1 Einleitung

Computersysteme bestehen aus einer Anzahl unterschiedlicher Hard- und Softwarekomponenten, deren Zusammenspiel erst die Abarbeitung komplexer Programme ermöglicht. Zu den Hardwarekomponenten gehören beispielsweise die eigentliche Verarbeitungseinheit, der Mikroprozessor mit dem Speicher, aber auch die sogenannte Peripherie, wie Tastatur, Maus, Monitor, LEDs oder Schalter. Diese Peripherie wird über Hardwareschnittstellen an die Verarbeitungseinheit angeschlossen. Hierfür haben sich Schnittstellen wie beispielsweise USB (Universal Serial Bus) oder im Bereich der eingebetteten Systeme auch I^2C, SPI oder GPIO-Interfaces etabliert. Im PC-Umfeld ist PCI (Peripheral Component Interconnect) und PCI-Express verbreitet. Die Netzwerk- oder Grafikkarte wird beispielsweise über PCI-Express, Tastatur, Maus oder auch Drucker über USB mit dem Rechner verbunden.

Zentrale Komponente für den HW-Zugriff

Zu den Softwarekomponenten gehören das BIOS, das den Rechner nach dem Anschalten initialisiert, und das Betriebssystem. Das Betriebssystem koordiniert sowohl die Abarbeitung der Applikationen als auch die Zugriffe auf die Peripherie. Vielfach ersetzt man in diesem Kontext den Begriff Peripherie durch *Hardware* oder einfach durch *Gerät*, so dass das Betriebssystem den Zugriff auf die Hardware bzw. die Geräte steuert. Dazu muss es die unterschiedlichen Geräte kennen, bzw. es muss wissen, wie auf diese Geräte zugegriffen wird. Derartiges *Wissen* ist innerhalb des Betriebssystems in den Gerätetreibern hinterlegt. Sie stellen damit als Teil des Betriebssystemkerns die zentrale Komponente für den Hardwarezugriff dar. Ein Gerätetreiber ist eine Softwarekomponente, die aus einer Reihe von Funktionen besteht. Diese Funktionen wiederum steuern den Zugriff auf das Gerät.

Für jedes unterschiedliche Gerät wird ein eigener Treiber benötigt. So gibt es beispielsweise jeweils einen Treiber für den Zugriff auf die Festplatte, das Netzwerk oder die serielle Schnittstelle.

Da das Know-how über das Gerät im Regelfall beim Hersteller des Gerätes und nicht beim Programmierer des Betriebssystems liegt, sind innerhalb des Betriebssystemkerns Schnittstellen offengelegt, über die der

vom Hersteller erstellte Treiber für das Gerät integriert werden kann. Kennt der Treiberprogrammierer diese Schnittstellen, kann er seinen Treiber erstellen und den Anwendern Zugriff auf die Hardware ermöglichen.

Der Anwender selbst greift auf die Hardware über ihm bekannte Schnittstellen zu. Bei einem Unix-System ist der Gerätezugriff dabei auf den Dateizugriff abgebildet. Jeder Programmierer, der weiß, wie er auf normale Dateien zugreifen kann, ist imstande, auch Hardware anzusprechen.

Für den Anwender eröffnen sich neben dem einheitlichen Applikationsinterface noch weitere Vorteile. Hält sich ein Gerätetreiber an die festgelegten Konventionen zur Treiberprogrammierung, ist der Betriebssystemkern in der Lage, die Ressourcen zu verwalten. Er stellt damit sicher, dass die Ressourcen – wie Speicher, Portadressen, Interrupts oder DMA-Kanäle – nur einmal verwendet werden. Der Betriebssystemkern kann darüber hinaus ein Gerät in einen definierten, sicheren Zustand überführen, falls eine zugreifende Applikation beispielsweise durch einen Programmierfehler abstürzt.

Reale und virtuelle Geräte

Treiber benötigt man jedoch nicht nur, wenn es um den Zugriff auf reale Geräte geht. Unter Umständen ist auch die Konzeption sogenannter virtueller Geräte sinnvoll. So gibt es in einem Unix-System das Gerät `/dev/zero`, das beim lesenden Zugriff Nullen zurückgibt. Mit Hilfe dieses Gerätes lassen sich sehr einfach leere Dateien erzeugen. Auf das Gerät `/dev/null` können beliebige Daten geschrieben werden; sämtliche Daten werden vom zugehörigen Treiber weggeworfen. Dieses Gerät wird beispielsweise verwendet, um Fehlerausgaben von Programmen aus dem Strom sinnvoller Ausgaben zu filtern.

Kernelprogrammierung

Der Linux-Kernel lässt sich aber nicht nur durch Gerätetreiber erweitern. Erweiterungen, die nicht gerätezentriert sind, die vielleicht den Systemzustand überwachen, Daten verschlüsseln oder den zeitkritischen Teil einer Applikation darstellen, sind in vielen Fällen sinnvoll als Kernelcode zu realisieren.

Zur Kernelprogrammierung und zur Erstellung eines Gerätetreibers ist weit mehr als nur das Wissen um Programmierschnittstellen im Kernel notwendig. Man muss sowohl die Möglichkeiten, die das zugrunde liegende Betriebssystem bietet, kennen als auch die prinzipiellen Abläufe innerhalb des Betriebssystemkerns. Eine zusätzliche Erfordernis ist die Vertrautheit mit der Applikationsschnittstelle. Das gesammelte Know-how bildet die Basis für den ersten Schritt vor der eigentlichen Programmierung: die Konzeption.

Ziel dieses Buches ist es damit,

- den für die Kernel- und Treiberprogrammierung notwendigen theoretischen Unterbau zu legen,
- die durch Linux zur Verfügung gestellten grundlegenden Funktionalitäten vorzustellen,
- die für Kernelcode und Gerätetreiber relevanten betriebssysteminternen und applikationsseitigen Schnittstellen zu erläutern,
- die Vorgehensweise bei Treiberkonzeption und eigentlicher Treiberentwicklung darzustellen und
- Hinweise für ein gutes Design von Kernelcode zu geben.

Scope

Auch wenn viele der vorgestellten Technologien unabhängig vom Betriebssystem bzw. von der Linux-Kernel-Version sind, beziehen sich die Beispiele und Übungen auf den Linux-Kernel 4.x.

Ubuntu und Kernel 4.0.3

Die Beispiele sind auf einem Ubuntu-Linux (Ubuntu 14.04) und dem Kernel 4.0.3 beziehungsweise einem Raspberry Pi 2 unter dem Betriebssystem Raspbian in der Version 2015.03 und dem Kernel in Version 4.0.3 getestet worden. Welche Distribution, ob Debian (pur, in der Ubuntu- oder Raspbian-Variante), Arch Linux, Fedora, SuSE, Red Hat oder ein Selbstbau-Linux (beispielsweise auf Basis von Buildroot), dabei zum Einsatz kommt, spielt im Grunde aber keine Rolle. Kernelcode ist abhängig von der Version des Betriebssystemkerns, nicht aber direkt abhängig von der verwendeten Distribution (Betriebssystemversion).

UP und SMP

Das Gleiche gilt bezüglich des Einsatzfeldes. Dank seiner hohen Skalierbarkeit ist Linux das erste Betriebssystem, das in eingebetteten Systemen, in Servern, auf Desktop-Rechnern oder sogar auf der Mainframe läuft. Die vorliegende Einführung deckt prinzipiell alle Einsatzfelder ab. Dabei spielt es keine Rolle, ob es sich um eine Einprozessormaschine (Uniprocessor System, UP) oder um eine Mehrprozessormaschine (Symmetric Multiprocessing, SMP) handelt.

Aufbau des Buches

Zu einer *systematischen* Einführung in die Treiberprogrammierung gehört ein solider theoretischer Unterbau. Dieser soll im folgenden Kapitel gelegt werden. Wer bereits gute Betriebssystemkenntnisse hat und für wen Begriffe wie *Prozesskontext* und *Interrupt-Level* keine Fremdwörter sind, kann diesen Abschnitt überspringen. Im Anschluss werden die Werkzeuge und Technologien vorgestellt, die zur Entwicklung von Treibern notwendig sind. In der vierten Auflage wurde dieses Kapitel um einen Abschnitt über die Cross-Entwicklung ergänzt.

Bevor mit der Beschreibung des Treiberinterface im Betriebssystemkern begonnen werden kann, muss das Applikationsinterface zum Treiber hin vorgestellt werden. Denn was nützt es, einen Gerätetreiber zu schreiben, wenn man nicht im Detail weiß, wie die Applikation später auf den Treiber zugreift? Immerhin muss die von der Applikation geforderte Funktionalität im Treiber realisiert werden.

Das folgende Kapitel beschäftigt sich schließlich mit der Treiberentwicklung als solcher. Hier werden insbesondere die Funktionen eines Treibers behandelt, die durch die Applikation aufgerufen werden. In diesem Abschnitt finden Sie auch ein universell einsetzbares Treibertemplate.

Darauf aufbauend werden die Komponenten eines Treibers behandelt, die unabhängig (asynchron) von einer Applikation im Kernel ablaufen. Stichworte hier: Interrupts, Softirqs, Tasklets, Kernel-Threads oder auch Workqueues. Ergänzend finden Sie hier das notwendige Knowhow zum Sichern kritischer Abschnitte, zum Umgang mit Zeiten und zur effizienten Speicherverwaltung.

Mit diesen Kenntnissen können bereits komplexere Treiber erstellt werden, Treiber, die sich jetzt noch harmonisch in das gesamte Betriebssystem einfügen sollten. Diese Integration des Treibers ist folglich Thema eines weiteren Kapitels.

Neben den bisher behandelten Treibern für zeichenorientierte Geräte (Character Devices) werden für die Kernelprogrammierung relevante Subsysteme wie GPIO, I^2C, USB, Netzwerk und Blockgeräte vorgestellt. Hier zeigen wir Ihnen auch, wie Sie im Kernel existierende und eigene Verschlüsselungsverfahren verwenden.

Einen Treiber zu entwickeln, ist die eine Sache, gutes Treiberdesign eine andere. Dies ist Thema des letzten Kapitels.

Im Anhang schließlich finden sich Hinweise zur Generierung und Installation des Kernels für die PC-Plattform und für den Raspberry Pi. Die Referenzliste der wichtigsten Funktionen, die im Kontext der Kernelprogrammierung eine Rolle spielen, lassen das Buch zu einem Nachschlagewerk werden.

Notwendige Vorkenntnisse

C-Kenntnisse

Das vorliegende Buch ist primär als eine systematische Einführung in das Thema gedacht. Grundkenntnisse im Bereich der Betriebssysteme sind empfehlenswert. Kenntnisse in der Programmiersprache C sind

zum Verständnis unabdingbar. Vor allem der Umgang mit Pointern und Funktionsadressen sollte vertraut sein.

Zusätzliche Informationsquellen

Errata und Beispielcode zum Buch

Errata und vor allem auch den Code zu den im Buch vorgestellten Beispieltreibern finden Sie unter https://ezs.kr.hsnr.de/TreiberBuch/.

Quellcode online

Die sicherlich wichtigste Informationsquelle zur Erstellung von Gerätetreibern ist der Quellcode des Linux-Kernels selbst. Wer nicht mit Hilfe der Programme find und grep den Quellcode durchsuchen möchte, kann auf die »Linux Cross-Reference« (http://lxr.free-electrons.com/) zurückgreifen. Per Webinterface kann der Quellcode angesehen, aber auch nach Variablen und Funktionen durchsucht werden.

Dokumentation als Teil der Kernel-Quellen

In den Kernel-Quellen befindet sich eine sehr hilfreiche Dokumentation. Ein Teil der Dokumentation besteht aus Textdateien, die sich mit jedem Editor ansehen lassen. Ein anderer Teil der Dokumentation muss erst erzeugt werden. Dazu wird im Hauptverzeichnis der Kernel-Quellen (`/usr/src/linux/`) eines der folgenden Kommandos aufgerufen:

```
(root)# make psdocs    # für Dokumentation in Postscript
(root)# make pdfdocs   # für Dokumentation in PDF
(root)# make htmldocs  # für HTML-Dokumentation
```

Sind die notwendigen DocBook-Pakete installiert (unter Ubuntu 14.04 unter anderem das Paket `docbook-utils`), werden eine Reihe unterschiedlicher Dokumente generiert und in das Verzeichnis `/usr/src/linux/Documentation/DocBook/` abgelegt. Insbesondere sind hier die folgenden Dokumente zu finden:

device-drivers Dieses Dokument enthält die Beschreibung von Betriebssystemkern-Funktionen, die insbesondere für Entwickler von Gerätetreibern interessant sind.

gadget Eine Einführung in die Erstellung von USB-Slavetreibern

genericirq Eine Einführung in die Interruptverarbeitung, insbesondere auch der Programmierschnittstellen, im Linux-Kernel

kernel-api Dieses Dokument enthält die Beschreibung von Funktionen des Betriebssystemkerns.

kernel-hacking Kernel-Entwickler Rusty Russell führt in einige Grundlagen der Kernel-Entwicklung ein. Leider ist das Dokument nicht mehr aktuell.

kernel-locking Rusty Russell: »Unreliable Guide to kernel-locking«. Hier finden sich einige Aspekte wieder, die die Vermeidung beziehungsweise den Schutz kritischer Abschnitte betreffen.

librs Dieses Dokument enthält die Beschreibung der Reed-Solomon-Bibliothek, die Funktionen zum Kodieren und Dekodieren enthält.
mac80211 Beschreibung des mac80211-Subsystems
parportbook Eine Einführung in die Erstellung von Treibern, die auf die parallele Schnittstelle über das Parport-Subsystem von Linux zugreifen
regulator Eine Beschreibung des Spannungs- und Regulator-Interface (linkstart;regulators driver interface)
uio-howto Howto zu Userspacetreiber (UIO siehe auch [KuQu11/07])
writing_usb_driver Eine Einführung in die Erstellung von USB-Hosttreibern

Neben der Dokumentation, die den Kernel-Quellen beiliegt, gibt es noch diverse Informationsquellen im Internet:

Online-Quellen

http://www.lwn.net Immer donnerstags gibt es hier aktuelle Kernel-News sowie Tipps und Tricks rund um die Kernel- und Treiberprogrammierung. Die ganz aktuelle Ausgabe steht jeweils nur der zahlenden Klientel zur Verfügung. Wer ohne Obolus auskommen will, kann die jeweils vorherige Ausgabe kostenlos lesen.
http://free-electrons.com Sehr wertvolle, praxisorientierte Infos zur Kernel- und Treiberprogrammierung in Form von Tutorials und Foliensätzen. Das Material ist allerdings vorwiegend in Englisch.
http://www.kernel.org Der Server »kernel.org« ist die zentrale Stelle für aktuelle und auch für alte Kernelversionen. Darüber hinaus finden sich hier die Patches einiger Kernelentwickler.
http://www.lkml.org Hier lässt sich die Kernel-Mailing-Liste aktuell mitlesen, ohne selbst eingeschrieben sein zu müssen.
http://www.kernelnewbies.org Hier finden sich viele Einsteigerinformationen und Programmiertricks.
http://www.heise.de/open Unter dem Titel »Kernel-Log« wird hier mit jeder Kernelversion eine detaillierte Zusammenfassung der neuen Features veröffentlicht.

Zu jeweiligen Spezialgebieten der Kernelprogrammierung und Treiberentwicklung gibt es im Internet überdies einige Texte oder Artikel. Hier ist der Leser allerdings selbst gefordert, mit Hilfe einer Suchmaschine Zusatzmaterial zu finden.

Ergänzungen

Zur Abrundung des Themas werden noch die beiden folgenden Bücher empfohlen:

- Quade, Mächtel: Moderne Realzeitsysteme kompakt. Eine Einführung mit Embedded Linux. dpunkt.verlag 2012 ([QuMä2012]). Das Buch behandelt verstärkt die Userland-Aspekte, also beispielsweise die Konzeption und Realisierung von realzeitfähigen Applikationen.
- Quade: Embedded Linux lernen mit dem Raspberry Pi. Linux-Systeme selber bauen und programmieren. dpunkt.verlag 2014 ([Quade2014]). Das Buch behandelt vor allem die Systemaspekte und zeigt, wie aus den einzelnen Komponenten (Kernel, Treiber, Userland, Applikation) komplette Systeme gebaut werden.

Zu guter Letzt bleibt noch der Verweis auf unsere Artikelserie im Linux-Magazin ab Ausgabe 8/2003, die das Thema Kernelprogrammierung behandelt. In dieser Reihe sind inzwischen weit über 80 Artikel erschienen, die neben der Treiberentwicklung auch praxisorientiert den Linux-Kernel selbst vorstellen. Die Mehrzahl der Artikel kann kostenlos im Internet gelesen werden.