HANSER

C und Linux

Martin Gräfe

Die Möglichkeiten des Betriebssystems mit eigenen Programmen nutzen

ISBN 3-446-22973-6

Leseprobe

6 Devices – das Tor zur Hardware

Eine der größten Stärken von Linux (und anderen Unix-Varianten) liegt in den *Devices*. Bei welchem anderen Betriebssystem können Sie mit einer Kommandozeile eine MIDI[1]-Sequenz aufzeichnen[2] oder etwa den Master-Boot-Record (MBR) einer Festplatte auslesen?[3] Devices bilden die Schnittstelle zwischen der Hardware (z.B. dem CD-ROM-Laufwerk) und dem Applikationsprogramm (z.B. einem CD-Player wie „`cda`“ oder „`workman`“).
Dieses Kapitel soll einen Einblick in den Umgang mit Devices aus Sicht des Programmierers vermitteln. Es werden exemplarisch das CD-ROM-Laufwerk, Sound-Karte, Video-Device (z.B. eine WebCam) und die serielle Schnittstelle RS 232 angesprochen.

6.1 Das Device-Konzept unter Linux

Ein Großteil der Hardware-Komponenten wie Festplatten, CD-ROM-Laufwerk, Sound-Karte, serielle und parallele Schnittstellen sind unter Linux als *Devices* – im Deutschen häufig als „Geräte“ bezeichnet – verfügbar. Diese besitzen einen Eintrag im Dateisystem, für gewöhnlich in dem Verzeichnis „`/dev`“. Die erste IDE-Festplatte ist beispielsweise mit „`/dev/hda1`“ verknüpft. Die Einträge im Dateisystem – auch als *Inodes* bezeichnet –, die mit Devices verknüpft sind, sind vom Typ „*character special file*“ oder „*block special file*“ (siehe auch Abschnitt 4.2). Bei „`ls -l`“ wird dieser Typ entsprechend durch ein „c“ oder „b“ am Zeilenanfang gekennzeichnet.

[1] Music Instrument Digital Interface – Schnittstelle für Keyboards und andere elektronische Musikinstrumente
[2] „`cat /dev/sequencer > midi-file`“ (Abbruch der Aufzeichnung mit Ctrl-C)
[3] Für (E)IDE-Festplatten: „`dd if=/dev/hda of=mbr bs=512 count=1`“

Über zwei Gerätenummern, der „*major*“ und der „*minor device number*“, sind diese Inodes mit dem entsprechenden Treiber verknüpft; diese Gerätenummern können ebenfalls mit „`ls -l`“ angezeigt werden:

```
brw-rw----   1 root   disk    3,   1 Nov  8  1999 /dev/hda1
└ block special file            │     └ minor device number
                                └ major device number
```

Für Gerätetreiber, die als Kernel-Modul ausgeführt sind, ist die Verknüpfung zwischen der *major device number* und dem Modul in der Datei „`/etc/modules.conf`“ registriert. Beispiel:

```
alias block-major-2  floppy
```

Hier werden alle *block special files*, die eine *major device number* von 2 haben, mit dem Kernel-Modul „`floppy.o`“ verknüpft.

6.1.1 Devices öffnen und schließen

Der Vorteil, der sich durch die Integration der Devices in das Dateisystem ergibt, liegt auf der Hand: dadurch wird es möglich, mit gängigen Dateioperationen auf Hardware-Komponenten zuzugreifen. So kann man z.B. in einer Shell mit dem Kommando „`cat /dev/audio > test.au`“ ohne ein spezielles Programm eine Audio-Datei aufzeichnen – vorausgesetzt, eine Sound-Karte ist installiert und unter Linux korrekt eingerichtet. In gleicher Weise kann diese Datei wieder abgespielt werden: „`cat test.au > /dev/audio`“. Manche Gerätetreiber stellen auch ein zusätzliches Device zur Verfügung, das Informationen über die Hardware und den Treiber liefert, Beispiel: „`cat /dev/sndstat`“.
Für den Programmierer bedeutet dies, dass er mit Hilfe der üblichen Funktionen für den Dateizugriff auch die Devices ansprechen kann. Dies gilt natürlich auch für das Öffnen und Schließen eines Devices, das mit den Befehlen `fopen()` bzw. `fclose()` für gepufferte Ein-/Ausgabe erfolgen kann (vgl. Kapitel 4). Häufiger verwendet man für den Zugriff auf Devices jedoch die entsprechenden Befehle für *un*gepufferte Ein-/Ausgabe:

```
int open(char *pathname, int flags);
int close(int fd);
```

Die Funktion `open()` benötigt als ersten Parameter den zu öffnenden Datei- bzw. Device-Namen. Als zweiter Parameter kann eine der Konstanten `O_RDONLY` (nur lesen), `O_WRONLY` (nur schreiben), `O_RDWR` (lesen

und schreiben) angegeben werden. Als Rückgabewert liefert `open()` entweder den neuen Dateideskriptor, also eine positive, ganze Zahl, oder -1, falls die angegebene Datei nicht geöffnet werden konnte. Die Funktion `close()` schließt die Datei, deren Deskriptor als Parameter angegeben ist.
Während sich Dateizugriffe durch die Verwendung von Schreib- und Lesepuffern deutlich beschleunigen lassen, bringt diese Pufferung bei Devices im Allgemeinen keine Vorteile. Ein Lesepuffer kann bei Devices sogar eher hinderlich sein – insbesondere bei Geräten wie Sound-Karte oder WebCam, bei denen die Daten nur mit einer bestimmten Datenrate „eintreffen“, würde das Füllen des Puffers mit einer Wartezeit verbunden sein. Solche Devices werden in der Regel durch *character special files* repräsentiert – im Gegensatz zu den *block special files* wie etwa für das Device einer Festplatte.

6.1.2 Ungepuffertes Lesen und Schreiben

Im Gegensatz zur gepufferten Ein-/Ausgabe, für die es zahlreiche Schreib-/Lesefunktionen unterschiedlicher Komplexität gibt, stehen für die *un*gepufferte Ein- und Ausgabe nur je zwei Funktionen zur Verfügung:

```
ssize_t read(int fd, void *buffer, size_t count);
ssize_t pread(int fd, void *buffer, size_t count,
              off_t offset);
ssize_t write(int fd, void *buffer, size_t count);
ssize_t pwrite(int fd, void *buffer, size_t count,
               off_t offset);
```

Die Funktionen `read()` und `pread()` lesen aus der Datei mit dem Deskriptor `fd` die Anzahl `count` Bytes und schreiben sie in den Speicher an die mit `buffer` angegebene Stelle. Dabei liest `pread()` nicht ab der aktuellen Position in der Datei, sondern ab der durch `offset` angegebenen Position. Als Rückgabewert liefern beide Funktionen die Anzahl der tatsächlich gelesenen Bytes bzw. -1 im Fehlerfall.
Analog werden die Funktionen `write()` und `pwrite()` verwendet, um `count` Bytes aus dem Speicher ab Adresse `buffer` in die Datei mit dem Deskriptor `fd` zu schreiben. Zuvor setzt `pwrite()` die aktuelle Position in der Datei auf `offset`. Beide Funktionen geben die Anzahl der tatsächlich geschriebenen Bytes zurück oder -1, falls ein Fehler aufgetreten ist.

Insbesondere beim Lesen aus einem *character special file* mit `read()` oder `pread()` ist die Anzahl der tatsächlich gelesenen Zeichen häufig geringer als die mit `count` angegebene Zahl. Dies deutet nicht auf einen Fehler hin, sondern liegt lediglich daran, dass im Moment noch keine weiteren Zeichen verfügbar sind.

Im Zusammenhang mit dem Lesen aus einem Device ist auch die Funktion `select()` interessant:

```
int select(int n, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

Mit dieser Funktion kann bis zu einer vorgegebenen, maximalen Zeit gewartet werden, bis mindestens ein Dateideskriptor bereit zum Lesen oder zum Schreiben ist. Ein Beispiel für die Verwendung dieser Funktion findet sich in Abschnitt 6.5.

6.1.3 Devices steuern mit `ioctl()`

Wenn auch die Einbindung der Devices in das Dateisystem den Lese- und Schreibzugriff auf die verschiedenen Hardware-Komponenten deutlich vereinfacht, kann über die Schreib-/Lesefunktionen dennoch nur ein kleiner Teil der Möglichkeiten der Devices genutzt werden. So kann man z.B. zwar eine Audio-Datei von der Sound-Karte aufzeichnen, aber zum Einstellen der Sampling-Rate ist ein erweiterter Zugriff auf das Device erforderlich. Dies ist Aufgabe der Funktion `ioctl()` (die Abkürzung steht für *Input-Output-Control*):

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

Auch diese Funktion benötigt als ersten Parameter den Dateideskriptor `fd` des Devices. Der Parameter `request` gibt das auf dieses Device anzuwendende Kommando an, optional gefolgt von weiteren Parametern, die von dem Kommando benötigt werden. Die möglichen Kommandos hängen von dem jeweiligen Device ab, eine (unvollständige) Liste der Kommandos erhält man mit

```
man ioctl_list
```

Im Fehlerfall liefert die Funktion `ioctl()` eine -1, sonst ist der Rückgabewert 0.[4]

[4] Einige Kommandos nutzen den Rückgabewert und liefern deshalb auch andere Werte als 0 oder -1.

Hinweis:

Soll die Funktion `ioctl()` auf ein mit `fopen()` geöffnetes Device angewendet werden, kann der Dateideskriptor mit der Funktion `fileno()` erfragt werden:

```
FILE *stream;
ioctl(fileno(stream), ...);
```

6.2 Das CD-ROM-Laufwerk

Das CD-ROM-Laufwerk lässt sich über das Device „`/dev/cdrom`“ ansprechen. Dabei handelt es sich um einen symbolischen Link auf das tatsächliche Device, z.B. „`/dev/hdc`“. Bevor Sie auf das CD-ROM-Laufwerk zugreifen können, müssen Sie sicherstellen, dass die Zugriffsrechte entsprechend gesetzt sind! Dazu können Sie das Device (nicht den Link!) entweder für alle Benutzer freigeben (z.B. `chmod a+r /dev/hdc`), oder – was auf Systemen mit mehreren Benutzern sinnvoller ist – Sie tragen Ihren Benutzernamen in der Datei „`/etc/group`“ in die Gruppe „`disk`“ ein. Wenn Sie sich als Benutzer „`root`“ anmelden, haben Sie natürlich auch Zugriff auf das CD-ROM-Laufwerk, doch davon rate ich dringend ab.
Die für das CD-ROM-Laufwerk zulässigen `ioctl()`-Kommandos sind in der Include-Datei „`/usr/include/linux/cdrom.h`“ aufgelistet.

6.2.1 Die CD „auswerfen“

Als erstes Anwendungsbeispiel für den Umgang mit dem CD-ROM-Laufwerk soll hier das Programm „`eject`“ vorgestellt werden, das die eingelegte CD „auswirft“:

```
 1  /*
 2      eject.c - CDROM-Laufwerk oeffnen
 3  */
 4
 5  # include <stdio.h>
 6  # include <unistd.h>
 7  # include <sys/ioctl.h>
 8  # include <fcntl.h>
 9  # include <linux/cdrom.h>
10
```

```
11  int main()
12   {
13    int fd;
14
15    if ((fd = open("/dev/cdrom", O_RDONLY | O_NONBLOCK))
16        == -1)
17     {
18      perror("eject: Can't open /dev/cdrom");
19      return(1);
20     }
21
22    if (ioctl(fd, CDROMEJECT) == -1)
23     {
24      perror("eject: ioctl() failed");
25      return(1);
26     }
27
28    close(fd);
29    return(0);
30   }
```

Betrachten wir zunächst die Zeile 15, in der das Device geöffnet wird. Als „Flags“ ist hier die Oder-Verknüpfung aus `O_RDONLY` und `O_NONBLOCK` angegeben. Letzteres bewirkt, dass die Funktion `open()` auch ausgeführt werden kann, wenn keine CD im Laufwerk liegt.
Nach erfolgreichem Öffnen des Devices wird in Zeile 22 mit Hilfe der Funktion `ioctl()` das Kommando `CDROMEJECT` an das Device geschickt. Dieses Kommando benötigt keinen Parameter. Weitere Kommandos dieser Art sind `CDROMSTART`, `CDROMSTOP`, `CDROMRESET` und `CDROMCLOSETRAY`.

6.2.2 Fähigkeiten des Laufwerks auslesen

Nicht alle CD-ROM-Laufwerke bieten die Möglichkeit, das Laufwerk selbsttätig wieder zu schließen – z.B. bei Notebooks muss das Laufwerk meist „von Hand“ wieder geschlossen werden. Auch andere Kommandos lassen sich nicht auf alle CD-ROM-Laufwerke anwenden. Daher gibt es das Kommando `CDROM_GET_CAPABILITY`, mit dem die Fähigkeiten des Devices abgefragt werden können. Das folgende Programm „`cdromcap`“ zeigt die Anwendung dieses Kommandos:

```
1  /*
2      cdromcap.c - Faehigkeiten des CDROM-Laufwerks
```

```
 3  */
 4
 5  # include <stdio.h>
 6  # include <unistd.h>
 7  # include <sys/ioctl.h>
 8  # include <fcntl.h>
 9  # include <linux/cdrom.h>
10
11  int main()
12   {
13    int fd, caps;
14
15    if ((fd = open("/dev/cdrom", O_RDONLY | O_NONBLOCK))
16        == -1)
17     {
18      perror("cdromcap: Can't open /dev/cdrom");
19      return(1);
20     }
21
22    if ((caps = ioctl(fd, CDROM_GET_CAPABILITY)) == -1)
23     {
24      perror("cdromcap: ioctl() failed");
25      return(1);
26     }
27
28    printf("Drive is a CD-R: %s, CD-RW: %s, DVD: %s, "
29           "DVD-R: %s.\n",
30           (caps & CDC_CD_R)? "yes" : "no",
31           (caps & CDC_CD_RW)? "yes" : "no",
32           (caps & CDC_DVD)? "yes" : "no",
33           (caps & CDC_DVD_R)? "yes" : "no");
34
35    printf("It can close tray: %s, lock: %s, "
36           "select disc: %s.\n",
37           (caps & CDC_CLOSE_TRAY)? "no" : "yes",
38           (caps & CDC_LOCK)? "yes" : "no",
39           (caps & CDC_SELECT_DISC)? "yes" : "no");
40
41    close(fd);
42    return(0);
43   }
```

Der Rückgabewert der Funktion `ioctl()` in Zeile 22 enthält bei erfolgreicher Ausführung des Kommandos `CDROM_GET_CAPABILITY` die Fähigkeiten des Devices, wobei jede Fähigkeit durch ein bestimmtes Bit repräsentiert wird. Eine Liste der möglichen Features erhält man mit (Benutzereingaben sind *schräg* dargestellt):

```
> grep CDC_ /usr/include/linux/cdrom.h
#define CDC_CLOSE_TRAY        0x1 /* caddy systems _can't_ close */
#define CDC_OPEN_TRAY         0x2 /* but _can_ eject. */
#define CDC_LOCK              0x4 /* disable manual eject */
#define CDC_SELECT_SPEED      0x8 /* programmable speed */
#define CDC_SELECT_DISC      0x10 /* select disc from juke-box */
#define CDC_MULTI_SESSION    0x20 /* read sessions>1 */
#define CDC_MCN              0x40 /* Medium Catalog Number */
#define CDC_MEDIA_CHANGED    0x80 /* media changed */
#define CDC_PLAY_AUDIO      0x100 /* audio functions */
#define CDC_RESET           0x200 /* hard reset device */
#define CDC_IOCTLS          0x400 /* driver has non-std ioctls */
#define CDC_DRIVE_STATUS    0x800 /* driver implements drive
                                     status */
#define CDC_GENERIC_PACKET 0x1000 /* driver implements generic
                                     packets */
#define CDC_CD_R           0x2000 /* drive is a CD-R */
#define CDC_CD_RW          0x4000 /* drive is a CD-RW */
#define CDC_DVD            0x8000 /* drive is a DVD */
#define CDC_DVD_R         0x10000 /* drive can write DVD-R */
#define CDC_DVD_RAM       0x20000 /* drive can write DVD-RAM */
```

Achtung: Die Bedeutung von `CDC_CLOSE_TRAY` ist sozusagen „invertiert“. Ist dieses Bit gesetzt, lässt sich das Laufwerk *nicht* automatisch schließen.

6.2.3 Audio-CDs abspielen

Eine Anwendung des CD-ROM-Laufwerks, die den direkten Device-Zugriff erfordert, ist das Abspielen von Audio-CDs. Dazu benötigen Sie entweder eine Sound-Karte, die mit dem CD-Laufwerk verbunden ist, oder ein Laufwerk mit einem Kopfhörerausgang. Bei Verwendung der Sound-Karte sollten Sie zudem ein Programm zum Einstellen der Audioquellen und der Lautstärke installiert haben, z.B. „`xmix`“ oder „`aumix`“.

Das Inhaltsverzeichnis einer Audio-CD

Bevor wir jedoch zum Abspielen einer CD kommen, soll zunächst gezeigt werden, wie das „Inhaltsverzeichnis“ einer Audio-CD gelesen wird und in welcher Weise die Positionen der einzelnen Stücke (*Tracks*) codiert sind. Dazu sei das folgende Programm betrachtet:

```
1  /*
2      cdtoc.c - "Inhaltsverzeichnis" einer Audio-CD
3  */
4
5  # include <stdio.h>
6  # include <unistd.h>
7  # include <errno.h>
8  # include <sys/ioctl.h>
9  # include <fcntl.h>
10 # include <linux/cdrom.h>
11
12 int main()
13  {
14   int fd, i;
15   struct cdrom_tochdr toc_hdr;
16   struct cdrom_tocentry toc_entry;
17
18   if ((fd = open("/dev/cdrom", O_RDONLY)) == -1)
19    {
20     if (errno == ENOMEDIUM)
21       fprintf(stderr, "cdtoc: No CD in drive.\n");
22     else
23       perror("cdtoc: Can't open /dev/cdrom");
24     return(1);
25    }
26
27   if (ioctl(fd, CDROMREADTOCHDR, &toc_hdr) == -1)
28    {
29     perror("cdtoc: Can't get header");
30     return(1);
31    }
32   printf("First track: %d, last track: %d\n",
33          toc_hdr.cdth_trk0, toc_hdr.cdth_trk1);
34
35   for (i=toc_hdr.cdth_trk0; i<=toc_hdr.cdth_trk1; i++)
```

```
36      {
37       toc_entry.cdte_track = i;
38       toc_entry.cdte_format = CDROM_MSF;
39       if (ioctl(fd, CDROMREADTOCENTRY, &toc_entry) == -1)
40        {
41         perror("cdtoc: Can't get table of contents");
42         return(1);
43        }
44       printf(" %2d) %02d:%02d.%02d\n", i,
45              toc_entry.cdte_addr.msf.minute,
46              toc_entry.cdte_addr.msf.second,
47              (toc_entry.cdte_addr.msf.frame*100+37)/75);
48      }
49
50     close(fd);
51     return(0);
52    }
```

In den Zeilen 18 bis 25 wird das Device geöffnet, wobei hier im Gegensatz zu den beiden vorangegangenen Abschnitten bewusst auf die Option `O_NONBLOCK` verzichtet wurde. Dadurch führt der Aufruf der Funktion `open()` zum Fehler `ENOMEDIUM`, wenn keine CD eingelegt ist.
Als Nächstes wird in den Zeilen 27 bis 33 der Kopf („`HDR`“ für *Header*) des Inhaltsverzeichnisses („`TOC`“ für *Table of Contents*) gelesen. Dies wird durch das `ioctl()`-Kommando `CDROMREADTOCHDR` erreicht, das als dritten Parameter des `ioctl()`-Aufrufs einen Zeiger auf eine Variable vom Typ `struct cdrom_tochdr` erwartet. Diese Struktur wird in „`/usr/include/linux/cdrom.h`“ definiert und enthält zwei Einträge:

```
struct cdrom_tochdr
 {
  __u8 cdth_trk0;      /* start track */
  __u8 cdth_trk1;      /* end track */
 };
```

Durch den `ioctl()`-Aufruf werden die Nummer des ersten Stücks (in der Regel ist das 1) und die Nummer des letzten Stücks auf der CD in diese Struktur eingetragen.
In den Zeilen 35 bis 48 werden dann in einer `for()`-Schleife Informationen zu allen Stücken eingeholt. Dazu dient das `ioctl()`-Kommando `CDROMREADTOCENTRY`, das als dritten Parameter des `ioctl()`-Aufrufs

einen Zeiger auf eine Variable vom Typ `struct cdrom_tocentry` erwartet. Diese Struktur muss vor dem `ioctl()`-Aufruf mit einer gültigen Nummer für das Stück und der Angabe des Formates für die Position initialisiert werden (Zeile 37 und 38). Als Format stehen `CDROM_MSF` und `CDROM_LBA` zur Verfügung. Ersteres steht für *Minute-Second-Frame*, Letzteres für *Logical Block Address*. Während die Adresse im LBA-Format durch eine Integerzahl repräsentiert wird, wird das MSF-Format durch die Struktur `cdrom_msf0` abgebildet:

```
struct cdrom_msf0
 {
  __u8 minute;
  __u8 second;
  __u8 frame;
 };
```

Die Einträge `minute` und `second` sind selbsterklärend, der Eintrag `frame` gibt den Bruchteil einer Sekunde an, wobei eine Sekunde aus `CD_FRAMES` (= 75) Frames besteht. Der Zusammenhang zwischen LBA- und MSF-Adresse lässt sich beschreiben durch

$$\texttt{LBA} = (\texttt{minute} \cdot 60 + \texttt{second} - 2) \cdot \texttt{CD\_FRAMES} + \texttt{frame}$$

Nach dem `ioctl()`-Aufruf in Zeile 39 des Programms enthält die Variable `toc_entry` unter anderem die Position des angegebenen Stücks mit der Nummer `i` im MSF-Format. Diese wird in der `printf()`-Anweisung in den Zeilen 44 bis 47 ausgegeben, wobei die Frames in $^1/_{100}$ Sekunden umgerechnet werden.

Wiedergabe einer Audio-CD

Bevor Sie eine Audio-CD wiedergeben wollen, stellen Sie bitte sicher, dass das CD-ROM-Laufwerk als Audioquelle aktiv ist. Dies kann z.B. mit dem Programm „`xmix`“ erfolgen oder in der Shell mit

```
aumix -c 100
```

Alternativ bieten manche CD-Laufwerke auch einen Kopfhörerausgang mit einem separaten Lautstärkeregler.
Das Abspielen einer Audio-CD erfolgt mit dem `ioctl()`-Kommando `CDROMPLAYMSF`, das als Parameter einen Zeiger auf eine Struktur vom Typ `cdrom_msf` erwartet:

```
struct cdrom_msf
 {
  __u8 cdmsf_min0;     /* start minute */
  __u8 cdmsf_sec0;     /* start second */
  __u8 cdmsf_frame0;   /* start frame */
  __u8 cdmsf_min1;     /* end minute */
  __u8 cdmsf_sec1;     /* end second */
  __u8 cdmsf_frame1;   /* end frame */
 };
```

Hier muss im MSF-Format Anfangs- und Endposition des abzuspielenden Teils der CD angegeben werden. Die Anfangsposition eines Stücks kann wie im vorangegangenen Beispielprogramm mit Hilfe des Kommandos `CDROMREADTOCENTRY` erfolgen. Als Ende eines Stückes kann die Anfangsposition des darauf folgenden Stücks angegeben werden. Die Endposition des letzten Stückes kann über die Position von `CDROM_LEADOUT` ermittelt werden. Das folgende Programm spielt eine Audio-CD vollständig ab:

```
1  /*
2      cdplay.c - Audio-CD abspielen
3  */
4
5  # include <stdio.h>
6  # include <unistd.h>
7  # include <sys/ioctl.h>
8  # include <fcntl.h>
9  # include <linux/cdrom.h>
10
11 void err_exit(char *err_text, int return_code)
12  {
13   perror(err_text);
14   exit(return_code);
15  }
16
17 int main()
18  {
19   int fd;
20   struct cdrom_tocentry toc_entry;
21   struct cdrom_msf start_stop;
22
23   if ((fd = open("/dev/cdrom", O_RDONLY)) == -1)
```

```
24      err_exit("cdplay: Can't open /dev/cdrom", 1);
25
26                          /* Anfang des 1. Stuecks */
27    toc_entry.cdte_track = 1;
28    toc_entry.cdte_format = CDROM_MSF;
29    if (ioctl(fd, CDROMREADTOCENTRY, &toc_entry) == -1)
30      err_exit("cdplay: ioctl() failed", 1);
31    start_stop.cdmsf_min0
32      = toc_entry.cdte_addr.msf.minute;
33    start_stop.cdmsf_sec0
34      = toc_entry.cdte_addr.msf.second;
35    start_stop.cdmsf_frame0
36      = toc_entry.cdte_addr.msf.frame;
37
38                          /* Ende des letzten Stuecks */
39    toc_entry.cdte_track = CDROM_LEADOUT;
40    toc_entry.cdte_format = CDROM_MSF;
41    if (ioctl(fd, CDROMREADTOCENTRY, &toc_entry) == -1)
42      err_exit("cdplay: ioctl() failed", 1);
43    start_stop.cdmsf_min1
44      = toc_entry.cdte_addr.msf.minute;
45    start_stop.cdmsf_sec1
46      = toc_entry.cdte_addr.msf.second;
47    start_stop.cdmsf_frame1
48      = toc_entry.cdte_addr.msf.frame;
49
50    if (ioctl(fd, CDROMPLAYMSF, &start_stop) == -1)
51      err_exit("cdplay: ioctl() failed", 1);
52
53    printf("Press <RETURN> to stop playing.\n");
54    getchar();
55
56    if (ioctl(fd, CDROMSTOP) == -1)
57      err_exit("cdplay: ioctl() failed", 1);
58
59    close(fd);
60    return(0);
61  }
```

Nach dem Öffnen des Devices in Zeile 23 wird in den Zeilen 27 bis 36 die MSF-Adresse des ersten Stücks abgefragt und in die Start-Position

der Variablen `start_stop` eingetragen. Analog wird in den Zeilen 39 bis 48 die Endadresse der CD erfragt und als Stop-Position in die Variable `start_stop` eingetragen. Danach erfolgt in Zeile 50 der `ioctl()`-Aufruf mit dem Kommando `CDROMPLAYMSF`, der das Abspielen der CD bewirkt. Mit dem Kommando `CDROMSTOP` (Zeile 56) lässt sich das Abspielen vorzeitig beenden.

Weitere Möglichkeiten

Ein „echter“ CD-Player bietet neben den Funktionen „play“ und „stop“ auch eine „pause“-Funktion für eine (kurze) Unterbrechung. Auch das Linux-Device bietet diese Möglichkeit mit dem `ioctl()`-Kommando `CDROMPAUSE`, das keinen weiteren Parameter benötigt. Das Abspielen lässt sich danach mit `CDROMRESUME` an der gleichen Stelle wieder fortsetzen.
Unser Programm `cdplay` wartet, nachdem es das Abspielen der CD gestartet hat, auf eine Benutzereingabe. Es „bemerkt“ nicht, ob die CD vielleicht schon zu Ende ist. Um den aktuellen Status des CD-Laufwerks abzufragen, dient das `ioctl()`-Kommando `CDROMSUBCHNL`. Als Parameter wird bei diesem Kommando ein Zeiger auf eine Variable vom Typ `struct cdrom_subchnl` erwartet. Vor dem `ioctl()`-Aufruf muss das Element `cdsc_format` dieser Struktur mit `CDROM_MSF` oder `CDROM_LBA` initialisiert werden. Nach erfolgreicher Ausführung des Kommandos `CDROMSUBCHNL` enthält die Struktur unter anderem Informationen über den Audio-Status, die Nummer des (laufenden) Stücks und die Position im vorgegebenen Format. Das folgende Programm demonstriert das Abfragen und Ausgeben dieser Informationen:

```
 1  /*
 2      cdstat.c - Status des CD-Laufwerks abfragen
 3  */
 4
 5  # include <stdio.h>
 6  # include <unistd.h>
 7  # include <sys/ioctl.h>
 8  # include <fcntl.h>
 9  # include <linux/cdrom.h>
10
11  int main()
12   {
13    int fd;
14    char *status_string;
```

```
15    struct cdrom_subchnl subch;
16
17    if ((fd = open("/dev/cdrom", O_RDONLY | O_NONBLOCK))
18        == -1)
19     {
20      perror("cdstat: Can't open /dev/cdrom");
21      return(1);
22     }
23
24    subch.cdsc_format = CDROM_MSF;
25    if (ioctl(fd, CDROMSUBCHNL, &subch) == -1)
26     {
27      perror("cdstat: ioctl() failed");
28      return(1);
29     }
30    switch(subch.cdsc_audiostatus)
31     {
32      case CDROM_AUDIO_PLAY:
33        status_string = "playing"; break;
34      case CDROM_AUDIO_PAUSED:
35        status_string = "paused"; break;
36      case CDROM_AUDIO_COMPLETED:
37        status_string = "completed"; break;
38      case CDROM_AUDIO_ERROR:
39        status_string = "error"; break;
40      default: status_string = "---";
41     }
42    printf("CD status:\t\t%s\n", status_string);
43    printf("current track:\t\t%d\n", subch.cdsc_trk);
44    printf("current position:\t%02d:%02d\n",
45           subch.cdsc_absaddr.msf.minute,
46           subch.cdsc_absaddr.msf.second);
47
48    close(fd);
49    return(0);
50   }
```

Einige CD-ROM-Laufwerke bieten die Möglichkeit, bei Wiedergabe einer Audio-CD die Lautstärke per Software einzustellen. Dazu bietet das Device zwei `ioctl()`-Kommandos: `CDROMVOLREAD` und `CDROMVOLCTRL`, Ersteres zum Auslesen der Einstellungen und Letzteres zur Modifizierung der Einstellungen. Beide Kommandos benötigen als dritten Para-

meter des `ioctl()`-Aufrufs den Zeiger auf eine Variable vom Typ `struct cdrom_volctrl`:

```
struct cdrom_volctrl
 {
  __u8 channel0;        /* left channel */
  __u8 channel1;        /* right channel */
  __u8 channel2;
  __u8 channel3;
 };
```

Nach erfolgreicher Ausführung von `CDROMVOLREAD` enthält diese Struktur die aktuell eingestellten Werte für alle vier (!) Kanäle – Kanal 2 und Kanal 3 sind in der Regel nicht belegt und deren Lautstärke ist dementsprechend 0. Mit dem Kommando `CDROMVOLCTRL` werden die in der Struktur eingetragenen Lautstärke-Werte (zwischen 0 und 255) für die entsprechenden Kanäle eingestellt.

6.3 Ansteuerung einer Sound-Karte

Eine Sound-Karte bietet vielfältige Möglichkeiten: Neben dem Analog-Digital- und Digital-Analog-Wandler zur Aufnahme und Wiedergabe von Audio-Signalen enthalten die meisten Karten einen Mixer, also eine Art Mischpult, einen Synthesizer sowie einen Chip (UART) für das Senden und Empfangen von MIDI-Daten. Die verschiedenen Funktionseinheiten einer Sound-Karte sind unter Linux auf mehrere Devices abgebildet:

Device	Beschreibung
`/dev/dsp`	Schnittstelle zum A/D- und D/A-Wandler
`/dev/audio`	Schnittst. zum A/D- und D/A-Wandler (8 Bit, log.)
`/dev/mixer`	elektronisches „Mischpult“
`/dev/sequencer`	MIDI-Rekorder
`/dev/mpu401`	MIDI-Schnittstelle (UART)
`/dev/sndstat`	Infos zur Sound-Karte

Neben diesen Devices sind noch eine ganze Reihe weiterer *character special files* in Verbindung mit einer Sound-Karte verfügbar – beispielsweise `/dev/music` oder `/dev/pcaudio`. Die folgenden Abschnitte beschränken sich jedoch auf die „gebräuchlichsten“ Devices der Sound-Karte.