Ralf
Jesse

Embedded Linux
mit Raspberry Pi und Co.

Hinweis des Verlages zum Urheberrecht und Digitalen Rechtemanagement (DRM)

Ralf Jesse

Embedded Linux

mit Raspberry Pi und Co.

mitp

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

ISBN 978-3-95845-062-2
1. Auflage 2016

www.mitp.de
E-Mail: mitp-verlag@sigloch.de
Telefon: +49 7953 / 7189 - 079
Telefax: +49 7953 / 7189 - 082

Lektorat: Sabine Schulz
Sprachkorrektorat: Irmgard Böger
Coverbild: © Dolnikov – fotolia.com
Satz: III-satz, www.drei-satz.de

Inhaltsverzeichnis

Einleitung

Embedded Linux hat nicht zuletzt durch die Artikel in Spiegel Online (2012) und Focus Online (2013) einen sehr hohen Bekanntheitsgrad erlangt. Dass Embedded Linux Programmierern, »Makern« und professionellen Hard- und Softwareentwicklern bereits bekannt war, kann man getrost voraussetzen. Anders ist dies bei »normalen« Anwendern: Dass die Hersteller aktueller Blueray-Disk-Player oder des neuen Smart-TV sowie von Wasch- und Geschirrspülmaschinen oder Wäschetrocknern Embedded Linux sehr häufig für die Steuerung der Elektronik und die Benutzerführung einsetzen, war der Allgemeinheit vor Erscheinen dieser Artikel vermutlich nicht geläufig. Dass Android, das Betriebssystem für Smartphones, ebenfalls einen (Embedded-)Linux-Kern hat, war vielen Anwendern wahrscheinlich ebenfalls unbekannt.

Die sehr beliebten Mini-Computer *Raspberry Pi*, *BeagleBone Black* und »Kollegen« haben dazu beigetragen, das Interesse an den Grundlagen von Embedded Linux sowie dessen Programmierung und Nutzung für alltägliche Dinge zu wecken, wie die Steuerung von Markisen und der Wohnraum-Beleuchtung oder die Regelung von Heizungsanlagen während der Abwesenheit: Viele Anwender, die früher nicht daran gedacht haben, sich mit diesem – zugegebenermaßen – komplexen Thema zu befassen, haben in dieser Beschäftigung ein interessantes Hobby und möglicherweise den Einstieg in eine neue berufliche Herausforderung gefunden.

Dieses Buch soll Ihnen unter Einsatz des Raspberry Pi den Einstieg in die Welt von Embedded Linux ebnen. Da die vermittelten Grundlagen allgemeingültig sind, können Sie die Erkenntnisse ebenfalls auf andere Embedded PCs übertragen. Sie werden erfahren, wie Sie eine Cross-Entwicklungsumgebung aufsetzen, wie Sie Daten zwischen dem Entwicklungs-PC und dem Zielgerät (dem Embedded PC) austauschen, wie Sie den Entwicklungsaufwand durch den Einsatz von »Das U-Boot« weiter reduzieren und optimieren können, und Sie lernen den Einstieg in die Treiberprogrammierung kennen. Ein vereinfachter Einstieg in die Shellprogrammierung zeigt Ihnen die Nutzung der wichtigsten Systemkommandos, sodass Sie in die Lage versetzt werden, eigene Shellscripts zu entwickeln bzw. bereits vorhandenes Wissen aufzufrischen. Das Ende jedes Kapitels umfasst eine Übersicht über ausgewählte weiterführende Literatur zum behandelten Stoff, sodass Sie die erworbenen Kenntnisse problemlos entsprechend Ihren eigenen Wünschen und Vorstellungen vertiefen können.

Als zwingend werden mittlere bis gute Kenntnisse der Programmiersprache C und der englischen Sprache vorausgesetzt. Die Gründe hierfür sind, dass der Kernel von Embedded Linux zu einem bedeutenden Teil in dieser Programmiersprache geschrieben wurde und wesentliche weiterführende Literatur – auch in Form von Datenblättern – nur in englischer Sprache verfügbar ist. Ebenfalls vorausgesetzt wird, dass Sie wissen, wie Ihr Mini-Computer in Betrieb genommen wird: Informationen zur Beschaffung und Installation des Betriebssystems vermittelt dieses Buch nicht!

Besondere Kenntnisse der Elektronik werden nicht vorausgesetzt, obwohl gewisse Grundlagen hier sehr hilfreich sind! Sie werden später beispielsweise Leuchtdioden sowohl direkt (Kapitel 4) als auch unter Einsatz von Schieberegistern ansteuern (Teil IV, Praxisteil I): Dass ohmsche Widerstände den elektrischen Strom begrenzen und somit eine Überlastung der GPIO-Leitungen oder hieran angeschlossener elektronischer Komponenten vermeiden, sollte als Basiswissen vorhanden sein.

Zielgruppe dieses Buches

Dieses Buch richtet sich an alle, die »mehr« aus ihrem Embedded System herausholen wollen. Diese Gruppe umfasst mit Ausnahme »echter« Linux-Profis alle Anwender vom Hobbyisten und interessierten Laien über Studenten aller technischen Fachrichtungen bis hin zu denjenigen, die ihr Geld mit der Programmierung von Computern oder der Entwicklung elektronischer Schaltungen verdienen, sich aber bisher noch nicht vertiefend mit Linux oder Embedded Linux beschäftigt haben.

An der Definition der Zielgruppe für dieses Buch erkennen Sie, dass Linux-Kenntnisse nicht erforderlich sind: Die erforderlichen Grundlagen für einen erfolgreichen Einsatz sowie für die Einarbeitung in vertiefende Aspekte werden in diesem Buch erarbeitet.

Das Buch ist in vier Teile gegliedert, von denen der erste Teil einen »sanften« Einstieg in die Embedded-Linux-Welt darstellt.

Teil I

In Kapitel 1 werden Sie den Unterschied zwischen sogenannten Bare-Metal-Mikrocontroller-Systemen und Mikrocontroller-Systemen mit Betriebssystem sowie den Unterschied zwischen dem Kernel »Linux« und Linux-Distributionen kennenlernen. Sie erfahren hier auch den Unterschied zwischen Mikroprozessoren und Mikrocontrollern. Im Anschluss erhalten Sie – ebenfalls noch in Kapitel 1 – einen Überblick über die grundlegende Architektur von Linux und Embedded Linux. Den Abschluss von Kapitel 1 bildet eine Beschreibung der Parallelinstallation von Linux Mint zu einem bereits vorhandenen Windows-Betriebssystem bzw. der Installation von Linux Mint in einer virtuellen Maschine. Die Durchführung einer dieser Varianten ist sehr empfehlenswert, da die »Rechenpower« des Raspberry Pi im Vergleich zu Standard-PCs – gelinde gesagt – unterdurchschnittlich ist: Wenn es später im Buch darum geht, einen eigenen Kernel zu entwickeln, werden Sie die Vorteile des sogenannten Cross-Developments zu schätzen wissen.

Kapitel 2 befasst sich im ersten Schritt mit der Installation und der Basiskonfiguration von Samba. Hierdurch lässt sich der Mini-Computer in ein bereits vorhandenes (Heim-) Netzwerk integrieren. Dies ist nicht immer erwünscht oder erforderlich. Es muss dann aber eine andere Möglichkeit vorhanden sein, um die Software vom Host auf das Target zu übertragen und dort zu testen. Für diese Anforderung liefert Kapitel 2 ebenfalls eine Antwort. Der überwiegende Teil von Kapitel 2 beschreibt die Verzeichnisstruktur eines typischen Unix- oder Linux-Betriebssystems, also auch die eines Embedded Linux.

In Kapitel 3 steht die Programmierung der Shell im Vordergrund. Die Shell ist ein wesentlicher Aspekt für die Beherrschung von Linux/Embedded Linux. Steht auf einem Desktop-Linux in der Regel eine grafische Benutzeroberfläche auch für die Systemverwaltung zur Verfügung, so existiert diese Möglichkeit auf Embedded-Systemen in der

Regel aufgrund der stark eingeschränkten Ressourcen (CPU-Taktung, Mangel an Arbeitsspeicher, Festplattenspeicher etc.) nicht. Dieses Kapitel liefert eine grundlegende Einführung in die wichtigsten Aspekte der Shell-Programmierung.

Bevor der Einstieg in die Shell-Programmierung erfolgt, werden aber zunächst die Programme `man` und `info` vorgestellt, da sie sehr ausführliche Informationen zu den einzelnen Kommandos zur Verfügung stellen. Darüber hinaus geht es vorher um die Ein- und Ausgabeumlenkung. Im Anschluss werden dann aber endlich die wichtigsten Features zum Einsatz der `bash` beschrieben. Hierzu zählen Shell-Variablen, die Programmierung von Alternativen (`if-then-elif-else-fi` sowie `case-esac`) bzw. Abfragen sowie die Programmierung von Schleifen.

Kapitel 4 beschreibt den Aufbau einer Cross-Entwicklungsumgebung für Software, die später auf dem Embedded PC ausgeführt werden soll. Ich habe mich hier für Code::Blocks entschieden, einer integrierten Entwicklungsumgebung, die für Windows-, Linux- und Mac-OS-X-Systeme verfügbar ist. Der besondere Charme dieser Entwicklungsumgebung liegt darin, dass sie vollständig in der Programmiersprache C++ geschrieben wurde, sodass die Abhängigkeit von weiterer Software entfällt, wie einer Java-Laufzeitumgebung: Durch die Installation von systemspezifischer Software, die grundsätzlich immer für die Softwareentwicklung benötigt wird, sind alle Abhängigkeiten automatisch erfüllt. Der Brite Gordon Henderson hat für den Raspberry Pi die Bibliothek `wiringPi` entwickelt und als Open-Source-Software der Community zur Verfügung gestellt. Kapitel 4 beschreibt die Beschaffung dieser Bibliothek und bietet darüber hinaus einen ersten Einstieg in ihre Nutzung.

Teil II

Kapitel 5 beschreibt, wie Sie einen eigenen Embedded-Linux-Kernel bauen und in Betrieb nehmen. Hierbei handelt es sich um einen Standardkernel ohne weitere Modifikationen. Wenn Sie zu einem späteren Zeitpunkt einen eigenen Kernel entwickeln wollen, so zeigt Ihnen dieses Kapitel, worauf Sie zu achten haben. Diese Beschreibung liegt in einer Schritt-für-Schritt-Form vor, sodass Sie sie auch als eine Art Checkliste betrachten können. Dass auch Alternativen für die meisten der erforderlichen Schritte erläutert werden, erweitert den Horizont und zeigt, dass im Regelfall immer mehrere Varianten zum gewünschten Ergebnis führen.

In Kapitel 6 wird gezeigt, wie Sie ein root-Dateisystem von Hand erzeugen können. Mit dem Wissen aus Kapitel 3, dem Verstehen des Shellscripts `mkrpi` aus Kapitel 5 sowie der in diesem Kapitel beschriebenen Vorgehensweise sollten Sie in der Lage sein, die hier durchgeführten Schritte selbst mit einem Shellscript zu automatisieren. Der Fokus liegt hierbei auf dem als Debootstrap bezeichneten Verfahren. Zum Schluss des Kapitels wird mit Buildroot noch eine Alternative zur Erzeugung eines root-Dateisystems von Hand aufgezeigt.

Kapitel 7 beschreibt den Bootprozess eines Embedded Linux. Hier geht es um die derzeit am meisten verbreiteten Embedded PCs Raspberry Pi A/B/B+/2, den BeagleBone Black sowie um die verschiedenen Cubieboards. Stellt dieser Teil noch einen eher theoretischen Überblick dar, wird im weiteren Verlauf der Bootloader `Das U-Boot` praktisch eingesetzt. Detaillierte Anleitungen liefern Vorschläge zum Booten eines Kernels von einer SD-Karte und anschließend von einem TFTP-Server. Insbesondere der zweite Teil – das Booten eines Kernels über das Netzwerk – stellt eine sehr interessante und weit verbrei-

tete Vorgehensweise dar, da das permanente Wechseln der SD-Karte beim Entwickeln von Kernel-Images und Treibern der Lebensdauer der SD-Karte und des Kartenslots nicht gerade zuträglich ist. Darüber hinaus beschleunigt dies die Entwicklung enorm.

Teil III

Die Kapitel 8, 9 und 10 befassen sich mit den Grundlagen der Entwicklung von Treibern und Kernelmodulen. In Kapitel 8 werden die minimalen Grundlagen gelegt. Die Entwicklung von Gerätetreibern und -modulen ist nicht schwierig – und viele Tutorials, die in großer Anzahl im Internet zu finden sind, zeigen Beispiele zur Vorgehensweise. Nur die wenigsten der Tutorials gehen auf die Grundlagen in dem Maße ein, die das Schreiben eigener Treiber vereinfacht.

In Kapitel 9 werden die Grundlagen aus Kapitel 8 erweitert und miteinander in Verbindung gesetzt. Zwar ist es nicht möglich in einem Buch, das sich nicht ausschließlich mit der Entwicklung von Treibern befasst, »in große Tiefen abzutauchen«: In diesem Buch wird aber angestrebt, die Zusammenhänge aufzuzeigen, und es nennt zusätzliche Informationsquellen, die mit zunehmender Erfahrung immer wichtiger werden.

Kapitel 10 fasst die Erkenntnisse der beiden vorangegangenen Kapitel zusammen. Auf dieser Basis wird eine Schablone (Template) entwickelt, die Sie später für die Entwicklung eigener Treiber einsetzen können. Zudem bietet Ihnen Kapitel 10 einige Tipps und Hinweise, wie die GPIO-Ports korrekt verwendet werden. Dies erfolgt getrennt für digitale Ein- und Ausgänge, wobei in besonderem Maße auf die Sicherheit eingegangen wird, damit weder die verschiedenen Bausteine noch der Mikrocontroller zerstört werden.

Teil IV

Teil IV legt besonderen Wert auf praktische Beispiele, welche die Entwicklung von Gerätetreibern bzw. Modulen verdeutlichen.

Kapitel 11 verwendet hierfür das serielle 8-Bit-Schieberegister SN74HC595 von Texas Instruments, das (hier) für die Ansteuerung von acht Leuchtdioden mit nur einem GPIO-Port genutzt wird. Hier wird die Funktionsweise des Bausteins beschrieben sowie ein kompletter Schaltplan entwickelt. Entsprechende Hinweise erläutern, worauf beim Einsatz dieses Bausteins zu achten ist. Der wichtigste Part dieses Kapitels ist die Entwicklung der vollständigen Treibersoftware, die auf Basis der Schablone aus Kapitel 10 entwickelt wurde, sowie die User-Space-Anwendung, die das Modul dann nutzt. Auszüge aus dem Datenblatt des SN74HC595 von Texas Instruments runden dieses Kapitel ab.

In Kapitel 12 wird mit dem MAXIM 7219 ein Schieberegister eines anderen Herstellers verwendet. Dieser Baustein eignet sich in besonderem Maße zur Ansteuerung von LED-Matrizen, wobei – abgesehen von der 8 x 8-LED-Matrix – nur ein zusätzlicher Widerstand benötigt wird. Auch hier wird das Treibermodul mitsamt der User-Space-Anwendung von Grund auf entwickelt.

Kapitel 13 liefert einen bereits recht komplexen Treiber zur Ansteuerung eines LC-Displays zur Anzeige von Texten (also kein grafisches Display!). Als besonderes Highlight wird hier nicht die eigentliche Funktionalität des Treibers betrachtet – sie unterscheidet sich nicht wesentlich von den anderen Treibern in diesem Buch. Wichtiger ist hier, dass sich das Verhalten des Treibers durch die User-Space-Anwendung steuern lässt. In der User-Space-Anwendung können Sie beispielsweise festlegen, in welcher Zeile des Dis-

plays Text angezeigt werden soll. Ebenfalls von außen können Sie bestimmen, ob Sie einen Cursor anzeigen wollen. Ist dies gewünscht, so können Sie entscheiden, ob der Cursor statisch angezeigt werden oder blinken soll. Weitere Funktionen, wie die Festlegung der Textausrichtung (links- oder rechtsbündig bzw. zentriert) oder ein Textticker, wurden vorbereitet, sind aber noch nicht implementiert. Anhand der Beispielimplementierung der übrigen Funktionen sollte es Ihnen aber nicht allzu schwer fallen, diese Funktionen selbst zu implementieren. Als »Endausbau« wäre denkbar, diese »Komfortfunktionen« in einer Bibliothek zusammenzufassen und sie der Allgemeinheit zur Verfügung zu stellen.

In eigener Sache

Zu diesem Buch habe ich eine E-Mail-Adresse eingerichtet, über die ich für Ihre Fragen zum Buch, Hinweise auf Fehler oder Wünsche nach Verbesserungen und Unterstützung erreichbar bin. Schreiben Sie einfach an

`embedded@ralf-jesse.de`.

Unter der Internetadresse `http://www.ralf-jesse.de` finden Sie Korrekturen von Fehlern, die erst nach dem Druck des Buches erkannt wurden sowie Bonusmaterial (z. B. eine Einführung in das Debugging oder einfache Grundlagen der Elektrotechnik). Ich empfehle Ihnen, zunächst immer die Rubrik NEWS anzusteuern, da Sie hier immer über Erweiterungen der Website informiert werden, die seit Ihrem letzten Besuch hinzugefügt wurden.

Über den Autor

Ralf Jesse ist Diplom-Ingenieur der Elektrotechnik mit fast 30 Jahren beruflicher Praxis im Einsatz von Mikroprozessoren und -controllern. Nach drei Jahren im industriellen Maschinenbau im Bereich der Anlagensicherheit folgten mehr als 20 Jahre Tätigkeit als Entwicklungs- und Supportingenieur in einem großen japanischen Konzern. Seit zwei Jahren ist Ralf Jesse in der Entwicklung und im Support industrieller Feldbusse tätig.

Teil I

Einführung und Einrichtung einer Entwicklungsumgebung

Der erste Teil bietet einen sanften Einstieg in die Welt der Mikrocontroller. Dabei erfahren Sie den Unterschied zwischen sogenannten Bare-Metal-Systemen und solchen Systemen, die über ein eigenes Betriebssystem verfügen. Da Embedded Systeme aufgrund eingeschränkter Ressourcen komfortables Arbeiten nahezu unmöglich machen, wird hier auf der Basis einer VirtualBox-Installation von Linux Mint eine Cross-Development-Umgebung eingerichtet. Um die auf diesem System entwickelte Software auf das Embedded System übertragen zu können, gibt es eine kurze Einführung in Samba. Ebenfalls zu den Grundlagenkenntnissen gehört die Shellprogrammierung. Auf die Auffrischung bekannter Grundlagen geht Kapitel 3 ein. Kapitel 4 befasst sich dann mit der konkreten Installation meiner Cross-Development-Umgebung.

In diesem Teil:

Kapitel 1

Embedded Linux

In diesem Kapitel geht es um einige grundlegende Fragen, die sich mit Linux im Allgemeinen und mit Embedded Linux im Speziellen befassen. Zunächst soll aber geklärt werden, was man üblicherweise unter einem Desktop-Betriebssystem versteht.

1.1 Desktop-Betriebssysteme

Wenn in der Literatur von Linux gesprochen wird, so ist hiermit üblicherweise eines der sogenannten *Desktop-Betriebssysteme* gemeint. Dabei spielt Linux bei Desktop-Betriebssystemen nur eine untergeordnete Rolle. Zu den Desktop-Betriebssystemen zählen *Windows* in seinen verschiedenen Ausprägungen (Windows 3.x, Windows 95/98/ME/2000/Vista/7/8.x sowie Windows 10 und auch Windows CE), *Mac OS X* und natürlich viele verschiedene Varianten von Linux (z. B. *Ubuntu*, *RedHat*, *openSuse*, *Fedora*, *Mint*, *Debian* usw.). Besonders bei den Linux-Versionen sticht eine gewisse Ungenauigkeit – um nicht zu sagen: Schlampigkeit – hervor, denn in den meisten Fällen wird der Begriff »Linux« genannt, obwohl die Bezeichnung »Linux-Distribution« korrekt wäre.

Hinweis

Will man ganz exakt sein, dann sollte man nicht einfach nur »Linux« sagen, wenn das Betriebssystem gemeint ist, sondern »GNU/Linux«, wobei »GNU« ein rekursives (sich also selbst wiederholendes) Wortspiel für »GNU is not Unix« ist. Die korrekte Bezeichnung »GNU/Linux« wird aber häufig einfach nur zu »Linux« abgekürzt.

Wichtig

Korrekterweise handelt es sich bei Ubuntu, openSuse, Fedora, Mint, Debian usw. nicht um das Betriebssystem »*Linux*«, sondern um sogenannte »*Linux-Distributionen*«!

Deshalb soll zunächst der Unterschied zwischen GNU/Linux und einer Linux-Distribution definiert werden:

- GNU/Linux besteht prinzipiell aus zwei Teilen: dem Kernel und dem root-Dateisystem rootfs.
- Eine Linux-Distribution ist im Allgemeinen viel umfangreicher: Neben dem Kernel und dem rootfs enthält eine Distribution eine Vielzahl zusätzlicher Anwendungen. Hierzu zählen Office-Programme (z. B. OpenOffice oder LibreOffice), Audio- und Videoanwendungen, Internet-Browser, Mail-Clients und vieles mehr. Die bekannten Linux-Distributionen gehen sogar noch einen Schritt weiter: Bei der Installation

einer solchen Distribution richten sie eine sehr umfassende Menge nützlicher und wichtiger Software automatisch ein.

Hinweis

Entsprechend dieser Definition handelt es sich bei den verschiedenen Windows-Systemen oder Mac OS X also um Betriebssysteme, weil die zum sinnvollen Arbeiten erforderliche Software kein Bestandteil von Windows oder Mac OS X ist, sondern separat beschafft und installiert werden muss.

Nachdem der Unterschied zwischen Linux und einer Linux-Distribution geklärt ist, »dürfen« Sie jetzt auch wieder »Linux« sagen, wenn Sie eigentlich eine Linux-Distribution meinen: Sie können den Unterschied schließlich auf Nachfrage sofort erläutern.

1.2 Bare-Metal vs. Betriebssystem

Wenn Sie einen Standard-PC oder Macintosh-Computer kaufen, so erhalten Sie Computer, die im Regelfall bereits betriebsbereit eingerichtet sind. Dies liegt aber nur daran, dass die Hersteller Ihnen die Arbeit der Installation von Betriebssystemen und Anwendungen zu einem großen Teil abnehmen. Wenn Sie sich aber einen PC selber »zusammenschrauben«, ist dies nicht der Fall: Nach erfolgtem Zusammenbau fängt die Arbeit nämlich erst richtig an! Prinzipiell könnte man solche PCs als sogenannte *Bare-Metal-Systeme* bezeichnen, da sie noch nicht betriebsbereit sind. Der Begriff »Bare-Metal-System« wird aber üblicherweise in einem anderen Zusammenhang verwendet: Er bezeichnet Computer, die ohne Betriebssystem genutzt werden!

Standard-PCs haben aufgrund leistungsstarker Mikroprozessoren genügend »Power« und verfügen über sehr große Ressourcen, wie Arbeitsspeicher, Grafikkarten, Festplatten usw. Aufgrund ihrer Leistungsstärke eignen sie sich, komplexe Betriebssysteme und die zugehörigen Programme auszuführen. Hierbei sticht ebenfalls hervor, dass sie prinzipiell universell für die verschiedensten Aufgaben einsetzbar sind und dies bei modernen Computern sogar nahezu gleichzeitig. Dies sind alles Eigenschaften, die sie für den Begriff »Bare-Metal-System« disqualifizieren.

»Bare-Metal-Systeme« verfügen nämlich in der Regel nicht über diese großen Ressourcen: Sie sind teilweise noch nicht einmal in der Lage, ein vollständiges Betriebssystem auszuführen, weil z. B. der integrierte Arbeitsspeicher oder der Flashspeicher bei weitem nicht ausreichend dimensioniert ist. Als »Bare-Metal-Systeme« werden vor allem Microcontroller-Steuerungen bezeichnet, die in den allermeisten Fällen für eine ganz spezielle Aufgabe konstruiert wurden. Hierzu zählen insbesondere viele Einplatinen-Computer. Ihre Einsatzgebiete liegen auch in ganz anderen Bereichen, wo Ressourcen, wie Tastaturen, DVD- oder Blue-Ray-Laufwerke, Computermäuse oder Großbildschirme, gar nicht erforderlich sind. Typische Einsatzgebiete sind Geräte der Unterhaltungselektronik (Smart-TV, Blue-Ray-Player), Geräte für die Telekommunikation (Smartphones), Haushaltsgeräte (Kaffeevollautomaten, Wasch- und Geschirrspülmaschinen oder Wäschetrockner) und nicht zuletzt Steuerungen innerhalb moderner Kraftfahrzeuge (ABS, ESB, Navigationssysteme etc.) und natürlich auch Maschinensteuerungen in der Industrie.

1.2.1 Mikroprozessoren vs. Mikrocontroller

Im letzten Abschnitt wurden mit *Mikroprozessor* und *Mikrocontroller* zwei Begriffe eingeführt, die bisher noch nicht gegeneinander abgegrenzt wurden.

- Unter einem Mikroprozessor versteht man üblicherweise einen hochintegrierten Halbleiterbaustein mit teilweise mehreren Milliarden Transistoren, die in einem Gehäuse zusammengefasst sind. Bei Mikroprozessoren unterscheidet man – je nach Architektur – *RISC-Prozessoren* (RISC = Reduced Instruction Set Computer) und *CISC-Prozessoren* (CISC = Complex Instruction Set Computer). Sie sind imstande, sehr schnelle sogenannte binäre Operationen durchzuführen. Mikroprozessoren benötigen für einen sinnvollen Einsatz zusätzliche Komponenten, die in der Literatur allgemein als *Northbridge* und als *Southbridge* bezeichnet werden. Nur soviel sei gesagt: Die Northbridge bedient sehr schnelle externe Hardware, wie den Arbeitsspeicher oder Grafikkarten. Zu diesem Zweck ist sie unmittelbar an den Mikroprozessor angeschlossen. Die Southbridge lässt es hingegen »etwas gemütlicher« angehen: Sie ist zuständig für den Datentransfer zwischen »langsamer« Peripherie, wie (in früheren Versionen) dem PCI-Bus oder externen Schnittstellen, z. B. USB. Gemeinsam werden North- und Southbridge im Allgemeinen als *Chipsatz* (*Chipset*) benannt und hat bei Intel z. B. die Bezeichnung *ICH* (*I/O Controller Hub*).

Hinweis

Die Beschreibung von Mikroprozessoren soll nicht weiter verfolgt werden. Wollte man tiefer in die Materie einsteigen, müssten weitere Begriffe, wie von-Neumann- und Harvard-Architektur, eingeführt werden.

- Mikrocontroller haben eine gewisse Ähnlichkeit mit Mikroprozessoren. Der wesentliche Unterschied besteht darin, dass Mikrocontroller weder eine North- noch eine Southbridge benötigen, da die Schnittstellen zur Peripherie bereits im Mikrocontroller integriert sind. Moderne Mikrocontroller verfügen beispielsweise über verschiedene unabhängige integrierte serielle Schnittstellen, Timer, Arbeitsspeicher (wenn auch nur relativ wenig), Flashspeicher zur Speicherung der Firmware usw. Seit einiger Zeit hat sich für Mikrocontroller auch die Bezeichnung *SoC* (*Silicon on a Chip*, Silizium auf einem Chip) eingebürgert. Viele sehr große Halbleiterunternehmen (z. B. NXP – das ist die ehemalige Halbleitersparte von Philips –, Texas Instruments, Atmel, Infineon, NEC, Hitachi, Motorola etc.) stellen für die verschiedensten Anwendungen optimierte Varianten von Mikrocontrollern her. Besonders bekannt sind (natürlich) auch die Firmen Intel und AMD, die neben Desktop-CPUs auch Mikrocontroller herausbringen. Dabei entwickeln die genannten Betriebe nicht nur völlig eigene Mikroprozessoren und Mikrocontroller, sondern lizenzieren zusätzliche Technologien, wie sie z. B. die Firma ARM entwickelt hat. Besonderer Beliebtheit erfreuen sich die Mikrocontroller, die auf der ARM-Cortex-Serie basieren, da sie stromsparend arbeiten und sich daher sehr gut für akkubetriebene Geräte eignen, z. B. Smartphones.

1.3 Embedded Betriebssysteme

Neben den hochspezialisierten Mikrocontrollern für Haushaltsgeräte, Unterhaltungs- und Telekommunikations-Elektronik usw. gibt es aber auch solche, die im Vergleich zu einfachen Mikrocontrollern geradezu üppig mit Arbeitsspeicher ausgestattet sind. In Verbindung mit einer relativ leistungsstarken *CPU* (*Central Processing Unit*), die aber im direkten Vergleich zu modernen Mikroprozessoren dennoch als »schneckenlahm« bezeichnet werden muss, einem bis zu mehrere Gigabyte großen Flashspeicher und allen in Abschnitt 1.2.1 genannten integrierten Peripherieschnittstellen, bieten diese genügend Ressourcen, »abgespeckte« und speziell angepasste vollständige Betriebssysteme verwenden und bedienen zu können. Diese Betriebssysteme werden allgemein als *Embedded Betriebssysteme* (»eingebettete« Systeme) bezeichnet.

Raspberry Pi, BeagleBone Black, Cubietruck, Banana Pi und viele mehr sind solche Mini-Computer, die in der Lage sind, Embedded Betriebssysteme zu verwenden. Wenn bei diesen Mini-Computern bevorzugt Embedded Linux eingesetzt wird, ist dies vor allem damit begründet, dass Embedded Linux – genau, wie sein großer Bruder GNU/Linux – kostenlos und im Quelltext verfügbar ist. Darüber hinaus ist Embedded Linux sehr sparsam, was den Gebrauch der Ressourcen erheblich effizienter macht, als dies mit anwenderorientierten Betriebssystemen möglich ist, wie Windows oder Mac OS X.

Embedded Linux ist aber nicht das einzige Embedded Betriebssystem: Viele weitere völlig unterschiedliche Betriebssysteme zählen ebenfalls zu den Embedded Systemen: *Windows CE*, eine Vielzahl sogenannter *Echtzeitbetriebssysteme*, die unter dem Sammelbegriff *RTOS* (RTOS = *Real Time Operating System*) geführt werden, *Embedix*, *eCos*, *VxWorks*, *QNX* oder *Epoc*.

Für die meisten der genannten Embedded Betriebssysteme gilt, dass sie im Regelfall nicht für sicherheitskritische Anwendungen, z. B. in Kraftfahrzeugen oder Flugzeugen bzw. in Kraftwerken, eingesetzt werden können, weil sie nicht in der Lage sind, auf kritische Situationen in Echtzeit zu reagieren.

Bei den RTOS-Betriebssystemen verhält sich dies anders: Sie gehören zur Gruppe der Echtzeitbetriebssysteme. Der Begriff »Echtzeit« ist hier aber nicht zwingend wörtlich zu nehmen: Selbstverständlich treten auch hierbei technisch bedingte Zeitverzögerungen zwischen verschiedenen Ereignissen auf, die genau gegeneinander abgewogen und sehr genau definiert werden müssen. Ganz besonders wichtig ist hier, dass die möglichen Verzögerungen, die allgemein als *Latenz* bezeichnet werden, vorhersehbar sein müssen und dass die Einhaltung dieser Latenzen garantiert werden muss. Das Deutsche Institut für Normung (DIN) hat die Vorgaben in der DIN 44300 genau definiert, wobei diese Norm eng verwandt mit den internationalen Normen ISO 2381-1 bis ISO 2381-22 ist.

Hinweis

Embedded Linux in seinen verschiedenen Ausprägungen gehört nicht zu den echtzeitfähigen Betriebssystemen.

Für einen ersten Einblick mag dies reichen. In diesem Buch werden wir uns ausschließlich mit einem Standard-Betriebssystem aus der Gruppe Embedded Linux befassen. Auf Bare-Metal- oder gar Echtzeitsysteme wird nicht näher eingegangen.

1.4 Die Architektur von Linux

In diesem Kapitel soll der Einstieg in das Verständnis der Architektur von Linux erfolgen.

Hinweis

Die folgenden Ausführungen sind wirklich nur als Einstieg zu verstehen. Spätere Kapitel beschäftigen sich auch mit einzelnen Aspekten, die dann schon »ans Eingemachte« gehen. An dieser Stelle ist es hierfür aber noch zu früh.

Linux ist, wie alle anderen modernen Betriebssysteme auch, sehr komplex. Um diese Komplexität beherrschbar zu machen, ist der Kernel in sogenannte *Layer* (*Schichten*) unterteilt, die widerum eine Vielzahl verschiedener Programme enthalten. Abbildung 1.1 zeigt den prinzipiellen Aufbau eines Linux-Systems.

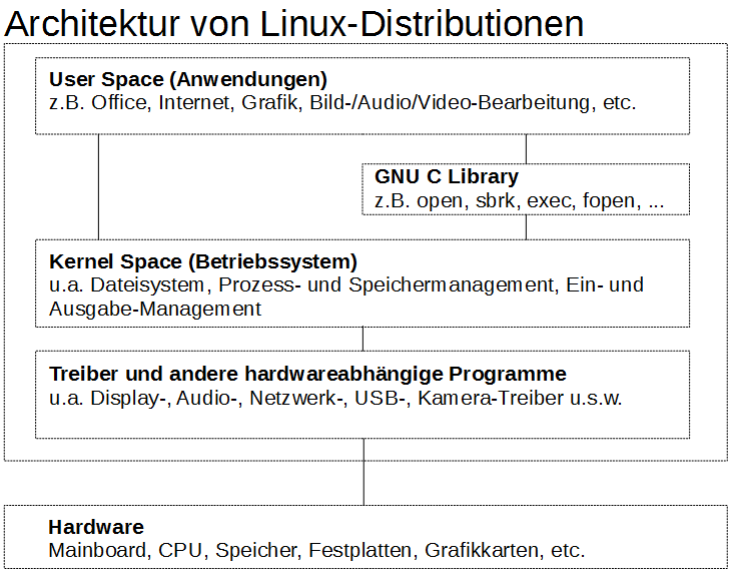

Abb. 1.1: Die Architektur einer Linux-Distribution

1.4.1 Erläuterungen

In Abbildung 1.1 erkennen Sie zwei voneinander abgegrenzte Kästen: Der obere Kasten zeigt den typischen Aufbau jeder Linux-Distribution, während der untere Kasten die Hardwareschicht darstellt.

Hinweis

Die Hardwareschicht ist nicht Bestandteil einer Linux-Distribution. Sie ist hier nur zum besseren Verständnis eingezeichnet.

Der obere Kasten gliedert sich in vier Teile, von denen die unteren beiden Teile das Linux-Kernsystem darstellen, also den Kernel sowie die Treiberschicht. Mit diesen beiden Teilen kommen nur Linux-Entwickler unmittelbar in Berührung, während sich reine Anwender hierum nicht kümmern müssen: Anwender werden beim Einsatz von Linux-Distros ausschließlich mit dem *User Space* in Kontakt kommen. Der Kasten, der mit *GNU C Library* bezeichnet ist, dient als Bindeglied zwischen dem User Space und dem *Kernel Space*: Auch hiermit kommt der typische Anwender nur indirekt in Berührung, wenn er beispielsweise in seiner Office-Anwendung ein neues Dokument erstellt, öffnet oder speichert. Wichtig ist in diesem Zusammenhang, dass Zugriffe auf diese Bibliothek für den Anwender unsichtbar sind: Man verwendet hierfür auch den Begriff *Transparenz*.

Die als *User Space* (häufig auch *Userland* genannt) bezeichnete Schicht wird im weiteren Verlauf ausgeblendet. Stattdessen folgt eine Erläuterung der abstrakten Begriffe Dateisystem, Prozess-, Speicher- sowie Ein- und Ausgabemanagement.

Dateisystem/Dateimanagement

Vollständige Anwendungen oder Teile hiervon werden, ebenso wie selbst erstellte Dokumente (z. B. Briefe, Programme, Fotos, Grafiken etc.), unter dem Sammelbegriff »Dateien« geführt. Diese Dateien müssen erzeugt, bearbeitet, sortiert und gegebenenfalls auch wieder gelöscht werden. Darüber hinaus muss der unberechtigte Zugriff auf diese Dateien verhindert werden. Diese Aufgaben übernehmen *Dateisysteme* (englisch: *File Systems*). Dateisysteme sorgen außerdem dafür, dass die Verwaltung von Dateien auf besonderen Geräten erfolgen kann, wie Festplatten, USB-Sticks oder SD-Karten. Der Fokus liegt hierbei auf dem schnellen, sicheren und zuverlässigen Zugriff auf diese Dateien.

Hinweis

Dateisysteme kümmern sich zudem auch darum, dass nach einem unzulässigen (und unerwünschten) Zwischenfall, wie einem Stromausfall oder dem unbeabsichtigten Ausschalten des Computers, ohne das System ordnungsgemäß »herunterzufahren«, die Dateien erhalten bleiben. Moderne *Journaling File Systems*, wie `ext4` eines ist, sind häufig sogar dazu in der Lage, beschädigte Dateien (man bezeichnet diese englisch auch als »corrupted files«) wieder in einen solchen Zustand zu versetzen, dass im schlimmsten Fall nur die zuletzt durchgeführten Arbeiten verloren sind. Dieses Szenario, das ich oft genug selbst erlebt habe, hat mich daher gelehrt: Sichere ständig die Dateien! Viele Programme führen die automatische Sicherung der letzten Änderungen regelmäßig aus, z. B. alle fünf Minuten.

Bekannte Dateisysteme auf Windows-Betriebssystemen sind *FAT16* (FAT = File Allocation Table), *FAT32* und *NTFS*. Da die genannten Dateisysteme durch Patente geschützt sind (bzw. nicht frei im Sinne der GNU Public License GPL sind), verwenden moderne Linux-Systeme andere Dateisysteme. Besonders beliebt ist hier derzeit das Dateisystem *ext4*, wobei *ext* für *Extended File System* steht.

Hinweis

In Unix- und Linux-Betriebssystemen werden über die oben genannten Dokumente hinaus auch Geräte, wie der Bildschirm, die Tastatur, Maus, Laufwerke, Drucker, Scanner usw., als Datei bezeichnet und wie eine solche behandelt. Das bedeutet, dass mit einer *open()*-Funktion unter Einsatz der Treiberschicht zunächst eine Verbindung zwischen dem Kernel und der »echten« Hardware hergestellt wird. Erst wenn dies erfolgreich geschehen ist, kann der Kernel unter Verwendung einer *read()*-Funktion Daten von diesem Gerät entgegennehmen oder durch Einsatz einer *write()*-Funktion Daten an das Gerät übertragen. Wird das Gerät für die aktuellen Aufgaben nicht mehr benötigt, so muss die Verbindung durch Einsatz der *close()*-Funktion wieder abgebaut und die verwendeten Ressourcen müssen wieder freigegeben werden. Selbstverständlich ist es auch hier möglich, die Funktionalität durch die geeignete Vergabe von Zugriffsrechten gezielt einzuschränken oder zu vergrößern. Als Beispiel mag hier der Zugriff auf einen Drucker gelten: Während »normalen« Sachbearbeitern nur der Monochrom-Druck erlaubt sein soll, dürfen Mitarbeiter des mittleren und gehobenen Managements auch die Farbfunktionalität von Druckern nutzen.

Prozessmanagement

Linux ist ein sogenanntes *Multiuser-* und *Multitasking*-Betriebssystem. Dies bedeutet, dass gleichzeitig mehrere Anwender das System nutzen können, wobei jeder Nutzer mehrere verschiedene Anwendungen ausführen kann. Die Nutzung verschiedener Anwendungen erfolgte bei älteren Computern, deren CPU nur einen Prozessorkern hatte, nur scheinbar gleichzeitig, da die vorhandenen Ressourcen (CPU, Arbeitsspeicher etc.) auf alle angemeldeten Benutzer und den von ihnen genutzten Anwendungen verteilt werden müssen.

Da das Betriebssystem intern weitere Dienste ausführt, werden die zur Verfügung stehenden Ressourcen noch weiter aufgeteilt. Hierfür ist das sogenannte *Prozessmanagement* zuständig: Jedem Dienst wird eine bestimmte Zeit zur Verfügung gestellt, in der er exklusiv ausgeführt wird. Die den Diensten zur Verfügung gestellte Zeit richtet sich nach den Prioritäten, die ihnen zugewiesen werden. So verfügen systeminterne Dienste über höhere Prioritäten als Anwendungen aus dem Userland und werden daher bevorzugt ausgeführt. Ist die Zeit für einen Dienst abgelaufen, so schaltet das Prozessmanagement über den sogenannten *Scheduler* auf den nächsten Dienst oder Prozess um. Die »Zeitscheiben«, die der Scheduler den einzelnen Diensten zuweist, sind so kurz, dass Anwender den Eindruck gewinnen, dass sämtliche Dienste gleichzeitig ausgeführt werden.

Hinweis

Moderne CPUs verfügen häufig über mehrere Prozessorkerne, sodass hier mehrere Prozesse wirklich gleichzeitig (Real time) ausgeführt werden. Überschreitet die Anzahl der laufenden Dienste aber die Anzahl der Kerne (was bereits der Fall ist, wenn nur das Betriebssystem läuft und keine weitere Anwendung gestartet wurde), so tritt auch hier wieder der Effekt auf, dass die Mehrzahl aller Prozesse wieder nur quasi-gleichzeitig ausgeführt wird.

Speichermanagement

Das *Speichermanagement* hat verschiedene Aufgaben. Zunächst einmal erhält jeder der im Abschnitt »Prozessmanagement« erwähnten Dienste einen eigenen Bereich des gesamten zur Verfügung stehenden Arbeitsspeichers. Hierdurch soll der Einfluss eines unerwarteten Ereignisses (z. B. der »Absturz« eines Prozesses) auf andere Prozesse verhindert werden: Wären die Speicherbereiche nicht gegeneinander »abgeschottet«, so würden diese unerwarteten Ereignisse im günstigsten Fall den Computer vollständig zum Stillstand bringen (»aufhängen«). Wesentlich schlimmer wäre aber der Fall, wenn ein »Amok laufender« Dienst den von einem anderen Dienst genutzten Speicherbereich überschreiben würde: Dies würde dazu führen, dass die Ergebnisse dieses zweiten Dienstes nicht mehr zuverlässig sind.

Die zweite wichtige Aufgabe des Speichermanagements besteht darin, den verschiedenen Diensten einen eindeutigen Speicherbereich zuzuweisen. (Arbeits-)Speicher wird über Adressen angesprochen. Die Zuweisung eines eindeutigen Adressbereichs an die verschiedenen Dienste wird durch das Speichermanagement gewährleistet, das letztendlich auch dafür verantwortlich ist, die Speicherbereiche verschiedener Dienste gegeneinander abzusichern.

Physikalischer Speicher (Arbeitsspeicher) ist eine begrenzte Ressource. Abhängig von der Anzahl der ausgeführten Prozesse und Anwendungen ist die Kapazität dieses Speichers irgendwann ausgeschöpft. In solchen Fällen sorgt das Speichermanagement dafür, dass von nicht aktiven Anwendungen belegter Speicher auf andere Geräte, wie Festplatten oder SD-Karten, ausgelagert wird. Hierdurch erhöht sich der zur Verfügung stehende Arbeitsspeicher scheinbar (virtuell), sodass man deshalb auch von *virtuellem Arbeitsspeicher* spricht. Die Beschaffung virtuellen Speichers ist die dritte Aufgabe des Speichermanagements.

Hinweis

Virtueller Speicher wird nicht nur von nicht-aktiven Anwendungen genutzt. Denkbar ist beispielsweise ebenfalls, dass eine Anwendung für ihre Ausführung mehr Arbeitsspeicher benötigt, als physikalisch vorhanden ist. In solchen Fällen beschafft das Speichermanagement ebenfalls virtuellen Speicher.

Unter Linux – und somit auch unter Embedded Linux – wird zu diesem Zweck die sogenannte *swap*-Partition oder eine sogenannte swap-Datei verwendet. Auf Windows-Systemen dient hierzu die Datei *pagefile.sys*.

Ein- und Ausgabemanagement

Die Aufgabe des *Ein-* und *Ausgabemanagements* liegt darin, die Kommunikation zwischen externen Geräten (Tastaturen, Computermäusen, Bildschirmen, Druckern, CD-, DVD- oder BD-Laufwerken, SD-Karten etc.) und dem Betriebssystem zu ermöglichen und zu koordinieren. Hierzu bietet das Ein- und Ausgabemanagement Funktionen zum Öffnen und Schließen eines Kommunikationskanals zwischen einem solchen Gerät und dem Betriebssystem sowie zum Lesen von Daten von einem Gerät und zum Schreiben von Daten auf ein Gerät. Dies erfolgt aber nicht unmittelbar, da diese Geräte selbst über besondere Eigenschaften verfügen: So kann einem Drucker mitgeteilt werden, wel-

che Abmessungen ein Dokument haben soll oder in welcher Schriftart ein Dokument ausgegeben werden soll. Für diese Art von Aufgaben liegt zwischen Betriebssystem und Gerät mit den Gerätetreibern eine weitere Schicht, die in Abbildung 1.1 als »Treiber und andere hardwareabhängige Programme« aufgeführt ist.

1.5 Beliebte Linux-Distributionen

Linux hat eine sehr große Beliebtheit erlangt. Dies gilt zwar für den Desktop nur sehr eingeschränkt, im Bereich der (Web-)Server und nicht zuletzt in der Industrieautomatisierung gilt Linux aber als Marktführer. Trotz seiner geringen Verbreitung auf heimischen Arbeitsplätzen oder im Büroeinsatz sind derzeit circa 140 verschiedene Distributionen frei verfügbar. Neben sehr speziellen Varianten, z. B. für Ton- oder Videostudioanwendungen, haben sich einige Distributionen auf den »normalen« PC-Anwender konzentriert, der im Allgemeinen wenig Lust verspürt, sich in die Interna von Betriebssystemen einzuarbeiten, sondern sie ausschließlich als »Mittel zum Zweck«, als Werkzeug, betrachten und entsprechend nutzen will. Insbesondere die Ubuntu-Distribution des südafrikanischen Unternehmens Canonical und seinem Besitzer Mark Shuttleworth hat hier lange Zeit den Ton angegeben. Einige Entscheidungen zum Wechsel der Benutzeroberfläche von Gnome nach Unity haben dazu geführt, dass sich die Verhältnisse etwas verschoben haben. Seit nunmehr etwa zwei Jahren liegt die auf Ubuntu basierende Distribution Linux Mint in der Rangliste ganz vorne. Die Webseite `http://www.distrowatch.com`, die viele EDV-Fachzeitschriften als Referenz verwenden, führt regelmäßig die Rangliste der beliebtesten Linux-Distributionen auf. Stand Mitte Oktober 2015 sind die ersten zehn Plätze folgendermaßen belegt:

Rang	Name	Aktuelle Version (Mitte Oktober 2015)	Kommentar
1	Mint	Linux Mint 17.1	Führt die Beliebtheitsskala mindestens seit Anfang 2013 an. Setzt auf den Xfce-Desktop, der sich durch besonders sparsamen Umgang mit den zur Verfügung stehenden Ressourcen auch für leistungsschwächere Computer sehr gut eignet. Basiert auf Debian und Ubuntu und bietet neben Xfce auch die Desktops Cinnamon, Gnome, KDE und Mate an. Wendet sich an Einsteiger.
2	Ubuntu	Ubuntu 14.10	Standardmäßig wird seit einiger Zeit Unity als Desktop eingerichtet, obwohl weiterhin auch Gnome unterstützt wird. Wendet sich hauptsächlich an Einsteiger, wird aber auch im Serverumfeld eingesetzt. Basiert auf Debian, bietet aber im Gegensatz hierzu auch sogenannte Closed-Source-Softwarepakete an. War mehrere Jahre die klare Nummer eins dieser Liste, musste diesen Platz aber (möglicherweise wegen des Unity-Desktops) an Linux Mint abtreten.

Tabelle 1.1: Die zehn beliebtesten Linux-Distributionen gemäß Distrowatch

Rang	Name	Aktuelle Version (Mitte Oktober 2015)	Kommentar
3	Debian	Debian GNU/ Linux 7.0	Basis-Distribution für Linux Mint und Ubuntu. Setzt ausschließlich auf Open-Source-Software. Die meiner Meinung nach beliebteste Linux-Distribution für den Raspberry Pi ist *Raspbian*, die vollständig auf Debian basiert.
4	openSuse	openSuse 13.2	Deutsche Distribution aus Nürnberg. Verwendet standardmäßig KDE als Desktop, der besonders viele individuelle Einstellungsmöglichkeiten bietet. Das Projekt wird von der Firma Novell unterstützt und bietet ebenfalls eine große Zahl verschiedener Desktops. Laut Distrowatch gibt es auch eine angepasste Version für den Raspberry Pi.
5	Fedora	Fedora 21	Der Open-Source-Wegbereiter für die kommerzielle Distribution Red Hat. Legt besonderen Wert auf hohe Aktualität insbesondere auf die Unterstützung neuer Hardware. Bewährte Software und Treiber wird dann von Red Hat in die kommerzielle Version übernommen.
6	Mageia	Mageia 5 Beta 1	Wurde im September 2010 von ehemaligen Mitarbeitern der französischen Firma Mandriva gegründet. Unterstützt eine Vielzahl von Desktops.
7	CentOS	CentOS 6.6	Erhebt den Anspruch einer 100-prozentigen Kompatibilität der kommerziellen Distribution RHEL (Red Hat Enterprise Linux, siehe Fedora), dabei aber völlig kostenlos zu sein. Unterstützt die Desktops Gnome und KDE.
8	Arch	2012.11.01	Bietet besonders viele Möglichkeiten, tief in die Linux-Welt »einzutauchen«. Wendet sich in erster Linie an erfahrene Linux-Anwender. Eine Raspberry-Pi-Version ist verfügbar.
9	elementary	elementary OS 0.3 Beta 1	Basiert ebenfalls auf Ubuntu. Wendet sich an Linux-Einsteiger.
10	Zorin	Zorin OS 9	Basiert auf Ubuntu und erhebt den Anspruch, noch mehr als Ubuntu selbst, sich besonders für Einsteiger zu eignen. Es verfügt über eine Windows-ähnliche Benutzeroberfläche und will hiermit erreichen, dass Linux als »echte« Alternative zu Microsoft Windows wahrgenommen wird.

Tabelle 1.1: Die zehn beliebtesten Linux-Distributionen gemäß Distrowatch (Forts.)

Alle in Tabelle 1.1 aufgeführten Distributionen sind Desktop-Distributionen für x86- bzw. ia64-Prozessoren. Auf den bereits mehrfach erwähnten Mini-Computern Raspberry Pi, BeagleBone Black, Cubietruck oder Banana Pi kommt keine dieser Varianten zum Einsatz: Die in Tabelle 1.1 genannten Distributionen sind hauptsächlich für den

Einsatz auf 32- oder 64-Bit-CPUs der Firmen Intel und AMD optimiert. Im Raspberry Pi wird mit dem BCM2835 hingegen ein auf einer ARM-Architektur basierender Mikrocontroller eingesetzt. Dies hat zur Folge, dass – selbst, wenn die Ressourcen vergleichbar wären – die genannten Linux-Distributionen auf dem Raspberry Pi gar nicht funktionieren würden: Viele Freiwillige haben sich aber der Aufgabe angenommen, einige der oben genannten Distributionen an die Raspberry-Pi-Hardware (und die der anderen genannten Mini-Computer) anzupassen und zu portieren. In der Kommentarspalte von Tabelle 1.1 finden Sie daher einige Hinweise auf Portierungen bekannter und beliebter Linux-Distributionen für den Raspberry Pi. Die vermutlich am häufigsten genutzte ist *Raspbian*, weshalb wir uns mit dieser Variante befassen werden.

Hinweis

Noch einmal sei herausgestellt: Dies ist nicht als Herabwürdigung anderer Distributionen zu verstehen. Bei Raspbian kann man aber aufgrund seiner großen Verbreitung davon ausgehen, dass im Internet Hilfe für dieses System leichter zu erhalten ist als für andere Systeme.

1.6 Linux installieren

Besonders als Windows- oder Mac-OS-X-Anwender werden Sie sich bei dieser Überschrift möglicherweise die Frage stellen: Was soll denn das? Schließlich habe ich doch eine Linux-Distribution, denn mein Mini-Computer funktioniert doch!

Diese Frage ist sehr berechtigt. Sie sollten aber berücksichtigen, dass die genannten Mini-Computer im Vergleich zu Desktop-PCs nicht sonderlich leistungsstark sind. Wenig Arbeitsspeicher, der zu einem guten Teil noch von der Hardware belegt wird, sowie weniger leistungsstarke Mikrocontroller lassen die Softwareentwicklung hiermit zu einem Geduldsspiel werden. Es ist daher sehr zu empfehlen, sich nicht ausschließlich auf das Embedded Linux Ihres Mini-Computers zu verlassen.

Die Installation eines weiteren Linux-Systems auf Ihrem Desktop-PC – z. B. in einer virtuellen Maschine – führt aufgrund der viel größeren Leistungsfähigkeit dazu, viele Arbeiten, wie die Entwicklung eines neuen Kernels oder von root-Dateisystemen, erheblich schneller durchführen zu können. Unter Einsatz der Distribution Linux Mint werden nachfolgend zwei verschiedene Möglichkeiten hierfür aufgezeigt.

Linux Mint können Sie von der Webseite `http://www.linuxmint.com/download.php` als ISO-Image herunterladen. Um den Download, der in der aktuellen Version etwa 1,4 Gigabyte umfasst, im Falle eines Datenverlustes nicht immer wieder aufs Neue durchführen zu müssen, sollten Sie das ISO-Image von Linux Mint auf eine DVD übertragen: Hierfür können Sie ein sogenanntes Imaging-Programm verwenden, von denen es einige im Internet sogar kostenlos gibt. Eines dieser Programme für Windows heißt Win32DiskImager.

Tipp

Alternativ können Sie natürlich das ISO-Image auf Ihrer Festplatte belassen und mit einem Programm wie *Virtual Clone Drive* ein DVD-Laufwerk emulieren. Äußeres Merkmal eines CD-, DVD- oder BD-Laufwerkes unter Windows ist ein Laufwerksbuchstabe:

Weisen Sie dem emulierten Laufwerk also einen Laufwerksbuchstaben zu, und Virtual Clone Drive wird ihn dann so verwenden, als handele es sich um ein physikalisches Laufwerk.

Die Software erkennt dann beim Starten einer Datei, ob es sich hierbei um ein sogenanntes Image handelt. Wann immer dies zutrifft, emuliert Virtual Clone Drive ein CD-, DVD- oder BD-Laufwerk (BD = Blue Ray Disk): Das Image wird dann genauso ausgeführt, als hätten Sie es auf eine CD/DVD/BD »gebrannt«.

1.6.1 Parallele Installation von Linux Mint zum vorhandenen Betriebssystem

Hinweis

Wenn im weiteren Verlauf dieses Buches die Rede von »Linux« ist, dann werden Sie inzwischen aus dem Kontext erkennen, ob nur der Kernel oder eine Distribution gemeint ist. Bitte verzeihen Sie mir also diese bereits oben erklärte »Schlampigkeit«.

Mit der parallelen Installation von Linux zu einem anderen vorhandenen Betriebssystem eröffnen sich Ihnen ohne Einschränkungen sämtliche Möglichkeiten, die Leistungsfähigkeit Ihres Computers auszunutzen. Das Verfahren wird allgemein als Dual-Boot-Installation bezeichnet. Dabei wird Linux in einem nicht-genutzten Bereich Ihrer Festplatten installiert, wobei die Daten der Standardinstallation (also Windows oder Mac OS X) nicht verloren gehen.

Hinweis

Bevor Sie den folgenden Anleitungen nachgehen, sollten Sie auf jeden Fall Ihr vorhandenes Betriebssystem sichern, mindestens aber Ihre wichtigen Daten. Auch wenn die Parallelinstallation eines neuen Betriebssystems inzwischen zu den Routinetätigkeiten zählt und im Allgemeinen problemlos funktioniert, können unvorhersehbare Situationen eintreten, die zu einem Verlust Ihrer wichtigen Daten führen können.

Legen Sie nun die oben erzeugte DVD in den DVD-Leser ein und starten Sie Ihren Computer neu (oder Sie verwenden das bereits erwähnte kostenlos nutzbare Programm *Virtual Clone Drive*). Gegebenenfalls müssen Sie die Bootreihenfolge im BIOS Ihres PCs ändern, bei moderneren Computern erkennt das BIOS aber automatisch, dass ein bootfähiges Medium im DVD-Leser eingelegt ist. Die Grundinstallation von Linux Mint ist dann in wenigen Minuten abgeschlossen.

Dual-Boot-Installation auf Windows-Rechnern

Sobald Linux Mint installiert ist, sollten Sie den Computer neu starten. Allgemein wird erwartet, dass *grub*, der sogenannte *Grand Unified Boot Loader*, Ihnen nach dem Neustart einen (wenig schönen) Auswahldialog anzeigt, mit dem Sie festlegen können, ob Sie nun Windows oder eben doch Linux Mint starten möchten.