gestattet schnellere I/O von und auf die Platte, umgeht aber das Blockdepot und erhöht die Wahrscheinlichkeit für Fehler im Dateisystem.
- *Terminaltreiber* bilden die Hauptschnittstelle zu den Benutzern.
- Der Kern verbindet *drei C-Listen* mit jedem Terminal:
  - für die Rohdateneingabe von der Tastatur
  - für die Eingabe unter Berücksichtigung von Lösch-, Kill- und Wagenrücklaufzeichen
  - für die Ausgabe
- Der Aufruf von *ioctl* gestattet Prozessen
  - die Kontrolle darüber, wie der Kern Eingabedaten bearbeitet,
  - das Versetzen des Terminals in den kanonischen Modus oder
  - das Setzen verschiedener Parameter für den Rohdatenmodus.

Zusammenfassung III

- Der Prozeß *getty* eröffnet Terminalleitungen und wartet auf eine Verbindung: Er setzt seine Prozeßgruppe so, daß die Login-Shell zum Prozeßgruppenführer wird, initialisiert mit *ioctl* Terminalparameter und führt den Benutzer durch die Login-Sequenz.
- *Streams* sind ein Schema zur Verbesserung der Modularität von Gerätetreibern und Protokollen. Ein Stream ist eine Vollduplexverbindung zwischen Prozessen und Gerätetreibern, die Module für Zeilendisziplin und Protokolle zur Bearbeitung der auf dem Weg befindlichen Daten enthalten kann.

Literatur

[1] Bach, Maurice J.: The Design of the UNIX Operating System, 1986, Prentice Hall

[2] Beck, Böhme, u. a.: Linux-Kernel-Programmierung: Algorithmen und Strukturen der Version 1.2, 3. Auflage, 1995, Addison-Wesley

[3] Sektion 4 (Devices) der Manual Pages unter Linux