Linux

Das umfassende Handbuch

Kapitel 1
Was ist Linux?

Um die einleitende Frage zu beantworten, erkläre ich in diesem Kapitel zuerst einige wichtige Begriffe, die im gesamten Buch immer wieder verwendet werden: Betriebssystem, Unix, Distribution, Kernel etc. Ein knapper Überblick über die Merkmale von Linux und die verfügbaren Programme macht deutlich, wie weit die Anwendungsmöglichkeiten von Linux reichen.

Es folgt ein kurzer Ausflug in die Geschichte von Linux. Von zentraler Bedeutung ist dabei natürlich die *General Public License* (kurz GPL), die angibt, unter welchen Bedingungen Linux weitergegeben werden darf. Erst die GPL macht Linux zu einem freien System, wobei »frei« mehr heißt als einfach »kostenlos«.

1.1 Einführung

Linux ist ein Unix-ähnliches Betriebssystem. Der wichtigste Unterschied gegenüber historischen Unix-Systemen besteht darin, dass Linux zusammen mit dem vollständigen Quellcode frei kopiert werden darf.

Ein Betriebssystem ist ein Bündel von Programmen, mit denen die Grundfunktionen eines Rechners realisiert werden. Dazu zählen die Verwaltung von Tastatur, Bildschirm, Maus sowie der Systemressourcen (CPU-Zeit, Speicher, SSDs etc.). Sie benötigen ein Betriebssystem, damit Sie ein Anwendungsprogramm überhaupt starten und eigene Daten in einer Datei speichern können. Populäre Betriebssysteme sind Windows, Linux, macOS, Android und iOS.

Schon lange vor Windows, Linux und den Smartphones gab es Unix. Dieses Betriebssystem war technisch gesehen seiner Zeit voraus: echtes Multitasking, eine Trennung der Prozesse voneinander, Zugriffsrechte für Dateien etc. Allerdings bot Unix anfänglich nur eine spartanische Benutzeroberfläche, stellte hohe Hardware-Anforderungen und lief nur auf teuren Workstations.

Inzwischen hat Linux Unix verdrängt: Große Teile des Internets werden von Linux-Servern getragen. Linux läuft in Form von Android auf Smartphones, in Routern und NAS-Festplatten sowie in Supercomputern: Die 500 schnellsten Rechner der Welt laufen heute *alle* unter Linux (*https://top500.org/statistics/list*).

Genau genommen bezeichnet der Begriff Linux nur den Kernel: Er ist der innerste Teil (also der Kern) eines Betriebssystems mit ganz elementaren Funktionen, wie Speicherverwaltung, Prozessverwaltung und Steuerung der Hardware. Die Informationen in diesem Buch beziehen sich auf den Kernel 6.*n*.

1.2 Hardware-Unterstützung

Linux unterstützt beinahe die gesamte gängige PC-Hardware und läuft darüber hinaus auch auf anderen Hardware-Plattformen, z. B. auf Smartphones oder Embedded Devices. Dennoch müssen Sie beim Kauf eines neuen Rechners aufpassen. Es gibt einige Hardware-Komponenten, die im Zusammenspiel mit Linux oft Probleme machen:

- **CPUs:** Linux ist kompatibel zu den meisten marktüblichen CPUs. Das gilt auch für ARM-CPUs. Linux läuft auf Milliarden von Android-Smartphones und Millionen von Raspberry Pis!

  Zu den Ausnahmen zählen aber die 2024 vorgestellten Notebooks auf der Basis von Qualcomm-ARM-CPUs (z. B. »Snapdragon X«). Auf solchen Geräten läuft Linux zwar prinzipiell, für den Praxisbetrieb gibt es (Stand Mitte 2025) aber noch zu viele Probleme. Lesenswert ist dieser Testbericht:

  https://www.phoronix.com/review/snapdragon-x-elite-linux-benchmarks

  Ähnliche Einschränkungen gelten für Apple-Notebooks mit den CPUs M1, M2 usw. Asahi Linux (*https://asahilinux.org*) hat sich den Betrieb von Linux auf modernen Apple-Geräten zum Ziel gesetzt. Stand Mitte 2025 ist Asahi Linux aber nur mit den ersten beiden CPU-Generationen kompatibel und auch dann mit vielen Einschränkungen verbunden.
- **Grafikkarten:** Fast alle auf dem Markt vertretenen Grafikkarten bzw. in die CPU integrierten Grafik-Cores funktionieren unter Linux. Für viele Linux-Anwender ohne besondere Anforderungen an das Grafiksystem sind Intel- oder AMD-CPUs mit eingebautem Grafik-Core die optimale Lösung. Grafikkarten von NVIDIA erfordern hingegen oft Zusatztreiber, damit die Karte perfekt genutzt werden kann. Die Installation dieser Treiber kann Probleme bereiten.
- **WLAN- und Bluetooth-Adapter:** Für relativ neue WLAN-, LAN- und Bluetooth-Controller gibt es mitunter keine Linux-Treiber.
- **Energiesparfunktionen:** Gerade neue Notebooks haben unter Linux oft deutlich kürzere Akku-Laufzeiten als unter Windows. Dieses Ärgernis resultiert daraus, dass das Zusammenspiel diverser Energiesparfunktionen optimale Treiber voraussetzt, die für Linux oft gar nicht oder erst ein, zwei Jahre nach der Markteinführung verfügbar sind.

Stellen Sie also *vor* dem Kauf eines neuen Rechners bzw. einer Hardware-Erweiterung sicher, dass alle Komponenten von Linux unterstützt werden. Auch eine Internetsuche nach *linux hardwarename* kann nicht schaden. Lesenswert sind zudem Testberichte der Zeitschrift c't:

Deren Redakteure machen sich bei den meisten Geräten die Mühe, auch die Linux-Kompatibilität zu testen.

Lieber etwas älter

Um ganz neue Notebooks mache ich beim Kauf in der Regel einen großen Bogen, auch wenn die Spezifikationen noch so verlockend sind. Sie verursachen allzu oft Treiberprobleme und verursachen Zusatzarbeit bei Installation und Konfiguration. Sie sparen Geld und vermeiden es, sich beim Konfigurieren zu ärgern, wenn Sie sich für ein Vorjahresmodell entscheiden!

1.3 Distributionen

Noch immer ist die einleitende Frage – Was ist Linux? – nicht ganz beantwortet. Viele Anwender interessiert der Kernel nämlich herzlich wenig. Für sie umfasst der Begriff Linux, wie er umgangssprachlich verwendet wird, neben dem Kernel auch das riesige Bündel mitgelieferter Programme: Dazu zählen unzählige Kommandos, ein Desktop-System (z. B. KDE oder Gnome), LibreOffice, Firefox, GIMP sowie zahllose Programmiersprachen und Server-Programme (Webserver, Mail-Server etc.).

Als Linux-Distribution wird die Einheit bezeichnet, die aus dem eigentlichen Betriebssystem (Kernel) und den vielen Zusatzprogrammen gebildet wird. Eine Distribution ermöglicht eine rasche und bequeme Installation von Linux. Die meisten Distributionen können kostenlos aus dem Internet heruntergeladen werden.

Distributionen unterscheiden sich vor allem durch folgende Punkte voneinander:

- **Umfang, Aktualität:** Die Anzahl, Auswahl und Aktualität der mitgelieferten Programme und Bibliotheken variiert stark. Manche Distributionen setzen bewusst auf etwas ältere, stabile Versionen – z. B. Debian.
- **Installations- und Konfigurationswerkzeuge:** Die mitgelieferten Programme zur Installation, Konfiguration und Wartung des Systems helfen dabei, die Konfigurationsdateien einzustellen. Das kann viel Zeit sparen.
- **Konfiguration des Desktops (KDE, Gnome):** Manche Distributionen lassen dem Anwender die Wahl zwischen KDE, Gnome und anderen Desktop-Systemen. Auch die Detailkonfiguration und optische Gestaltung variiert je nach Distribution.
- **Hardware-Unterstützung:** Linux kommt mit den meisten PC-Hardware-Komponenten zurecht. Dennoch gibt es im Detail Unterschiede zwischen den Distributionen, insbesondere wenn es darum geht, Nicht-Open-Source-Treiber (z. B. für NVIDIA-Grafikkarten) in das System zu integrieren.

- **Zielplattform (CPU-Architektur):** Viele Distributionen sind nur für x86- und ARM-kompatible Prozessoren erhältlich. Es gibt aber auch Distributionen für andere Prozessorplattformen (SPARC etc.). Besonders viele Plattformen unterstützt Debian.
- **Lizenz:** Die meisten Distributionen sind kostenlos erhältlich. Bei einigen Distributionen gibt es aber Einschränkungen: Beispielsweise ist bei den Enterprise-Distributionen von Red Hat und SUSE ein Zugriff auf das Update-System nur für registrierte Kunden möglich. Sie zahlen hier nicht für die Software an sich, wohl aber für das Service-Angebot rundherum.

Sie können eine Linux-Distribution nur so lange sicher betreiben, wie Sie Updates bekommen. Danach sollten Sie auf eine neue Version der Distribution wechseln. Deswegen ist es bedeutsam, wie lange es für eine Distribution Updates gibt. Hier gilt meist die Grundregel: je teurer der kommerzielle Support, desto länger der Zeitraum (siehe Tabelle 1.1).

Distribution	Wartungszeitraum
Debian	3 Jahre (mit Einschränkungen 5 Jahre)
Fedora	13 Monate
Red Hat Enterprise Linux (RHEL)	10 Jahre (mit Einschränkungen 13 Jahre)
RHEL-Klone	bis zu 10 Jahre
SUSE Enterprise Server	10 Jahre (mit Einschränkungen 13 Jahre)
Ubuntu LTS	3 bis 5 Jahre
Ubuntu LTS mit Pro-Upgrade	10 Jahre (mit Einschränkungen 12 Jahre)
Ubuntu (sonstige Versionen)	9 Monate

Tabelle 1.1 Maximale »Lebenszeit« verschiedener Linux-Distributionen (Stand: Sommer 2025)

Live-System

Manche Distributionen ermöglichen den Linux-Betrieb direkt von einem USB-Stick. Das ermöglicht ein einfaches Ausprobieren. Außerdem bieten derartige Live-Systeme eine gute Möglichkeit, um ein defektes Linux-System zu reparieren bzw. die betreffende Distribution neu zu installieren.

Rolling-Release-Konzept

Bei den meisten Linux-Distributionen werden mit den regulären Updates nur Sicherheitsprobleme und offensichtliche Fehler behoben,aber keine Versionssprünge durchgeführt. Bei-

spielsweise wurde Ubuntu 24.04 mit dem Versionsverwaltungsprogramm `git` in der Version 2.43 ausgeliefert. Sie können einen Ubuntu-Server mit Version 24.04 bis in die 2030er-Jahre sicher betreiben, aber `git` wird bei Version 2.43 bleiben. Zu einer neueren Version kommen Sie nur, wenn Sie alternative Paketquellen einrichten oder ein Upgrade auf eine neuere Distributions-Version durchführen (z. B. auf Ubuntu 26.04).

Es gibt aber auch Distributionen, die das *Rolling-Release*-Modell anwenden, z. B. Arch Linux oder openSUSE Tumbleweed: Dort erhalten Sie mit Updates stets die neueste Version *jeder* installierten Software-Komponente. Das klingt praktisch, kann aber zu Stabilitätsproblemen führen. Deswegen sind Rolling-Release-Distributionen im Server-Bereich unüblich. Sie sprechen eher fortgeschrittene Linux-Anwender an, die Software entwickeln oder Systeme administrieren und die kein Problem damit haben, nach einem Update die eine oder andere Konfigurationsdatei anzupassen, wenn etwas nicht mehr funktioniert.

Gängige Linux-Distributionen

Der folgende Überblick über die wichtigsten verfügbaren Distributionen soll Ihnen eine erste Orientierungshilfe geben. Die Liste ist alphabetisch geordnet und erhebt keinen Anspruch auf Vollständigkeit.

AlmaLinux ist ein RHEL-Klon, also eine zu Red Hat Enterprise Linux kompatible Distribution. AlmaLinux hat zusammen mit Rocky Linux die Nachfolge von CentOS Linux angetreten.

Android ist eine von Google entwickelte Plattform für Mobilfunkgeräte und Tablets. Android hat damit Linux zu der Weltdominanz verholfen, über die Linux-Entwickler in der Vergangenheit gescherzt haben. Android ist aber ungeeignet für eine PC-Installation und insofern keine »echte« Distribution.

Arch Linux ist eine für technische Anwender optimierte Rolling-Release-Distribution. Wegen der relativ komplizierten, im Textmodus durchzuführenden Installation machen Einsteigerinnen und Einsteiger zumeist einen großen Bogen um Arch Linux. Dafür zählen *https://wiki.archlinux.org* und *https://wiki.archlinux.de* zu den besten Quellen für Linux-Konfigurationsdetails im Netz. ArchLinux ist auf x86-Architekturen fokussiert. Es gibt zwar eine ARM-Variante, diese war zuletzt aber in keinem guten Zustand.

Die Arch-Linux-Derivate **CachyOS**, **Manjaro** und **EndeavourOS** mit grafischen Installations- und Konfigurationsprogrammen haben Arch Linux in der Top-10-Liste von *distrowatch.com* verankert.

CentOS war eine kostenlose Variante zu Red Hat Enterprise Linux (RHEL) und hatte eine riesige Installationsbasis. Allerdings hat Red Hat im Dezember 2020 das Ende von CentOS in seiner bisherigen Form verkündet. Sein Nachfolger **CentOS Stream** unterscheidet sich in zwei wichtigen Details vom ursprünglichen CentOS: Zum einen ist der Wartungszeitraum wesentlich kürzer und beträgt nur 4 bis 5 Jahre anstelle von bisher 10 Jahren; zum anderen werden die meisten Paket-Updates (ausgenommen sind Sicherheits-Updates, die einem *Non-*

disclosure Agreement unterliegen) zuerst für CentOS freigegeben, bevor sie für RHEL zum Einsatz kommen. Das scheint auf den ersten Blick ein Vorteil zu sein. Tatsächlich geht damit aber die vollständige Kompatibilität zu RHEL verloren. CentOS Stream ist für den längerfristigen Produktiveinsatz ungeeignet.

Das **Chrome OS** wird wie Android von Google entwickelt. Es ist für Notebooks optimiert und setzt zur Nutzung eine aktive Internetverbindung voraus. Die Benutzeroberfläche basiert auf dem Webbrowser Google Chrome. Chrome OS spielt aktuell in Europa keine große Rolle, wohl aber auf dem Bildungsmarkt in den USA: Dort werden billige Chrome-Books (also Notebooks mit Chrome OS) häufig in Schulen eingesetzt.

Debian ist die älteste vollkommen freie Distribution. Sie wird von engagierten Linux-Entwicklern zusammengestellt, wobei die Einhaltung der Spielregeln »freier« Software eine hohe Priorität genießt. Die strikte Auslegung dieser Philosophie hat in der Vergangenheit mehrfach zu Verzögerungen geführt.

Debian richtet sich an fortgeschrittene Linux-Anwender und hat einen großen Marktanteil bei Server-Installationen. Im Vergleich zu anderen Distributionen ist Debian stark auf maximale Stabilität hin optimiert und enthält deswegen oft nicht die neuesten Programmversionen. Dafür steht Debian für neun Hardware-Plattformen zur Verfügung. Viele andere Distributionen sind von Debian abgeleitet, z. B. Raspberry Pi OS und Ubuntu.

Fedora ist der kostenlose Entwicklungszweig von Red Hat Linux. Die Entwicklung wird von Red Hat unterstützt und gelenkt. Für Red Hat ist Fedora eine Art Spielwiese, auf der neue Funktionen ausprobiert werden können, ohne die Stabilität der Enterprise-Versionen zu gefährden. Programme, die sich unter Fedora bewähren, werden später in die Enterprise-Versionen integriert. Bei technisch interessierten Linux-Fans ist Fedora beliebt, weil diese Distribution oft eine Vorreiterrolle spielt: Neue Linux-Funktionen finden sich oft zuerst in Fedora und erst später in anderen Distributionen. Neue Fedora-Versionen erscheinen alle sechs Monate. Updates werden einen Monat nach dem Erscheinen der übernächsten Version eingestellt, d. h., die Lebensdauer ist mit 13 Monaten sehr kurz.

Das auf Debian basierende **Kali Linux** enthält eine riesige Sammlung von Hacking- und Pen-Testing-Werkzeugen. Die Distribution gilt als *der* Werkzeugkasten für Hacker und Sicherheitsexperten.

Oracle bietet unter dem Namen **Oracle Linux** eine Variante zu Red Hat Enterprise Linux (RHEL) an. Das ist aufgrund der Open-Source-Lizenzen eine zulässige Vorgehensweise. Technisch gibt es nur wenige Unterschiede zu RHEL, die Oracle-Variante ist aber billiger und ohne Support sogar kostenlos verfügbar.

Raspberry Pi OS ist die Standarddistribution für den beliebten Minicomputer Raspberry Pi. Raspberry Pi OS basiert auf Debian, wurde für den Raspberry Pi aber speziell adaptiert und erweitert.

Die 2018 von IBM übernommene Firma **Red Hat** ist das international bekannteste und erfolgreichste Linux-Unternehmen. Red-Hat-Distributionen dominieren insbesondere den amerikanischen Markt. Die Paketverwaltung auf der Basis des *Red Hat Package Formats* (RPM) wurde von vielen anderen Distributionen übernommen.

Red Hat ist überwiegend auf Unternehmenskunden ausgerichtet. Die Enterprise-Versionen (RHEL = **Red Hat Enterprise Linux**) sind vergleichsweise teuer. Sie zeichnen sich durch hohe Stabilität und einen zehnjährigen Update-Zeitraum aus. Für Linux-Enthusiasten und -Entwickler, die ein Red-Hat-ähnliches System zum Nulltarif suchen, bieten sich AlmaLinux, CentOS Stream, Fedora, Oracle Linux oder Rocky Linux an.

SUSE ist nach diversen Übernahmen die wichtigste deutsche Linux-Firma. Ihr Hauptprodukt, **SUSE Enterprise**, ist vor allem im europäischen Markt verankert. **openSUSE** ist eine kostenlose Variante, die sich aber speziell an Privatanwender und Entwicklerinnen wendet.

Ubuntu ist die zurzeit populärste Distribution für Privatanwender. Ubuntu verwendet als Basis Debian, ist aber besser für Desktop-Anwender optimiert (Motto: *Linux for human beings*). Die kostenlose Distribution erscheint im Halbjahresrhythmus. Für gewöhnliche Versionen werden Updates über neun Monate zur Verfügung gestellt. Für die alle zwei Jahre erscheinenden Versionen mit Long Time Support (LTS) gibt es sogar 3 bis 5 Jahre lang Updates.

Für kommerzielle Kunden bietet die Firma Canonical diverse Support-Angebote, unter anderem **Ubuntu Pro**: Dieses zahlungspflichtige Upgrade erweitert den Update-Zeitraum von LTS-Versionen auf zehn Jahre. Canonical hat sich damit vor allem im Server- und Cloud-Sektor zu den weltweit wichtigsten Linux-Firmen entwickelt.

Zu Ubuntu gibt es eine Menge offizieller und inoffizieller Varianten. Etabliert und weit verbreitet sind **Ubuntu Server**, **Kubuntu**, **Xubuntu**, **Ubuntu MATE** und **Linux Mint**. Die amerikanische Firma System76 pflegt mit **Pop!_OS** eine Ubuntu-Variante, die speziell für Notebooks optimiert ist und NVIDIA-Grafikkarten besonders gut unterstützt. Interessant ist auch **KDE Neon**: Diese Distribution kombiniert Ubuntu LTS mit stets aktuellen KDE-Paketen und ist insofern bei KDE-Fans beliebt.

Mini-Distributionen

Neben den oben aufgezählten »großen« Distributionen gibt es im Internet zahlreiche Zusammenstellungen von Miniatursystemen. Sie sind vor allem für Spezialaufgaben konzipiert, etwa für Wartungsarbeiten (Emergency-Systeme) oder um ein Linux-System ohne eigentliche Installation verwenden zu können (Live-Systeme). Populäre Vertreter dieser Linux-Gattung sind **Parted Magic** und **TinyCore**.

Einen ziemlich guten Überblick über alle momentan verfügbaren Linux-Distributionen, egal ob kommerziellen oder anderen Ursprungs, finden Sie im Internet auf der folgenden Seite:

https://distrowatch.com

Die Qual der Wahl

Eine Empfehlung für eine bestimmte Distribution ist schwierig. Für Linux-Einsteiger ist es zumeist von Vorteil, sich vorerst für eine weitverbreitete Distribution wie Debian, Fedora oder Ubuntu zu entscheiden. Zu diesen Distributionen sind sowohl im Internet als auch im Buch- und Zeitschriftenhandel viele Informationen verfügbar. Bei Problemen ist es vergleichsweise leicht, Hilfe zu finden.

Kommerzielle Linux-Anwender bzw. Server-Administratoren müssen sich entscheiden, ob sie bereit sind, für professionellen Support Geld auszugeben. In diesem Fall spricht wenig gegen die Marktführer Red Hat, Ubuntu (mit Pro-Abo) und SUSE. Andernfalls sind AlmaLinux, Debian, Rocky Linux und Ubuntu (ohne Abo) attraktive kostenlose Alternativen.

Linux Standard Base

Egal, für welches Linux Sie sich entscheiden – es gibt einen riesigen gemeinsamen Nenner. Auch wenn jede Distribution ihre Eigenheiten hat, gibt es noch viel mehr Gemeinsamkeiten. Das Linux-Standard-Base-Projekt (LSB) definiert dafür die Regeln. Fast alle Distributionen sind LSB-konform:

https://wiki.linuxfoundation.org/lsb/start

1.4 Open-Source-Lizenzen (GPL & Co.)

Die Grundidee von »Open Source« besteht darin, dass der Quellcode von Programmen frei verfügbar ist und von jedem erweitert bzw. geändert werden darf. Allerdings ist damit auch eine Verpflichtung verbunden: Wer Open-Source-Code zur Entwicklung eigener Produkte verwendet, muss den gesamten Code ebenfalls wieder frei weitergeben.

Die Open-Source-Idee verbietet übrigens keinesfalls den Verkauf von Open-Source-Produkten. Auf den ersten Blick scheint das ein Widerspruch zu sein. Tatsächlich bezieht sich die Freiheit in »Open Source« mehr auf den Code als auf das fertige Produkt. Zudem regelt die freie Verfügbarkeit des Codes auch die Preisgestaltung von Open-Source-Produkten: Nur wer neben dem Kompilat eines Open-Source-Programms weitere Zusatzleistungen anbietet (Handbücher, Support etc.), wird überleben. Sobald der Preis in keinem vernünftigen Verhältnis zu den Leistungen steht, werden sich andere Firmen finden, die es günstiger machen.

General Public License (GPL) und Lesser General Public License (LGPL)

Das Ziel der Open-Source-Entwickler ist es, Software zu schaffen, deren Quellen frei verfügbar sind und es auch bleiben. Um einen Missbrauch auszuschließen, sind viele Open-Source-Programme durch die *GNU General Public License* (kurz GPL) geschützt. Hinter der GPL steht die *Free Software Foundation* (FSF). Diese Organisation wurde von Richard Stallman gegrün-

det, um hochwertige Software frei verfügbar zu machen. Richard Stallman ist übrigens auch der Autor des Editors Emacs, der in Kapitel 18 beschrieben wird.

Die Kernaussage der GPL besteht darin, dass zwar jeder den Code verändern und sogar die resultierenden Programme verkaufen darf, dass aber gleichzeitig der Anwender/Käufer das Recht auf den vollständigen Code hat und diesen ebenfalls verändern und wieder kostenlos weitergeben darf. Jedes GNU-Programm muss zusammen mit dem vollständigen GPL-Text weitergegeben werden. Die GPL schließt damit aus, dass jemand ein GPL-Programm weiterentwickeln und verkaufen kann, *ohne* die Veränderungen öffentlich verfügbar zu machen. Jede Weiterentwicklung ist somit ein Gewinn für *alle* Anwender. Den vollständigen Text der GPL finden Sie hier:

https://gnu.org/licenses/gpl.html

Das Konzept der GPL ist recht einfach zu verstehen, im Detail treten aber immer wieder Fragen auf. Viele davon werden hier beantwortet:

https://gnu.org/licenses/gpl-faq.html

Wenn Sie glauben, dass Sie alles verstanden haben, sollten Sie das GPL-Quiz ausprobieren:

https://gnu.org/cgi-bin/license-quiz.cgi

Neben der GPL existiert noch die Variante LGPL (Lesser GPL). Der wesentliche Unterschied zur GPL besteht darin, dass eine derart geschützte Bibliothek auch von kommerziellen Produkten genutzt werden darf, deren Code *nicht* frei verfügbar ist. Ohne die LGPL könnten GPL-Bibliotheken nur wieder für GPL-Programme genutzt werden, was in vielen Fällen eine unerwünschte Einschränkung für kommerzielle Programmierer wäre.

Andere Lizenzen

Durchaus nicht alle Teile einer Linux-Distribution unterliegen den gleichen Copyright-Bedingungen! Der Kernel und diverse Linux-Tools unterliegen der GPL, aber für andere Komponenten und Programme gilt eine Fülle anderer Lizenzen.

- **MIT- und BSD-Lizenz:** Die MIT- und BSD-Lizenzen erlauben die kommerzielle Nutzung des Codes *ohne* die Verpflichtung, Änderungen öffentlich weiterzugeben. Die Lizenzen sind damit wesentlich liberaler als die GPL und eher mit der LGPL vergleichbar.
- **Doppellizenzen:** Für manche Programme gelten Doppellizenzen. Beispielsweise können Sie den Datenbank-Server MySQL für Open-Source-Projekte, auf einem eigenen Webserver bzw. für die innerbetriebliche Anwendung gemäß der GPL kostenlos einsetzen. Wenn Sie hingegen ein kommerzielles Produkt auf der Basis von MySQL entwickeln und samt MySQL verkaufen möchten, ohne Ihren Quellcode zur Verfügung zu stellen, dann kommt die kommerzielle Lizenz zum Einsatz. Die Weitergabe von MySQL wird in diesem Fall kostenpflichtig.

- **Business Source Licenses (BSL) und Server Side Public Licenses (SSPL):** Traditionelle Doppellizenzen beschränken die Weitergabe von Software. Bei *Software as a Service* (SAAS) kommt es aber nie zu einer Weitergabe. Für Server-Software ist das zum Problem geworden (z. B. für Datenbanksysteme wie MongoDB). SAAS-Firmen verdienen Geld mit Dienstleistungen auf der Basis von Software, nicht nur ohne zu zahlen, sondern auch ohne jede Notwendigkeit, sich an der Weiterentwicklung des Codes zu beteiligen. Dieses Geschäftsmodell wird von vielen Entwicklerinnen und Entwicklern als unfair betrachtet.

  Abhilfe schaffen Lizenzen, bei denen der Code zwar öffentlich publiziert wird, seine Anwendung aber an Bedingungen geknüpft wird. Bei manchen Lizenzen wird der Quellcode nach einer längeren Zeitspanne (z. B. nach vier oder fünf Jahren) automatisch frei von allen Einschränkungen. Nach dem OSI-Standard (siehe den folgenden Kasten) handelt es sich dabei *nicht* um echte Open-Source-Produkte.

- **Kommerzielle Lizenzen:** Es gibt im Linux-Umfeld einige Treiber und Programme, die zwar kostenlos genutzt werden dürfen, deren Quellcode aber ein Firmengeheimnis bleibt. Die NVIDIA-Grafiktreiber sind ein Beispiel dafür. (NVIDIA hat zuletzt damit begonnen, »echte« Open-Source-Treiber zu entwickeln. Der Erfolg/Abschluss dieses Projekts bleibt abzuwarten.)

Manche Distributionen kennzeichnen die Produkte, bei denen die Nutzung oder Weitergabe eventuell lizenzrechtliche Probleme verursachen könnte. Bei Debian befinden sich solche Programme in der Paketquelle *non-free*.

Das Dickicht der zahllosen, mehr oder weniger »freien« Lizenzen ist schwer zu durchschauen. Groß ist die Bandbreite zwischen der manchmal fundamentalistischen Auslegung von »frei« im Sinne der GPL und den verklausulierten Bestimmungen mancher Firmen, die ihr Software-Produkt zwar frei nennen möchten (weil dies gerade modern ist), in Wirklichkeit aber uneingeschränkte Kontrolle über den Code behalten möchten. Eine gute Einführung in das Thema finden Sie hier:

https://opensource.org
https://heise.de/-221957

Was ist die Open Source Initiative (OSI)?

Die Organisation *Open Source Initiative* hat Kriterien dafür aufgestellt, was als Open-Source-Lizenz gilt und was nicht. Lizenzen, die *OSI approved* sind, entsprechen dem Ideal der Open-Source-Bewegung:

https://opensource.org/licenses

Lizenzkonflikte zwischen Open- und Closed-Source-Software

Wenn Sie Programme entwickeln und diese zusammen mit Linux bzw. in Kombination mit Open-Source-Programmen oder -Bibliotheken verkaufen möchten, müssen Sie sich in die bisweilen verwirrende Problematik der unterschiedlichen Software-Lizenzen tiefer einarbeiten. Viele Open-Source-Lizenzen erlauben die Weitergabe nur, wenn auch Sie Ihren Quellcode im Rahmen einer Open-Source-Lizenz frei verfügbar machen. Auf je mehr Open-Source-Komponenten mit unterschiedlichen Lizenzen Ihr Programm basiert, desto komplizierter wird die Weitergabe.

Es gibt aber auch Ausnahmen, die die kommerzielle Nutzung von Open-Source-Komponenten erleichtern: Beispielsweise gilt für Apache und PHP sinngemäß, dass Sie diese Programme auch in Kombination mit einem Closed-Source-Programm frei weitergeben dürfen.

Manche proprietären Treiber für Hardware-Komponenten (z. B. für NVIDIA-Grafikkarten) bestehen aus einem kleinen Kernelmodul (Open Source) und diversen externen Programmen oder Bibliotheken, deren Quellcode nicht verfügbar ist (Closed Source). Das Kernelmodul hat nur den Zweck, eine Verbindung zwischen dem Kernel und dem Closed-Source-Treiber herzustellen.

Diese Treiber sind aus Sicht vieler Linux-Anwender eine gute Sache: Sie sind kostenlos verfügbar und ermöglichen es, diverse Hardware-Komponenten zu nutzen, zu denen es entweder gar keine oder zumindest keine vollständigen Open-Source-Treiber für Linux gibt.

Die Frage ist aber, ob bzw. in welchem Ausmaß die Closed-Source-Treiber wegen der engen Verzahnung mit dem Kernel, der ja der GPL untersteht, diese Lizenz verletzen. Viele Open-Source-Entwickler dulden die Treiber nur widerwillig. Eine direkte Weitergabe mit GPL-Produkten ist nicht zulässig, weswegen der Benutzer die Treiber in der Regel selbst herunterladen und installieren muss.

Die Frage ist allerdings, ob man der Open-Source-Idee mit dieser engen Auslegung der GPL-Regeln nützt oder schadet: Optimisten glauben, dass die Hardware-Firmen dadurch gezwungen wären, selbst Open-Source-Treiber zu entwickeln oder zumindest die erforderlichen Informationen an die Entwicklergemeinde freizugeben. Pessimisten befürchten, dass derartige Hardware dann unter Linux einfach nicht mehr nutzbar wäre, oft ohne gute Alternativen.

1.5 Die Geschichte von Linux

In den folgenden Punkten habe ich versucht, die wichtigsten Meilensteine aufzuzeigen, die zu dem Linux geführt haben, das wir heute kennen.

- **1982 – GNU:** Da Linux ein Unix-ähnliches Betriebssystem ist, müsste ich an dieser Stelle eigentlich mit der Geschichte von Unix beginnen – aber dazu fehlt hier der Platz. Stattdes-

sen beginnt diese Geschichtsstunde mit der Gründung des GNU-Projekts durch Richard Stallman. GNU steht für *GNU is not Unix*. In diesem Projekt wurden seit 1982 Open-Source-Werkzeuge entwickelt. Dazu zählen der GNU-C-Compiler, der Texteditor Emacs sowie diverse GNU-Utilities wie `find` und `grep` etc.

- **1989 – GPL:** Erst sieben Jahre nach dem Start des GNU-Projekts war die Zeit reif für die erste Version der *General Public License*. Diese Lizenz stellt sicher, dass freier Code frei bleibt.
- **1991 – Linux-Kernel 0.01:** Die allerersten Teile des Linux-Kernels (Version 0.01) entwickelte Linus Torvalds. Er gab seinen Code im September 1991 über das Internet frei. Schnell fanden sich weltweit Programmierer, die an der Idee Interesse hatten und Erweiterungen dazu schrieben. Als der Kernel von Linux die Ausführung des GNU-C-Compilers erlaubte, stand auch die gesamte Palette der GNU-Tools zur Verfügung. Weitere Komponenten waren das Dateisystem Minix, Netzwerk-Software von BSD-Unix, das X Window System des MIT und dessen Portierung XFree86 etc.

  Linux ist also nicht nur Linus Torvalds zu verdanken. Hinter Linux stehen vielmehr eine Menge engagierter Menschen, die in ihrer Freizeit, im Rahmen ihres Studiums oder bezahlt von Firmen wie Google, IBM oder HP freie Software produzieren.
- **1994 – Erste Distributionen:** Informatik-Freaks an Universitäten konnten sich Linux und seine Komponenten selbst herunterladen, kompilieren und installieren. Eine breite Anwendung fand Linux aber erst mit Linux-Distributionen, die Linux und die darum entstandene Software auf Disketten bzw. CD-ROMs verpackten und mit einem Installationsprogramm versahen. Vier der zu dieser Zeit entstandenen Distributionen existieren heute noch: Debian, Red Hat, Slackware und SUSE.
- **1996 – Pinguin:** 1996 wurde der Pinguin Tux zum Linux-Logo.
- **1998 – Microsoft sieht Linux als Bedrohung:** Mit dem rasanten Siegeszug des Internets stieg auch die Verbreitung von Linux, vor allem auf Servern. Gewissermaßen zum Ritterschlag für Linux wurde der legendäre Ausspruch von Steve Ballmer: *Microsoft is worried about free software* … Ein Jahr später ging Red Hat spektakulär an die Börse.
- **2000er-Jahre – Fundament für Internet und Cloud:** Der gemeinsame Nenner von Amazon (gegründet 1994), Google (1998), Wikipedia (2001), Facebook (2006) und Dropbox (2007) besteht darin, dass all diese Firmen bzw. Organisationen für ihren Server-Betrieb auf Linux setzen. Das Internet, wie es sich seit den 2000er-Jahren entwickelt, und die daraus entwachsene Cloud-Infrastruktur sind ohne Linux undenkbar.
- **2009 – Android:** Mit der Android-Plattform brachte Google Linux zuerst auf das Handy (2009), danach auch auf Tablets und in TV-Geräte.
- **2012 – Raspberry Pi:** 2012 eroberte der Minicomputer Raspberry Pi die Herzen von Elektronikbastlern. Für nur rund 40 EUR können Sie mit dem Raspberry Pi selbst Hardware-Experimente durchführen, in die Welt der Heimautomation einsteigen, ein Medien-Center oder einen Home-Server betreiben. Der Raspberry Pi macht Embedded Linux zu einem Massenphänomen.

- **2018 – IBM kauft Red Hat, Microsoft umarmt Open Source und Linux:** 2018 erwarb IBM die Firma Red Hat für stattliche 34 Mrd. US-Dollar. Gleichzeitig wandte sich Microsoft unter der Führung von Satya Nadella immer stärker der Open-Source-Idee und Linux zu: Das *Windows Subsystem for Linux* erlaubt die Ausführung von Linux direkt in Windows. Mit dem Kauf von GitHub, der ebenfalls 2018 über die Bühne ging, beherrscht Microsoft nun das wichtigste Repository für Open-Source-Projekte.

Ist seither nichts mehr passiert? Ja und nein! Linux und Open-Source-Software gehören mittlerweile so sehr zum IT-Mainstream, dass die markanten Meilensteine seltener wurden. Vielmehr hat sich Linux inkrementell weiterentwickelt und neue Marktsegmente erobert, beispielsweise in Form von Containern (Docker, Kubernetes) oder als Infrastruktur für KI-Firmen.

Eine technisch sehr spannende Entwicklung geht von **Immutable** oder **Atomic Distributions** aus, die ähnlich wie Container funktionieren. Beispiele sind *Vanilla OS*, *Fedora Silverblue*, *Image Mode for RHEL* oder *SUSE Adaptable Linux Platform*. Die ganze Distribution oder zumindest größere Komponenten sind dabei wie statische Blöcke zu sehen, die im Betrieb nie verändert werden. Für ein Update wird ein komplett neues Image vorbereitet und per Reboot gestartet. Die Konfigurationsdateien und Daten befinden sich außerhalb der eigentlichen Distribution in eigenen Dateisystemen.

Eine Sonderstellung nimmt NixOS ein, das durch ein funktionales Paketmanagement und vollständig deklarative Systemkonfiguration ähnliche Ziele verfolgt, technisch aber einen anderen Weg geht. Alle genannten Systeme ermöglichen einfache Rollbacks auf frühere Systemzustände und versprechen höhere Zuverlässigkeit. Der große Durchbruch solcher Systeme, am ehesten im Cloud-Segment, steht noch aus.

Kapitel 26
Kernel und Module

Dieses Kapitel beschäftigt sich mit dem Linux-Kernel und seinen Modulen. Module sind Teile des Kernels, die bei Bedarf geladen werden – etwa wenn eine bestimmte Hardware-Komponente zum ersten Mal angesprochen wird. Vorweg ein Überblick:

- **Kernelmodule:** Abschnitt 26.1 erklärt, warum Kernelmodule automatisch geladen werden und was Sie tun müssen, wenn dieser Automatismus versagt.
- **Device Trees:** Auf Smartphones und in Embedded Devices gelten für die Verwaltung der Kernelmodule ganz andere Voraussetzungen als auf PCs. Dort haben sich Device Trees zur Verwaltung von Kernelmodulen durchgesetzt. Eine Einführung in diese auch auf dem Raspberry Pi übliche Technik gibt Abschnitt 26.2.
- **Kernelmodule selbst kompilieren:** Wenn Sie spezielle Hardware einsetzen, müssen Sie eventuell ein eigenes Kernelmodul kompilieren. Wie das gelingt, verrät Abschnitt 26.3.
- **Den ganzen Kernel kompilieren:** Auch wenn eher selten die Notwendigkeit besteht, den ganzen Kernel neu zu kompilieren, beweist Abschnitt 26.4, dass dies durchaus keine Hexerei ist.
- **Kernel-Live-Patches:** Die Installation eines neuen Kernels wird erst mit einem Neustart wirksamer. Für Sicherheits-Updates ist es eleganter, diese im laufenden Betrieb durchzuführen. Abschnitt 26.5 stellt Mechanismen zur Aktivierung derartiger Live-Patches vor.
- **/proc- und /sys-Dateisystem:** Abschnitt 26.6 zeigt, wie Sie aus dem `/proc`- bzw. `/sys`-Dateisystem aktuelle Informationen über den Kernel ermitteln.
- **Kerneloptionen und -parameter:** Abschnitt 26.7 und Abschnitt 26.8 erklären, wie Sie während des Rechnerstarts Optionen an den Kernel übergeben bzw. im laufenden Betrieb Kernelparameter ändern.
- **Spectre, Meltdown & Co.:** Abschnitt 26.9 beschreibt schließlich, mit welchen Maßnahmen der Kernel Sie vor Sicherheitslücken in aktuellen CPUs schützt.

Dieses Kapitel richtet sich explizit an fortgeschrittene Linux-Anwenderinnen und -Anwender. Einsteiger sind gut beraten, nur den für ihre Distribution vorgesehenen Kernel zu verwenden! Die Informationen in diesem Kapitel gelten für alle Kernelversionen ab 4.*n*.

26.1 Kernelmodule

Der Kernel ist jener Teil von Linux, der für elementare Funktionen wie Speicherverwaltung, Prozessverwaltung, Zugriff auf SSDs und Netzwerkkarten etc. zuständig ist. Der Kernel verfolgt dabei ein modularisiertes Konzept: Anfänglich – also beim Hochfahren des Rechners – wird ein Basiskernel geladen, der nur jene Funktionen enthält, die zum Rechnerstart erforderlich sind.

Wenn im laufenden Betrieb Zusatzfunktionen benötigt werden, z. B. für spezielle Hardware, wird der erforderliche Code als Modul mit dem Kernel verbunden. Werden diese Zusatzfunktionen eine Weile nicht mehr benötigt, kann das Modul wieder aus dem Kernel entfernt werden. Dieses modularisierte Konzept hat viele Vorteile:

- Kernelmodule können nach Bedarf eingebunden werden. Wenn ein bestimmtes Modul auf Ihrem Rechner nicht benötigt wird, kann so Speicher gespart werden, d. h., der Kernel ist nicht größer als unbedingt notwendig und optimal an Ihre Hardware angepasst.
- Bei einer Änderung der Hardware (z. B. nach dem Einbau einer neuen Netzwerkkarte) muss kein neuer Kernel kompiliert, sondern nur das entsprechende Modul geladen werden.
- Bei der Entwicklung eines Kernelmoduls muss nicht ständig der Rechner neu gestartet werden. Es reicht, ein Modul neu zu kompilieren. Anschließend kann es bei laufendem Betrieb getestet werden.

Eine Menge Hintergrundinformationen zum Umgang mit Kernelmodulen finden Sie im Module-HOWTO-Dokument. Es ist zwar bald 20 Jahre alt, an den Prinzipien der Modulverwaltung hat sich seither aber wenig geändert.

https://www.tldp.org/HOWTO/Module-HOWTO

Module automatisch laden

Dafür, dass Kernelmodule tatsächlich automatisch geladen werden, sobald sie benötigt werden, ist die in den Kernel integrierte Komponente `kmod` verantwortlich. `kmod` wird durch die Dateien `/etc/modprobe.conf` und `/etc/modprobe.d/*.conf` gesteuert. Diese Konfigurationsdateien werden weiter unten genauer beschrieben.

In den Anfangstagen von Linux mussten der Kernel und seine Module exakt zusammenpassen: Es war nicht möglich, ein Modul zu laden, das für eine andere, vielleicht nur geringfügig veränderte Kernelversion kompiliert wurde. Aus diesem Grund gab und gibt es bis heute für jede Kernelversion ein eigenes Modulverzeichnis `/lib/modules/n.n`.

2006 hat man mit *Module Versioning* versucht, zusammen mit einem Modul Zusatzinformationen zu speichern, die Aufschluss darüber geben, ob eine Zusammenarbeit zwischen dem Modul und dem Kernel auch bei unterschiedlicher Versionsnummer möglich ist. Damit konnten nicht exakt zur Kernelversion passende Module genutzt werden. Dieser Mechanismus funktioniert allerdings nur, wenn es zwischen der Kernel- und der Modulversion keine

Änderungen an den Schnittstellen gegeben hat. In der Praxis hat sich dieser Mechanismus nicht besonders gut bewährt, weswegen seine Anwendung heute unüblich ist.

Kommandos zur Modulverwaltung

Alle gängigen Distributionen sind so eingerichtet, dass Module bei Bedarf automatisch geladen werden. Ein Beispiel: Sie binden mit `mount` das Dateisystem eines USB-Sticks in den Verzeichnisbaum ein. Daraufhin wird automatisch das `vfat`-Modul aktiviert, das zum Lesen des Dateisystems erforderlich ist.

Im Regelfall erfolgt die Modulverwaltung also automatisch und transparent, ohne dass Sie mit den im Folgenden beschriebenen Kommandos zur manuellen Modulverwaltung eingreifen müssen. Dennoch sollten Sie die Modulkommandos kennen, um Module zur Not auch manuell laden zu können.

Alle Module befinden sich im Verzeichnis `/lib/modules/n.n`. Dabei ist `n.n` die Version des laufenden Kernels. Moduldateien haben die Dateiendung `*.ko` bzw. `.ko.xz`, wenn sie mit `xz` komprimiert sind. Das Kommando `uname -r` liefert die Versionsnummer des laufenden Kernels:

```
uname -r

  6.15.10-200.fc42.aarch64

ls /lib/modules/6.15.10-200.fc42.aarch64

  config
  dtb/
  kernel/
  modules.alias
  modules.alias.bin
  modules.block
  modules.builtin
  ...
```

`modprobe` lädt das angegebene Modul in den Kernel. Dabei muss nur der Modulname angegeben werden. Zusätzlich können Parameter (Optionen) an das Modul übergeben werden. Hexadezimalen Werten müssen Sie `0x` voranstellen, also etwa `option=0xff`.

```
sudo modprobe cifs
```

Im Vergleich zum Low-Level-Kommando `insmod`, auf das ich hier nicht eingehe, agiert `modprobe` relativ intelligent:

- `modprobe` sucht die Moduldatei selbst im verzweigten Modulverzeichnisbaum des gerade aktiven Kernels. Bei Modulen, die schon beim Kompilieren in den Kernel integriert wurden, endet `modprobe` ohne Fehlermeldung.

- `modprobe` lädt gegebenenfalls auch alle Module, die als Voraussetzung für das gewünschte Modul benötigt werden.
- `modprobe` berücksichtigt alle in `/etc/modprobe*` angegebenen Moduloptionen.

`modprobe` setzt eine korrekte Modulkonfiguration durch die Dateien `/etc/modprobe*` und `/lib/modules/n.n/modules.dep` voraus.

`lsmod` liefert eine normalerweise recht lange Liste aller momentan geladenen Kernelmodule. Beachten Sie, dass `lsmod` in den Kernel einkompilierte Module nicht anzeigt, sondern nur solche Module, die nachträglich geladen wurden!

```
lsmod | sort

  Module                  Size  Used by
  8250_dw                24576  0
  ac97_bus               16384  1 snd_soc_core
  acpi_pad               24576  0
  acpi_thermal_rel       16384  1 int3400_thermal
  aesni_intel           401408  12
  ...
```

Sehr hilfreich bei der Zuordnung von Kernelmodulen ist das Kommando `lspci` mit der Option `-k` (*Kernel driver in use*). Es listet alle über den PCI-Bus angeschlossenen Hardware-Komponenten auf und zeigt an, welche Treiber geeignet wären, um das Gerät zu steuern, und welcher Treiber tatsächlich verwendet wird:

```
lspci -k

  00:00.0 Host bridge: Intel Corporation 8th Gen ...
    Subsystem: Lenovo 8th Gen Core Processor Host ...
    Kernel driver in use: skl_uncore
  00:01.0 PCI bridge: Intel Corporation Xeon ...
    Kernel driver in use: pcieport
  00:02.0 VGA compatible controller ...
    Subsystem: Lenovo UHD Graphics 630 (Mobile)
    Kernel driver in use: i915
    Kernel modules: i915
  ...
```

`modinfo` liefert eine Menge Informationen über ein Modul. Das Modul muss sich nicht im Kernel befinden.

```
sudo modinfo cifs

  filename:       /lib/modules/6.15.10-200.fc42.aarch64/kernel/fs/smb/
                  client/cifs.ko.xz
  version:        2.54
```

```
description:    VFS to access SMB3 servers e.g. Samba ...
license:        GPL
author:         Steve French
...
parm:           cifs_min_rcv: Network buffers in pool.
                Default: 4 Range: 1 to 64 (uint)
parm:           cifs_min_small: Small network buffers in pool.
                Default: 30 Range: 2 to 256 (uint)
...
```

`rmmod` entfernt das angegebene Modul wieder aus dem Kernel und gibt den belegten Speicher frei. Das Kommando kann nur erfolgreich ausgeführt werden, wenn das Modul gerade nicht verwendet wird.

```
sudo rmmod cifs
```

Modulkonfiguration

Die Modulverwaltung funktioniert scheinbar wie von Zauberhand:

- Wenn Sie eine zusätzliche Partition in das Dateisystem einbinden und dabei ein bisher nicht genutztes Dateisystemformat zum Einsatz kommt, wird automatisch das Modul für dieses Dateisystem geladen.
- Wenn sich die Partition auf einem USB-Laufwerk befindet, werden auch die USB-Module aktiviert, sofern diese nicht ohnedies schon geladen sind.
- Während der Initialisierung von Netzwerkfunktionen wird automatisch der erforderliche Treiber für Ihre Netzwerkkarte geladen etc.

Für das automatische Laden von Kernelmodulen ist die in den Kernel integrierte Komponente `kmod` verantwortlich. Dafür, dass all das funktioniert, sorgen unterschiedliche Konfigurationsmechanismen:

- **Für den Rechnerstart erforderliche Module:** Manche Kernelmodule werden sofort beim Start des Rechners benötigt – etwa Module zum Zugriff auf das Dateisystem. Sofern diese Module nicht integrale Bestandteile des Kernels sind, müssen sie in einer Initrd-Datei durch GRUB beim Rechnerstart an den Kernel übergeben werden (siehe Abschnitt 24.1, »GRUB-Grundlagen«).
- **Module für Grundfunktionen:** Die Module für die Basisverwaltung von Hardware-Komponenten (z. B. für das USB-System) werden von verschiedenen Scripts des Init-Prozesses direkt durch `modprobe`-Anweisungen geladen.
- **Module für Schnittstellen:** Eine Reihe weiterer Module wird dann geladen, wenn eine Schnittstelle zum ersten Mal benutzt wird. Hier tritt allerdings das Problem auf, dass es für manche Schnittstellen je nach der eingesetzten Hardware unterschiedliche Module gibt.

Wenn Sie also die Schnittstelle `eth0` für die erste Netzwerkkarte im Rechner ansprechen, muss das zu dieser Karte passende Modul geladen werden.

Da der Kernel nicht hellsehen kann, benötigt er eine Information darüber, welches Modul das richtige ist. Diese Information befindet sich in `/etc/modprobe.conf` sowie in den Dateien der Verzeichnisse `/etc/modprobe.d` und `/etc/modules-load.d`. Dort befinden sich installations- oder distributionsspezifische Optionen sowie Anweisungen, welche Module *nicht* automatisch zu laden sind (`blacklist`-Datei). Bei systemd-Distributionen werden diese Informationen durch die Service-Datei `systemd-modules-load.service` ausgewertet. Auf die Syntax der `modprobe`-Dateien gehe ich gleich im Detail ein.

Auch die automatische Device-Verwaltung durch das `udev`-System lädt bei Bedarf die notwendigen Module. Die entsprechenden Regeln finden Sie in `/etc/udev/rules.d`.

- **Module für USB-Geräte etc.:** Derartige Hardware-Komponenten nehmen eine Sonderrolle ein. Die Datei `/lib/modules/n.n/modules.alias` enthält eine Referenz mit Identifikationscodes der Komponenten und entscheidet darüber, welches Modul geladen wird.
- **Modulabhängigkeiten:** Eine Menge Module sind voneinander abhängig. Beispielsweise funktioniert das Modul `nfs` für das NFS-Dateisystem nur, wenn auch die Module `lockd`, `nfs_acl` und `sunrpc` geladen sind. Derartige Modulabhängigkeiten sind zentral in der Datei `/lib/modules/n.n/modules.dep` verzeichnet.

Module beim Rechnerstart laden

Manchmal wollen Sie unabhängig von den hier zusammengefassten Konfigurationswegen erreichen, dass beim Rechnerstart ein bestimmtes Kernelmodul geladen wird – und das, ohne sich auf irgendwelche Automatismen zu verlassen. Die optimale Vorgehensweise hängt von Ihrer Distribution ab.

Besonders einfach ist es bei Debian und Ubuntu: Dort kümmert sich das durch systemd einmalig ausgeführte Kommando `kmod` darum, alle in `/etc/modules` zeilenweise aufgelisteten Module zu laden. Sie müssen also lediglich das gewünschte Modul in einer neuen Zeile in `/etc/modules` angeben.

Bei vielen anderen Distributionen sowie bei aktuellen Debian- und Ubuntu-Versionen existiert zusätzlich das Verzeichnis `/etc/modules-load.d`. Es ist anfänglich zumeist leer. Wenn Sie Module automatisch laden möchten, erzeugen Sie in dem Verzeichnis eine Datei mit der Kennung `*.conf` und speichern dort die Modulnamen zeilenweise.

Viele Kernelmodule werden ganz früh im Bootprozess geladen. Damit Ihre Änderungen in `/etc/modules` oder in der `modprobe`-Konfiguration wirksam werden, müssen Sie die Initrd-Datei aktualisieren (siehe auch Abschnitt 24.2, »Initrd-Dateien«)!

```
sudo update-initramfs -u     # Debian, Ubuntu <= 25.04
sudo dracut --force          # Fedora, RHEL, Ubuntu >= 25.10
sudo mkinitcpio -P           # Arch Linux, CachyOS
```

modprobe-Syntax

Die folgenden Absätze beschreiben die wichtigsten Schlüsselwörter für die Dateien `modprobe.conf`, `/etc/modprobe.d/*` sowie `/lib/modules/n.n/modules.alias`. Weitere Details liefert `man modprobe.conf`.

- `alias`-Anweisungen geben an, welche Kernelmodule für welche Devices eingesetzt werden. Dazu ein Beispiel: Für das Device `/dev/eth0` soll das Modul `8139too` verwendet werden:

  ```
  alias eth0 8139too
  ```

  Der Zugriff auf viele Hardware-Komponenten erfolgt durch block- und zeichenorientierte Device-Dateien im `/dev`-Verzeichnis. Aus der Sicht des Kernels werden diese Device-Dateien nicht durch ihren Namen, sondern durch die Major- und Minor-Device-Nummer charakterisiert (siehe auch Abschnitt 12.9, »Device-Dateien«). Zahlreiche `alias`-Anweisungen stellen den Zusammenhang zwischen Device-Nummern und Modulen her.

  Analog sieht auch die Definition von Netzwerkprotokollen aus: Um ein bestimmtes Protokoll zu nutzen, sucht der Kernel nach einer Protokollfamilie mit dem Namen `net-pf-N`, wobei `N` die ID-Nummer des Protokolls ist. Das folgende Beispiel bewirkt, dass für die Protokollfamilie 5 das AppleTalk-Modul geladen wird:

  ```
  alias net-pf-5 appletalk
  ```

  Wenn Sie dieses veraltete Protokoll nicht brauchen und womöglich das entsprechende Modul gar nicht installiert ist, dann erspart Ihnen die folgende Anweisung lästige Fehlermeldungen:

  ```
  alias net-pf-5 off
  ```

- `options`-Anweisungen geben an, mit welchen Optionen ein bestimmtes Modul geladen werden soll. Die folgende Anweisung bewirkt, dass das Modul `ne` (für NE-2000-kompatible Ethernet-Karten) mit der Option `io=0x300` geladen wird:

  ```
  options ne io=0x300
  ```

- `include`-Anweisungen laden weitere Konfigurationsdateien.

- Mit `install`-Anweisungen geben Sie Kommandos an, die ausgeführt werden, anstatt das betreffende Modul einfach zu laden. Auch hierzu sehen Sie ein Beispiel, das aus Platzgründen auf zwei Zeilen verteilt wurde. Wenn das ALSA-Modul `snd` benötigt wird, sollen die folgenden Kommandos ausgeführt werden:

  ```
  install snd modprobe --ignore-install snd $CMDLINE_OPTS && \
    { modprobe -Qb snd-ioctl32 ; : ; }
  ```

- Mit `remove` geben Sie Kommandos an, die beim Entfernen eines Moduls ausgeführt werden sollen.

- `blacklist` bewirkt, dass modulinterne Alias-Definitionen nicht berücksichtigt werden. `blacklist`-Anweisungen befinden sich üblicherweise in der Datei `/etc/modprobe.d/`

`blacklist`. Sie enthält Module, die beispielsweise wegen Kompatibilitätsproblemen oder aufgrund von besseren Alternativen *nicht* geladen werden sollen. Beispielsweise verhindert die folgende Zeile, dass das Modul `usbmouse` geladen wird:

```
blacklist usbmouse
```

26.2 Device Trees

Der Begriff »Device Tree« bezeichnet die hierarchische Darstellung von Hardware-Komponenten. Der Device Tree wird während des Boot-Vorgangs vom Kernel geladen und teilt diesem mit, welche Hardware-Komponenten zur Verfügung stehen und über welche Anschlüsse diese Komponenten genutzt werden.

Device Trees wurden von den Linux-Kernelentwicklern ersonnen, um der Vielfalt von Chips und Geräten (sprich: Smartphones) auf Basis von ARM-CPUs Herr zu werden. Dank Device Trees ist es möglich, dass ein für ARM-Geräte kompilierter Kernel auf unterschiedlichen Geräten mit unterschiedlichen Zusatzkomponenten laufen kann. Bei PCs ist das selbstverständlich; in der ARM-Welt war dies aufgrund der Vielfalt von CPU- und Hardware-Varianten aber bisher unmöglich.

Sofern Sie nicht gerade ein Kernelentwickler für Smartphones sind, kommen Sie am ehesten bei der Arbeit mit Minicomputern wie dem Raspberry Pi mit Device Trees in Kontakt. Device Trees werden beispielsweise in Raspberry Pi OS standardmäßig eingesetzt. Beim Raspberry Pi beschreibt der Device Tree den Chip BCM27xx/28xx mit all seinen vielen Steuerungsmöglichkeiten, Bus-Systemen, GPIOs etc. Die Beschreibung erfolgt in einem Textformat, das dann aus Platz- und Effizienzgründen in ein binäres Format umgewandelt wird. Die Details dieses Formats sind aus Anwendersicht nicht relevant. Wenn Sie sich dennoch dafür interessieren, finden Sie auf den folgenden Seiten eine umfassende technische Referenz:

https://devicetree.org
https://elinux.org/Device_Tree
https://linux-magazin.de/Ausgaben/2013/06/Kern-Technik

Device Trees sind auch für den Betrieb von Notebooks mit ARM-CPUs relevant (im Unterschied zu Notebooks mit Intel- und AMD-CPUs, deren Hardware-Eigenschaften per ACPI Discovery (*Advanced Configuration and Power Interface*) ermittelt werden). Die Integration der korrekten Device-Tree-Daten in einen Boot-Prozess, der allgemeingültig für verschiedene Notebook-Modelle funktioniert, hat sich als großes Problem für Linux-Distributionen herausgestellt. Ein Lösungsansatz ist das Project *Stubble* von Canonical/Ubuntu, das die gerätespezifischen Device Trees in das Kernel-Image integriert. Stubble ist kompatibel zu `systemd-stub` und `ukify`. Mehr Details können Sie hier nachlesen:

https://github.com/ubuntu/stubble
https://lwn.net/Articles/1034579

Device-Tree-Konfiguration beim Raspberry Pi

Die Device-Tree-Konfiguration des Raspberry Pi ist über mehrere Orte verteilt. Das Verzeichnis `/boot/firmware` enthält Device-Tree-Beschreibungen für alle momentan gängigen Raspberry-Pi-Modelle. Die Dateikennung `*.dtb` steht dabei für *Device Tree Blob*, wobei ein *Blob* einfach ein binäres Objekt ist. Die Nummern 2708 bis 2711 sind Broadcom-interne Nummern zur Beschreibung der Chip-Familie. Die auf dem Raspberry Pi eingesetzten Modelle BCM2835 bis BCM2837 sind konkrete Implementierungen (»Familienmitglieder«, wenn Sie so wollen).

- `bcm2708-rpi-0-w.dtb`: Raspberry Pi Zero W/WH
- `bcm2710-rpi-3-b.dtb/bcm2710-rpi-3-b-plus.dtb`: Raspberry Pi Modell 3B und 3B+
- `bcm2711-rpi-4-b.dtb`: Raspberry Pi Modell 4B
- `bcm2711-rpi-400.dtb`: Raspberry Pi 400
- `bcm2712-rpi-5-b.dtb`: Raspberry Pi 5
- `bcm2711-rpi-500.dtb`: Raspberry Pi 500

Während des Boot-Prozesses übergibt das in der Boot-Datei `start.elf` enthaltene Programm den für das jeweilige Raspberry-Pi-Modell geeigneten Device Tree an den Kernel.

Das Verzeichnis `/boot/firmware/overlays` enthält fast 400 Device-Tree-Blob-Overlays (DTBOs), von denen ich hier nur einige wenige exemplarisch nenne:

- `hifiberry-dacplus-overlay.dtbo`: für die HiFiBerry-Erweiterung DAC+
- `lirc-rpi-overlay.dtbo`: für Infrarot-Fernbedienungen
- `i2c-rtc-overlay.dtbo`: für I^2C-Komponenten mit Real Time Clock
- `w1-gpio-overlay.dtbo`: für 1-Wire-Temperatursensoren

Overlays sind also Ergänzungen zum Haupt-Device-Tree `bcmxxx`. Diese DTBs werden *nicht* automatisch geladen, sondern nur, wenn die Konfigurationsdatei `config.txt` entsprechende Hinweise enthält. Sie ergänzen den Device Tree des BCM28xx bzw. BCM2711.

Die Raspberry-Pi-spezifische Datei `/boot/firmware/config.txt` kennt drei Schlüsselwörter zum Umgang mit Device Trees:

- `dtparam=i2c_arm=on/off,i2s=on/off,spi=on/off` aktiviert oder deaktiviert die Bussysteme I^2C, I^2S und SPI. Die Beschreibung dieser Bussysteme ist im Haupt-Device-Tree bereits enthalten. Standardmäßig sind alle drei Bussysteme inaktiv (entspricht `dtparam=i2c_arm=off,i2s=off,spi=off`). Wenn ein Bussystem oder mehrere genutzt werden sollen, müssen sie explizit aktiviert werden.
  - `dtoverlay=name,key1=val1,key2=val2...` bewirkt, dass die genannte Overlay-Datei geladen wird, wobei die übergebenen Parameter berücksichtigt werden.
  - `device_tree=`, also ohne die Zuweisung eines konkreten Werts, deaktiviert das gesamte Device-Tree-System.

Die Schlüsselwörter `dtparam` und `dtoverlay` können mehrfach verwendet werden.

Anhand der Device-Tree-Konfiguration entscheidet der Linux-Kernel, welche Module er lädt. Daher ersetzt die Device-Tree-Konfiguration die bei älteren Raspbian-Installationen erforderlichen Einstellungen in `/etc/modules` bzw. in `/etc/modprobe.d/*`.

In einfachen Fällen können Sie `config.txt` mit dem Programm `raspi-config` im Untermenü Advanced Options korrekt einrichten. Das gilt insbesondere, wenn Sie die Bussysteme I^2C und SPI verwenden möchten.

In allen anderen Fällen müssen Sie die entsprechenden Zeilen hingegen selbst in `config.txt` eintragen. Das folgende Listing illustriert die Syntax anhand einiger Beispiele. Beachten Sie, dass mehrere Optionen nur durch Kommata, aber nicht durch Leerzeichen voneinander getrennt werden dürfen (siehe z. B. `dtoverlay=...`).

```
# Datei /boot/firmware/config.txt
# Beispiele zur Steuerung des Device-Tree-Systems (weitere Details
# siehe /boot/overlays/README in einer Raspberry-Pi-OS-Installation)

# Audio-System aktivieren
dtparam=audio=on

# SPI-Bus aktivieren
dtparam=spi=on

# I2C-Bus aktivieren
dtparam=i2c_arm=on

# HiFiBerry DAC+ verwenden
dtoverlay=hifiberry-dacplus

# 1-Wire-Temperatursensor mit Standardeinstellungen verwenden
dtoverlay=w1-gpio-pullup

# 1-Wire-Temperatursensor verwenden, der mit
# GPIO X verbunden ist (per Default GPIO 4),
# und dabei den internen Pull-up-Widerstand aktivieren
dtoverlay=w1-gpio-pullup,gpiopin=X,pullup=y

# Echtzeituhr-Modell ds1307 verwenden
dtoverlay=rtc-i2c,ds1307

# IR-Empfänger verwenden
dtoverlay=lirc-rpi
```

26.3 Kernelmodule selbst kompilieren

Wenn Sie Linux in Kombination mit VirtualBox einsetzen, die proprietären Grafiktreiber von NVIDIA nutzen möchten oder ein anderes hardware-spezifisches Kernelmodul brauchen, das im Kernel Ihrer Distribution fehlt, dann müssen Sie das Modul passend zum laufenden Kernel kompilieren.

Zum Kompilieren eines Moduls sind neben dem C-Compiler `gcc` und `make` auch weitere grundlegende Entwicklungswerkzeuge erforderlich. Die meisten Distributionen erleichtern die Sache durch fertige Paketselektionen oder Meta-Pakete, die auf alle relevanten Pakete verweisen (siehe Tabelle 26.1).

Distribution	Kommando
Arch Linux	`pacman -S base devel`
Debian, Ubuntu	`apt install build-essential`
Fedora, RHEL	`dnf group install development-tools`
SUSE	`zypper install -t pattern devel_basis`

Tabelle 26.1 Kommandos zur Installation grundlegender Entwicklungswerkzeuge

Außerdem brauchen Sie die Include-Dateien (Header-Dateien) zum aktuellen Kernel (siehe Tabelle 26.2). Diese Dateien sind Teil des Kernelcodes. Bei einigen Distributionen befinden sich die Include-Dateien und der Rest des Codes in zwei getrennten Paketen. Das hat den Vorteil, dass Sie nicht gleich den riesigen Kernelcode installieren müssen, wenn Sie nur die vergleichsweise kleinen Include-Dateien brauchen.

Distribution	Kommando
Arch Linux	`pacman -S linux-headers`
Debian, Ubuntu	`apt install linux-headers-$(uname -r)`
Fedora, RHEL	`dnf install kernel-devel`
SUSE	`zypper install kernel-devel`

Tabelle 26.2 Kommando zur Installation der Kernel-Header-Dateien

Natürlich landen die Header-Dateien je nach Distribution in unterschiedlichen Verzeichnissen (siehe Tabelle 26.3). `n.n` ist dabei ein Platzhalter für die installierte Kernelversion. Diese Information ermitteln Sie mit dem Kommando `uname -r`.

Distribution	Pfad
Arch Linux	/usr/lib/modules/n.n/build
Debian, Ubuntu	/usr/src/linux-headers-n.n
Fedora, RHEL	/usr/src/kernels/n.n
SUSE	/usr/src/linux-n.n-obj/arch/default

Tabelle 26.3 Installationsort der Kernel-Header-Dateien

Wenn Sie den Kernel selbst kompilieren (siehe Abschnitt 26.4), landen die zum Kernel passenden Include-Dateien automatisch im Verzeichnis `/lib/modules/n.n/build/include`.

Modul kompilieren

Die meisten Programme, die eigene Kernelmodule benötigen, enthalten ein Installations-Script, das sich um das Kompilieren und Einrichten des Moduls kümmert. Das gilt beispielsweise für VirtualBox oder die Grafiktreiber von NVIDIA. Bei manchen Distributionen ist der Prozess sogar dahin gehend automatisiert, dass nach jedem Kernel-Update automatisch das Modul neu kompiliert wird (siehe *DKMS* weiter unten).

Wenn Sie dagegen den Quellcode für eine noch nicht offiziell unterstützte Hardware-Komponente heruntergeladen haben, müssen Sie sich um den Kompilierprozess selbst kümmern. Dazu führen Sie in der Regel die folgenden Kommandos aus. Nur das `make`-Kommando erfordert `root`-Rechte.

```
cd quellcodeverzeichnis
make clean
make
sudo make install
```

Modul-Updates mit DKMS automatisieren

DKMS steht für *Dynamic Kernel Module Support* und hilft dabei, nach einem Kernel-Update selbst kompilierte Kernelmodule automatisch zu aktualisieren. DKMS besteht aus einigen Shell-Scripts und wurde von Dell entwickelt. Die entsprechenden `dkms`-Pakete stehen gegenwärtig für die Distributionen Debian, Fedora und Ubuntu zur Verfügung.

Um DKMS zu nutzen, muss der Quellcode des Moduls in einem Verzeichnis der Form `/usr/src/NAME-VERSION` installiert werden. Das Verzeichnis muss die Datei `dkms.conf` enthalten, die DKMS erklärt, wie es mit dem Code umgehen soll. Die folgenden Zeilen stammen vom NVIDIA-Treiber für Ubuntu, wobei ich die Formatierung des Listings geändert habe, um die Lesbarkeit zu verbessern. Tatsächlich sind vor und nach dem Zeichen = keine Leerzeichen erlaubt!

```
# Datei /usr/src/nvidia-580.76.05/dkms.conf
PACKAGE_NAME            = "nvidia#"
PACKAGE_VERSION         = "580.76.05"
CLEAN                   = "make clean"
BUILT_MODULE_NAME[0]    = "nvidia"
DEST_MODULE_LOCATION[0] = "/kernel/drivers/char/drm"
PROCS_NUM               = `nproc`
[ $PROCS_NUM -gt 16 ] && PROCS_NUM=16
MAKE[0]                 = "unset ARCH; [ ! -h /usr/bin/cc ] && ..."
BUILT_MODULE_NAME[1]    = "nvidia-modeset"
DEST_MODULE_LOCATION[1] = "/kernel/drivers/char/drm"
BUILT_MODULE_NAME[2]    = "nvidia-drm"
DEST_MODULE_LOCATION[2] = "/kernel/drivers/char/drm"
...
```

Sind diese Voraussetzungen erfüllt, übergeben Sie das Kernelmodul mit `dkms add` der Kontrolle von DKMS, kompilieren es mit `dkms build` für den aktuellen Kernel und installieren es mit `dkms install`. Die folgenden Beispiele beziehen sich wieder auf den NVIDIA-Kerneltreiber. Die Kommandos werden normalerweise nach dem Update des Kernels oder eines Treibers automatisch ausgeführt (Kommando `dkms autoinstall`). Nur wenn dieser Automatismus nicht funktioniert, müssen Sie manuell nachhelfen und sich bei eventuellen Fehlern auf die Suche nach deren Ursachen machen. (Mitunter sind die Versionen des Kernels und des Treibers nicht kompatibel zueinander.)

```
sudo dkms add     -m nvidia -v 580.76.05
sudo dkms build   -m nvidia -v 580.76.05
sudo dkms install -m nvidia -v 580.76.05
```

Die obigen Kommandos gelten für den gerade laufenden Kernel. Um Kernelmodule für eine andere Kernelversion zu kompilieren und zu installieren, fügen Sie den obigen Kommandos die Option `-k n.n` hinzu, wobei `n.n` die Versionsnummer eines auf Ihrem Rechner installierten Kernels ist.

`dkms status` bzw. ein Blick in das Verzeichnis `/var/lib/dkms` verrät, welche Kernelmodule sich momentan unter der Kontrolle von DKMS befinden.

Bei Debian und Ubuntu gibt es eine ganze Reihe von `xxx-name-dkms`-Paketen, mit denen Kernelmodule für Hardware-Treiber kompiliert werden können:

```
sudo apt-cache --names-only search '.*-dkms' | sort
```

Bei Debian befinden sich die Pakete zum Teil in den Paketquellen *contrib* und *non-free*, die Sie eventuell vorher aktivieren müssen. Die Installation eines Kernelmoduls sieht dann wie folgt aus, wobei Sie einfach `nvidia` durch den Namen des gewünschten Treibers und `amd64` durch Ihre CPU-Architektur ersetzen:

```
sudo apt update
sudo apt install linux-headers-n.n-amd64 nvidia-nnn-dkms
```

Im Rahmen der Installation werden alle abhängigen Pakete installiert sowie das Kernelmodul kompiliert und als DKMS-Modul eingerichtet. Bei zukünftigen Kernel-Updates wird somit automatisch eine neue Version des Moduls erzeugt.

DKMS oder module-assistant?

Vor allem in Debian kamen zum Kompilieren von Kernelmodulen in der Vergangenheit häufig die Werkzeuge aus dem Paket `module-assistant` zum Einsatz. Das gleichnamige Kommando bzw. dessen Kurzform `m-a` existiert weiterhin, muss aber extra installiert werden; nach Möglichkeit sollten Sie DKMS-Pakete vorziehen!

akms

Die RPMFusion-Paketquelle für Fedora verwendet anstelle von DKMS einen eigenen Mechanismus zum Kompilieren von Kernelmodulen: Das Kommando `akms` wird nach Kernel-Updates aufgerufen. Es kompiliert zu allen RPM-Source-Paketen, die sich in `/usr/src/akmods` befinden, die entsprechenden Kernelmodule und installiert diese.

https://rpmfusion.org/Packaging/KernelModules/Akmods

26.4 Kernel selbst konfigurieren und kompilieren

Der durchschnittliche Linux-Anwender muss seinen Kernel nicht selbst kompilieren. Bei allen aktuellen Distributionen werden ein brauchbarer Standardkernel und eine umfangreiche Sammlung von Modulen mitgeliefert. Dennoch kann es Gründe geben, den Kernel neu zu kompilieren:

- Sie wollen Ihr System besser kennenlernen. Das Motto dieses Buchs ist es ja, Ihnen auch einen Blick hinter die Linux-Kulissen zu ermöglichen.
- Sie brauchen besondere Funktionen, die weder in den mitgelieferten Kernel integriert sind noch als Modul vorliegen.
- Sie möchten eine aktuellere Version des Kernels verwenden als die, die mit Ihrer Distribution mitgeliefert wurde.
- Sie möchten selbst an der Kernelentwicklung teilnehmen und daher mit dem neuesten Entwicklerkernel experimentieren.
- Sie wollen in Ihrem Bekanntenkreis mit Insider-Wissen auftrumpfen: »Ich habe den neuesten Linux-Kernel selbst kompiliert!«

Hürden

Es gibt allerdings gewichtige Gründe, die gegen das Kompilieren eines eigenen Kernels sprechen:

- Viele Distributionen verwenden nicht den Originalkernel, wie er von Linus Torvalds freigegeben wird, sondern eine gepatchte Version mit diversen Zusatzfunktionen, wobei natürlich jede Distribution andere Patches verwendet.

  An sich ist das eine feine Sache für den Anwender: Er bekommt auf diese Weise Zusatzfunktionen, von denen der Distributor glaubt, dass sie schon ausreichend stabil funktionieren. Wenn Sie sich nun aber selbst den Quellcode des Originalkernels herunterladen, fehlen diese Patches. Einzelne Funktionen Ihrer Distribution, die bisher einwandfrei gearbeitet haben, machen plötzlich Probleme oder funktionieren gar nicht mehr.

- Das Kompilieren eines eigenen Kernels ist nicht schwierig. Schwierig ist aber die vorherige Konfiguration des Kompilationsprozesses. Dabei stehen Tausende von Optionen zur Auswahl. Sie können mit diesen Optionen beeinflussen, welche Funktionen direkt in den Kernel integriert werden, welche als Module und welche gar nicht zur Verfügung stehen sollen.

  Wenn Sie sich – mangels Detailwissen – für die falschen Optionen entscheiden, ist das Ergebnis wie oben: Einzelne Funktionen verweigern den Dienst, und es ist relativ schwierig, die Ursache herauszufinden. Gerade für Linux-Einsteiger ist es praktisch unmöglich, die passenden Einstellungen für alle Optionen richtig zu erraten.

Aus diesen Gründen verweigern die meisten Distributoren jeden Support, wenn Sie nicht den mit der Distribution mitgelieferten Kernel verwenden. Lassen Sie sich von diesen Warnungen aber nicht abschrecken, es einmal selbst zu versuchen. Wenn Sie nach der in diesem Abschnitt präsentierten Anleitung vorgehen, können Sie Ihren Rechner anschließend sowohl mit dem alten als auch mit dem neuen Kernel hochfahren – es kann also nichts passieren!

Entwicklungswerkzeuge

Zur Kompilierung des Kernels sind dieselben Entwicklungswerkzeuge wie zum Kompilieren eines einzelnen Moduls erforderlich (siehe Abschnitt 26.3).

Je nach Distribution sind darüber hinaus weitere Pakete notwendig. Unter Debian und Ubuntu müssen Sie beispielsweise in `/etc/apt/sources.list` die `deb-src`-Paketquellen aktivieren und dann die folgenden Kommandos ausführen:

```
sudo apt update
sudo apt install flex bison
sudo apt build-dep linux
```

Unter Fedora sind häufig die folgenden Pakete nicht automatisch installiert:

```
sudo dnf install dwarves ncurses-devel zstd
```

Kernelversionen

Die Weiterentwicklung des Linux-Kernels erfolgt seit Jahren im gleichen Rhythmus: Sobald Linus Torvalds befindet, dass die gerade in Entwicklung befindliche Kernelversion ausreichend stabil ist, wird diese offiziell freigegeben. Gleichzeitig beginnen die Arbeiten an der nächsten Version: Entwickler können in den nächsten zwei Wochen Patches für neue Funktionen bei Linus Torvalds einreichen. Danach dauert es typischerweise fünf bis sieben Wochen, bis alle Probleme und Fehler im neuen Code behoben sind und die nächste Kernelversion fertig wird.

Die Versionsnummer des Kernels besteht aus drei Teilen: *Major Release Number*, *Minor Release Number* und *Bugfix Release Number*. Als ich dieses Kapitel Ende August 2025 überarbeitete, war die Kernelversion 6.16 aktuell. Diese hatte Linus Torvalds am 27. Juli freigegeben. Bis Ende August gab es nur ein kleineres Update zur Behebung von Bugs und Sicherheitsproblemen. Die vollständige Versionsnummer lautete zu diesem Zeitpunkt 6.16.2.

Long Term Releases

Grundsätzlich wird die Wartung alter Kernelversionen mit der Fertigstellung der jeweils neuesten Version beendet. Ausgenommen von dieser Regel sind ausgewählte Kernelversionen, die durch die Kernelentwicklergemeinde über mehrere Jahre gepflegt werden. Im August 2025 genossen die folgenden Kernelversionen Langzeitunterstützung: 5.4, 5.10, 5.15, 6.1, 6.6 und 6.12.

Für Linux-Distributoren gibt es drei Varianten, mit Kernel-Updates umzugehen:

- Vordergründig am einfachsten wäre es, stets die gerade aktuellste Kernelversion als Update anzubieten. Das kann aber zu Stabilitätsproblemen führen (vor allem dann, wenn weitere Komponenten, z. B. das Grafiksystem, nicht ebenfalls aktualisiert werden), weswegen die meisten Distributoren vor diesem Weg zurückschrecken. Zu den wenigen Ausnahmen zählen Fedora sowie Rolling-Release-Distributionen wie Arch Linux oder openSUSE Tumbleweed.
- Die nächstbeste Variante besteht darin, für die Distribution die gerade aktuellste Long-Term-Linux-Version zu verwenden. Das hat für den Distributor den großen Vorteil, dass er sich um die Pflege des Kernelcodes nicht selbst kümmern muss.
- Am aufwendigsten ist es, wenn ein Distributor nicht auf einen Langzeit-Kernel zurückgreift, sondern den ursprünglich mit der Distribution ausgelieferten Kernel selbst pflegt. Dazu müssen Sicherheits-Updates und fallweise auch Zusatzfunktionen aus aktuellem Kernelcode »rück-portiert« werden. Besonders bemerkenswert ist diesbezüglich Red Hat, das für seine Enterprise-Distributionen mit immensem Arbeitsaufwand ein ganzes Jahrzehnt lang alte Kernelversionen mit Sicherheits-Updates versorgt.

Statistik und Links

Der Kernel besteht zurzeit aus über 40 Millionen Zeilen Code. Der Großteil davon ist in C geschrieben, ein kleiner Teil in Assembler sowie in der Sprache Rust. Wenn Sie wissen möchten, wer bzw. welche Firmen zur Kernelentwicklung beitragen, verfolgen Sie einfach die Linux-News-Site *https://lwn.net*. Dort finden Sie zu jedem Kernel-Release eine statistische Aufarbeitung, wer die meisten Änderungen durchgeführt hat:

https://lwn.net/Articles/1031161 (für Version 6.16)

Tipps zur Kompilierung des Kernels finden Sie auf der folgenden Seite:

https://kernelnewbies.org/FAQ

Wenn Sie sich für technische Interna interessieren, sind die Dokumentationsdateien des Kernelcodes sehr aufschlussreich. Gerade neue Funktionen des Kernels werden zuerst an dieser Stelle beschrieben, noch bevor die entsprechenden `man`-Seiten aktualisiert werden:

https://www.kernel.org/doc/Documentation

Kernelcode installieren

Der Quellcode für den Kernel befindet sich üblicherweise im Verzeichnis `/usr/src/linux`; nur bei Red Hat und Fedora gibt es abweichende Gepflogenheiten, die weiter unten behandelt werden. Falls dieses Verzeichnis leer ist, haben Sie den Kernelcode nicht installiert. Sie können nun wahlweise den Kernelquellcode Ihrer Distribution installieren oder den gerade aktuellen offiziellen Kernelcode herunterladen. Weniger Probleme bereitet zumeist die erste Variante, insbesondere für Einsteiger.

Ist genug Platz auf der SSD?

Der Platzbedarf für den Kernelcode ist beachtlich! Ich habe für meine Tests das offizielle Git-Repository geklont, das an sich schon ca. 8 GiB Platz beansprucht. Beim Kompilieren kommen unzählige Binärdateien hinzu, der Platzbedarf steigt auf ca. 35 GiB. Zuletzt können Sie mit `make clean` zahllose Objektdateien wieder löschen, aber zwischenzeitlich brauchen Sie genug freien Speicherplatz.

Kernelcode der Distribution installieren

Bei den meisten Distributionen gibt es ein eigenes Paket, das den Kernelquellcode enthält (siehe Tabelle 26.4). Dabei ist `n.n` ein Platzhalter für die installierte Kernelversion.

Bei Debian und Ubuntu wird der Kernelcode als `tar`-Archiv in das Verzeichnis `/usr/src` installiert. Sie müssen das Archiv selbst mit `tar xjf linux-n.n.tar.bz2` auspacken. Die Kennung `.bz2` deutet darauf hin, dass der Quellcode mit dem besonders effizienten bzip2-Verfahren komprimiert wurde.

Bei RHEL finden Sie nach `dnf download --source kernel` das Paket `kernel-n.n.src.rpm` im aktuellen Verzeichnis. Mit `rpm -iv` packen Sie es im Verzeichnis `rpmbuild/SOURCE` aus.

Distribution	Kommando
Debian, Ubuntu	`apt install linux-source`
Fedora	`fedpkg clone -a kernel` (Details siehe unten)
RHEL	`dnf download --source kernel` (Quellcodepaket!)
SUSE	`zypper install kernel-source`

Tabelle 26.4 Installation des distributionsspezifischen Kernelquellcodes

Den aktuellen Kernel-Code für Arch Linux laden Sie am besten mit `git` aus dem entsprechenden Arch-Linux-Repo herunter:

```
git clone \
  https://gitlab.archlinux.org/archlinux/packaging/packages/linux.git
```

Fedora ist unter Kernelentwicklern eine besonders beliebte Distribution. Bevor Sie loslegen, müssen Sie diverse Entwicklerpakete installieren und den aktuellen User-Account für `pesign` freischalten. (`pesign` ist ein Kommando zum Signieren von UEFI-Programmen.)

```
sudo dnf install fedpkg fedora-packager rpmdevtools \
      ncurses-devel pesign openssl

sudo pesign

  pesign: Nothing to do.
```

Die Fedora-Entwickler empfehlen, den Quellcode des Kernels sowie aller Fedora-Patches mit `fedpkg clone -a kernel` aus einem Git-Repository herunterzuladen. Details können Sie hier nachlesen:

https://fedoraproject.org/wiki/Building_a_custom_kernel

Offiziellen Kernelcode installieren

Der mit der Distribution mitgelieferte Kernel ist oft schon veraltet. Den aktuellen Kernelcode in Form von komprimierten `tar`-Archiven finden Sie hier:

https://www.kernel.org

Anstatt eine bestimmte Kernelversion herunterzuladen und mit ihr zu arbeiten, kann es sinnvoller sein, einen Klon des offiziellen Git-Repositorys für stabile Kernelversionen zu erzeugen. `git tag` ermittelt in Kombination mit `sort -V` (numerische Sortierordnung) die aktuellsten

zehn im Code enthaltenen Versionen. `git checkout` aktiviert die Version, die Sie anschließend tatsächlich nutzen (kompilieren) möchten – hier den Release Candidate 3 der damals noch in Entwicklung befindlichen Version 6.17:

```
git clone \
  git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git

cd linux-stable

git tag -l | sort -V | tail -n 10

  v6.16-rc4
  v6.16-rc5
  v6.16-rc6
  v6.16-rc7
  v6.16.1
  v6.16.2
  v6.16.3
  v6.17-rc1
  v6.17-rc2
  v6.17-rc3

git checkout v6.17-rc3
```

Anfänglich wird durch `git clone` zwar deutlich mehr Code heruntergeladen (Download-Umfang knapp 6 GiB, Platzbedarf auf der SSD ca. 8 GiB); dafür verlaufen spätere Updates bzw. Versionswechsel aber einfacher und schneller – elementare Grundkenntnisse des `git`-Kommandos einmal vorausgesetzt.

Beachten Sie, dass es sich hier um den originalen Kernelcode handelt, wie ihn Linus Torvalds freigegeben hat – also ohne distributionsspezifische Patches. Dieser Kernel wird oft *Vanilla Kernel* genannt.

Varianten und Entwicklerzweige

Es gibt im Internet unzählige weitere Git-Repositorys mit diversen Varianten und Entwicklungszweigen des Kernelcodes. Werfen Sie insbesondere einen Blick in das Blog des Kernelentwicklers Greg Kroah-Hartman (vierter Link)!

https://git.kernel.org
https://github.com/torvalds/linux
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
http://www.kroah.com/log/blog/2019/06/15/linux-stable-tree-mirror-at-github

Kernelkonfiguration

Der Kernel besteht aus Tausenden von Einzelfunktionen bzw. Komponenten. Bei nahezu allen Funktionen können Sie vor dem Kompilieren angeben, ob sie direkt in den Kernel integriert werden, als Modul kompiliert werden oder gar nicht verfügbar sein sollen. Dieser Vorgang heißt den »Kernel konfigurieren«.

Die Kernelkonfiguration wird durch die Datei `.config` im Verzeichnis mit dem Kernelquellcode bestimmt. Dabei handelt es sich um eine aktuell ca. 13.000 Zeilen lange Textdatei, die angibt, ob eine Funktion direkt in den Kernel integriert (`name=y`) oder als Modul kompiliert werden soll (`name=m`). Nicht benötigte Funktionen erscheinen in der Konfigurationsdatei nicht bzw. nur in Kommentarzeilen. Die Datei kann auch zusätzliche Einstellungen enthalten (`name=wert`). Die folgenden Zeilen zeigen einige Zeilen einer `.config`-Datei:

```
CONFIG_64BIT=y
CONFIG_X86_64=y
CONFIG_X86=y
CONFIG_INSTRUCTION_DECODER=y
CONFIG_OUTPUT_FORMAT="elf64-x86-64"
...
```

Wenn Sie bei der manuellen Kernelkonfiguration (siehe den folgenden Abschnitt) keinen Ausgangspunkt haben, müssen Sie sich wirklich um alle Kerneloptionen kümmern. Gerade beim ersten Mal ist es so gut wie sicher, dass Sie irgendetwas übersehen werden. Sie sparen eine Menge Zeit und Mühe, wenn Sie die mit Ihrer Distribution mitgelieferte Kernelkonfigurationsdatei als Startbasis verwenden:

```
cp old-config /pfad/zum/code/linux-n.n/.config
```

Dieses Verfahren hat leider einen Nachteil: Wenn der ursprüngliche Kernelcode andere Patches enthält als der neu zu kompilierende Code, enthält auch die ursprüngliche Konfigurationsdatei Optionen, die im neuen Code nicht vorgesehen sind. Das kann zu Problemen führen. Manche Distributoren bauen Patches in ihren Kernel ein, die im Standardkernel nicht enthalten sind. Deswegen müssen Sie anschließend in das Quellcodeverzeichnis wechseln und dort das folgende Kommando ausführen:

```
cd /pfad/zum/code/linux-n.n
make oldconfig                  # bei Unklarheiten rückfragen
```

`make oldconfig` wertet die vorhandene `.config`-Datei aus. Fehlen dort Optionen, die der aktuelle Kernelcode vorsieht, dann werden entsprechende Rückfragen angezeigt. In der Regel können Sie einfach mit ↵ antworten; dann gelten für diese Optionen die von den Entwicklern vorgeschlagenen Defaulteinstellungen. Genau das macht – ohne jede Rückfrage – das `make`-Kommando `olddefconfig`:

```
make olddefconfig               # bei Unklarheiten Defaults verwenden
```

Aktuelle Konfiguration feststellen

Jetzt bleibt noch die Frage offen, woher Sie die aktuelle Kernelkonfigurationsdatei für das Kommando `cp old-config` nehmen. Bei nahezu allen Distributionen befindet sich im Verzeichnis `/boot` die zum laufenden Kernel passende Konfigurationsdatei, also z. B. `/boot/config-n.n`. Somit wird aus `cp old-config` beispielsweise:

```
cd /pfad/zum/code/linux-n.n
cp /boot/config-6.2.11-300.fc38.x86_64.config .config
make oldconfig
```

Der mit SUSE mitgelieferte Kernel verwendet die `cloneconfig`-Option (Gruppe *General setup*). Das bedeutet, dass `/proc/config.gz` den komprimierten Inhalt der `.config`-Datei enthält, mit der der gerade laufende Kernel kompiliert wurde. Mit `make cloneconfig` kopieren Sie die zuletzt verwendete Konfiguration in die Datei `.config`.

Kernel manuell konfigurieren

Egal, ob Sie mit einer Grundkonfiguration oder bei null starten – mit `make xxxconfig` (auf die verschiedenen Varianten gehe ich im nächsten Abschnitt ein) können Sie nun sämtliche Kerneloptionen nach Ihrem Bedarf verändern bzw. einstellen. Dabei müssen Sie sich zwischen zwei prinzipiellen Kerneltypen entscheiden: einem monolithischen Kernel oder einem modularisierten Kernel. Ein monolithischer Kernel enthält alle benötigten Treiber und unterstützt keine Module. Modularisierte Kernel sind über die integrierten Treiber hinaus in der Lage, im laufenden Betrieb zusätzliche Module aufzunehmen. Ein modularisierter Kernel ist in fast allen Fällen die bessere Entscheidung.

Bei den meisten Kernel-Komponenten und -Treibern haben Sie die Wahl zwischen drei Optionen: YES, MODULE und NO.

- YES bedeutet, dass diese Komponente direkt in den Kernel integriert wird.
- MODULE bedeutet, dass diese Komponente als Modul kompiliert wird (nur sinnvoll bei einem modularisierten Kernel).
- NO bedeutet, dass die Komponente überhaupt nicht kompiliert wird.

Es gibt auch eine Reihe von Funktionen, die nicht als Module zur Verfügung gestellt werden können – dort reduziert sich die Auswahl auf YES oder NO.

Die übliche Vorgehensweise besteht darin, in den modularisierten Kernel nur relativ wenige elementare Funktionen zu integrieren und alle anderen Funktionen als Module verfügbar zu machen. Der Vorteil: Der Kernel an sich ist relativ klein, Module werden nur nach Bedarf nachgeladen.

Eine alternative Strategie besteht darin, einen monolithischen Kernel möglichst exakt für die eigenen Hard- und Software-Ansprüche zu optimieren. Alle Funktionen, die genutzt werden

sollen, integrieren Sie direkt in den Kernel. Bei allen anderen Komponenten entscheiden Sie sich für No.

Generell wird ein monolithischer Kernel immer etwas größer als ein modularisierter Kernel. Dafür funktioniert er ohne die dynamische Modulverwaltung, und der Rechnerstart ist unter Umständen ohne Initrd-Datei möglich. (Das ist abhängig von den Randbedingungen. Eine Initrd-Datei kümmert sich, losgelöst von den Kernelmodulen, auch um die initiale Einstellung der Tastatur, die eventuell notwendige Eingabe von LUKS-Passwörtern für verschlüsselte Dateisysteme etc. Nur wenn Sie all diese Funktionen nicht brauchen, ist ein Start ohne Initrd-Datei denkbar.)

Der Nachteil eines monolithischen Kernels ist offensichtlich: Wenn Sie eine bestimmte Funktion später brauchen, müssen Sie den Kernel neu kompilieren. Und nur echte Linux-Profis können abschätzen, welche Funktionen sie nutzen werden.

Werkzeuge zur manuellen Kernelkonfiguration

Um abweichend von der aktuellen Konfiguration einzelne Einstellungen zu verändern, können Sie `.config` manuell editieren. Das ist aber fehleranfällig und erfordert eine gute Kenntnis der Namen der diversen Optionen. Besser ist es daher, mit `make xxxconfig` ein spezielles Konfigurationsprogramm zu starten. Dabei stehen unterschiedliche Varianten zur Verfügung, die Sie mit einem der aufgelisteten `make`-Kommandos starten:

```
cd /pfad/zum/code/linux-n.n
make menuconfig        # dialoggeführte Konfiguration im Textmodus
make nconfig           # dialoggeführte Konfiguration im Textmodus
make localmodconfig    # autom. Konfiguration für die aktuelle Hardware
```

`make menuconfig` und `make nconfig` setzen voraus, dass Sie vorher das Paket `ncurses-devel` bzw. `libncurses6-dev` installiert haben. Die Konfiguration erfolgt ebenfalls im Textmodus. Der große Vorteil im Vergleich zu `make config` besteht darin, dass die Einstellung der unzähligen Optionen durch verschachtelte Dialoge strukturiert ist (siehe Abbildung 26.1). Die Gestaltung der Dialoge und die Tastenkürzel variieren ein wenig, in ihrer Funktionsweise sind beide Varianten aber recht ähnlich.

`make localmodconfig` ist eine interessante Kompiliervariante für alle, die es eilig haben. Dabei werden nur die Module kompiliert, die im gerade laufenden Kernel tatsächlich genutzt werden. Das hat Vor- und Nachteile: Der offensichtliche Vorteil besteht darin, dass wirklich nur der Teil des Kernelcodes übersetzt wird, der tatsächlich benötigt wird. Das kann die Übersetzungszeit auf ein Drittel senken!

Allerdings läuft der so kompilierte Kernel auf einem anderen Rechner unter Umständen nicht, wenn für seine Hardware-Komponenten relevante Treiber fehlen. Auch das Nachladen eines Moduls, das zur Kompilierzeit nicht aktiv war, wird scheitern. Der Kernel ist also nur

zu Testzwecken geeignet, nicht aber für eine längerfristige Nutzung. Detailinformationen zu dieser `make`-Variante können Sie hier nachlesen:

https://heise.de/-1402386

Wenn Sie nur einzelne Optionen an einer vorgegebenen Kernelkonfiguration ändern möchten, können Sie auf `make xxxconfig` ganz verzichten. Stattdessen geben Sie die gewünschten Einstellungen in der getrennten Datei `config-local` an. Die hier gewählten Optionen haben Vorrang gegenüber `.config`.

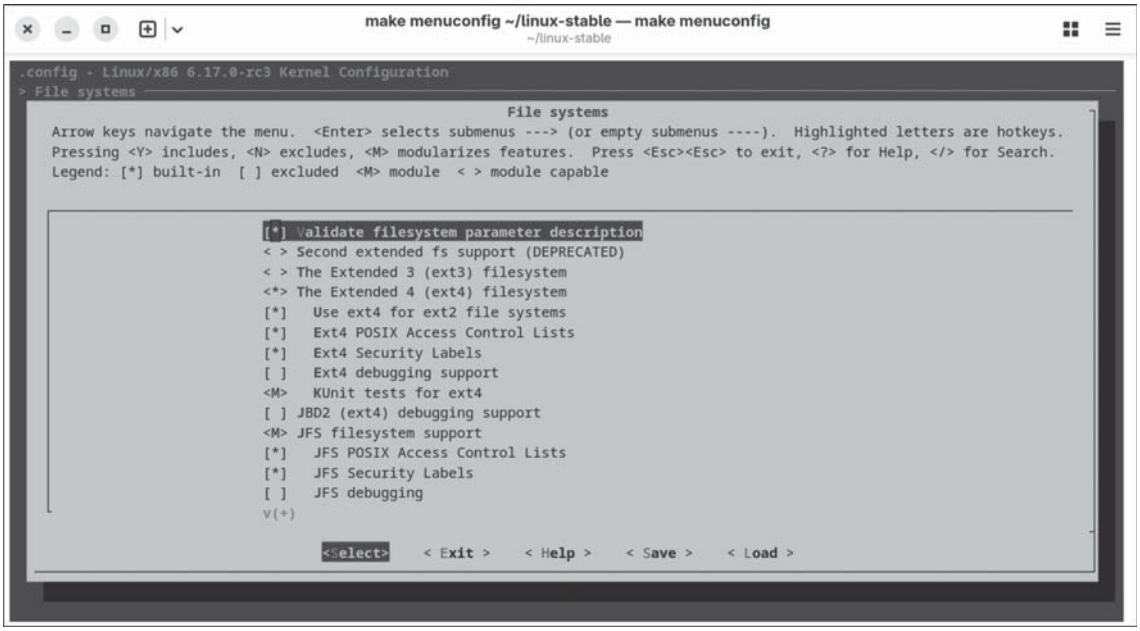

Abbildung 26.1 Kernelkonfiguration mit »make menuconfig«

Kernel kompilieren und installieren

Nachdem Sie mit der Konfiguration des Kernels vermutlich einige Zeit verbracht haben, muss jetzt der Rechner arbeiten. Die folgenden Kommandos beschäftigen einen zeitgemäßen Rechner circa 15 bis 30 Minuten. Die Option `-j N` gibt an, wie viele Prozesse `make` parallel starten soll. Das Kommando `nproc` ermittelt, wie viele virtuelle Cores zur Verfügung stehen. (Ich habe meine Tests auf einem Mini-PC mit der AMD-CPU 8745H durchgeführt. Die CPU besitzt 8 Cores und nutzt Hyperthreading, d. h. `nproc` liefert 16. Das `make`-Kommando nutzt somit alle Threads und führte zu einer nahezu hundertprozentigen CPU-Auslastung für 18 Minuten.)

```
cd /pfad/zum/code/linux-n.n
make -j$(nproc) all          # alles kompilieren, alle CPU-Cores nutzen
```

Das Ergebnis am Ende dieses Prozesses ist die Datei `bzImage` im Verzeichnis `/pfad/zum/code/linux-n.n/arch/x86/boot`. Die Größe der Datei liegt meist zwischen 10 und 20 MiB und hängt davon ab, wie viele Funktionen direkt in den Kernel inkludiert sind und wie viele als Module bzw. überhaupt nicht kompiliert wurden.

Module und Kernel installieren

`make modules_install` kopiert die Moduldateien dorthin, wo die Kommandos zur Modulverwaltung (etwa `insmod`) diese erwarten: in das Verzeichnis `/lib/modules/n.n`. Dabei ist `n.n` die Versionsnummer des soeben kompilierten Kernels. Während das Kompilieren ohne `root`-Rechte klappt, muss dieses Kommando mit `sudo` ausgeführt werden.

```
sudo make modules_install        # Module samt Debug-Infos installieren
```

Standardmäßig enthalten die neu erzeugten Kernelmodule eine Menge Debugging-Informationen. Sie sind deswegen viel größer als »gewöhnliche« Module und führen in der Folge zu riesigen Initrd-Dateien (bei meinen Tests z. B. 270 MiB anstatt 80 MiB!). Sofern Sie nicht vorhaben, sich auf die Suche nach Kernelfehlern zu begeben, sollten Sie die Debugging-Informationen aus den Modulen entfernen. (In der Kerneldokumentation wird dieser Vorgang »stripping« genannt.) Anstelle des obigen Kommandos führen Sie dazu die folgende Variante aus:

```
sudo make INSTALL_MOD_STRIP=1 modules_install # nur Module installieren
```

Der frisch erzeugte neue Kernel ist natürlich noch nicht aktiv. Bisher wurde nur eine Menge neuer Dateien erstellt, sonst nichts! Der neue Kernel kann erst beim nächsten Start von Linux aktiviert werden und auch dann nur, wenn Sie den Kernel in das `/boot`-Verzeichnis kopieren und eine dazu passende Initrd-Datei erzeugen. Außerdem ist es üblich, die beim Kompilieren verwendete Datei `.config` unter dem Namen `config-n.n` im `/boot`-Verzeichnis zu speichern. Damit wird dokumentiert, wie Ihr Kernel kompiliert wurde.

Dankenswerterweise kümmert sich `make install` um sämtliche gerade zusammengefassten Punkte:

```
sudo make install      # Kerneldateien installieren
```

Falls Ihr System Kernelmodule benötigt, deren Code außerhalb des offiziellen Kernelcodes liegt (typischerweise der NVIDIA-Treiber), müssen auch diese Module neu kompiliert und berücksichtigt werden. Das erfolgt bei vielen Distributionen mittels DKMS. Beachten Sie, dass das Kompilieren des NVIDIA-Treibers bei neuen Kernelversionen oft scheitert bzw. die allerneueste Version des NVIDIA-Treibers voraussetzt!

Ich habe meine Tests auf einem Rechner mit deaktiviertem UEFI Secure Boot durchgeführt. Wenn Sie Secure Boot nutzen möchten, müssen Sie einen *Machine Owner Key* (MOK) erstellen, mit `mokutil` in die MOK-Datenbank einfügen und den Kernel sowie alle Module damit signieren.

Details zum Signiervorgang können Sie hier nachlesen:

https://docs.fedoraproject.org/en-US/quick-docs/kernel-build-custom/#_secure_boot
https://wiki.archlinux.org/title/Unified_Extensible_Firmware_Interface/Secure_Boot
https://wiki.debian.org/SecureBoot#MOK_-_Machine_Owner_Key

Zuletzt müssen Sie die GRUB-Konfiguration aktualisieren:

```
sudo update-grub                                    # Debian, Ubuntu
sudo grub2-mkconfig -o /boot/grub2/grub.cfg         # Fedora, RHEL
```

Neustart und Aufräumarbeiten

Ob alles funktioniert hat, merken Sie beim Neustart:

```
uname -a

  Linux fedora 6.17.0-rc3 ...
```

Sollte der neue Kernel aus irgendeinem Grund nicht funktionieren, starten Sie den Rechner einfach mit dem bisherigen Kernel und unternehmen einen weiteren Versuch, den Kernel richtig zu konfigurieren und neu zu kompilieren. Läuft der neue Kernel dagegen zufriedenstellend, sollten Sie die nun nicht mehr benötigten Objekt-Dateien des Compilers aufräumen. Sie gewinnen auf diese Weise 25 bis 30 GiB Platz auf Ihrer SSD zurück! Allerdings verlängert sich ein eventueller weiterer Kompiliervorgang dadurch erheblich.

```
cd /pfad/zum/code/linux-n.n
make clean
```

26.5 Kernel-Live-Patches

Die meisten Updates eines Linux-Systems können im laufenden Betrieb erfolgen. Aktualisierte Netzwerkdienste müssen zwar anschließend neu gestartet werden, aber es besteht keine Notwendigkeit, den ganzen Rechner neu zu starten. Die Ausnahme von dieser Regel ist der Kernel: Damit Sicherheits-Updates im Kernel wirksam werden, müssen Sie einen neuen Kernel und neue Module installieren und anschließend den ganzen Rechner neu starten.

Auf Rechnern, die jeden Tag oder zumindest einmal pro Woche ein- und ausgeschaltet werden, ist das egal. Aber bei Servern, die möglichst ohne Unterbrechung ständig verfügbar sein sollen, ist ein Neustart zumeist unerwünscht. Und selbst Administratoren, die Updates automatisieren, scheuen in der Regel davor zurück, auch die erforderlichen Neustarts zu automatisieren. Zu groß ist die Gefahr, dass etwas schiefgeht und dann (zumeist mitten in der Nacht) keiner Zeit hat, das Problem sofort zu beheben.

Ksplice (Oracle)

Die erste Lösung für dieses Problem bot die Funktion *Ksplice*: Bei vielen Updates ist es möglich, die betreffende Kernelfunktion im laufenden Betrieb zu deaktivieren und durch neuen Code zu ersetzen. Die nicht ganz trivialen technischen Hintergründe des Verfahrens sind auf den folgenden Seiten beschrieben:

https://ksplice.oracle.com
https://lwn.net/Articles/340477
https://en.wikipedia.org/wiki/Ksplice

2011 übernahm Oracle die Firma Ksplice. Mit der Funktion Ksplice durchgeführte Kernel-Updates waren über mehrere Jahre ein durchaus gewichtiges Unterscheidungsmerkmal zu anderen Enterprise-Linux-Versionen. Oracle bietet Ksplice allerdings nur zahlenden Kunden an. Ksplice steht zwar als Open-Source-Code zur Verfügung, ist aber nicht Bestandteil des offiziellen Linux-Kernels.

kPatch und kGraft (Red Hat und SUSE)

Red Hat und SUSE wollten in dieser Hinsicht natürlich nicht zurückstecken und entwickelten unter den Namen *kPatch* und *kGraft* vergleichbare Update-Mechanismen. Die beiden Mechanismen stehen seit 2014 in den Enterprise-Versionen von Red Hat und SUSE zur Verfügung. Die für kPatch und kGraft erforderlichen Funktionen (gewissermaßen der größte gemeinsame Nenner) wurden von Linus Torvalds 2015 in den offiziellen Linux-Kernel aufgenommen. Ob Ihr Kernel die Funktionen enthält, erkennen Sie am Vorhandensein des Verzeichnisses `/sys/kernel/livepatch`. Allerdings stellen auch Red Hat und SUSE geeignete Live-Kernel-Patches nur zahlenden Kunden zur Verfügung.

https://github.com/dynup/kpatch
https://en.wikipedia.org/wiki/KGraft

Ubuntu

Canonical trat zuletzt in das Live-Patching-Geschäft ein. Seit Ende 2016 bietet Canonical seinen kommerziellen Kunden für kritische Sicherheitsprobleme in Ubuntu-LTS-Versionen Live-Patches an, zuerst im Rahmen der Landscape-Dienste, mittlerweile unter der Bezeichnung »Ubuntu Pro«. Das zugrunde liegende Programm `canonical-livepatch` greift dabei auf die vorhin erwähnte Live-Patch-Infrastruktur im Kernel zurück.

Erfreulicherweise stellt Canonical die Live-Patches in beschränktem Umfang auch nichtkommerziellen Anwenderinnen und Anwendern zur Verfügung: Sofern Sie über ein kostenloses Ubuntu-One-Konto verfügen, erhalten Sie einen Token, mit dem Sie fünf Rechner in das Ubuntu-Pro-Programm aufnehmen können:

```
sudo apt install pro      # ist in der Regel bereits installiert

sudo pro attach <token>

  ...
  This machine is attached to 'Ubuntu Pro - free personal subscription'

  SERVICE     ENTITLED  STATUS    DESCRIPTION
  esm-apps         yes  enabled   Expanded Security Maintenance for ...
  esm-infra        yes  enabled   Expanded Security Maintenance for ...
  fips             yes  disabled  NIST-certified core packages
  fips-updates     yes  disabled  NIST-certified core packages with ...
  livepatch        yes  enabled   Canonical Livepatch service
  usg              yes  disabled  Security compliance and audit tools
```

Im Rahmen der Ubuntu-Pro-Aktivierung wird auch der Live-Patch-Dienst eingerichtet (siehe die vorletzte Zeile im obigen Listing). Die erforderliche Software steht ausschließlich als Snap-Paket zur Verfügung. Den Live-Patch-Status können Sie mit `canonical-livepatch status` ermitteln:

```
sudo canonical-livepatch status

  last check: 22 minutes ago
  kernel: 6.8.0-64.67-generic
  server check-in: succeeded
  kernel state: kernel series 6.8 is covered by Livepatch
  patch state:  no livepatches available for kernel 6.8.0-64.67-generic
  tier: updates (Free usage; This machine beta tests new patches.)
```

Achten Sie auf das Kleingedruckte!

Die Statusmeldung enthält einen Hinweis, der leicht zu übersehen ist: *This machine beta tests new patches*. Konkret bedeutet das, dass Sie als nicht zahlender Ubuntu-Pro-Nutzer ein Beta-Tester für Kernel-Updates sind! Canonical liefert Kernel-Patches zuerst an die nicht zahlenden Kunden aus. Wenn das zu keinen Problemen führt, werden die gleichen Updates später auch den zahlenden Kunden zur Verfügung gestellt.

Grundsätzlich ist diese Vorgehensweise nachvollziehbar. Ich nutze das kostenlose Ubuntu-Pro-Angebot auf mehreren Rechnern/Servern und bin bisher sehr zufrieden damit. Ärgerlich ist aber die intransparente Informationspolitik von Canonical. Obwohl ich eine Weile danach gesucht habe, bin ich im Internet auf kein offizielles Dokument gestoßen, das deutlich auf den Beta-Charakter der Kernel-Updates bei der kostenlosen Nutzung hinweist.

Wenn Sie detaillierte Informationen zu den behobenen Sicherheitslücken wünschen, geben Sie zusätzlich die Option `--verbose` an. `dmesg | grep livepatch` liefert Meldungen über zuletzt durchgeführte Live-Patches.

Live-Patches nur für Notfälle

Den Code des laufenden Kernels zu verändern, ist ein diffiziler Vorgang (wenn Sie dramatische Vergleiche mögen: wie die Operation am offenen Herzen). Deswegen wird dieser Mechanismus aktuell von allen Distributoren nur für gefährliche Sicherheitslücken im Kernel verwendet.

Bei sonstigen Korrekturen wird wie bisher ein neuer Kernel installiert, auf die Aktivierung der dort durchgeführten Änderungen via Live-Patches aber verzichtet. Somit kann es trotz aktivierter Live-Patches sein, dass Ihre Distribution nach der Installation von (Kernel-)Updates meldet, dass ein Neustart erforderlich ist. Lassen Sie sich davon nicht verunsichern.

26.6 Die Verzeichnisse /proc und /sys

Die Verzeichnisse `/proc` und `/sys` werden während des Systemstarts in das Dateisystem eingebunden. Sie dienen dazu, Informationen über den Kernel, laufende Prozesse, geladene Module und viele andere Parameter auf eine transparente Art und Weise sichtbar zu machen. Intern sind die Verzeichnisse `/proc` und `/sys` als virtuelle Dateisysteme realisiert. Sie enthalten also keine echten Dateien und beanspruchen daher auch keinen Platz auf der Festplatte. Das gilt auch für die scheinbar sehr große Datei `/proc/kcore`, die den Arbeitsspeicher abbildet.

Die meisten der `/proc`- und `/sys`-Dateien liegen im Textformat vor. Um die Dateien zu lesen, müssen Sie unter Umständen `cat` statt `less` verwenden, weil manche `less`-Versionen mit virtuellen Dateien nicht zurechtkommen.

Das `/proc`-Verzeichnis liefert interne Kernelinformationen sowie Daten zu allen laufenden Prozessen (siehe Tabelle 26.5). Unter anderem ist dort jedem Prozess ein eigenes Unterverzeichnis zugeordnet. Innerhalb des Prozessverzeichnisses befinden sich dann einige Dateien mit diversen Verwaltungsdaten (z. B. die zum Start verwendete Kommandozeile). Diese Daten werden von diversen Kommandos zur Prozessverwaltung (z. B. `top`, `ps` etc.) ausgewertet.

Datei	Bedeutung
`/proc/N/*`	Informationen zum Prozess mit der PID=`N`
`/proc/asound`	ALSA (Advanced Linux Sound Architecture)
`/proc/bus/usb/*`	USB-Informationen

Tabelle 26.5 Wichtige /proc-Dateien

Datei	Bedeutung
/proc/bus/pccard/*	PCMCIA-Informationen
/proc/bus/pci/*	PCI-Informationen
/proc/cmdline	an den Kernel übergebene Parameter
/proc/config.gz	Kernelkonfigurationsdatei (SUSE)
/proc/cpuinfo	CPU-Informationen
/proc/devices	Nummern von aktiven Devices
/proc/fb	Informationen zum Frame-Buffer
/proc/filesystems	im Kernel enthaltene Dateisystemtreiber
/proc/interrupts	Nutzung der Interrupts
/proc/lvm/*	Nutzung des Logical Volume Managers
/proc/mdstat	RAID-Zustand
/proc/modules	aktive Module
/proc/mounts	aktive Dateisysteme
/proc/net/*	Netzwerkzustand und -nutzung
/proc/partitions	Partitionen der Festplatten
/proc/scsi/*	SCSI/SATA-Geräte und -Controller
/proc/sys/*	System- und Kernelinformationen
/proc/uptime	Zeit in Sekunden seit dem Rechnerstart
/proc/version	Kernelversion

Tabelle 26.5 Wichtige /proc-Dateien (Forts.)

Das /sys-Verzeichnis enthält teilweise dieselben Informationen wie /proc, allerdings sind die Daten systematischer organisiert (siehe Tabelle 26.6). Das Ziel des /sys-Verzeichnisses ist es, den Zusammenhang zwischen dem Kernel und der Hardware abzubilden.

Datei	Bedeutung
/sys/block/*	Informationen über alle Block-Devices (Festplatten etc.)
/sys/bus/*	Informationen über alle Bus-Systeme (PCI, SCSI, USB etc.)

Tabelle 26.6 Wichtige /sys-Dateien

Datei	Bedeutung
/sys/class/*	Informationen über Device-Klassen (Bluetooth, Grafik etc.)
/sys/devices/*	Informationen über angeschlossene Hardware-Komponenten
/sys/firmware/*	Informationen über Hardware-Treiber und -Firmware
/sys/kernel/*	Informationen über den Kernel
/sys/module/*	Informationen über geladene Module
/sys/power/*	Informationen über die Energieverwaltung

Tabelle 26.6 Wichtige /sys-Dateien (Forts.)

26.7 Kernel-Boot-Optionen

Nicht immer, wenn ein Detail im Kernel geändert werden soll, muss der Kernel gleich neu kompiliert werden! Es gibt zwei Möglichkeiten, ohne ein Neukompilieren auf den Kernel Einfluss zu nehmen:

- Zum einen können Sie mit dem Bootloader während des Systemstarts Parameter an den Kernel übergeben. Dieser Mechanismus ist Thema dieses Abschnitts.
- Zum anderen können Sie eine Reihe von Kernelfunktionen dynamisch – also im laufenden Betrieb – verändern. Diese Art des Eingriffs ist insbesondere zur Steuerung von Netzwerkfunktionen gebräuchlich und wird in Abschnitt 26.8, »Kernelparameter verändern«, beschrieben.

GRUB

Bei der Konfiguration von GRUB können Sie in der Zeile `linux` bzw. in der Datei `/etc/default/grub` Kernel-Boot-Optionen angeben. Derartige Optionen können Sie auch interaktiv beim Start eines Linux-Installationsprogramms oder beim Start von GRUB über die Tastatur eintippen (siehe Abschnitt 24.3, »GRUB-Bedienung (Anwendersicht)«). Die Syntax für die Angabe von Optionen sieht so aus:

```
linux /boot/vmlinux-n.n optionA=parameter optionB=parameter1,parameter2
```

Die Parameter zu einer Option müssen ohne Leerzeichen angegeben werden. Mehrere Optionen müssen durch Leerzeichen voneinander getrennt werden, nicht durch Kommata. Hexadezimale Adressen werden in der Form `0x1234` angegeben. Ohne vorangestelltes `0x` wird die Zahl dezimal interpretiert.

Kernel-Boot-Optionen helfen oft dabei, Hardware-Probleme zu umgehen. Wenn der Linux-Kernel beispielsweise aufgrund eines fehlerhaften BIOS nicht erkennt, wie viel RAM Ihr Rechner hat, geben Sie den korrekten Wert mit dem Parameter `mem=` an.

Beachten Sie, dass die beim Linux-Start angegebenen Parameter nur Einfluss auf die in den Kernel integrierten Treiber haben! Parameter für Kernelmodule müssen dagegen in der Datei `/etc/modprobe.conf` bzw. in den Verzeichnissen `/etc/modprobe.d` oder `/etc/modules-load.d` angegeben werden.

Dieser Abschnitt beschreibt nur die wichtigsten Kernel-Boot-Optionen. Weitere Informationen erhalten Sie mit `man bootparam` sowie auf den folgenden Seiten:

https://tldp.org/HOWTO/BootPrompt-HOWTO.html
https://www.kernel.org/doc/Documentation/admin-guide/kernel-parameters.txt

Wichtige Kernel-Boot-Optionen

- `root=/dev/sdb3`: Die `root`-Option gibt an, dass nach dem Laden des Kernels die dritte primäre Partition des zweiten SATA-Laufwerks als Systempartition für das Root-Dateisystem verwendet werden soll. Analog können natürlich auch andere Laufwerke und Partitionen angegeben werden.

  Wenn die Partition mit einem Label bezeichnet ist, kann die Systempartition auch in der Form `root=LABEL=xxx` angegeben werden. Bei `ext`-Partitionen ermitteln Sie den Partitionsnamen mit `e2label` bzw. verändern ihn mit `tune2fs`.

  Eine weitere Variante ist die Angabe der Systempartition durch `root=UUID=nnn`, wobei `nnn` die UUID des Dateisystems ist. Im interaktiven Betrieb von GRUB ist die Eingabe von UUIDs allerdings sehr mühsam. Die UUID ermitteln Sie im laufenden Betrieb mit `blkid <device>`.

- `ro`: Die Option `ro` gibt an, dass das Dateisystem vorerst *read-only* gemountet werden soll. Das ist (in Kombination mit einer der beiden folgenden Optionen) praktisch, wenn ein defektes Dateisystem manuell repariert werden muss.

- `init`: Nach dem Kernelstart wird bei den meisten Distributionen systemd gestartet (siehe Kapitel 25). Wenn Sie dies nicht wollen, können Sie mit der Option `init` ein anderes Programm angeben.

  Mit `init=/bin/sh` erreichen Sie beispielsweise, dass eine Shell gestartet wird. Die Option kann Linux-Profis helfen, ein Linux-System wieder zum Laufen zu bringen, wenn bei der Init-Konfiguration etwas schiefgegangen ist. Beachten Sie, dass das root-Dateisystem nur read-only zur Verfügung steht. (Das können Sie mit `mount -o remount` ändern, siehe Abschnitt 23.8, »mount und /etc/fstab«.) Beachten Sie außerdem, dass in der Konsole das US-Tastaturlayout gilt und dass die `PATH`-Variable noch leer ist.

- `single` oder `emergency`: Wenn Sie eine dieser Optionen verwenden, startet der Rechner im Single-User-Modus (Init-V) bzw. im Rescue-Modus (systemd). Genau genommen werden diese Optionen nicht vom Kernel ausgewertet, sondern so wie alle unbekannten Optionen an das erste vom Kernel gestartete Programm weitergegeben (siehe Kapitel 25, »systemd«).
- `initrd=name`: `initrd` gibt den Namen der zu ladenden Initial-RAM-Disk-Datei an. Wenn Sie *keine* Initrd-Datei verwenden möchten, geben Sie `initrd=` oder `noinitrd` an.
- `ipv6.disable=1`: Diese Option deaktiviert alle IPv6-Funktionen des Kernels.
- `reserve=0x300,0x20`: Diese Option gibt an, dass die 32 Bytes (hexadezimal 0x20) zwischen 0x300 und 0x31F von keinem Hardware-Treiber angesprochen werden dürfen, um darin nach irgendwelchen Komponenten zu suchen. Die Option ist bei manchen Komponenten notwendig, die auf solche Tests allergisch reagieren. Sie tritt im Regelfall in Kombination mit einer zweiten Option auf, die die exakte Adresse der Komponente angibt, die diesen Speicherbereich für sich beansprucht.
- `pci=bios|nobios|nommconf`: Diese Option steuert, wie die Hardware-Erkennung von PCI-Komponenten erfolgt. Sollten dabei Probleme auftreten, hilft manchmal `pci=bios` oder `pci=nommconf`.
- `quiet`: Diese Option bewirkt, dass während des Kernelstarts keine Meldungen auf dem Bildschirm dargestellt werden.
- `video=1024x768`: Mit dieser Option kann per *Kernel Mode Setting* (KMS) die gewünschte Grafikauflösung eingestellt werden, falls der Kernel nicht selbst die optimale Auflösung wählt, z. B. wenn das Video-Signal über einen KVM-Switch geleitet wird. Die Option kann auch helfen, die Installation einer Linux-Distribution in einer geringeren Auflösung durchzuführen (praktisch bei HiDPI-Monitoren, wenn die Schrift unleserlich klein ist).

  Wenn Sie auch die Farbtiefe (z. B. 24 Bit) und die Bildfrequenz angeben möchten, sieht die Syntax so aus: `video=1280x800-24@60` Die Einstellung der Grafikdaten funktioniert nur bei KMS-kompatiblen Treibern. Die `video`-Einstellung gilt normalerweise für alle angeschlossenen Monitore. Wenn Sie die Auflösung nur für einen einzelnen Monitor ändern möchten, geben Sie den entsprechenden Signalausgang an, z. B. `video=VGA-1:1024x768`.
- `nomodeset`: Diese Option deaktiviert das Kernel Mode Setting (KMS). Sie verhindert, dass bei einem Notebook mit NVIDIA-Grafikkarte, aber ohne die erforderlichen NVIDIA-Treiber der Bildschirm beim Start des Grafiksystems schwarz wird.
- `mitigations=auto|auto,nosmt|off`: Diese Option steuert, welche Schutzmaßnahmen der Kernel gegen CPU-Fehler ergreifen soll (siehe Abschnitt 26.9, »Spectre und Meltdown«).
- `dis_ucode_ld`: Diese Option verhindert, dass der Kernel Microcode-Updates an die CPU weitergibt (siehe Abschnitt 21.7, »Firmware-, BIOS- und EFI-Updates«). Sie sollte nur verwendet werden, wenn es aufgrund eines derartigen Updates zu Abstürzen oder Instabilitäten kommt.

SMP-Optionen

SMP steht für *Symmetric Multiprocessing* und bezeichnet die Fähigkeit des Kernels, mehrere CPUs bzw. CPU-Cores gleichzeitig zu nutzen. Sollten dabei Probleme auftreten, können die folgenden Optionen hilfreich sein:

- `maxcpus=1`: Wenn Sie bei einem Multiprozessorsystem Boot-Probleme haben, können Sie mit dieser Option die Anzahl der genutzten Prozessoren auf 1 reduzieren. Der Wert 0 entspricht der Option `nosmp`.
- `nosmp`: Diese Option deaktiviert die SMP-Funktionen. Der Kernel nutzt nur eine CPU.
- `noht`: Diese Option deaktiviert die Hyper-Threading-Funktion. Dank dieser Funktion verhalten sich viele CPUs so, als stünden mehrere Cores zur Verfügung. Daraus ergibt sich eine etwas höhere Rechenleistung, wenngleich die Steigerung nicht so hoch ist wie bei echtem SMP.
- `nolapic`: APIC steht für *Advanced Programmable Interrupt Controller* und bezeichnet ein Schema, um Hardware-Interrupts an die CPUs weiterzuleiten. Bei aktuellen Kernelversionen wird APIC immer aktiviert. Wenn Sie Probleme mit APIC vermuten, verhindern Sie durch `nolapic`, dass der Kernel den lokalen APIC aktiviert bzw. nutzt.
- `noapic`: Diese Option reicht etwas weniger weit als `nolapic` und deaktiviert nur den IO-Teil von APIC.
- `lapic`: Diese Option aktiviert APIC explizit. Das ist dann notwendig, wenn APIC durch das BIOS deaktiviert ist, aber dennoch genutzt werden soll.

ACPI-Optionen

Das Energieverwaltungssystem ACPI (*Advanced Configuration and Power Interface*) ist nicht nur für das Ein- und Ausschalten verantwortlich, sondern auch für den sparsamen Umgang mit Energie, für die Verwaltung verschiedener Hibernate-Modi etc. Im Folgenden sind die wichtigsten Optionen zur Steuerung der ACPI-Funktionen des Kernels zusammengefasst:

- `acpi=on/off`: Diese Option (de)aktiviert die ACPI-Funktionen im Kernel.
- `acpi=oldboot`: Damit werden die ACPI-Funktionen nur während des Boot-Vorgangs genutzt. Sobald der Rechner läuft, werden die ACPI-Funktionen aber nicht mehr verwendet.
- `pci=noacpi`: Diese Option deaktiviert die Interrupt-Zuweisungen durch ACPI.
- `noresume`: Diese Option bewirkt, dass vorhandene Hibernate-Daten in der Swap-Partition ignoriert werden. Sie ist also dann sinnvoll, wenn der Rechner nicht mehr richtig aufwacht, z. B., weil die Hibernate-Daten defekt sind.

26.8 Kernelparameter verändern

Viele Parameter des Kernels können im laufenden Betrieb über das /proc-Dateisystem verändert werden. Das folgende Beispiel zeigt, wie Sie die Masquerading-Funktion aktivieren, um den Rechner als Internet-Gateway für andere Rechner einzusetzen:

```
sudo sh -c 'echo 1 > /proc/sys/net/ipv4/ip_forward'
```

Einen eleganteren Weg bietet das Kommando sysctl, das mit den meisten aktuellen Distributionen mitgeliefert wird. Das analoge Kommando, um das Masquerading wieder abzuschalten, würde so aussehen:

```
sudo sysctl -w net.ipv4.ip_forward=0
```

sysctl -a liefert eine Liste aller Kernelparameter zusammen mit ihren aktuellen Einstellungen. Mit sysctl -p können die in einer Datei gespeicherten sysctl-Einstellungen aktiviert werden. Als Dateiname wird üblicherweise /etc/sysctl.conf verwendet.

Die Syntax ist in der Manual-Seite zu sysctl.conf beschrieben. Viele Distributionen sehen vor, dass diese Datei während des Init-Prozesses automatisch ausgewertet und ausgeführt wird.

26.9 Spectre und Meltdown

2018 wurden gravierende Design-Schwächen in den CPUs mehrerer Hersteller bekannt. Besonders betroffen war Intel. Durch Exploits konnte ein Prozess unerlaubterweise die Daten anderer Prozesse lesen. Die ersten derartigen Sicherheitslücken erhielten die klingenden Namen *Spectre* und *Meltdown*. Seither wurden diverse weitere verwandte Probleme entdeckt: *Fallout*, *Foreshadow*, *RIDL*, *ZombieLoad* usw.:

https://en.wikipedia.org/wiki/Transient_execution_CPU_vulnerability

Da ein Austausch aller betroffenen CPUs weder technisch noch wirtschaftlich möglich ist, wird seither versucht, die Fehler auf verschiedene Arten zu umgehen. Die vier Hauptansatzpunkte sind dabei:

- Microcode-Updates für die CPU (siehe auch Abschnitt 21.7, »Firmware-, BIOS- und EFI-Updates«)
- Änderungen an den Compilern, um Code zu generieren, der eine Ausnutzung der Fehler verhindert
- Änderungen im Kernel (das betrifft neben Linux auch Windows, macOS etc.) mit derselben Zielsetzung
- Änderungen an besonders betroffenen Programmen, z. B. an Webbrowsern und an Virtualisierungssystemen

Tatsächlich ist es mit diesen Maßnahmen gelungen, viele (wenn auch nicht alle) Fehler zu umgehen. Der Preis dafür ist allerdings hoch: Benchmarktests von Michael Larabel haben gezeigt, dass die Performance der so abgesicherten Rechner je nach Anwendung und CPU erheblich gesunken ist:

https://www.phoronix.com/search/Spectre

Das führt zur absurden Situation, dass die CPU-Hersteller nicht etwa für ihre Designfehler verantwortlich gemacht werden, sondern indirekt sogar davon profitieren: Die verminderte Rechenleistung macht Hardware-Updates früher erforderlich als geplant.

Um festzustellen, von welchen Problemen Ihr Rechner bzw. Ihre CPU betroffen ist und welche Sicherheitslücken durch den Kernel behoben werden, werfen Sie einen Blick in die Dateien des Verzeichnisses `/sys/devices/system/cpu/vulnerabilities`. Die folgenden Ergebnisse stammen von einem etwas älteren Notebook mit einer CPU von Intel (i7-8750H). Ich habe die Spalten des Listings eingerückt, um die Ergebnisse besser lesbar zu präsentieren.

```
cd /sys/devices/system/cpu/vulnerabilities

grep "" *

  gather_data_sampling:   Mitigation: Microcode
  ghostwrite:             Not affected
  indirect_target_sel:    Not affected
  itlb_multihit:KVM:      Mitigation: Split huge pages
  l1tf:                   Mitigation: PTE Inversion;
                          VMX: conditional cache flush, SMT vulnerable
  mds:                    Mitigation: Clear CPU buffers; SMT vulnerable
  meltdown:               Mitigation: PTI
  mmio_stale_data:        Mitigation: Clear CPU buffers; SMT vulnerable
  old_microcode:          Not affected
  reg_file_data_sampling: Not affected
  retbleed:               Mitigation: IBRS
  spec_rstack_overflow:   Not affected
  spec_store_bypass:      Mitigation: Speculative Store Bypass disabled
                          via prctl
  spectre_v1:             Mitigation: usercopy/swapgs barriers and
                          __user pointer sanitization
  spectre_v2:             Mitigation: IBRS;
                          IBPB: conditional; STIBP: conditional; RSB
    filling;
                          PBRSB-eIBRS: Not affected;
                          BHI: Not affected
  srbds:Mitigation:       Microcode
  tsa:                    Not affected
  tsx_async_abort:        Not affected
```

Wesentlich mehr Details verrät das Script `spectre-meltdown-checker.sh`:

```
git clone https://github.com/speed47/spectre-meltdown-checker.git

cd spectre-meltdown-checker

sudo ./spectre-meltdown-checker.sh


  Spectre and Meltdown mitigation detection tool v0.46-29-g34c6095

  Checking for vulnerabilities on current system
  Kernel is Linux 6.16.0-5-cachyos #1 SMP PREEMPT_DYNAMIC
  CPU is Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz

  Hardware check
  * Hardware support (CPU microcode) for mitigation techniques
    * Indirect Branch Restricted Speculation (IBRS)
      * SPEC_CTRL MSR is available: YES
      * CPU indicates IBRS capability: YES (SPEC_CTRL feature bit)
    * Indirect Branch Prediction Barrier (IBPB)
      * CPU indicates IBPB capability: YES (SPEC_CTRL feature bit)
    * Single Thread Indirect Branch Predictors (STIBP)
      * SPEC_CTRL MSR is available: YES
      * CPU indicates STIBP capability: YES (Intel STIBP feature bit)
  ...
  * CPU vulnerability to the speculative execution attack variants
  * Affected by CVE-2017-5753 (Spectre Variant 1): YES
  * Affected by CVE-2017-5715 (Spectre Variant 2): YES
  * Affected by CVE-2017-5754 (Variant 3, Meltdown): YES
  ...

  > SUMMARY:
    CVE-2017-5753:OK   CVE-2018-3640:OK   CVE-2018-3620:OK    ...
    CVE-2017-5715:OK   CVE-2018-3639:OK   CVE-2018-3646:OK
    CVE-2017-5754:OK   CVE-2018-3615:OK   CVE-2018-12126:OK

  Need more details about mitigation options? Use --explain
  A false sense of security is worse than no security at all,
  see --disclaimer
```

Schutzmechanismen deaktivieren

Standardmäßig sind im Kernel fast alle gerade verfügbaren Schutzmaßnahmen aktiv. Die einzige Ausnahme betrifft Hyperthreading bzw. *Simultaneous Multithreading* (SMT). Das ist

ein Verfahren, mit dem eine CPU mehr Threads parallel ausführen kann, als echte Cores zur Verfügung stehen. SMT bringt zwar einen deutlichen Performance-Schub, es müsste aus Sicherheitsgründen aber eigentlich deaktiviert werden. Aktuell sind Angriffe, die darauf abzielen, aber nur schwierig umzusetzen.

Für jedes der im Kernel implementierten Schutzverfahren gibt es eigene Kernelparameter, um das jeweilige Verfahren zu deaktivieren: `nospectre_v1`, `nospectre_v2`, `nopti` usw. Darüber hinaus gibt es den Parameter `mitigations`, der drei Einstellungen vorsieht:

- `mitigations=auto`: In der Defaulteinstellung sind alle verfügbaren Schutzmaßnahmen aktiviert. SMT bleibt aber aktiv.
- `mitigations=auto,nosmt`: Diese Einstellung deaktiviert SMT und ist noch sicherer. Sofern Ihre CPU Multithreading unterstützt, sinkt die Performance von Anwendungen mit vielen Threads aber nochmals deutlich.
- `mitigations=off`: Diese Einstellung deaktiviert alle Schutzmaßnahmen und macht Ihren Rechner schneller. Empfehlenswert ist das aber nur, wenn Sie sich vor eventuellen Angriffen sicher fühlen. Das lässt sich in manchen Fällen durchaus befürworten: So wurden die Sicherheitslücken Spectre, Meltdown & Co. bisher zwar theoretisch nachgewiesen, es gibt aber zurzeit kaum bekannte Schadsoftware, die diese Sicherheitslücken tatsächlich ausnutzt.

  Insofern ist die Versuchung groß, in bestimmten Anwendungsfällen auf den Schutz zu verzichten – z. B. auf Entwicklungs- oder Labor-Rechnern, auf denen häufig CPU-intensive Anwendungen laufen. Vollkommen tabu ist diese Option jedoch auf allen Servern, die virtuelle Maschinen unterschiedlicher Besitzer oder Kunden ausführen.

Um die `mitigations`-Einstellungen einmal auszuprobieren, öffnen Sie während des Boot-Vorgangs mit E den Editor für den betreffenden GRUB-Menüeintrag und fügen am Ende der `linux`-Zeile `mitigations=...` hinzu. Um die Option dauerhaft einzustellen, verändern Sie die Zeile `GRUB_CMDLINE_LINUX_DEFAULT` in `/etc/default/grub` und aktualisieren dann die GRUB-Konfiguration mit `update-grub` bzw. mit `grub2-mkconfig`.