edv-buchversand.de

Linux-Treiber entwickeln

Eine systematische Einführung in die Gerätetreiber- und Kernel-Programmierung

» Hier geht's direkt zum Buch

DAS VORWORT

Dieses Vorwort wird Ihnen von www.edv-buchversand.de zur Verfügung gestellt.

1 Einleitung

Hardware

Computersysteme bestehen aus einer Vielzahl unterschiedlicher Hard- und Softwarekomponenten, deren Zusammenspiel erst die Abarbeitung komplexer Programme ermöglicht. Zu den Hardwarekomponenten gehören die eigentliche Verarbeitungseinheit, der Mikroprozessor (CPU), der Arbeitsspeicher (RAM) sowie permanente Speichermedien wie SSDs (Solid State Drives) oder HDDs (Hard Disk Drives). Darüber hinaus spielt die sogenannte Peripherie eine zentrale Rolle, zu der Geräte wie Tastatur, Maus, Monitor, LEDs, Schalter oder auch Sensoren zählen. Diese Peripherie wird über Hardwareschnittstellen an die Verarbeitungseinheit angeschlossen.

Schnittstellen für Standard-Peripherie

Für die Verbindung von Peripheriegeräten haben sich verschiedene Schnittstellen etabliert. Im PC-Umfeld ist USB (Universal Serial Bus) nach wie vor der Standard für Geräte wie Tastaturen, Mäuse, Drucker und externe Speichermedien. USB hat sich weiterentwickelt, und die aktuellen Versionen wie USB 3.2 und USB4 bieten deutlich höhere Datenübertragungsraten und verbesserte Leistungsfähigkeit.

Im Bereich der eingebetteten Systeme sind Schnittstellen wie I^2C (Inter-Integrated Circuit), SPI (Serial Peripheral Interface) und GPIO (General-Purpose Input/Output) weit verbreitet. Diese Schnittstellen werden häufig eingesetzt, um Sensoren, Displays oder andere Komponenten anzusteuern.

Standards für hohe Datenraten

Für die Anbindung von Hochgeschwindigkeitskomponenten wie Grafikkarten, Netzwerkkarten oder SSDs hat sich PCI-Express (Peripheral Component Interconnect Express) als Standard durchgesetzt. PCI-Express ist die moderne Weiterentwicklung des älteren PCI-Standards und bietet deutlich höhere Bandbreiten, die für anspruchsvolle Anwendungen wie Gaming, maschinelles Lernen oder Datenverarbeitung erforderlich sind. Die aktuelle Version PCIe 5.0 und die kommende PCIe 6.0 ermöglichen noch schnellere Datenübertragungsraten und verbesserte Effizienz.

Zusätzlich zu diesen etablierten Schnittstellen gewinnen neue Technologien wie Thunderbolt (basierend auf USB4) und NVMe (Non-Volatile Memory Express) an Bedeutung. Thunderbolt ermöglicht die Anbindung von Hochleistungsgeräten wie externen GPUs oder Speichersystemen mit extrem hohen Datenraten, während NVMe speziell für die Anbindung von SSDs über PCI-Express optimiert ist und eine deutlich schnellere Datenverarbeitung im Vergleich zu älteren Standards wie SATA ermöglicht.

Software

Zu den Softwarekomponenten gehören die Startup-Software (UEFI, BIOS, Firmware), die 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, sodass 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, die den Zugriff auf ein Gerät steuern.

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

Schnittstellen

Da das Know-how über das Gerät im Regelfall beim Hersteller des Geräts und nicht bei den Programmiererinnen und Programmierern des Betriebssystems liegt, sind innerhalb des Betriebssystemkerns Schnittstellen offengelegt, über die der vom Hersteller erstellte Treiber für das Gerät integriert werden kann. Kennt die Treiberprogrammiererin oder der Treiberprogrammierer diese Schnittstellen, kann der Treiber erstellt und Anwenderinnen und Anwendern Zugriff auf die Hardware ermöglicht werden.

Diese greifen auf die Hardware über ihnen bekannte Schnittstellen zu. Bei einem Unix-System ist der Gerätezugriff dabei auf den Dateizugriff abgebildet. Dadurch ist jede Person, die den Zugriff auf normale Dateien kennt, imstande, auch Hardware anzusprechen.

Damit eröffnen sich 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 Be-

triebssystemkern 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. Mithilfe dieses Geräts 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,
- Hinweise für ein gutes Design von Kernelcode zu geben,
- Best Practices für die Erstellung von eigenem Kernelcode, wie z.B. die Vermeidung von Race Conditions, zu vermitteln und
- funktionstüchtigen Beispielcode als Grundlage für eigene Entwicklungen zur Verfügung zu stellen.

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 6.x.

Ubuntu und Kernel 6.8

Die Beispiele sind auf einem Ubuntu-Linux (Ubuntu 24.04) und dem Kernel 6.8 beziehungsweise einem Raspberry Pi 4 unter dem Betriebssystem Raspberry Pi OS in der Version 2024-11-19-raspios_bookworm-arm64 und dem Kernel in Version 6.8 getestet worden. Die Wahl der Distribution – sei es Debian (pur, Ubuntu oder Raspbian), Arch Linux, Fedora, Red Hat oder ein eigenes Buildroot-Linux – hat darauf keinen Einfluss. Denn Kernelcode orientiert sich an der Version des Betriebssystemkerns, nicht an der Distribution.

UP und SMP

Ähnlich verhält es sich mit dem Einsatzgebiet: Linux ist dank seiner Skalierbarkeit universell einsetzbar – von Embedded Devices über Server und Desktops bis hin zu Mainframes. Diese Einführung behandelt alle Szenarien, egal ob Einprozessor- (UP) oder Mehrprozessorsysteme (SMP).

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 den 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, inklusive der Cross-Entwicklung.

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 darauffolgende 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 Know-how 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^{2}C, USB, Netzwerk, Blockgeräte, Watchdog, und Industrial-IO vorgestellt. Hier wird auch gezeigt, wie im Kernel existierende und eigene Verschlüsselungsverfahren verwendet werden können.

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

Im letzten Kapitel schließlich finden sich Hinweise zur Generierung und Installation des Kernels für die PC-Plattform und für den Raspberry Pi.

Schnuppertour

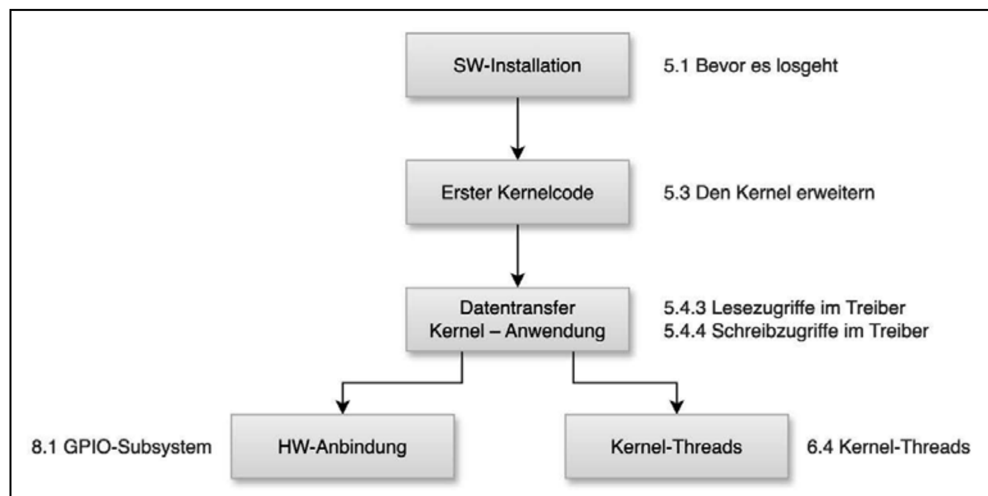

Abb. 1.1 *Die Schnuppertour vermittelt erste Einblicke.*

Schnelleinstieg

Bevor Sie in die komplexe Welt des professionellen Kernel- und Treiberprogrammierens eintauchen und sich mit den notwendigen Grundlagen vertraut machen, können Sie bei einer kleinen Schnuppertour erste Einblicke gewinnen. Starten Sie dazu mit der Installation der notwendigen Software (Abschnitt 5.1), erstellen und laden Sie Ihren ersten Kernelcode (Abschnitt 5.3) und tauschen Sie erste Daten zwischen Kernel und Anwendung aus (Abschnitt 5.4.3 und Abschnitt 5.4.4).

Planen Sie Treibercode auf einem Raspberry Pi zu schreiben, dann schließen Sie doch eine LED an einen GPIO-Port an. Abschnitt 8.1 ist mit dem Device Tree Beispiel 8.1 und dem Quellcode Beispiel 8.4 ein guter Startpunkt.

Interessieren Sie sich hingegen mehr für reinen Kernelcode, dann bietet Abschnitt 6.4 einen schnellen und praktischen Einstieg für Ihre ersten Erfolge.

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://codeberg.org/quade/LinuxTreiberEntwickeln.git*.

Die sicherlich wichtigste Informationsquelle zur Erstellung von Gerätetreibern ist der Quellcode des Linux-Kernels selbst. Wer nicht mithilfe der Programme `find` und `grep` den Quellcode durchsuchen möchte, kann auf die Linux Cross-Reference (*https://elixir.bootlin.com/linux/latest/source*) zurückgreifen. Per Webinterface kann der Quellcode angesehen, aber auch nach Variablen und Funktionen durchsucht werden.

Quellcode online

In den Kernel-Quellen befindet sich eine sehr hilfreiche Dokumentation. Ein Teil der Dokumentation besteht aus reinen Textdateien, die sich beispielsweise mit einem Editor ansehen lassen. Generell setzt Linux aber auf das reStructuredText-Format [SphinxDoc], das lesbare Quelldateien ebenso wie attraktive Ausgaben in unterschiedlichen Formaten bietet. Zu finden ist die HTML-Variante unter *https://www.kernel.org/doc/html/latest*. 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 latexdocs # fuer Dokumentation in LaTeX
(root)# make pdfdocs   # fuer Dokumentation in PDF
(root)# make htmldocs  # fuer HTML-Dokumentation
(root)# make epubdocs  # fuer EPUB-Dokumentation
```

Sind die notwendigen Pakete installiert (unter Ubuntu 24.04 unter anderem das Paket `sphinx-common`), werden eine Reihe unterschiedlicher Dokumente generiert und in das Verzeichnis `/usr/src/linux/Documentation/output/` abgelegt.

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

ChatGPT

Large Language Models (LLMs) wie beispielsweise ChatGPT und DeepSeek können Fragen rund um den Linux-Kernel und die Kernelprogrammierung beantworten. Das ist sehr hilfreich und spart sehr viel Recherchearbeit, wenn es beispielsweise um die Erklärung von Funktionen oder Subsystemen geht.

Die LLMs sind darüber hinaus aber auch in der Lage, Kernelcode mitsamt detaillierten Erläuterungen und Hinweisen, wie der Kernelcode zu generieren, zu laden und zu testen ist, zu erstellen. Das ist sehr beeindruckend, aber muss mit großer Vorsicht genossen werden [Quade-1-25].

ChatGPT reflektiert diesbezüglich selbstkritisch, dass es in Hinsicht auf Hardware, Nebenläufigkeit, Synchronisation und Speichermanagement unsicher ist. Konkret: Der generierte Code lässt sich häufig nicht out of the box generieren. Hinzu kommt, dass die KI gerne veraltete Interfaces benutzt. Partiell werden Funktionen erfunden, was aber nur auf Nachfragen offenbart wird. Der generierte Code enthält subtile Fehler, die ohne ausreichend Wissen von einem Laien nicht erkannt werden und zu schwer identifizierbaren Seiteneffekten führen. Hier fehlt ChatGPT Kontextverständnis. Die KI korrigiert auch schon mal Fehler, die eigentlich nicht vorhanden sind und der erzeugte Code ist komplizierter, als es notwendig wäre.

Online-Quellen

https://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.

https://bootlin.com
: Sehr wertvolle, praxisorientierte Infos zur Kernel- und Treiberprogrammierung in Form von Tutorials und Foliensätzen. Das Material ist vorwiegend in Englisch. Bootlin hostet auch eine der besten Linux-Cross-Reference-Seiten [BootlinLXR].

https://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 Kernelentwicklerinnen und -entwickler.

https://www.lkml.org
: Hier lässt sich die Kernel-Mailing-Liste aktuell mitlesen, ohne selbst eingeschrieben sein zu müssen.

https://www.kernelnewbies.org
Hier finden sich Einsteigerinformationen und Programmiertricks.

https://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 die Leserin oder der Leser allerdings selbst gefordert, mithilfe einer Suchmaschine Zusatzmaterial zu finden.

Ergänzungen

Zu guter Letzt bleibt noch der Verweis auf die Artikelserie im Linux-Magazin ab Ausgabe 8/2003, die das Thema Kernelprogrammierung behandelt. In dieser Reihe sind inzwischen weit über 140 Artikel erschienen, die neben der Treiberentwicklung auch praxisorientiert den Linux-Kernel selbst vorstellen. Die Mehrzahl der Artikel kann kostenlos im Internet gelesen werden.