Das /proc-(Pseudo-)Filesystem beinhaltet eine Menge Informationen zu Ihrem System. Ob Sie nun Informationen zum aktuellen Prozess, zur Hardware in Ihrem System oder zum Kernel benötigen, all dies finden Sie im /proc-Filesystem.

4 Zugriff auf Systeminformationen

Da der Kernel den kompletten Rechner verwaltet, besitzt dieser auch eine Menge Informationen zum System, auf dem dieser läuft. Diese Informationen werden vom Kernel in Pseudo-Dateien und Pseudo-Verzeichnissen angelegt. All diese Dateien liegen im `/proc`-Verzeichnis. Auf die Einträge im `/proc`-Verzeichnis können Sie wie auf gewöhnliche Dateien zugreifen; meistens allerdings nur lesend. Dieses (Pseudo-)Dateisystem belegt keinerlei Plattenplatz und befindet sich im Hauptspeicher (RAM). Sie können sich gerne einen Überblick zum `/proc`-Verzeichnis verschaffen.

```
$ ls -l /proc | less
```

Auf alle diese Informationen können Sie (fast) ohne weiteres lesend zugreifen. Wollen Sie z. B. den aktuellen Kernel Ihres Systems ermitteln und ausgeben lassen, können Sie auf die Datei `/proc/version` zugreifen:

```
$ cat /proc/version
Linux version 2.6.27-7-generic (buildd@palmer)
(gcc version 4.3.2 (Ubuntu 4.3.2-1ubuntu11))
#1 SPM Tue Jan 4 19:33:20 UTC 2009
```

Schon erhalten Sie die aktuelle Kernel-Version, das Datum, an dem dieser kompiliert wurde, und die Version des zugehörigen Compilers als String zurück.

4.1 Informationen aus dem /proc-Verzeichnis herausziehen

Als Programmierer dürfte Sie in erster Linie interessieren, wie Sie mit Ihrem Programm an diese Informationen kommen. Als Beispiel dient hier die Speicherauslastung des Systems. Hierbei soll lediglich ermittelt werden, wie viel RAM insgesamt auf Ihrem System zur Verfügung steht. Diese Informationen erhalten Sie aus `/proc/meminfo`:

```
$ cat /proc/meminfo
MemTotal:       514296 kB
MemFree:         10428 kB
Buffers:        122940 kB
Cached:         120112 kB
SwapCached:       1152 kB
Active:         337336 kB
```

```
Inactive:       122512 kB
HighTotal:           0 kB
HighFree:            0 kB
LowTotal:       514296 kB
LowFree:         10428 kB
SwapTotal:      409616 kB
SwapFree:       385128 kB
...
```

Uns interessiert hier der Wert »MemTotal«. Der einfachste Weg, diesen Wert auszulesen, ist es, `/proc/meminfo` mit `fopen()` zu öffnen, in einen Puffer einzulesen und nach der Stringfolge »MemTotal« zu suchen. Anschließend kann der Wert mit der `sscanf()` aus dem Puffer in eine dafür vorgesehene Variable übergeben werden. Hier das mögliche Beispiel:

```
/* memory.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

static long get_mem_total (void) {
   FILE *fp;
   char buffer[1024];
   size_t bytes_read;
   char *match;
   long mem_tot;

   if((fp = fopen("/proc/meminfo", "r")) == NULL) {
      perror("fopen()");
      exit(EXIT_FAILURE);
   }

   bytes_read = fread (buffer, 1, sizeof (buffer), fp);
   fclose (fp);
   if (bytes_read == 0 || bytes_read == sizeof (buffer))
      return 0;
   buffer[bytes_read] = '\0';
   /* Suchen nach der Stringfolge "Memtotal" */
   match = strstr (buffer, "MemTotal");
   if (match == NULL) /* Nicht gefunden */
      return 0;
   sscanf (match, "MemTotal: %ld", &mem_tot);
   return (mem_tot/1024); /* 1MB = 1024KB */
}

int main (void) {
  long memory = get_mem_total();
  if(memory == 0)
     printf("Konnte RAM nicht ermitteln\n");
  else
```

```
    printf("Vorhandener Arbeitsspeicher: %ldMB\n" ,memory);
  return EXIT_SUCCESS;
}
```

Das Programm bei der Ausführung:

```
$ gcc -o memory memory.c
$ ./memory
Vorhandener Arbeitsspeicher: 249MB
```

So oder so ähnlich können Sie in der Regel bei allen Informationen vorgehen, die Sie im `/proc`-Verzeichnis finden und benötigen. Sie sollten allerdings darauf achten, dass sich der Name des Eintrags im `/proc`-Verzeichnis bei neueren Kernel-Versionen ändern könnte. So könnte vielleicht in absehbarer Zeit aus `/proc/meminfo` `/proc/more_meminfo` (nur ein Beispiel) werden. Bedenken Sie dies, wenn Sie Ihr Programm auf dem Stand der Zeit halten wollen.

4.2 Hardware-/Systeminformationen ermitteln

Viele Einträge im `/proc`-Filesystem erlauben Ihnen, Informationen zur Hardware des Systems zu ermitteln. Das sind interessante Informationen für Systemadministratoren und auch für den Programmierer von Programmen. Hier einige häufig verwendete Einträge im Überblick.

Hinweis

Bei diesem Buch handelt es sich nicht um ein Buch zur Erklärung von Hardware. Sollten Sie keinerlei Kenntnisse vom Innenleben Ihres PC haben, dann wäre zusätzliche Literatur recht sinnvoll.

4.2.1 CPU-Informationen – /proc/cpuinfo

`/proc/cpuinfo` beinhaltet Informationen zur CPU (Prozessor) oder zu den CPUs, die auf Ihrem System laufen. Ausgegeben wird dabei die Anzahl der Prozessoren. Ist der Wert von »`processor`« wie im Beispiel 0, so kann dies bedeuten, dass es sich um ein Single-Prozessor-System handelt.

Hinweis

Auf SMP (Multi-CPU-Maschinen) ist der Wert von `processor ...` nicht direkt 0, sondern es wird für jeden Prozessor einzeln »`processor`, `vendor_id`« etc. angezeigt, weshalb »*auf 0 gesetzt*« nicht ganz korrekt ist.

Weitere Daten finden Sie zu der CPU-Familie, dem Modell, der Frequenz, der Revision und noch einigem mehr. Hierzu ein Ausschnitt aus meinem System:

```
$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
```

```
cpu family      : 15
model           : 2
model name      : Intel(R) Pentium(R) 4 CPU 1.60GHz
stepping        : 4
cpu MHz         : 1594.865
cache size      : 512 KB
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 2
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 sep mtrr pge mca cmov pat
  pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
bogomips        : 3185.04
```

Für mehr Informationen zur CPU sei die Dokumentation zu *cpuid* des Intel Architecture Software Developers Manual, Volume 2, empfohlen.

4.2.2 Geräteinformationen – /proc/devices

Hier sind Informationen zu allen verfügbaren Geräten (zeichen- und blockorientiert) wie der Festplatte, dem Diskettenlaufwerk, der seriellen und parallelen Schnittstelle usw. aufgelistet.

4.2.3 Speicherauslastung – /proc/meminfo

(S. o. unter 4.1.) In dieser Pseudo-Datei kann der aktuelle Speicherstatus ausgelesen werden. Angezeigt werden der vorhandene und der belegte Speicher (sowohl der physikalische als auch der Swap-Speicher). Ebenfalls lässt sich ermitteln, wie viel davon für geteilten Speicher (Shared Memory), Puffer und Caches belegt ist, die der Kernel benutzt.

4.2.4 Weitere Hardware-Informationen zusammengefasst

Es gibt natürlich noch eine Menge mehr Informationen zur Hardware im `/proc`-Verzeichnis, die allerdings zum Teil auch von der Konfiguration des Systems abhängt. Hierzu eine kleine Zusammenfassung zu weiteren gängigen Einträgen im `/proc`-Verzeichnis.

Verzeichniseintrag	Bedeutung
`/proc/dma`	Liste von belegten DMA-Kanälen, die für einen Treiber reserviert sind, und der Name des Treibers
`/proc/interrupts`	Liste der benutzten Interrupts mit Anzahl der passierten Interrupts seit Systemstart, Typ des Interrupts und Angabe, durch welche Module dieser Interrupt verwendet wird

Tabelle 4.1 Systeminformationen im /proc-Verzeichnis

Verzeichniseintrag	Bedeutung
/proc/ioports	Eine Liste aller belegten IO-Ports für die Ein-/Ausgabe wie die Festplatte, Ethernet, Soundkarte, Modem, USB etc. (gemappte Hardware-Speicherbereiche finden sich in /proc/iomem).
/proc/stat	Allgemeine Informationen über den Status der Prozessoren. Diese Informationen werden vom Programm procinfo verwendet und übersichtlich aufgelistet.
/proc/uptime	Anzahl der Sekunden, seitdem das System gestartet ist, und wie lange davon die CPU mit dem *Nichts*tun (HLT) (Leerlaufzeit = engl. *idle*) verbracht hat
/proc/scsi/	Unterverzeichnis mit Informationen zu SCSI-Geräten
/proc/scsi/scsi	Sofern Sie eine SCSI-Schnittstelle besitzten, finden Sie hier eine Auflistung aller Geräte.
/proc/ide/	Unterverzeichnis mit Informationen zu IDE-Geräten
/proc/apm	Informationen zum Advanced Power Management (z. B. ein Text wie »AC offline« bedeutet, dass ein Notebook im Batteriebetrieb läuft)
/proc/mounts	Liste aller »gemounteten« Dateisysteme
/proc/net/	Unterverzeichnis zu Netzwerkinformationen
/proc/loadavg	Duchschnittliche Systemauslastung (eine Minute, drei Minuten, fünf Minuten, aktive Prozesse/Anzahl Prozesse, und zuletzt benutzte PID)

Tabelle 4.1 Systeminformationen im /proc-Verzeichnis (Forts.)

Das folgende Listing soll Ihnen jetzt demonstrieren, wie einfach es ist, aus dem System nützliche Informationen zur Hardware ausgeben zu lassen. Das Beispiel stellt eine einfache Schnittstelle dar, die es zu erweitern gilt. Damit haben Sie schon die Grundlage für ein Systemprogramm zur Hand. Sollte es ein Konsolenprogramm werden, dann müssten Sie die Ausgabe benutzerfreundlich anpassen; ebenfalls dann, wenn Sie nur einzelne Informationen ausgeben wollen. Natürlich können Sie über das Programm auch ein grafisches Frontend ziehen. Denn auch die Systemprogramme von KDE oder GNOME machen nichts anderes, als die Daten aus dem /proc-Filesystem zu lesen und entsprechend auszugeben.

```
/* myinfo.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#define BUF 4096 /* Anpassen */

enum { CPU, DEV, DMA, INT, IOP, MEM, VERS, SCSI, EXIT };

static const char *info[] = {
   "/proc/cpuinfo", "/proc/devices", "/proc/dma",
   "/proc/interrupts", "/proc/ioports", "/proc/meminfo",
   "/proc/version", "/proc/scsi/scsi"
 };

static void get_info (const int inf) {
   FILE *fp;
```

```
   char buffer[BUF];
   size_t bytes_read;

   if((fp = fopen(info[inf], "r")) == NULL) {
      perror("fopen()");
      return;
   }
   bytes_read = fread (buffer, 1, sizeof (buffer), fp);
   fclose (fp);
   if (bytes_read == 0 || bytes_read == sizeof (buffer))
      return;
   buffer[bytes_read] = '\0';

   printf("%s",buffer);
   printf("Weiter mit ENTER");
   getchar();
   return;
}

int main (void) {
   int auswahl;
   do {
      printf("Wozu benötigen Sie Informationen?\n\n");
      printf("-%d- Prozessor\n", CPU);
      printf("-%d- Geräte\n", DEV);
      printf("-%d- DMA\n", DMA);
      printf("-%d- Interrupts\n", INT);
      printf("-%d- I/O-Ports\n", IOP);
      printf("-%d- Speicher\n", MEM);
      printf("-%d- Version\n", VERS);
      printf("-%d- SCSI\n", SCSI);
      printf("-%d- Programmende\n", EXIT);
      printf("\nIhre Auswahl : ");
      do{ scanf("%d",&auswahl); } while(getchar() != '\n');

      switch(auswahl) {
         case CPU : get_info(CPU);   break;
         case DEV : get_info(DEV);   break;
         case DMA : get_info(DMA);   break;
         case INT : get_info(INT);   break;
         case IOP : get_info(IOP);   break;
         case MEM : get_info(MEM);   break;
         case VERS: get_info(VERS);  break;
         case SCSI: get_info(SCSI);  break;
         case EXIT: printf("Bye\n"); break;
         default  : printf("Falsche Eingabe?\n");
      }
   } while(auswahl != EXIT);
   return EXIT_SUCCESS;
}
```

Das Programm im Einsatz:

```
$ ./myinfo
Wozu benötigen Sie Informationen?

-0- Prozessor
-1- Geräte
-2- DMA
-3- Interrupts
-4- I/O-Ports
-5- Speicher
-6- Version
-7- SCSI
-8- Programmende

Ihre Auswahl : 7
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00
  Vendor: VMware,  Model: VMware Virtual S Rev: 1.0
  Type:   Direct-Access    ANSI  SCSI revision: 02
Host: scsi2 Channel: 00 Id: 00 Lun: 00
  Vendor: MATSHITA Model: DVD-RAM UJ-850S  Rev: 1.60
  Type:   CD-ROM            ANSI  SCSI revision: 05
Weiter mit ENTER
```

Hinweis

Fragen zum Löschen des Bildschirms in einem Terminal werden in Kapitel 13, »Terminal E/A und Benutzerschnittstellen für die Konsole«, beantwortet.

4.3 Prozessinformationen

Neben Hardware-Informationen können Sie aus dem `/proc`-Filesystem auch Informationen zu den einzelnen Prozessen ermitteln. Auch Programme wie `ps` oder `top` nutzen diese Möglichkeit, um Ihnen Informationen zu einem Prozess zu liefern. Den Prozessen und deren Verwaltung ist extra noch ein Kapitel gewidmet. Das `/proc`-Filesystem beinhaltet für jeden Prozess extra ein Unterverzeichnis. Der Name des Verzeichnisses entspricht der Prozess-ID. Lassen Sie sich am besten das `/proc`-Verzeichnis ausgeben:

```
$ ls -l /proc | less
dr-xr-xr-x    3 root     root     0 2003-11-13 23:46 1
dr-xr-xr-x    3 root     root     0 2003-11-13 23:46 12
dr-xr-xr-x    3 root     root     0 2003-11-13 23:46 1201
dr-xr-xr-x    3 root     root     0 2003-11-13 23:46 1278
dr-xr-xr-x    3 bin      root     0 2003-11-13 23:46 1303
...
dr-xr-xr-x    4 root     root     0 2003-11-13 23:47 tty
-r--r--r--    1 root     root     0 2003-11-13 23:47 uptime
```

```
-r--r--r--    1 root     root     0 2003-11-13 23:47 version
dr-xr-xr-x    3 root     root     0 2003-11-13 23:47 video
```

Diese Verzeichnisse werden als Prozessverzeichnisse bezeichnet, da sich diese auf die Prozess-ID (PID) beziehen (komischerweise taucht es in den 2.6er-Sources als »TGID« auf ...) und Informationen zu diesem Prozess beinhalten. Eigentümer und Gruppe eines jeden solchen PID-Verzeichnisses werden auf die ID des Benutzers, der den Prozess ausführt, gesetzt. (Eine nachträgliche Änderung über `setuid()` ändert nur den Eigentümer des Verzeichnisses, nicht den der Dateien.) Nach Beendigung des Prozesses wird der Eintrag im `/proc`-Verzeichnis ebenfalls wieder gelöscht. Während der Ausführung eines Prozesses können Sie aus dem Verzeichnis die nun folgenden nützlichen Informationen dazu erhalten:

```
#include <unistd.h>
int main(void) { setuid(25); while(1); }
```

Im Hintergrund ausführen (»`./a.out &`«) und reingucken:

```
# ./a.out &
[1] 2275
# ls -l /proc/2275/
insgesamt 0
-r--r--r--  1 tot  users   0 2004-08-14 02:36   cmdline
-r--------  1 tot  users   0 2004-08-14 02:36   environ
lrwxrwxrwx  1 tot  users   0 2004-08-14 02:36   exe->./a.out
dr-x------  2 tot  users   0 2004-08-14 02:36
fd-rw------ 1 tot  users   0 2004-08-14 02:36   mapped_base
-r--r--r--  1 tot  users   0 2004-08-14 02:36   maps
-rw-------  1 tot  users   0 2004-08-14 02:36   mem
-r--r--r--  1 tot  users   0 2004-08-14 02:36   mounts
lrwxrwxrwx  1 tot  users   0 2004-08-14 02:36   root -> /

-r--r--r--  1 tot  users   0 2004-08-14 02:36   stat
-r--r--r--  1 tot  users   0 2004-08-14 02:36   statm
-r--r--r--  1 tot  users   0 2004-08-14 02:36   status
```

Gilt auch für `seteuid()` und `setreuid()`.

4.3.1 /proc/$pid/cmdline

Sie finden z. B. ein Verzeichnis mit der ID 2167 und wollen wissen, was in der Kommandozeile des Prozesses steht, dann müssen Sie nur `cmdline` abfragen:

```
$ cat /proc/2167/cmdline
anjuta
```

Der Prozess mit der ID 2167 ist also das Programm (die Entwicklungsumgebung) *Anjuta* bei der Ausführung. Wollen Sie alle Prozesse ausgeben lassen, können Sie dies auch folgendermaßen machen:

```
# cat /proc/[0-9]*/cmdline
```

Da die Ausgabe allerdings zu wünschen übrig lässt, sollten Sie hierfür das Kommando `strings` einsetzen:

```
# strings -f /proc/[0-9]*/cmdline
```

Und schon bekommen Sie alle zu den Prozess-IDs gehörenden Namen ausgegeben. Mit `cmdline` können Sie eine einzelne Zeile des Prozesses ausgeben, worin der Name (oder auch das Kommando) des Programms mit allen Argumenten enthalten ist.

Falls *strings* bei Ihnen nicht machen will, was es soll, können Sie auch folgende Eingabe verwenden:

```
# grep -Ha "" /proc/[0-9]*/cmdline | tr '\0' " "
```

4.3.2 /proc/$pid/environ

Jeder Prozess hat außerdem auch eine Prozessumgebung. Welche Umgebungsvariablen für welchen Prozess gesetzt sind, können Sie mit `environ` im `/proc`-Verzeichnis der Prozess-ID erfragen. Hier soll weiterhin der Prozess mit der ID 2167, die Entwicklungsumgebung *Anjuta*, beobachtet werden:

```
$ strings -f /proc/2167/environ
/proc/2167/environ: LESSKEY=/etc/lesskey.bin
/proc/2167/environ:MANPATH=/usr/local/man:/usr/share/man:/..
/proc/2167/environ:INFODIR=/usr/local/info:/usr/share/...
/proc/2167/environ: NNTPSERVER=news
...
/proc/2167/environ: KDE_FULL_SESSION=true
/proc/2167/environ: KDE_MULTIHEAD=false
/proc/2167/environ: SESSION_MANAGER=local/linux:/tmp/.IC...
/proc/2167/environ: KDE_STARTUP_ENV=linux;1068759934;528706;1890
```

Wenn es auch hier wieder nicht mit *strings* klappen sollte, dann sollten Sie es mit Folgendem probieren:

```
# tr '\0' '\n' < /proc/2167/environ
```

4.3.3 /proc/self

Wollen Sie die aktuelle Prozess-ID Ihres eigenen Programms ermitteln, so können Sie `/proc/self` auswerten. Bei diesem Eintrag handelt es sich um einen symbolischen Link des aktuell laufenden Prozesses. Das aktuelle Programm sollten Sie aber nicht mit der Shell verwechseln (versuchen Sie selbst: `ls -dl /proc/self` und `ls -dl /proc/$$`). Diesen symbolischen Link können Sie mit der Funktion `readlink()` auslesen. Hier ein Listing, das demonstriert, wie Sie die Prozess-ID des aktuell laufenden Programms ermitteln können:

```
/* my_getpid.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>

int main (void) {
```

```
    char buf[64];
    int pid;

    memset(buf, '\0', sizeof(buf));
    readlink("/proc/self", buf, sizeof(buf) - 1);
    sscanf(buf, "%d", &pid);
    printf("Meine Prozess-ID lautet: %d\n", pid);
    return EXIT_SUCCESS;
}
```

Das Programm bei seiner Ausführung:

```
$ gcc -o my_getpid my_getpid.c
$ ./my_getpid
Meine Prozess-ID lautet: 2256
```

Hinweis

Das Beispiel stellt natürlich nur eine Demonstration dar. In der Praxis werden Sie auf die Funktion `getpid()` für die Ermittlung der Prozess-ID des laufenden Programms zurückgreifen.

4.3.4 /proc/$pid/fd/

Ein weiterer Eintrag ist `fd`, ein Unterverzeichnis, das alle Einträge von Filedeskiptoren beinhaltet, die ein Prozess geöffnet hat. Jeder Eintrag darin ist ein symbolischer Link, der auf eine bestimmte Datei lesend oder schreibend zurückgreift. Natürlich sind hierbei auch immer die Standard-Filedeskriptoren (0, 1, 2) enthalten (es sei denn, sie wurden geschlossen, was viele Dämonen etc. tun!). Öffnen Sie eine Konsole, und geben Sie `ps` ein:

```
$ ps
 PID TTY          TIME CMD
 1988 pts/1    00:00:00 bash
 2827 pts/1    00:00:00 ps
```

Wollen Sie jetzt wissen, welche Filedeskiptoren *bash* offen hat, machen Sie Folgendes:

```
$ ls -l /proc/1988/fd/
insgesamt 0
lrwx------ 1 tot   users   64 2003-11-14 00:50 0->/dev/pts/1
lrwx------ 1 tot   users   64 2003-11-14 00:50 1->/dev/pts/1
lrwx------ 1 tot   users   64 2003-11-14 00:50 2->/dev/pts/1
```

Hierbei können Sie die drei Standard-Filedeskriptoren als symbolischen Link auf dem Pseudo-Terminal `pts/1` wieder erkennen. Sie können jetzt ohne weiteres auf diese Filedeskriptoren zugreifen:

```
$ echo "Ein Beweis gefällig?" > /proc/1988/fd/1
Ein Beweis gefällig?
```

Hier wurde die Standardausgabe von `echo` auf die eigentliche Standardausgabe des Pseudo-Terminals »geleitet«.

4.3.5 /proc/$pid/statm

Informationen zur Speicherbelegung eines Prozesses werden im `statm`-File hinterlegt. Wollen Sie z. B. die Speicherbelegung der Entwicklungsumgebung *Anjuta* ermitteln, gehen Sie folgendermaßen vor:

```
$ cat /proc/2167/statm
2671 2670 1678 302 0 2368 992
```

Sie bekommen dabei sieben verschiedene Zahlen zurück. Hier die Bedeutung der einzelnen Zahlen von links nach rechts:

- Gesamte Programmgröße in Kilobyte
- Größe von Speicherteilen in Kilobyte
- Anzahl von »geshareten« Seiten (so genannte Pages)
- Anzahl Seiten von Programmcode
- Anzahl Seiten von Stack/Daten
- Anzahl Seiten von Bibliotheksprogrammcode
- Anzahl von unsauberen Seiten

Es gibt noch weitere nützliche Informationen zu den Prozessen, die Sie jetzt in der folgenden Tabellen aufgelistet finden. `$pid` ersetzen Sie bitte für eine echte Prozess-ID.

Verzeichniseintrag	Bedeutung
`/proc/$pid/status`	Darin finden Sie die formatierten Informationen, die Sie ebenfalls mit `/proc/$pid/stat` (allerdings nicht formatiert) ermitteln können. Dies sind Informationen wie die Prozess-ID, die reale und effektive User- und Group-ID, Speicherverbrauch, Bitmasken usw.
`/proc/$pid/stat`	Siehe `/proc/$pid/status`. Dies ist eine etwas kompaktere Datei, die sich besser als »status« mit `sscanf()` oder `fscanf()` verwenden lässt.
`/proc/$pid/cwd`	Ein symbolischer Link zum aktuellen Arbeitsverzeichnis des Prozesses
`/proc/$pid/exe`	Ein Verweis auf die ausführbare Programmdatei
`/proc/$pid/root`	Ein Link zum Root-Verzeichnis, das als Wurzelverzeichnis für den Prozess gilt (siehe `chroot()`)
`/proc/$pid/maps`	Beinhaltet Speicher-Mappings zu den verschiedenen laufenden Dateien und Bibliotheken, die mit diesem Prozess zusammenhängen. Die Datei kann sehr lang werden, wenn ein umfangreicher Prozess ausgeführt wird.

Tabelle 4.2 Prozessinformationen in den individuellen PID-Verzeichnissen

4.4 Kernel-Informationen

Viele Einträge im `/proc`-Verzeichnis geben auch Informationen zum laufenden Kernel aus. Der Ort der Kernel-Informationen ist unterschiedlich, viele liegen im `/proc`-Verzeichnis selbst und weitere in den Unterverzeichnissen wie `/proc/sys` oder `/proc/sys/kernel`.

Hierzu einige häufig benötigte Informationen, die in viele Anwendungen integriert sind:

```
$ cat /proc/sys/kernel/ostype
Linux
$ cat /proc/sys/kernel/osrelease
2.6.27-7-generic
$ cat /proc/sys/kernel/version
#1 SMP Tue Jan 4 19:33:20 UTC 2009
$ cat /proc/sys/kernel/ctrl-alt-del
0
```

Zuerst wurde abgefragt, was für ein Betriebssystem genau hier läuft (`ostype`), dann die Version des Kernels (`osrelease`) und wann dieser Kernel kompiliert wurde (`version`). Bei der letzten Abfrage wird überprüft, ob `ctrl-alt-del` gesetzt ist. Damit können Sie beeinflussen, ob *init* bei der Tastenkombination Ctrl+Alt+Del eine Aktion ausführen soll. Steht »`ctrl-alt-del`« auf 1, wird bei Tastendruck sofort ein BIOS-Reboot angeordnet. Steht »`ctrl-alt-del`« allerdings auf 0, wird *init* dazu angehalten, das System ordentlich herunterzufahren Wollen Sie diesen Parameter verändern, müssen Sie sich schnell als Superuser darstellen:

```
$ su
Password:********
# echo 1 >> /proc/sys/kernel/ctrl-alt-del
# exit
exit
$ cat /proc/sys/kernel/ctrl-alt-del
1
```

Ein weiteres Beispiel. Sie haben eine CD-ROM eingelegt und gemountet. Nachdem Sie die Daten von der CD-ROM gelesen haben, werfen Sie die CD-ROM ordnungsgemäß mit *umount* wieder aus und fahren anschließend das System herunter. Jetzt benötigen Sie aber die CD-ROM für einen Bekannten. Somit müssen Sie den PC leider nochmals anschalten. Wäre doch nett, wenn beim Ausgeben der CD-ROM die CD ausgeworfen wird. Für solche Fälle ist der Eintrag in `/proc/sys/dev/cdrom/autoeject` zuständig. Der Wert ist mit der Voreinstellung 0 versehen. Folgendermaßen können Sie dabei vorgehen, um den Wert auf 1 zu setzen:

```
$ mount /media/cdrom
$ cat /proc/sys/dev/cdrom/autoeject
0
$ su
Password:********
# echo 1 >> /proc/sys/dev/cdrom/autoeject
# exit
exit
$ cat /proc/sys/dev/cdrom/autoeject
1
$ umount /media/cdrom
```

Beim Aushängen des CD-ROM-Laufwerkes müsste jetzt die CD ausgeworfen werden. Dieses nette Feature zeigt allerdings schon recht früh seine Schattenseiten, z. B. wenn die Hardwareerkennung oder das Installationsprogramm das Device mehrmals öffnet und schließt.

Besonders fies auf engem Raum, wo Ihr CD-Laufwerk leiden wird, wenn es sich nicht vollständig öffnen kann.

Sie hacken übrigens damit nicht im System herum. Viele dieser Kernel-Parameter wurden bewusst so implementiert, damit Sie den Bedürfnissen der Anwender angepasst werden können, um ein höchstes Maß an Flexibilität zu erreichen. Da dabei in der Regel Superuser-Rechte benötigt werden, sind diese Einstellungen meistens für den Systemadministrator vorgesehen. Daher macht es relativ wenig Sinn, wenn Sie in Ihrer Anwendung, die Sie schreiben, versuchen, die Kernel-Parameter zu verändern. Dies würde bedeuten, dass Ihre Anwendung im Superuser-Modus laufen müsste. Bedenken Sie, dass dies nicht immer möglich ist – und auch nicht sein sollte.

Für administrative Zwecke soll hierfür jedoch eine solche Anwendung erstellt werden. Aber wie schon erwähnt, Sie benötigen Superuser- Rechte (bzw. root-Rechte) für dieses Programm, sollten Sie etwas verändern wollen. Versucht der normale Anwender, etwas zu verändern, bekommt er ein »Permission denied« zurückgegeben. Hier das Beispiel mit anschließender Erläuterung.

```
/* kernelinf.c */
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h.>
#include <string.h>
#define BUF 4096

enum { EJECT, FILE_MAX, SHARED_MAX  };
enum { CDINFO, OS, RELEASE, VERSION };

static const char *sys[] = {
  /* Bei umount CD auswerfen */
  "/proc/sys/dev/cdrom/autoeject",
  /* max. Anzahl geöffneter Dateien pro Prozess */
  "/proc/sys/fs/file-max",
  /*max. Größe geteilter Speicher (Shared Memory) */
  "/proc/sys/kernel/shmmax"
};

static const char *info[] = {
  "/proc/sys/dev/cdrom/info",   /* Infos zur CD-ROM        */
  "/proc/sys/kernel/ostype",    /* Welches Betriebssystem  */
  "/proc/sys/kernel/osrelease"  /* Kernel-Version          */
  "/proc/sys/kernel/version"    /* Kernel von wann         */
};

static char *get_info (const char *inf) {
  FILE *fp;
  static char buffer[BUF];
```

```
  size_t bytes_read;

  fp = fopen (inf, "r");
  if (fp == NULL) {
    perror("fopen()");
    return NULL;   /* Fehler beim Öffnen */
  }
  bytes_read = fread (buffer, 1, sizeof (buffer), fp);
  fclose (fp);
  if (bytes_read == 0 || bytes_read == sizeof (buffer))
    return NULL;
  buffer[bytes_read] = '\0';
  return buffer;
}


static void set_sys (const char *sys, unsigned long set) {
  FILE *fp;
  char buf[32];

  fp = fopen (sys, "w");
  if (fp == NULL) {
    perror ("fopen()");
    printf ("Weiter mit ENTER\n");
    getchar ();
    return;
  }
  sprintf(buf, "%ld", set);
  fprintf (fp, "%s", buf);
  fclose (fp);
  return;
}

int main (int argc, char **argv) {
  int auswahl;
  unsigned int file_max;
  unsigned long shared_max;

  do {
    printf ("Aktueller Zustand\n");
    printf ("Betriebssystem : %s", get_info (info[OS]));
    printf ("Kernel-Version  : %s",
       get_info (info[RELEASE]));
    printf ("Datum           : %s",
       get_info (info[VERSION]));
    printf ("-----------------------------------\n");
    printf ("Verändern können Sie Folgendes ...\n");
    printf ("-0- Bei \"umount\"  CD auswerfen"
            " Aktuell:%s", get_info (sys[EJECT]));
    printf ("-1- Max. Anzahl geöffneter Dateien pro"
```

```
            " Prozess    Aktuell:%s",
               get_info (sys[FILE_MAX]));
    printf ("-2- Max. Größe des geteilten Speichers"
            " (KB)     Aktuell %s",
               get_info (sys[SHARED_MAX]));
    printf ("-------------------------------------\n");
    printf ("Informationen bekommen Sie zu ...\n");
    printf ("-3- CD-ROM\n");
    printf ("-------------------------------------\n");
    printf ("-4- ENDE\n");
    printf ("Ihre Auswahl bitte (0-4) : ");

    do { scanf ("%d", &auswahl); } while (getchar ()!='\n');
    switch (auswahl) {
    case EJECT:
      if (strncmp ("0", get_info (sys[EJECT]),1) == 0)
        set_sys (sys[EJECT], 1); /* Setzen      */
      else                       /* Zurücksetzen */
        set_sys (sys[EJECT], 0);
      break;
    case FILE_MAX:
      printf ("Welcher Wert soll gesetzt werden : ");
      do{ scanf("%d", &file_max); } while (getchar()!='\n');
      set_sys (sys[FILE_MAX], file_max);
      break;
    case SHARED_MAX:
      printf ("Welcher Wert soll gesetzt werden : ");
      do { scanf("%ld", &shared_max); }
      while (getchar()!='\n');
      set_sys (sys[SHARED_MAX], shared_max);
      break;
    case 3:
      printf ("%s", get_info (info[CDINFO]));
      printf ("Weiter mit ENTER\n");
      getchar ();
      break;
    case 4:
      printf("Programm wird beendet\n");
      break;
    default:
      printf ("Unbekannte Eingabe\n");
    }
  } while (auswahl != 4);
  return EXIT_SUCCESS;
}
```

Das Programm bei der Ausführung:

```
$ gcc -o kernelinf kernelinf.c
$ ./kernelinf
Aktueller Zustand
```

```
Betriebssystem : Linux
Kernelversion  : 2.6.27-7-generic
Datum          : #1 SMP Tue Jan 4 19:33:20 UTC 2009
-----------------------------------
Verändern können Sie folgendes ...
-0- Bei "umount"  CD auswerfen Aktuell:0
-1- Max. Anzahl geöffneter Datei pro Prozess  Aktuell:49594
-2- Max. Größe des geteiltem Speicher (KB)    Aktuell 33554432
-----------------------------------
Informationen bekommen Sie zu ...
-3- CD-ROM
-----------------------------------
-4- ENDE
Ihre Auswahl bitte (0-4) :
```

Neben dem Auswerfen der CD-ROM bei *umount* können hier noch die maximale Anzahl gleichzeitig geöffneter Dateien pro Prozess und die maximale Größe des geteilten Speichers (Shared Memory) verändert werden. Diese Veränderungen können aber wie bereits erwähnt nur mit speziellen Rechten vorgenommen werden. Informationen zum CD-ROM-Laufwerk, was es alles kann, können wieder für alle Anwender aufgelistet werden.

Bitte beachten Sie, dass diese Art von Veränderungen der Kernel-Parameter nur für den laufenden Betrieb gültig ist. Sobald Sie das System herunterfahren und wieder starten, sind die Werte wieder auf ihren Ursprungszustand gestellt. Wollen Sie eine dauerhafte Veränderung bewirken, müssen Sie sich die Datei `sysctl.conf` im Verzeichnis `/etc` vornehmen. Diese Datei können Sie auch mit dem gleichnamigen Kommando `sysctl` verändern. Wollen Sie z. B. mit `sysctl` autoeject verändern, gehen Sie wie folgt vor (root-Rechte):

```
# sysctl -w dev.cdrom.autoeject=0
dev.cdrom.autoeject = 0
# sysctl -w dev.cdrom.autoeject=1
dev.cdrom.autoeject = 1
```

Auf die Angaben von `/proc/sys` können Sie dabei verzichten, und anstelle eines Slashs wird ein Punkt verwendet.

Wollen Sie wissen, mit welchem Befehl der Kernel beim Booten gestartet wurde, können Sie ihn mit `cmdline` erfragen:

```
$ cat /proc/cmdline
root=/dev/hda6 vga=0x0314  hdc=ide-scsi hdclun=0 splash=silent
```

Oder Linux 2.6 mit LILO 22.3.4:

```
auto BOOT_IMAGE=2.6.4-HX ro root=301
```

Zu den verschiedenen Bootoptionen sei hier der Verweis auf die *bootparam(7)* Manual Page gegeben. Es gibt noch weitaus mehr zu den Kernel-Informationen zu sagen, aber das würde den Rahmen des Buches bei weitem sprengen und am Thema der Linux-Programmierung vorbeigehen.

4.4.1 /proc/locks

Mit dieser Datei werden alle Dateien angezeigt, die im Augenblick vom Kernel gesperrt werden (vgl. Kapitel 2 über `flock()` und `lockf()`). Diese Ausgabe enthält interne Kernel-Debugging-Daten und kann daher logischerweise je nach System stark variieren.

```
1: FLOCK ADVISORY WRITE 807 03:05:308731 0 EOF c2ac0 c0248 c2a220
2: POSIX ADVISORY WRITE 708 03:05:308720 0 EOF c2a1c c2ac4 c02548
```

In der ersten Spalte besitzt jeder Lock eine einmalige Zahl gefolgt vom Typ des Locks. Mögliche Ausgaben sind hier FLOCK (meist bei älteren UNIX-Datei-Locks mit dem Systemaufruf `flock()`) und POSIX (bei neueren POSIX-Locks mit dem Systemaufruf `lockf()`).

Der Wert ADVISORY der dritten Spalte bedeutet, dass ein weiterer Datenzugriff für andere Benutzer möglich ist, aber keine weiteren Lock-Versuche mehr erlaubt sind. Ein anderer Wert hierfür wäre MANDATORY, was bedeutet, dass hier keine Datenzugriffe mehr möglich sind, solange der Lock vorhanden ist. In der vierten Spalte befinden sich die Lese- oder Schreibrechte (READ, WRITE) des Eigentümers. Die fünfte Spalte enthält die PID des Lock-Eigentümers. Die ID der gelockten Datei finden Sie in der sechsten Spalte mit folgendem Format:

```
MAJOR-DEVICE:MINOR-DEVICE:INODE-NUMBER
```

Die Spalten sechs und sieben zeigen Anfang und Ende der gelockten Region. Im Beispiel gilt die Sperre vom Anfang 0 bis zum Ende (EOF) der Datei. Die letzten Spalten zeigen auf Kernel-interne Datenstrukturen für spezielle Debugging-Funktionen.

4.4.2 /proc/modules

In `/proc/modules` finden Sie eine Liste von allen Modulen, die vom System geladen wurden. Auch hierbei hängt die Ausgabe von den Einstellungen des Systems ab.

```
$ cat /proc/modules
isofs 40100 1 - Live 0xe0c36000
udf 88356 0 - Live 0xe0c94000
crc_itu_t 10112 1 udf, Live 0xe0bff000
btusb 19736 0 - Live 0xe0c23000
nls_iso8859_1 12032 1 - Live 0xe0c1f000
nls_cp437 13696 1 - Live 0xe0c09000
vfat 18816 1 - Live 0xe0c03000
fat 57376 1 vfat, Live 0xe0c0f000
ipv6 263972 15 - Live 0xe0c52000
....
....
```

In der ersten Spalte befindet sich jeweils der Name des geladenen Moduls, gefolgt von der Speichergröße in Bytes. In der dritten Spalte wird angezeigt, wie oft das Modul gerade benutzt wird (*usage count*). In der letzten Spalte finden Sie die Module, die benötigt werden, damit andere Module funktionieren. Zum Beispiel benötigt das Modul *cdrom* das Modul *ide-cd*, um ordentlich ausgeführt zu werden. »`autoclean`« gibt an, ob sich das Modul nach einer gewissen Zeit selbst deaktiviert, »`unused`«, dass es nicht benutzt wird.

4.5 Filesysteme

Weitere Informationen, die Sie im `/proc`-Verzeichnis finden, sind die über verschiedene Filesysteme und welche dabei eingehängt (»gemountet«) sind. Das Verzeichnis `/proc/filesystems` z. B. listet alle Filesysteme auf, die dem Kernel bekannt sind und womit dieser arbeiten kann. Einige davon sind intern und können nicht gemountet werden, auch wenn sie hier aufgelistet werden. Dazu zählt z. B. *pipefs*; wobei diese Auflistung nicht vollständig ist. Denn es gibt noch viele Filesysteme, die nachgeladen werden können und gerade nicht aktiv sind. Andere wiederum sind nur statisch gelinkt und werden erst bei Bedarf aktiviert.

4.5.1 /proc/mounts

Was alles gerade eingehängt ist, finden Sie in `/proc/mounts` verzeichnet, z. B.:

```
rootfs / rootfs rw 0 0
/dev/root / reiserfs rw 0 0
proc /proc proc rw 0 0
sysfs /sys sysfs rw 0 0
devpts /dev/pts devpts rw 0 0
tmpfs /dev/shm tmpfs rw 0 0
/dev/hda2 /C vfat rw,nodiratime,fmask=0022,dmask=0022,codepage=cp437 0 0
usbdevfs /proc/bus/usb usbdevfs rw 0 0
/dev/hdb /F iso9660 ro,nosuid,nodev 0 0
```

Die Ausgabe von `/proc/mount` entspricht (fast) exakt der Ausgabe von `/etc/mtab`, nur dass `/etc/mtab` von *mount* verwaltet wird und somit nur Dateisysteme auflistet, die mit */bin/mount* eingehängt worden sind, nicht jedoch z. B. der Systemcall `mount()`.

In der ersten Spalte einer jeden Zeile finden Sie das Gerät und in der zweiten Spalte den dazugehörenden Mountpoint. Der Dateisystemtyp wird in der dritten Spalte aufgelistet. Ob darauf nur lesend (`ro`) oder auch schreibend (`rw`) zugegriffen werden kann, wird – nebst anderen möglichen Optionen – in der vierten Spalte angezeigt. Die letzten beiden Spalten sind Dummy-Werte, damit das `/proc/mount`-Format exakt dem von `/etc/mtab` entspricht.

4.6 Weiterführendes

Zum `/proc`-Verzeichnis könnten noch viele Seiten geschrieben werden, aber irgendwo muss Schluss sein. Sie wissen jetzt, wenn Sie bestimmte Informationen für Ihr Programm oder auch zur Administration benötigen, wo und wie Sie an diese Informationen herankommen. Weitere Informationen zum `/proc`-Verzeichnis finden Sie auf der Manual Page *»man 5 proc«*.

Dass Linux frei im Quellcode ist (Open Source), weiß – denke ich – schon jeder. Daher hierzu einige Anwendungen, deren Quellcode zu studieren sinnvoll erscheint, da diese kräftig Gebrauch vom `/proc`-Verzeichnis machen.

Anwendung	Beschreibung
`mount`	Informationen über gemountete Datenträger
`ps`	Informationen über Prozesse
`top`	Auslastung der CPU
`xload`	Durchschnittliche Auslastung des Systems
`xosview`	Durchschnittliche Auslastung des Systems, der CPU, des Speicherbedarf, Interrupts ...

Tabelle 4.3 Anwendungen, die vom /proc-Verzeichnis profitieren

Damit auch einfach autorisierte Prozesse Zugriff auf die Hardware-Komponenten des Systems haben, können die Devices (Gerätedateien) verwendet werden.

5 Devices – eine einfache Verbindung zur Hardware

Damit das Betriebssystem mit der Hardware kommunizieren kann (vereinfacht ausgedrückt), wird in Linux (wie in den meisten anderen Betriebssystemen auch) ein Gerätetreiber dafür verwendet. Unter Linux ist dieser Treiber immer ein Teil des Kernels und kann entweder statisch hinzugelinkt oder auf Anfrage als Kernel-Modul nachgeladen werden. Da ein Gerätetreiber ein Teil des Kernels ist, kann aus Sicherheitsgründen nicht direkt darauf mit einem normal laufenden Prozess zugegriffen werden. Um aber jetzt dennoch auf Hardware-Komponenten wie u. a. Festplatte, CD-ROM, Soundkarte oder der seriellen bzw. parallelen Schnittstelle zugreifen zu können, wurde unter UNIX ein Mechanismus eingeführt, womit recht problemlos mit einem Gerätetreiber über eine Hardware-Gerätedatei kommuniziert werden kann. Diese »Datei« können Sie mit einem gewöhnlichen Prozess öffnen und auf sie lesend und schreibend zugreifen – sprich, Sie können über normale Datei-E/A-Operationen mit dem Gerätetreiber dank dieser »Datei« so kommunizieren, als handle es sich um eine gewöhnliche Datei.

5.1 Die Gerätedateitypen

Die Einträge der Gerätedateien – auch als Devices bekannt – finden Sie im Verzeichnis `/dev`. Ein `ls -l /dev` verschafft Ihnen einen ersten Überblick und stiftet vielleicht zugleich auch ein wenig Verwirrung über diese Gerätedateien, die es zu entflechten gilt. Es wird hierbei zwischen zwei Devices-Typen unterschieden – zu erkennen am ersten Buchstaben der Auflistung der Gerätedatei(en) (`b` oder `c`):

- *character device* (`c`) – hierbei handelt es sich um Hardwaregeräte, mit der eine fortlaufende Folge von Datenbytes gelesen oder geschrieben wird. Dies sind Hardware-Komponenten wie Soundkarte, Tapes, Terminalgeräte, serielle oder parallele Schnittstelle(n), Maus, USB-Schnittstelle usw.
- *block device* (`b`) – Blockgeräte sind Devices, mit denen feste ganze Blöcke (anstatt Bytes) gelesen oder geschrieben werden können. Dabei wird erst ein Puffer aufgefüllt, bevor der ganze Datenblock an den Treiber gesendet wird. Daher sind blockorientierte Geräte auch meistens etwas schneller als zeichenorientierte, da gewartet werden muss, bis ein Block voll ist. Zu den Blockgeräten gehören z. B. die Festplatte, Diskette oder das CD-ROM-/

DVD-Laufwerk. Ein zeichenorientiertes Gerät hat mehr Aufwand, weil nach jedem `read`/`write` bereits eine Art *flush* stattfindet.

Hinweis

Es kann gefährlich werden, wenn Sie direkt auf Blockgeräte wie die Festplatte schreibend zugreifen. Sie müssen bedenken, dass alle Operationen wie bei einer Datei ausgeführt werden können; also auch Dinge wie `lseek()`, um den Schreibzeiger zu positionieren.

Wenn Sie hier nicht genau wissen, was Sie tun, dann lassen Sie die Finger davon. Die Folge kann sein, dass Sie den kompletten Inhalt Ihrer Festplatte unbrauchbar machen – sprich, auch Linux. Zwar verbieten die meisten Linux-Systeme einem normalen Prozess solch einen Zugriff – aber aus experimentellen Gründen versucht man es (erfahrungsgemäß) gerne als `root`. Bei Blockgeräten sollten Sie also immer zweimal nachdenken, was Sie tun!

5.2 Die Gerätedateinummern

Für Sie als Programmierer und auch für den Anwender stellt eine solche Gerätedatei im Verzeichnis `/dev` eine Schnittstelle zum eigentlichen Treiber der dazugehörenden Gerätedatei dar. Jegliche Aktion mit der Gerätedatei (schreiben, lesen, teilweise auch steuern) wird dann an den zuständigen Treiber weitergeleitet.

Damit eine Gerätedatei weiß, welcher Treiber für sie zuständig ist, enthält diese eine Nummer – die *major* Nummer – die Treibernummer des Systems. Zum Beispiel ist die *major* Nummer der ersten IDE-Festplatte 3:

```
---[Linux]---
$ ls -l /dev/hda /dev/hda1
brw-rw----    1 root     disk   3,   0 2003-03-14 14:07 /dev/hda
brw-rw----    1 root     disk   3,   1 2003-03-14 14:07 /dev/hda1
---[*BSD]---
$ ls -l /dev/ad0 /dev/ad0s1
crw-r----- 1 root operator 116, 0x00010002 Aug 19 /dev/ad0
crw-r----- 1 root operator 116, 0x00020002 Aug 19 /dev/ad0s1
```

Wenn Sie sich das komplette `/dev`-Verzeichnis auflisten lassen, fällt Ihnen sicherlich auch auf, dass viele Gerätedateien über die gleiche *major* Nummer verfügen. Dabei interagieren die Gerätedateien über den gleichen Treiber mit dem Betriebssystem. Dabei muss es nicht sein, dass eine andere *major* Nummer ein anderes Gerät bezeichnet. Bestes Beispiel: Die serielle Schnittstelle verfügt über mehrere Gerätenummern – obwohl es sich um ein und dieselbe Schnittstelle handelt. Der Unterschied ist also nicht das Gerät, sondern, wie schon erwähnt, der Treiber. Mit einem Treiber wird z. B. auf den Terminal zugegriffen, während mit dem anderen Treiber auf das Modem zugegriffen wird.

Jetzt haben Sie zwar für ein Gerät mit der *major* Nummer einen Treiber, aber häufig gibt es von einem Gerät der gleichen Bauart mehrere, die im Einsatz sind. Bestes Beispiel sind meh-

rere Festplatten oder Partitionen in Ihrem Rechner. Da die Festplatten zwar alle denselben Treiber verwenden, müssen diese dennoch voneinander unterschieden werden. Um dies zu realisieren, wurde die *minor* Nummer gleich mit eingeführt. Die *minor* Nummer befindet sich gleich rechts neben der *major* Nummer. Anhand dieser Nummer weiß der Treiber jetzt, welche Festplatte oder Partition er bedienen soll.

5.3 Zugriff auf die Gerätedateien

Wie schon erwähnt, erfolgt der Zugriff auf Gerätedateien (dank Integration im Dateisystem) über die gewöhnlichen Datei-E/A-Funktionen. Natürlich lässt sich dies auch via Shell mit einer Dateiumleitung erledigen. Wollen Sie z. B. einen Text an die Gerätedatei des Druckers senden, so gehen Sie einfach folgendermaßen vor (entsprechende Zugriffsrechte vorausgesetzt):

```
$ cat textdatei.txt > /dev/lp0
```

Selbiges in einem C-Programm angewandt, sieht so aus:

```
int fd;
fd = open("/dev/lp0", O_WRONLY);
if(fd >= 0)
   write(fd, buf, buf_size);
close(fd);
```

Zum Öffnen und Schließen einer Gerätedatei werden somit `open()` und `close()` und zum Lesen und Schreiben die Funktionen `read()` und `write()` verwendet.

Mit der Möglichkeit, bestimmte Gerätedateien zu öffnen und schreibend bzw. lesend darauf zuzugreifen, können Sie nur einen Bruchteil dessen machen, wozu der Treiber imstande ist. Denn neben dem Schreiben und Lesen ist es auch möglich, ein Gerät in seiner spezifischen Art zu steuern. Dies können Sie mit dem Funktionsaufruf `ioctl()` (*Input-Output-Control*) erledigen. Erst damit können Sie den Treiber unter Linux direkt ansprechen. Hierfür müssen Sie allerdings den entsprechenden Befehlssatz für den entsprechenden Treiber kennen. Zuerst die Syntax zu `ioctl()`:

```
#include <sys/ioctl.h>

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

Als ersten Parameter geben Sie den geöffneten Filedeskriptor an, mit dem Sie eine Gerätedatei geöffnet haben, und der zweite Parameter ist der entsprechende Befehl, der natürlich von der Gerätedatei abhängt, die geöffnet wurde. Sie können z. B. nicht für eine Festplatte den Befehl zum Auswerfen einer CD-ROM verwenden (groteskes Beispiel, es macht aber deutlich, worauf ich hinauswill). Im Fehlerfall liefert `ioctl()` -1, ansonsten bei Erfolg 0 zurück.

Beachten Sie allerdings, dass es sich hier um keinen Standard handelt. Argumente, Rückgabewerte und Semantik von `ioctl()` variieren je nach dem entsprechenden Gerätetreiber (der Aufruf wird als ein Allheilmittel für alle Operationen benutzt, die nicht sauber in das UNIX-Stream-E/A-Modell passen). Siehe `ioctl_list(2)` für eine Liste von vielen der bekannten `ioctl`-Aufrufe. Die Funktion `ioctl()` erschien in Version 7 von AT&T UNIX.

Die entsprechenden Befehlssätze für eine bestimmte Gerätedatei, um damit den entsprechenden Treiber zu steuern, finden Sie unter *man ioctl_list* – was allerdings mittlerweile keine vollständige Liste mehr darstellt. Wie Sie diese Befehle anwenden, werde ich Ihnen anhand eines Beispiels mit dem CD-ROM-Laufwerk demonstrieren.

5.4 Gerätenamen

Die Gerätenamen sind mittlerweile über die Systeme hinweg recht ähnlich geworden. Mittlerweile besitzen viele Einträge (die zum Standard einer normalen PC-Ausrüstung gehören) zum CD-ROM-Laufwerk, Diskettenlaufwerk, Modem oder zur Maus einen symbolischen Link mit einem aussagekräftigen Namen. Zur Übersicht sollen hier dennoch einige block- und zeichenorientierte Gerätedateien und deren Bedeutung aufgelistet werden:

Device	Name	Major	Minor
erstes Diskettenlaufwerk	`/dev/fd0`	2	0
zweites Diskettenlaufwerk	`/dev/fd1`	2	1
erster IDE-Controller (Hauptgerät)	`/dev/hda` (`/dev/ad0`)	3	0
erster IDE-Controller (erste Partition) (Hauptgerät)	`/dev/hda1` (`/dev/ad0s1a`)	3	1
erster IDE-Controller (zweites Gerät)	`/dev/hdb` (`/dev/ad1`)	3	64
erster IDE-Controller (erste Partition) (zweites Gerät)	`/dev/hdb1` (`/dev/ad1s1a`)	3	65
zweiter IDE-Controller (Hauptgerät)	`/dev/hdc` (`/dev/ad2`)	22	0
zweiter IDE-Controller (zweites Gerät)	`/dev/hdd` (`/dev/ad3`)	22	64
erstes SCSI-Laufwerk	`/dev/sda` (`/dev/da0`)	8	0
erstes SCSI-Laufwerk (erste Partition)	`/dev/sda1` (`/dev/da0s1a`)	8	1
zweites SCSI-Laufwerk	`/dev/sdb` (`/dev/da1`)	8	16
zweites SCSI-Laufwerk (erste Partition)	`/dev/sdb1` (`/dev/da1s1a`)	8	17
erstes SCSI-CDROM Laufwerk	`/dev/sr0` (`/dev/cd0`)	11	0
zweites SCSI-CDROM Laufwerk	`/dev/sr1` (`/dev/cd1`)	11	1

Tabelle 5.1 Blockorientierte Gerätedateien

Hinweis

Die Angaben in den Klammern stellen die Bezeichnungen der Gerätenamen unter FreeBSD da. Wobei FreeBSD ein anderes System verwendet als z. B. Linux. Hier gibt es in den Partitionen zusätzlich noch so genannte Slices. So bedeutet die Angabe von `/dev/ad0s1a` => 1. Partition im 1. Slice der ersten IDE-Platte oder `/dev/ad0s1b` => 2. Partition im 1. Slice der ersten IDE-Platte. Bei SCSI-Laufwerken ist das ähnlich, wie. `/dev/da1s2d` => 4. Partition im 2. Slice der 2. SCSI-(Wechsel-)Platte.

Device	Name	Major	Minor
1. Parallelport	`/dev/lp0` oder `/dev/par0`	6	0
2. Parallelport (falls vorhanden)	`/dev/lp1` oder `/dev/par1`	6	1
High-Level-Parallelport	`/dev/parport0`	99	0
erste serielle Schnittstelle	`/dev/ttyS0`	4	64
zweite serielle Schnittstelle	`/dev/ttyS1`	5	65
IDE-Tape-Laufwerk	`/dev/ht0`	37	0
erstes SCSI-Tape-Laufwerk	`/dev/st0`	9	0
Systemkonsole	`/dev/console`	5	1
erster viruteller Terminal	`/dev/tty1`	4	1
1. Soundkarte	`/dev/dsp0`	14	

Tabelle 5.2 Zeichenorientierte Gerätedateien

5.5 Spezielle Gerätedateien

Es befinden sich auch einige spezielle Einträge im `/dev`-Verzeichnis, die mit keiner Hardware verbunden sind. Hier einige der bekanntesten:

- `/dev/null` – wird gerne verwendet, um unerwünschte Ausgaben auf die Standardausgabe zu beseitigen. Dabei wird einfach die Ausgabe mit `programmname > /dev/null` umgeleitet. Kurz: wird als Müllschlucker verwendet :-)
- `/dev/zero` – dabei handelt es sich um eine sehr lange Gerätedatei gefüllt mit 0 Bytes. `/dev/zero` wird besonders gerne verwendet als eine andere Art der Speicherreservierung mit `mmap()`. Damit ist es möglich, zwischen zwei verwandten Prozessen zu kommunizieren. Der Vorteil an dieser Methode ...

  ```
  if((fd=open("/dev/zero", O_RDWR)) < 0) {
    /* Fehler */
  }
  memoin=mmap(0,8192 ... );
  ```

  ... ist der, dass keine neue Datei mit dem Memory-Mapped-Aufruf `mmap()` erstellt werden muss (mehr zum Systemaufruf `mmap()` finden Sie im Anhang bei der Funktionsreferenz).

  Folgendes geschieht durch diesen Aufruf und ist zu beachten:

  - Der Speicherbereich, der eingerichtet wird, ist namenlos.
  - Die Größe des Speicherbereichs wird mit dem zweiten Argument übergeben.

- Dieser Speicherbereich wird mit 0 initialisiert.
- Und WICHTIG! Wenn verwandte Prozesse ebenfalls auf diesen Speicherbereich zugreifen sollen, so müssen Sie das Flag `MAP_SHARED` angeben.

Natürlich kann `/dev/zero` auch als 2. Müllschlucker (im Falle des Schreibens) oder – als »sehr lange Gerätedatei« – als eine Gerätedatei, die unendlich viele binäre Nullen zurückgibt, verwendet werden.

- `/dev/full` – wird verwendet, um ein Programm beim Abspeichern einer Datei darauf zu testen, was passiert, wenn kein Platz mehr zum Speichern vorhanden ist. Jeder Kopiervorgang in die Datei `/dev/full` setzt `errno` auf `ENOSPC`, z. B.:

```
$ cp datei /dev/full
cp: Schreiben von "/dev/full":
Auf dem Gerät ist kein Speicherplatz mehr verfügbar
```

- `/dev/random` und `/dev/urandom` – wird zum Erzeugen von Zufallszahlen verwendet. `/dev/random` besteht aus reinen Zufallsquellen (also Maus, ansatzweise Tastatur, CPU-Temperatur oder Mainboard-Spannung etc.), wohingegen `/dev/urandom` wegen der Dauerverfügbarkeit auch aus weniger sicheren Datenquellen bezogen wird. `/dev/random` blockiert im Gegensatz zu `/dev/urandom` ggf. den Prozess, wenn keine Aktion mit der Tastatur oder der Maus stattfindet. Mit `/dev/urandom` hingegen wird die Erzeugung einer Zufallszahl nicht blockiert, da im Falle keiner Aktion ein Algorithmus zur Erzeugung der Zufallszahl verwendet wird. Zum besseren Verständnis wenden Sie einfach `od` auf beide Gerätedateien wie jeweils folgt an:

```
$ od -t u1 /dev/random
$ od -t u1 /dev/urandom
```

- `/dev/pts/*` – virtuelle Konsolen, die durch das Öffnen von `/dev/ptmx` entstehen (z. B. über `xterm`s).

5.6 Gerätedateien in der Praxis einsetzen

Auf den folgenden Seiten will ich Ihnen nun zeigen, wie Sie die Gerätedatei in der Praxis einsetzen können; also neben dem Lesen und Schreiben auch das Steuern des Treibers durch die Gerätedatei. Das Ziel des Abschnitts ist es, Ihnen ein Gefühl dafür zu geben, wie Sie die entsprechenden Informationen ermitteln können, um den unübersichtlichen Haufen von Gerätedateien steuern zu können.

Als Beispiel wird das CD-ROM-Laufwerk verwendet, da zum einen fast jeder Leser so etwas besitzen dürfte, und zum anderen lässt es sich damit relativ leicht nachvollziehen, da hierfür keine speziellen Hardware-Kenntnisse nötig sind. Es soll hierbei ein Listing erstellt werden, womit Sie die Lade öffnen können, um eine Musik-CD einzulegen. Weiterhin soll es möglich sein, die Track-Informationen einzulesen und auch wieder auszugeben. Die Musik soll ebenfalls wiedergegeben werden, mitsamt der Anzeige, welcher Track gerade läuft und wo gerade gelesen wird (zeitmäßig gesehen).

Das Ganze hört sich schlimmer an, als es ist, und lässt sich relativ einfach mit den Treiberbefehlen der Gerätedatei über `ioctl()` steuern. Das CD-ROM-Laufwerk lässt sich gewöhnlich

von Haus aus über den Mountpoint `/dev/cdrom` oder auch `/media/cdrom` ansprechen. Wenn dies bei Ihnen nicht der Fall ist, wäre es sinnvoll, selbst einen symbolischen Link zum CD-ROM-Laufwerk zu legen. In der Tabelle zu den Gerätedateien können Sie das Verzeichnis für Ihr CD-ROM-Laufwerk herauslesen.

Außerdem benötigen Sie mindestens Leserechte auf das CD-ROM-Laufwerk. Sofern diese bei Ihnen nicht vorhanden sind, sollten Sie dieses Recht mittels `chmod a+r /dev/hdc` (`/dev/hdc/` sei hier der Pfad zur Gerätedatei) setzen oder von Ihrem Administrator setzen lassen.

Zuerst müssen Sie die Informationen zu den Treiberbefehlen des CD-ROM-Laufwerks suchen. Einen ersten Überblick hierzu können Sie sich, wie schon erwähnt, unter *man ioctl_list* verschaffen. Folgende Informationen können hierbei schon ermittelt werden:

```
$ man ioctl_list
...
// <include/linux/cdrom.h>
0x00005301  CDROMPAUSE          void
0x00005302  CDROMRESUME         void
0x00005303  CDROMPLAYMSF        const struct cdrom_msf *
0x00005304  CDROMPLAYTRKIND     const struct cdrom_ti *
0x00005305  CDROMREADTOCHDR     struct cdrom_tochdr *
0x00005306  CDROMREADTOCENTRY   struct cdrom_tocentry *
0x00005307  CDROMSTOP           void
0x00005308  CDROMSTART          void
0x00005309  CDROMEJECT          void
0x0000530A  CDROMVOLCTRL        const struct cdrom_volctrl *
0x0000530B  CDROMSUBCHNL        struct cdrom_subchnl *
0x0000530C  CDROMREADMODE2      const struct cdrom_msf *
...
0x00005312  CDROMRESET          void
0x00005313  CDROMVOLREAD        struct cdrom_volctrl *
0x00005314  CDROMREADRAW        const struct cdrom_msf *
0x00005315  CDROMREADCOOKED     const struct cdrom_msf *
0x00005316  CDROMSEEK           const struct cdrom_msf *
...
```

In der ersten Spalte finden Sie den Hexwert für den Treiber. Als Mensch liest man gerne, und daher finden Sie in der zweiten Spalte die symbolische Konstante dazu. In der dritten Spalte finden Sie außerdem den entsprechenden Datentyp oder die Struktur, worauf sich der Befehl bezieht – sprich das dritte Argument für die Funktion `ioctl()`. Dies ist schon ein guter Überblick. Aber um einen Blick in die Headerdatei für den Linux CD-ROM-Treiber kommen Sie nicht mehr herum, wenn Sie die einzelnen Strukturvariablen verwenden wollen. Wo Sie diese finden können, ist gleich zu Beginn bei der Manual Page mit `<include/linux/cdrom.h>` angegeben. Sie müssen also ins Verzeichnis Ihrer `include`-Dateien gehen. Meistens ist dies `/usr/include` oder `/usr/local/include`. Darin finden Sie jetzt außer den Befehlen noch alles, was Sie für die weitere Verwendung benötigen – vor allem auch eine kurze Beschreibung der einzelnen Befehle, was man ja bei *man ioctl_list* schmerzlich vermisst.

Hinweis

Das folgende Beispiel lässt sich nur mit einem Linux-Rechner übersetzen und ausführen. Sofern Sie ein ähnliches Beispiel auf Systemen wie SunOS, NetBSD, OpenBSD oder FreeBSD erstellen wollen, benötigen Sie hierzu die Headerdatei `sys/cdio.h` anstatt `linux/cdrom.h`.

Leider gibt es hierbei einige (geringfügige) Differenzen, so dass ich mir hier nicht mit einer bedingten Kompilierung behelfen kann. Natürlich finden Sie als BSD-Leser auf der Buch-CD ein entsprechendes Listing (`bsd_cdrom.c`), das auf allen BSD- und eventuell (ungetestet) auch auf SunOS-Systemen laufen sollte.

5.6.1 CD auswerfen und wieder schließen

Bevor Sie das CD-ROM-Laufwerk steuern können, müssen Sie die entsprechende Gerätedatei zum Lesen öffnen. Der Übersichtlichkeit halber wurden die einzelnen Aktionen in kleine Routinen gepackt, womit Sie diese jederzeit für Ihre Zwecke verwenden können. Am Ende des Kapitels finden Sie die einzelnen Routinen nochmals als komplettes Listing wieder.

```
...
#define CDROM "/dev/cdrom"
...
static int open_cdrom (void) {
   int fd = open (CDROM, O_RDONLY | O_NONBLOCK);
   if (fd == -1) {
      if (errno == ENOMEDIUM)
         printf ("Keine CD im Laufwerk!\n");
      else
         perror ("Fehler bei open()");
      exit (EXIT_FAILURE);
   }
   return fd;
}
```

Damit die Funktion `open()` nicht blockiert, wenn Sie mal keine CD im Laufwerk haben, muss das Flag `O_NONBLOCK` verwendet werden. Die Funktion gibt den Filedeskriptor `cdrom` für die anschließende weitere Steuerung mit `ioctl()` zurück.

Nachdem Sie die Gerätedatei zum Lesen geöffnet haben, können Sie die ersten Steuerbefehle ausführen. Hierzu werden erst einmal die einfachsten Kommandos, die keine weiteren Parameter in `ioctl()` mehr benötigen, verwendet, zum einen das Auswerfen der CD, zum anderen – falls technisch möglich – das Einfahren des Caddys. Gerade bei Laptops ist das Zurückfahren des CD-Fachs nicht möglich. Dafür bekommen Sie – wenn technisch nicht möglich – eine Meldung zurück, dies doch selbst zu tun.

```
static void open_tray (int cdrom) {
   if (ioctl (cdrom, CDROMEJECT) == -1) {
      perror ("Eject yourself");
      exit (EXIT_FAILURE);
   }
```

```
}

/* Funktioniert nicht überall */
static void close_tray (int cdrom) {
   if (ioctl (cdrom, CDROMCLOSETRAY) == -1) {
      printf ("CD-Tray bitte von Hand schließen\n");
   }
}
```

5.6.2 CD-ROM-Fähigkeiten

Benötigen Sie die Fähigkeiten des CD-ROM-Laufwerks, dann können Sie dies mithilfe des Kommandos `CDROM_GET_CAPABILITY` ermitteln. In diesem Fall liefert `ioctl()` als Rückgabewert einen Integer zurück, dessen einzelne Bit-Werte es gilt auszulesen. In unserem Beispiel ermitteln wir zwar nur, um was für ein Laufwerk es sich handelt – sprich CD-R, CD-RW, DVD etc. –, aber Sie können gerne weitere Fähigkeiten, die alle mit dem Präfix `CDC_` beginnen, zur Überprüfung verwenden und auswerten. Sie müssen lediglich den Rückgabewert von `ioctl()` mit dem bitweisen UND-Operator und der entsprechenden symbolischen `CDC_`-Konstante als Bedingung verwenden. Ist die Bedingung wahr, dann trifft Entsprechendes zu.

```
static void capability_cdrom (int cdrom) {
   const char *j[] = {"nein", "ja"};
   int caps = ioctl (cdrom, CDROM_GET_CAPABILITY);
   if (caps == -1) {
      perror ("CDROM_GET_CAPABILITY");
      return;
   }
   printf ("CDROM-Fähigkeiten:\n"
      "\tCD-R    : %s\n"
      "\tCD-RW   : %s\n"
      "\tDVD     : %s\n"
      "\tDVD-R   : %s\n"
      "\tDVD-RAM : %s\n",
      j[!!(caps & CDC_CD_R)],
      j[!!(caps & CDC_CD_RW)],
      j[!!(caps & CDC_DVD)],
      j[!!(caps & CDC_DVD_R)],
      j[!!(caps & CDC_DVD_RAM)]);
}
```

5.6.3 Audio-CD abspielen – komplett und einzelne Tracks – Pause, Fortfahren und Stopp

Um das Inhaltsverzeichnis der CD einzulesen, können Sie die Kommandos `CDROMTOCHDR` und `CDROMREADTOCENTRY` verwenden (TOC = Table Of Contents). Verwenden Sie `ioctl()` mit `CDROMTOCHDR`, wird als dritter Parameter eine Adresse vom Typ `struct cdrom_tochdr` erwartet. Diese Struktur sieht in der Headerdatei `cdrom.h` wie folgt aus:

```
/* Struktur wird bei CDROMREADTOCHDR mit ioctl() erwartet */
struct cdrom_tochdr {
   __u8   cdth_trk0;   /* start track */
   __u8   cdth_trk1;   /* end track */
};
```

Wenn Sie das Kommando mit `ioctl()` verwenden, wird in der Struktur `cdrom_tochdr` der erste (`cdth_trk0`) und der letzte (`cdth_trk1`) Track eingetragen. Somit können Sie mit dem Auslesen des letzten Tracks schon ermitteln, wie viele Tracks sich auf der Audio-CD befinden.

Das zweite Kommando `CDROMREADTOCENTRY` hingegen benötigt als dritten Parameter für `ioctl()` eine Struktur vom Typ `cdrom_tocentry`, die folgendermaßen in der Headerdatei `cdrom.h` definiert ist.

```
/* Struktur wird bei CDROMREADTOCENTRY mit ioctl() erwartet */
struct cdrom_tocentry {
   __u8   cdte_track;
   __u8   cdte_adr  : 4;
   __u8   cdte_ctrl : 4;
   __u8   cdte_format;
   union  cdrom_addr cdte_addr;
   __u8   cdte_datamode;
};
```

Auf jeden Fall angeben müssen Sie in dieser Struktur den Track (`cdte_track`) und das Format der Ausgabe (`cdte_format`). Beim Format geben Sie `CDROM_MSF` (MSF = Minute, Sekunde, Frame) an. Möglich wäre auch `CDROM_LBA` (Logical Block Address), was eine Adresse, die durch eine Integerzahl dargestellt wird, zurückgeben würde. `cdte_ctrl` können Sie z. B. verwenden, um herauszufinden, ob es sich bei dem Track um einen Audio- oder Datentrack handelt. Hierzu müssen Sie nur wieder den bitweisen UND-Operator mit der Konstante `CDROM_DATA_TRACK` überprüfen. Ist der Ausdruck wahr, handelt es sich um einen Datentrack, ansonsten ist es ein Audiotrack.

Hinweis

Eine Sekunde besteht aus 75 Frames. Ein Frame ist 2352 Bytes lang.

Da wir uns für die Minuten, Sekunden und Framelänge der CD-ROM mit `CDROM_MSF` interessieren, benötigen Sie `union cdrom_addr cdte_addr` – was allerdings nicht mehr auf den ersten Blick ersichtlich ist. Die Union `cdrom_addr` wiederum beinhaltet eine Struktur mit dem Namen `struct cdrom_msf msf`, die das beinhaltet, wonach Sie suchen. Hier die beiden Strukturen:

```
union cdrom_addr {
   struct cdrom_msf0 msf;
   int lba;
};

struct cdrom_msf0 {
   __u8 minute;
```

```
   __u8 second;
   __u8 frame;
};
```

Hinweis

Sofern Ihnen die Dokumentation der Gerätedatei nicht mehr weiterhilft, müssen Sie, ausgehend von den Angaben des Kommandos in `ioctl()`, die Angaben einer Struktur selbst systematisch absuchen. Ein *Greenhorn* verfängt sich bei umfangreicheren Gerätedateien dabei recht schnell im Netz der Spinne.

Hier die Funktion, die das Inhaltsverzeichnis einer Audio-CD auslesen soll.

```
static void content_cdrom (int cdrom) {
   struct cdrom_tochdr tochdr;
   struct cdrom_tocentry tocentry;
   int track;

   if (ioctl (cdrom, CDROMREADTOCHDR, &tochdr) == -1) {
      perror ("Kann den Header nicht holen");
      exit (EXIT_FAILURE);
   }
   printf ("\nInhalt %d Tracks:\n", tochdr.cdth_trk1);

   track = tochdr.cdth_trk0;
   while (track <= tochdr.cdth_trk1) {
      tocentry.cdte_track = track;
      tocentry.cdte_format = CDROM_MSF;
      if (ioctl (cdrom, CDROMREADTOCENTRY, &tocentry)==-1) {
         perror ("Kann Inhalt der CD nicht ermitteln");
         exit (EXIT_FAILURE);
      }
      printf ("%3d: %02d:%02d:%02d (%06d) %s%s\n",
         tocentry.cdte_track,
         tocentry.cdte_addr.msf.minute,
         tocentry.cdte_addr.msf.second,
         tocentry.cdte_addr.msf.frame,
         tocentry.cdte_addr.msf.frame +
         tocentry.cdte_addr.msf.second * 75 +
         tocentry.cdte_addr.msf.minute * 75 * 60 - 150,
         (tocentry.cdte_ctrl & CDROM_DATA_TRACK)
            ? "data " : "audio",
         CDROM_LEADOUT == track ? " (leadout)" : "");
      track++;
   }
}
```

Im nächsten Schritt soll die Audio-CD abgespielt werden. Zur Auswahl steht, entweder die komplette CD oder nur einen bestimmten Track abzuspielen. Auch hierzu benötigen Sie das Kommando `CDROMREADTOCENTRY`, um den Inhalt der CD bzw. bestimmte Tracks abzuspielen.

Die Struktur `cdrom_tocentry` benötigen Sie, um den Anfang des ersten Tracks (`cdte_track`), das Format (`CDROM_MSF`) und das Ende des letzten bzw., im Falle einer Einzelauswahl, das Ende des nächsten Tracks anzuzeigen. Den letzten Track können Sie übrigens auch mit der symbolischen Konstante `CDROM_LEADOUT` angeben.

Abgespielt wird diese Position dann mit dem Kommando `CDROMPLAYMSF` und der Struktur `cdrom_msf` als dritten Parameter von `ioctl()`.

```
/* Struktur wird bei CDROMPLAYMSF mit ioctl() 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    */
};
```

Bevor Sie also `ioctl()` und das Kommando `CDROMPLAYMSF` auf die Struktur loslassen, müssen Sie zuerst die Anfangsposition an den einzelnen Strukturvariablen übergeben – was alle Strukturvariablen mit der `..0` am Ende beinhaltet (`cdmsf_min0`, `cdmsf_sec0`, `cdmsf_frame0`) – und die Endposition, was alle Strukturvariablen (wer hätte das gedacht) mit einer `..1` am Ende beinhaltet (`cdmsf_min1`, `cdmsf_sec1`, `cdmsf_frame1`). Hier die Funktion dazu:

```
static void play_cdrom ( int cdrom, int flag ) {
   struct cdrom_tocentry tocentry;
   struct cdrom_msf play;
   int track = flag;
   int lead  = flag;

   if(flag == FULL) {
      track = 1;
      lead  = CDROM_LEADOUT;
   }
   else {
      track = flag;
      lead  = flag+1;
   }

   /* Anfang des ersten Liedes */
   tocentry.cdte_track = track;
   tocentry.cdte_format = CDROM_MSF;
   if (ioctl (cdrom, CDROMREADTOCENTRY, &tocentry) == -1) {
      perror ("Kann den Inhalt der CD nicht ermitteln\n");
      exit (EXIT_FAILURE);
   }

   play.cdmsf_min0   = tocentry.cdte_addr.msf.minute;
   play.cdmsf_sec0   = tocentry.cdte_addr.msf.second;
   play.cdmsf_frame0 = tocentry.cdte_addr.msf.frame;
```

```
   /* Ende des letzten Liedes */
   tocentry.cdte_track  = lead;
   tocentry.cdte_format = CDROM_MSF;
   if (ioctl (cdrom, CDROMREADTOCENTRY, &tocentry) == -1) {
      perror ("Kann den Inhalt der CD nicht ermitteln\n");
      exit (EXIT_FAILURE);
   }

   play.cdmsf_min1   = tocentry.cdte_addr.msf.minute;
   play.cdmsf_sec1   = tocentry.cdte_addr.msf.second;
   play.cdmsf_frame1 = tocentry.cdte_addr.msf.frame;

   if (ioctl (cdrom, CDROMPLAYMSF, &play) == -1) {
      perror ("Kann CD nicht abspielen");
      exit (EXIT_FAILURE);
   }
}
```

Wollen Sie die CD stoppen, eine Pause einbauen und den Track wieder aufnehmen, so müssen Sie nur die Kommandos `CDROMSTOP`, `CDROMPAUSE` und `CDROMRESUME` ohne weitere Parameter mit `ioctl()` aufrufen.

```
static void stop_cdrom (int cdrom) {
   if (ioctl (cdrom, CDROMSTOP) == -1) {
      perror ("Kann CD nicht anhalten");
      return;
   }
}

static void pause_cdrom (int cdrom) {
   if (ioctl (cdrom, CDROMPAUSE) == -1) {
      perror ("Kann PAUSE nicht setzen");
      return;
   }
}

static void resume_cdrom (int cdrom) {
   if (ioctl (cdrom, CDROMRESUME) == -1) {
      perror ("Kann CD nicht mehr fortfahren");
      return;
   }
}
```

5.6.4 Aktuellen Status der Audio-CD ermitteln

Nachdem Sie eventuell die Audio-CD gestartet haben, möchten Sie sicherlich auch wissen, welcher Track gerade abgespielt wird oder wie lange der Track oder die CD insgesamt schon abgespielt wurde – oder ob die CD überhaupt gerade gespielt wird, falls jemand gerade Pause gedrückt hat, etc.

Den Status einer laufenden Audio-CD können Sie mit dem Kommando `CDROMSUBCHNL` und der Struktur `cdrom_subchnl` als dritten Parameter für `ioctl()` ermitteln.

```
/* Struktur wird bei CDROMSUBCHNL mit ioctl() erwartet */
struct cdrom_subchnl {
   __u8  cdsc_format;
   __u8  cdsc_audiostatus;
   __u8  cdsc_adr:   4;
   __u8  cdsc_ctrl:  4;
   __u8  cdsc_trk;
   __u8  cdsc_ind;
   union cdrom_addr cdsc_absaddr;
   union cdrom_addr cdsc_reladdr;
};
```

Bevor Sie `ioctl()` aufrufen, sollten Sie zuvor noch das Format (`cdsc_format=CDROM_MSF`) angeben. Nach dem Aufruf von `ioctl()` beinhaltet `cdsc_audiostatus` den aktuellen Status der Audio-CD. Dies können folgende symbolische Konstanten sein:

```
/* Audio-Status ermitteln wird nicht unterstützt */
#define CDROM_AUDIO_INVALID 0x00
/* Audio-CD wird gerade abgespielt */
#define CDROM_AUDIO_PLAY 0x11
/* Audio-CD wurde angehalten (Pause) */
#define CDROM_AUDIO_PAUSED 0x12
/* Audio-CD wurde komplett abgespielt */
#define CDROM_AUDIO_COMPLETED 0x13
/* Audio-CD abspielen wurde wegen Fehler unterbrochen */
#define CDROM_AUDIO_ERROR 0x14
/* Kein besonderer Status vorhanden */
#define CDROM_AUDIO_NO_STATUS 0x15
```

Über die Strukturvariable `cdsc_absaddr` können Sie dann die absolute Zeit und über `cdsc_reladdr` die relative Zeit der Audio-CD ermitteln. Auch hier finden Sie wieder die entsprechenden Werte über den Umweg zur Union `cdrom_addr`. Hier die Funktion dafür:

```
static void get_audio_status (int cdrom) {
   struct cdrom_subchnl sub;

   printf ("Audio status: ");
   fflush (stdout);
   sub.cdsc_format = CDROM_MSF;
   if (ioctl (cdrom, CDROMSUBCHNL, &sub))
      printf ("FAILED\n");
   else {
      switch (sub.cdsc_audiostatus) {
      case CDROM_AUDIO_INVALID:
         printf ("invalid\n");
         break;
      case CDROM_AUDIO_PLAY:
         printf ("playing");
         break;
```

```
      case CDROM_AUDIO_PAUSED:
         printf ("paused");
         break;
      case CDROM_AUDIO_COMPLETED:
         printf ("completed\n");
         break;
      case CDROM_AUDIO_ERROR:
         printf ("error\n");
         break;
      case CDROM_AUDIO_NO_STATUS:
         printf ("no status\n");
         break;
      default:
         printf ("Oops: unknown\n");
      }
      if (sub.cdsc_audiostatus == CDROM_AUDIO_PLAY ||
          sub.cdsc_audiostatus == CDROM_AUDIO_PAUSED) {
         printf (" at: %02d:%02d abs/ %02d:%02d track %d\n",
            sub.cdsc_absaddr.msf.minute,
            sub.cdsc_absaddr.msf.second,
            sub.cdsc_reladdr.msf.minute,
            sub.cdsc_reladdr.msf.second, sub.cdsc_trk);
      }
   }
}
```

5.6.5 Das komplette Listing

Sie haben nun gesehen, wie einfach es ist, mit `ioctl()` und den entsprechenden Kommandos auf die Treiberschnittstelle der Hardware über die Gerätedatei zuzugreifen. Allerdings haben Sie leider auch gemerkt, dass dies recht mühsam werden kann, wenn man den ganzen Strukturen folgen muss. Aber so, wie Sie hierbei vorgegangen sind, werden Sie in der Regel immer vorgehen müssen. Aufgrund des enormen Umfangs der Gerätedateien ist das immer noch eine recht undokumentierte Geschichte.

Hier nun nochmals die einzelnen Funktionen, eingebaut in einem kompletten Listing:

```
/* cdrom.c */
#include <stdio.h>
#include <unistd.h.>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <stdlib.h>
#include <linux/cdrom.h>
#include <errno.h>

/* Bitte ggf. Anpassen */
#define CDROM "/dev/cdrom"
#define FULL   0 /* komplette Audio-CD abspielen */
```

```
static int open_cdrom (void) {
   int fd = open (CDROM, O_RDONLY | O_NONBLOCK);
   if (fd == -1) {
      if (errno == ENOMEDIUM)
         printf ("Keine CD im Laufwerk!\n");
      else
         perror ("Fehler bei open()");
      exit (EXIT_FAILURE);
   }
   return fd;
}

static void open_tray (int cdrom) {
   if (ioctl (cdrom, CDROMEJECT) == -1) {
      perror ("Eject yourself");
      exit (EXIT_FAILURE);
   }
}

/* Funktioniert nicht überall */
static void close_tray (int cdrom) {
   if (ioctl (cdrom, CDROMCLOSETRAY) == -1) {
      printf ("CD-Tray bitte von Hand schließen\n");
   }
}

static void capability_cdrom (int cdrom) {
   const char *j[] = {"nein", "ja"};
   int caps = ioctl (cdrom, CDROM_GET_CAPABILITY);
   if (caps == -1) {
      perror ("CDROM_GET_CAPABILITY");
      return;
   }
   printf ("CDROM-Fähigkeiten:\n"
      "\tCD-R    : %s\n"
      "\tCD-RW   : %s\n"
      "\tDVD     : %s\n"
      "\tDVD-R   : %s\n"
      "\tDVD-RAM : %s\n",
      j[!!(caps & CDC_CD_R)],
      j[!!(caps & CDC_CD_RW)],
      j[!!(caps & CDC_DVD)],
      j[!!(caps & CDC_DVD_R)],
      j[!!(caps & CDC_DVD_RAM)]);
}

static void get_audio_status (int cdrom) {
   struct cdrom_subchnl sub;
   printf ("Audio-Status: ");
   fflush (stdout);
```

```
   sub.cdsc_format = CDROM_MSF;
   if (ioctl (cdrom, CDROMSUBCHNL, &sub))
      printf ("FAILED\n");
   else {
      switch (sub.cdsc_audiostatus) {
      case CDROM_AUDIO_INVALID:
         printf ("invalid\n");
         break;
      case CDROM_AUDIO_PLAY:
         printf ("playing");
         break;
      case CDROM_AUDIO_PAUSED:
         printf ("paused");
         break;
      case CDROM_AUDIO_COMPLETED:
         printf ("completed\n");
         break;
      case CDROM_AUDIO_ERROR:
         printf ("error\n");
         break;
      case CDROM_AUDIO_NO_STATUS:
         printf ("no status\n");
         break;
      default:
         printf ("Oops: unknown\n");
      }
      if (sub.cdsc_audiostatus == CDROM_AUDIO_PLAY ||
          sub.cdsc_audiostatus == CDROM_AUDIO_PAUSED) {
         printf (" at: %02d:%02d abs/ %02d:%02d track %d\n",
            sub.cdsc_absaddr.msf.minute,
            sub.cdsc_absaddr.msf.second,
            sub.cdsc_reladdr.msf.minute,
            sub.cdsc_reladdr.msf.second, sub.cdsc_trk);
      }
   }
}

static void content_cdrom (int cdrom) {
   struct cdrom_tochdr tochdr;
   struct cdrom_tocentry tocentry;
   int track;

   if (ioctl (cdrom, CDROMREADTOCHDR, &tochdr) == -1) {
      perror ("Kann den Header nicht holen");
      exit (EXIT_FAILURE);
   }
   printf ("\nInhalt %d Tracks:\n", tochdr.cdth_trk1);

   track = tochdr.cdth_trk0;
   while (track <= tochdr.cdth_trk1) {
```

```
      tocentry.cdte_track = track;
      tocentry.cdte_format = CDROM_MSF;
      if (ioctl (cdrom, CDROMREADTOCENTRY, &tocentry)==-1) {
         perror ("Kann den Inhalt der CD nicht ermitteln");
         exit (EXIT_FAILURE);
      }
      printf ("%3d: %02d:%02d:%02d (%06d) %s%s\n",
          tocentry.cdte_track,
          tocentry.cdte_addr.msf.minute,
          tocentry.cdte_addr.msf.second,
          tocentry.cdte_addr.msf.frame,
          tocentry.cdte_addr.msf.frame +
          tocentry.cdte_addr.msf.second * 75 +
          tocentry.cdte_addr.msf.minute * 75 * 60 - 150,
         (tocentry.cdte_ctrl & CDROM_DATA_TRACK) ?
         "data " : "audio",
         CDROM_LEADOUT == track ? " (leadout)" : "");
      track++;
   }

}
static void play_cdrom ( int cdrom, int flag ) {
   struct cdrom_tocentry tocentry;
   struct cdrom_msf play;
   int track = flag;
   int lead  = flag;

   if(flag == FULL) {
      track = 1;
      lead  = CDROM_LEADOUT;
   }
   else {
      track = flag;
      lead  = flag+1;
   }
   /* Anfang des ersten Liedes */
   tocentry.cdte_track = track;
   tocentry.cdte_format = CDROM_MSF;
   if (ioctl (cdrom, CDROMREADTOCENTRY, &tocentry) == -1) {
      perror ("Kann den Inhalt der CD nicht ermitteln\n");
      exit (EXIT_FAILURE);
   }

   play.cdmsf_min0 = tocentry.cdte_addr.msf.minute;
   play.cdmsf_sec0 = tocentry.cdte_addr.msf.second;
   play.cdmsf_frame0 = tocentry.cdte_addr.msf.frame;

   /* Ende des letzten Liedes */
   tocentry.cdte_track = lead;
   tocentry.cdte_format = CDROM_MSF;
```

```
   if (ioctl (cdrom, CDROMREADTOCENTRY, &tocentry) == -1) {
      perror ("Kann den Inhalt der CD nicht ermitteln\n");
      exit (EXIT_FAILURE);
   }
   play.cdmsf_min1 = tocentry.cdte_addr.msf.minute;
   play.cdmsf_sec1 = tocentry.cdte_addr.msf.second;
   play.cdmsf_frame1 = tocentry.cdte_addr.msf.frame;

   if (ioctl (cdrom, CDROMPLAYMSF, &play) == -1) {
      perror ("Kann CD nicht abspielen");
      exit (EXIT_FAILURE);
   }
}

static void stop_cdrom (int cdrom) {
   if (ioctl (cdrom, CDROMSTOP) == -1) {
      perror ("Kann CD nicht anhalten");
      return;
   }
}

static void pause_cdrom (int cdrom) {
   if (ioctl (cdrom, CDROMPAUSE) == -1) {
      perror ("Kann PAUSE nicht setzen");
      return;
   }
}

static void resume_cdrom (int cdrom) {
   if (ioctl (cdrom, CDROMRESUME) == -1) {
      perror ("Kann CD nicht mehr fortfahren");
      return;
   }
}

int main (void) {
   int fd_cdrom;
   int select;
   int track;

   fd_cdrom = open_cdrom();
   do{
      printf("-1- CD-Tray öffnen\n");
      printf("-2- CD-Tray schließen\n");
      printf("-3- CDROM-Fähigkeiten\n");
      printf("-4- Audio-CD abspielen (komplett)\n");
      printf("-5- Einzelnen Track abspielen\n");
      printf("-6- <PAUSE> Audio-CD abspielen\n");
      printf("-7- <FORTFAHREN> Audio-CD abspielen\n");
      printf("-8- <STOP> Audio-CD abspielen\n");
```

```
      printf("-9- Aktuellen Status ermitteln\n");
      printf("-10- CD-Inhalt ausgeben\n");
      printf("-11- Programmende\n");
      printf("\nIhre Auswahl : ");
      scanf("%d",&select);

      switch( select ) {
         case 1: open_tray( fd_cdrom ); break;
         case 2: close_tray( fd_cdrom ); break;
         case 3: capability_cdrom (fd_cdrom); break;
         case 4: play_cdrom (fd_cdrom, FULL); break;
         case 5: printf("Welchen Track wollen Sie hören: ");
                 scanf("%d",&track);
                 play_cdrom( fd_cdrom, track); break;
         case 6: pause_cdrom( fd_cdrom ); break;
         case 7: resume_cdrom( fd_cdrom ); break;
         case 8: stop_cdrom( fd_cdrom ); break;
         case 9: get_audio_status( fd_cdrom );break;
         case 10: content_cdrom( fd_cdrom ); break;
         case 11: printf("Bye\n"); break;
         default : printf("Häh!\n");
      }
   } while( select != 11 );
   stop_cdrom( fd_cdrom );
   close (fd_cdrom);
   return 0;
}
```

Das Programm bei der Ausführung:

```
$ gcc -o cdrom cdrom.c
$ ./cdrom
-1- CD-Tray öffnen
-2- CD-Tray schließen
-3- CDROM-Fähigkeiten
-4- Audio-CD abspielen (komplett)
-5- Einzelnen Track abspielen
-6- <PAUSE> Audio-CD abspielen
-7- <FORTFAHREN> Audio-CD abspielen
-8- <STOP> Audio-CD abspielen
-9- Aktuellen Status ermitteln
-10- CD-Inhalt ausgeben
-11- Programmende

Ihre Auswahl : ...
```

Jürgen Wolf

Diese Bonus-Kapitel stammen aus dem Buch

Linux-UNIX-Programmierung

Das umfassende Handbuch

Galileo Press