Konzepte von Betriebssystemkomponenten

−Gerätetreiber−

Mario Körner, 26.01.2004

1 Allgemein

Wichtige Anforderungen an große Software-Pakete sind oftmals Flexibilität und Anpaßbarkeit.

Betriebssysteme bilden dabei keine Ausnahme: Anpaßbarkeit bedeutet hier unter anderem, daß das System unabhängig von der zugrundeliegenden Hardware sein soll. Dies erreicht man mithilfe von speziellen Betriebssystemmodulen: den Gerätetreibern.

Treiber stellen eine standardisierte Sicht auf die Vielfältigkeit der tatsächlichen physikalischen Gegebenheiten eines Rechners dar und ermöglichen dem Betriebssystem damit eine einheitliche Verwaltung dieser Resourcen.

Praktisch ist ein Treiber nichts weiter als ein Softwaremodul, welches eine Reihe von Funktionen implementiert, die zusammengenommen seine Schnittstelle zum Betriebssystem bilden. Die Art dieser Funktionen ist abhängig vom Typ des Gerätes, welches durch den Treiber bedient werden soll und wird durch das Betriebssystem vorgegeben.

Beinahe jedes physikalische Gerät wird durch einen Hardwarecontroller - einem intelligenten, programmierbaren Chip - gesteuert: beispielsweise Tastatur, Maus und serielle Schnittstellen durch einen Super IO Chip, IDE-Festplatten durch einen IDE-Controller, SCSI-Festplatten durch einen SCSI-Controller usw. Die Programmierung dieser Hardware-Controller unterscheidet sich von Chip zu Chip. Die Umsetzung der Betriebssystemschnittstelle auf die Schnittstelle des Hardwarecontrollers ist die wesentliche Aufgabe eines Gerätetreibers.

2 Anforderungen

Gute Treiber sollten vollen Zugriff auf die Fähigkeiten der Hardware bieten und dabei stabil, schnell und flexibel sein. Sie können von mehreren Benutzern gleichzeitig verwendet werden und unterstützen dabei sowohl synchronen als auch asynchronen Betrieb.

Treiber laufen in den meisten Betriebssystemen auf privilegierter Ebene, da den Anwenderprozessen der direkte Zugriff auf die Hardware ganz oder teilweise verwehrt wird. Ein fehlerhafter Treiber kann das System zum Absturz bringen. Saubere Programmierung und Fehlerbehandung sind hier ein absolutes Muß.

Bei der Entwicklung ist besonders darauf zu achten, daß die durch konkurierende Anforderungen von Benutzern und Hardware entstehenden Nebenläufigkeiten richtig behandelt werden.

3 Schnittstelle zur Hardware

Die Kommunikation zwischen Gerät und Gerätetreiber erfolgt hauptsächlich über

- Control/Status-Register (kurz: CSR oder einfach Register)
- Interrupts

Darüber hinaus können einige Geräte selbständig mit anderen Systemkomponenten wie z.B. dem Speicher über DMA (Direct Memory Access) Daten austauschen.

Über die Register werden Geräte aktiviert, initialisiert und mit Daten versorgt. Umgekehrt können mit ihrer Hilfe Daten und Statusinformationen vom Gerät abgefragt werden.

Interrupts dienen dazu, asynchrone Ereignisse an die CPU zu melden.

3.1 Adressierung der Control/Status−Register

Im einfachsten Fall ist der Hardware-Controller direkt an den I/O-Bus der CPU angeschlossen, der Zugriff auf ein Control/Status-Register erfolgt dann durch Ausgabe einer Adresse auf dem Adreßbus und Lesen bzw. Schreiben auf dem Datenbus.

Jedes physikalische Gerät am I/O-Bus belegt einen gewissen Adreßbereich, in dem sich seine Register befinden. Man spricht in diesem Zusammenhang auch von I/O-Ports. Um auf den I/O-Bus zuzugreifen, stellen die meisten CPUs spezielle Instruktionen bereit, welche oftmals nur auf privilegierter Ebene verwendet werden dürfen (z.B. Intel 80x86: in bzw. out).

Alternativ können die I/O-Ports eines Gerätes auch in den physikalischen Adreßraum des Hauptspeichers eingeblendet werden. Die CPU kann in diesem Fall die normalen Speicherzugriffsbefehle (80x86: mov) verwenden, um mit dem Gerät zu kommunizieren.

Nicht alle Geräte sind direkt mit der CPU verbunden, sondern es können verschiedene Bussysteme wie SCSI, PCI oder USB dazwischengeschaltet sein. Hier sind spezielle Bustreiber notwendig, auf denen die eigentlichen Gerätetreiber dann aufsetzen.

3.2 Verwaltung der I/O−Adreßbereiche unter Linux 2.4

Nicht immer weiß ein Treiber, an welchen Adressen sich die Register „seines“ Gerätes befinden. Insbesondere bei älteren Technologien wie dem ISA-Bus muß ein Treiber gelegentlich raten, wo die entsprechenden I/O-Bereiche liegen.

Fatal wäre es, wenn dabei auf Register zugegriffen würde, die ein anderer Treiber benutzt. Deshalb bietet Linux die Möglichkeit, verwendete I/O-Bereiche beim System zu registrieren und abzufragen.

Registrierte I/O-Bereiche können in `/proc/ioports` bzw. `/proc/iomem` eingesehen werden. Diese Informationen sind hilfreich bei der Fehlersuche oder der Konfiguration (älterer) Hardware.

Neuere Bussysteme, z.B. PCI (Peripheral Component Interconnect) kennen dieses Problem nicht mehr. Ein PCI-Bus kann nach angeschlossenen Geräten durchsucht werden. Jedes Gerät verfügt ferner über einen standardisierten Konfigurationsregistersatz, worüber u.a. die Art des Gerätes und dessen Adreßbedarf abgefragt werden können. Die Konfiguration des PCI-Busses erfolgt im Rahmen des Systemstarts.

4 Schnittstelle zum Betriebssystem

Wie bereits erwähnt werden Treiber über wohldefinierte Interfaces in das Betriebssystem eingebunden. Im folgenden soll dies am Beispiel Linux gezeigt werden.

Gerätetreiber unter Linux sind größtenteils als Bestandteil des Kernels implementiert, je nach Anwendungsfall als Modul oder statisch gelinkt.

Linux kennt 3 verschiedene Geräteklassen mit jeweils eigener Schnittstelle:

- Zeichenorientierte Geräte
- Blockorientierte Geräte
- Netzwerkgeräte

4.1 Zeichenorientierte Geräte

Zeichenorientierte Treiber stellen die einfachste Form eines Gerätetreibers dar. Sie bieten die Möglichkeit, einzelne Zeichen (oder Ströme von Einzelzeichen) zum Gerät zu übertragen oder vom Gerät einzulesen. Typische Beispiele dafür sind serielle Schnittstellen, Tastatur und Textkonsole.

Die Schnittstelle eines zeichenorientierten Treibers zum Betriebssystem entspricht der Dateischnittstelle, wie sie auch für die Integration von Dateisystemen verwendet wird. Sie wird durch die Struktur `file_operations` (deklariert in .../linux/include/linux/fs.h) repräsentiert und besteht im wesentlichen aus den Funktionen `open, read, write` und `release`.

Wird ein zeichenorientierter Treiber geladen, muß er sich zunächst beim Kernel anmelden. Dies tut er mittels der Funktion

```
int register_chrdev(unsigned int major, const char* name, struct
file_operations* fpops),
```

an die er mithilfe des Parameters `fpops` Funktionszeiger auf seine Implementierung übergibt. Alle im System registrierten Treiber erscheinen in `/proc/devices`.

Nachdem ein Treiber geladen und registriert ist, kann mit dem Kommando `mknod` ein entsprechender Eintrag im Dateisystem angelegt werden. Der Treiber wird dabei durch die Nummer („major device number“) im System identifiziert, welche beim Aufruf der Registrierungsfunktion angegeben wurde. Es ist möglich, mehrere solche Gerätedateien anzulegen, die auf den gleichen Treiber verweisen (etwa für die erste und zweite serielle Schnittstelle eines PCs). Zu ihrer Unterscheidung existiert eine weitere Nummer („minor device number“), welche innerhalb des Treibers ausgewertet wird.

Über Gerätedateien erhalten Benutzerprogramme Zugriff auf die Hardware. Zeichenorientierte Gerätedateien kennzeichnet `ls` durch ein „c“ (char device). Unter Linux sind sie gewöhnlich im Verzeichnis `/dev` zu finden.

4.2 Blockorientierte Geräte

Blockorientierte Geräte transferieren Daten in Form von Blöcken fester Größe, auf die wahlfrei zugegriffen werden kann. Klassisches Beispiel sind Festplatten und ähnliche Massenspeicher mit ihren Sektoren als kleinste Dateneinheit.

Obwohl bockorientierte Geräte aus Anwendersicht den zeichenorientierten Geräten sehr ähnlich erscheinen - als Gerätedateien mit der Markierung „b“ - ist ihre Kernel-interne Verwaltung deutlich komplexer.

Hauptursache dafür sind die hohen Performanceanforderungen. Ist der Festplattenzugriff zu langsam, so kann das gesamte System unbrauchbar werden. Deshalb werden Daten auf langsamen Massenspeichern durch den Kernel im Hauptspeicher zwischengepuffert. Die Verwaltung dieses Pufferspeichers („buffer cache“) erfolgt in Blöcken fester Größe (vgl. Vortrag über „Disc caches“). Bei Bedarf müssen diese Blöcke möglichst schnell auf die Platte ausgelagert bzw. von dort eingelesen werden.

Im Gegensatz zu zeichenorientierten Treibern implementiert ein blockorientierter Treiber keine `read`– bzw. `write`-Funktion. Stattdessen kommuniziert der Kernel über Warteschlangen („request queues“) mit den Blocktreibern, welche Aufträge zum Lesen bzw. Schreiben eines Datenblocks enthalten. Jeder Treiber stellt eine Bearbeitungsfunktion für eine oder mehrere „request queue“ zur Verfügung. Um Daten zwischen „buffer cache“ und Festplatte zu transferieren, muß der Kernel die entsprechenden Blöcke in die „request queue“ des richtigen Treibers einhängen und die Bearbeitungsfunktion aufrufen.

Die Verwaltung der „request queues“ hat wesentlichen Einfluß auf die Performance der Festplattenzugriffe. Der Kernel optimiert die Reihenfolge der Anforderungen deshalb nach den folgenden Kriterien:

- Bei einem Festplattenzugriff sollten möglichst gleich mehrere physikalisch aufeinanderfolgende Sektoren gelesen bzw. geschrieben werden („clustering“)
- Der Schreib-/Lesekopf der Festplatte sollte möglichst selten die Richtung ändern („Fahrstuhl-Algorithmus“)

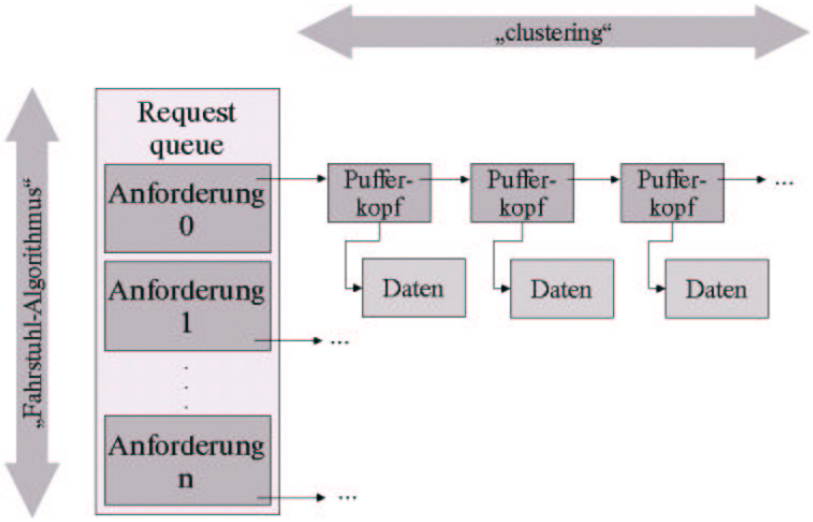

Abbildung 1 Struktur einer "request queue"

Die Schnittstelle zwischen Blocktreiber und Betriebssystem besteht aus mehreren Funktionen, Makros und (globalen) Variablen, welche der Registrierung und Parametrisierung des Treibers bzw. der Verwaltung der „request queues“ dienen. Eine umfassende Beschreibung findet sich in [1].

4.3 Netzwerkgeräte

Die Rolle eines Netzwerktreibers im System ist vergleichbar mit der eines Blocktreibers: Er registriert seine Dienste, um bei Bedarf Datenblöcke zu „senden“ und zu „empfangen“. Im Falle eines Netzwerktreibers handelt es sich dabei um Pakete, die vom darüber liegenden Netzwerksubsystem des Kernels geliefert werden.

Dennoch gibt es einige wesentliche Unterschiede:

- Ein Netzwerkgerät korrespondiert nicht mit einer Gerätedatei im `/dev` Verzeichnis, da es für Benutzerprogramme nicht viel Sinn macht, direkt auf ein Netzwerkgerät zuzugreifen. Netzwerktreiber übertragen Pakete, die zu verschiedenen logischen Verbindungen gehören; ihre Zuordnung erfolgt in den Protokollimplementierungen des Netzwerksubsystems mit der Socket-Schnittstelle als Frontend.
- Netzwerktreiber empfangen asynchron Daten von der Leitung und leiten sie selbständig an den Kernel weiter, wohingegen blockorientierte Treiber stets auf Anfrage handeln.
- Netzwerktreiber erledigen einige kommunikationsspezifische Aufgaben, z.B. die Behandlung von Verbindungstimeouts oder das Setzen von Adressen und Übertragungsparametern.

Netzwerktreiber sind vollständig protokollunabhängig. Das gilt sowohl für Protokolle auf Netzwerkebene, als auch auf physikalischer Ebene. Die wesentliche Aufgabe des Treibers besteht in der Übermittlung einzelner Netzwerkpakete zwischen dem Netzwerksubsystem des Kernels und der Netzwerkkarte.

5 Literaturverzeichnis

[1] Rubini A., Corbet J., Linux Device Drivers, Second Edition, O'Reilly & Associates, freier Download von http://www.oreilly.com/catalog/linuxdrive2

[2] Bovet D.P., Ceasti M., Understanding the Linux Kernel, Second Edition, O'Reilly & Associates

[3] Rusling D.A., The Linux Kernel, http://en.tldp.org/LDP/tlk/tlk-toc.html