IN DIESEM KAPITEL

lernen Sie, was ein Betriebssystem ist und wozu man es benötigt,

können Sie sich einen historischen Überblick über die Entwicklung der Betriebssysteme verschaffen,

erhalten Sie einen Eindruck von der Vielfalt von Betriebssystemen für die unterschiedlichsten Einsatzzwecke.

Kapitel 1
Ein bisschen Einführung

Bevor Sie in den späteren Kapiteln die »Innereien« von Betriebssystemen wie das Dateisystem oder den Scheduler näher kennenlernen, ist es hilfreich, zunächst eine Außensicht auf diese Form von Software zu entwickeln. Dazu muss das Betriebssystem von anderer Software abgegrenzt werden, und seine Aufgaben müssen definiert werden. Wie immer bei komplizierten technischen Systemen entstehen diese nicht ad hoc, sondern unterliegen einer Evolution. Daher geht es in diesem Kapitel auch um die Entwicklung dieser Softwarekategorie von den ersten Schedulern bis zur heutigen Vielfalt von Universal- und Spezialbetriebssystemen.

Was ist ein Betriebssystem?

In diesem Buch geht es um *Betriebssysteme*, und somit ist es sicherlich sinnvoll, zunächst einmal zu klären, was das überhaupt ist, ein Betriebssystem.

Ingenieure nutzen gern Normen, und interessanterweise gibt es eine DIN, die den Begriff *Betriebssystem* direkt definiert:

> »Die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften der Rechenanlage die Grundlage der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen.« [15]

Nun, das klingt ein bisschen antiquiert (kein Wunder, die Norm stammt aus dem Jahr 1988), trifft aber den Kern der Sache recht gut. Offenbar handelt es sich bei Betriebssystemen um *Software*, und zwar um die, die für den eigentlichen Betrieb des Rechensystems notwendig ist. Microsoft Word oder der Firefox-Browser sind zum Betrieb nicht unbedingt nötig (man könnte ja ganz ohne ihr Zutun beispielsweise ein Spielchen laden), aber die *darunterliegende* Softwareschicht, die bei einem normalen PC häufig *Windows* heißt und die beispielsweise das Laden des Spielchens oder einer anderen *Applikation* übernimmt, die ist essenziell. Das ist das Betriebssystem, um das es in diesem Buch geht.

Das war jetzt möglicherweise ein bisschen viel Fachjargon für den Anfang. Wieso gibt es Schichten von Software, die offenbar auch noch eine Art Stapel bilden, und was ist mit »Applikation« gemeint? Die Beantwortung der ersten Frage soll noch etwas verschoben werden. Fakt ist, dass auf heutigen Rechensystemen sehr viele verschiedene Softwarekomponenten existieren, und damit man den Überblick behält, muss man diesen Wust an Software strukturieren. Eine Möglichkeit dafür ist das Schichtenmodell, das Sie im Kapitel 4 »Grundlegende Begriffe und Abstraktionen« genauer kennenlernen werden. Die zweite Frage ist deutlich einfacher zu beantworten: Unter »Applikationen« versteht man die Gesamtheit der Programme, die dem Nutzer eines Systems zur Verfügung stehen und die nicht zum Betriebssystem gehören.

Seit dem Siegeszug der Smartphones hat sich für »Applikation« der Kurzbegriff *App* etabliert. Im Deutschen gibt es auch noch den Begriff »Anwendungsprogramm«.

Charakteristisch ist des Weiteren, dass der Nutzer von der Arbeit des Betriebssystems verhältnismäßig wenig mitbekommt. Den größten Anteil am Code eines Betriebssystems nehmen Funktionen ein, die audiovisuell nicht in Erscheinung treten, sondern gewissermaßen »unter der Haube« still und zuverlässig ihre Arbeit verrichten. Ihre Existenz wird häufig erst durch den Nutzer wahrgenommen, wenn einmal etwas schiefgeht, beispielsweise der Zugriff auf die abgelegten Daten unmöglich ist oder die Netzverbindung nicht zustande kommt.

Das Betriebssystem ist sozusagen ein mehr oder minder unsichtbares Helferlein, das Ihnen gestattet, auf einfache Weise mit dem System zu kommunizieren und dessen Ressourcen bestmöglich auszuschöpfen. Die »Abarbeitung von Programmen« spielt dabei eine große Rolle.

Zu einem Betriebssystem gehört in der Regel nicht nur der im Hauptspeicher abgelegte *Kern* (im Englischen häufig *Kernel* genannt), sondern eine große Anzahl von Hilfsprogrammen, die es überhaupt gestatten, effizient mit dem System zu arbeiten, es zu überwachen und zu diagnostizieren. Im Kontext sogenannter *eingebetteter Systeme* (das sind Rechensysteme, die Teil eines größeren Apparats oder einer Maschine sind, beispielsweise eines Fernsehers oder eines Computertomografen) wird das Betriebssystem häufig auch *Firmware* genannt.

Betriebssystem oder Firmware?

Die Abgrenzung beider Begriffe voneinander ist nicht ganz einfach.

Von »Betriebssystem« spricht man, wenn es um »universelle« Computer geht, die potenziell alle mögliche Software abarbeiten können. Der Nutzer kann zur Laufzeit des Systems entscheiden, welches konkrete Programm abgearbeitet werden soll. Insbesondere ist es möglich, mit Hilfe von Entwicklungswerkzeugen neue Software für das System herzustellen, die dann ebenfalls abgearbeitet werden kann. Es gibt also (mindestens) zwei Softwareebenen: das Betriebssystem und die mit dessen Hilfe abgearbeiteten Programme.

Der Begriff »Firmware« wird stattdessen verwendet, wenn die Software im Kontext eines eingebetteten Systems eingesetzt wird. Die Auswahl und das Abarbeiten von wie auch immer gearteten Applikationsprogrammen ist dabei normalerweise nicht vorgesehen. Die angebotene Funktionalität steht im Allgemeinen fest (das englische *firm* bedeutet »fest«). Eine Teilung in mehrere Ebenen ist somit unmöglich; die Firmware bildet einen monolithischen Block.

Es gibt natürlich viele weitere Definitionen des Begriffs »Betriebssystem«; jeder Autor eines entsprechenden Lehrbuches formuliert natürlich eine eigene. Stellvertretend sei Andrew S. Tanenbaum zitiert, der zwei verschiedene Rollen eines Betriebssystems unterscheidet: ([27]):

- ✔ das Betriebssystem als Verwalter der Systemressourcen und
- ✔ das Betriebssystem als Erweiterung des Systems, die deutlich einfacher zu programmieren ist als die reine Hardware.

Es muss betont werden, dass der Begriff »Betriebssystem« gar nicht so einfach zu definieren ist und im Regelfall ein gewisser Interpretationsfreiraum besteht.

Im Abschnitt »Aufgaben eines Betriebssystems« dieses Kapitels erfahren Sie genauer, welche Aufgaben ein Betriebssystem zu erfüllen hat (und was besser durch Applikationen realisiert wird). Zuvor wollen wir aber einen kurzen historischen Exkurs die Entwicklung von Betriebssystemen betreffend unternehmen.

Eine (ganz) kurze Geschichte der Betriebssysteme

Komplexe technische Systeme unterliegen einem historischen Entwicklungsprozess; Betriebssysteme bilden dabei keine Ausnahme. Viele zugrunde liegende Konzepte, Algorithmen und Mechanismen wurden über längere Zeiträume hinweg immer wieder

verbessert, adaptiert und optimiert. Um die Komplexität heutiger Systeme zu verstehen, ist es daher durchaus hilfreich, sich einen kurzen Überblick über wesentliche Entwicklungsetappen der Betriebssysteme zu verschaffen.

Nicht weiter überraschend ist, dass die Geschichte der Betriebssysteme sehr eng an die Geschichte der Rechentechnik, genauer: der Hardware, gekoppelt ist.

Im Anfang war das Nichts

In der Anfangsphase der Computer, die von 1941 bis etwa Anfang der 1960er-Jahre dauerte, gab es typischerweise kein Betriebssystem. Man war gewissermaßen froh, dass diese überaus komplexen Maschinen überhaupt funktionierten und plausible Ergebnisse lieferten. Das Rechensystem wurde mit einem Programm »gefüttert«, arbeitete dieses ab und lieferte Ergebnisse zunächst mittels Lampenfeldern, später über Lochkartendrucker. Die allerersten Rechner hatten Namen wie »ENIAC« und »EDVAC« und wurden infolge ihrer immensen Kosten zunächst durch durch Militär und Geheimdienste genutzt, beispielsweise zur Berechnung ballistischer Tabellen für die Artillerie oder der Objektive von Spionagekameras. Sehr bald kamen jedoch auch zivile wissenschaftliche und, ein bisschen später, auch wirtschaftsorientierte Anwendungsgebiete hinzu.

Großrechner (Mainframes)

Mitte der 1950er-Jahre etablierten sich die ersten Unternehmen, die kommerziell Rechensysteme anboten, nämlich in Gestalt einer neuen Gattung Hardware, der sogenannten *Großrechner* oder *Mainframes*. Die Software für diese Systeme war mittlerweile so komplex, dass man diese in zwei »Schichten« strukturierte. Direkt auf der Hardware setzte eine Softwareschicht auf, die von der »nackten« Hardware abstrahierte und gewisse Basisdienste anbot. Diese Schicht nannte man *Operation System* oder *Operating System* – das *Betriebssystem* war geboren. Oberhalb dieser Schicht residierte das *Applikationsprogramm* (zunächst immer nur eines!), das die Basisdienste des Betriebssystems benutzte und die vom Nutzer gewünschte Funktionalität erbrachte (Abbildung 1.1).

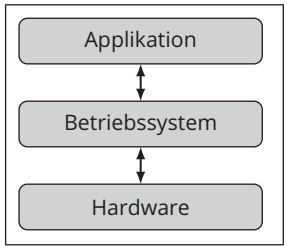

Abbildung 1.1: Einfache Schichtenarchitektur eines Rechensystems

Weite Verbreitung fanden Mainframes des Typs »IBM System/360«, die mit dem Betriebssystem OS/360 ausgerüstet waren. Über viele Entwicklungsetappen wurde dieses zur heute existierenden Mainframe-Architektur *z Systems* und dem dazugehörigen Betriebssystem

z/OS weiterentwickelt. Innerhalb der Klasse der Mainframes ist Kompatibilität, also die Übertragbarkeit von Programmen zwischen verschiedenen Generationen, ausgesprochen wichtig. Infolge der hohen Anschaffungs- und Betriebskosten bilden Mainframes für sich eine verhältnismäßig abgeschlossene Welt.

Minicomputer

Die fortschreitende Miniaturisierung führte etwa ab 1970 zu einer neuen Klasse von Rechnern, die etwas weniger leistungsfähig, dafür jedoch deutlich weniger kostenintensiv als Mainframes waren. Bezugnehmend auf ihre geringere Größe wurden sie *Minicomputer* genannt. Nach heutigen Maßstäben ist die Bezeichnung grob irreführend; sie spielt darauf an, dass der Rechner nicht mehr einen ganzen Raum einnahm, sondern etwa die Größe eines halbhohen Schrankes hatte.

Ein wichtiger Hersteller dieser Systeme war die Digital Equipment Corporation (DEC), deren PDP-Serie und insbesondere die VAX genannten Rechner sehr populär in Industrie und Forschung waren. Betriebssystemseitig wurden sie zum einen durch die proprietäre Eigenentwicklung VMS betrieben (VMS bedeutet *Virtual Memory System* und weist auf das hier erstmals verwirklichte Konzept des *virtuellen Speichers* hin, das Sie im Kapitel 9 ziemlich genau kennenlernen werden). Zum anderen wurden Minicomputer mit *UNIX* betrieben, einem damals brandneuen, in den Bell Laboratories von Ken Thompson und Dennis Ritchie entwickelten Betriebssystem, das zusammen mit der Programmiersprache C entworfen wurde. Während VMS heute nur noch ein Nischendasein führt, wurde UNIX zum Stammvater für eine ganze Betriebssystem-Familie, von denen Ihnen einige in diesem Buch noch viele Male begegnen werden.

Minicomputer als Geräteklasse wurden in den 1980er-Jahren zunächst von den wiederum preiswerteren Workstations verdrängt, die zumeist ebenfalls unter UNIX genutzt wurden und ihrerseits Mitte der 1990er-Jahre durch leistungsstarke PCs ersetzt wurden.

Mikrocomputer

Hardwaretechnisch gesehen entstand die nächste Geräteklasse, die der Mikrocomputer, gegen Ende der 1970er-Jahre. Sie zeichneten sich durch die erstmals verwendeten Mikroprozessoren aus, die im Vergleich zu Minicomputern wiederum bedeutende Kosten- und Größenreduktionen erlaubten. Nun wurde es möglich, dass einzelne Personen Computer für sich allein beanspruchten, sowohl in der Form von Heimcomputern als auch professionell als *Personal Computer*.

In dieser Gerätekategorie kann man drei Subkategorien unterscheiden:

Heimcomputer

Diese Systeme wurden mit preiswerten 8- oder 16-Bit-CPUs gebaut und waren als Hobbygeräte sowohl zum Spielen als auch für einfache »ernsthafte« Verarbeitungsaufgaben gedacht. Systeme wie Commodore 64 und Amiga, Sinclair ZX81 und Spectrum oder Atari ST wurden in ungeheuren Stückzahlen gebaut und verkauft. Die Betriebssysteme bestanden zunächst

häufig aus einer Sammlung wichtiger *Routinen*, die im ROM des Rechners abgelegt waren. Später entwickelte man auch für diese kleine Geräteklasse »richtige« Betriebssysteme wie CP/M von Digital Research oder AmigaOS des Commodore Amiga, das bereits eine grafische Oberfläche anbot sowie Multitasking beherrschte.

Die ungeheure Vielfalt an Herstellern und zueinander inkompatiblen Geräten und Betriebssystemen (das bedeutet, man konnte die Software eines Systems im Normalfall nicht auf einem anderen Gerät betreiben) sowie die zögerlich erfolgende Weiterentwicklung der Hardware sorgte für das weitgehende Aussterben dieser Rechnerkategorie in den 1990er-Jahren.

Apple-Computer

Nachdem die 1976 gegründete Firma Apple Inc. zunächst einen sehr erfolgreichen Heimcomputer, den Apple II, entwickelt hatte, konzentrierte man sich mit der *Macintosh*-Familie alsbald auf höherwertige Rechner, die in der Folge besonders von Kreativen geschätzt wurden. Apple gebührt der Verdienst, mit *macOS* als Erster eine grafische Nutzerschnittstelle (*Graphical User Interface*, GUI) für ein kommerzielles System etabliert zu haben. Dieses Betriebssystem, mittlerweile *OS X* genannt, wird bis heute entwickelt und ausschließlich für die Mac-Reihe von Apple genutzt.

Der IBM-PC und seine Nachfahren

Im Jahr 1981 entwickelte die Firma IBM auf der Basis des Intel-Prozessors 8086 einen preiswerten Computer für Büroanwendungen, dessen Architektur bewusst offen dokumentiert wurde, den sogenannten *IBM-PC*. In den folgenden Jahren etablierte sich diese Architektur als Quasistandard für Bürocomputer, wobei die in rascher Folge erscheinenden Prozessoren der Firma Intel (80286, 80386, Intel Pentium …) sowie dazu kompatible CPUs von weiteren Herstellern wie AMD als CPU Verwendung fanden. Die offene und modulare Architektur motivierte zahllose Mitbewerber, zum original IBM PC kompatible Rechner zu entwickeln. Zum ursprünglichen Desktop-Format kamen alsbald auch transportable Modelle hinzu; mittlerweile sind die Mehrzahl der verkauften Systeme Notebooks. Der enorme Wettbewerb und Kostendruck ließ IBM bereits Mitte der 1990er-Jahre aus diesem Markt aussteigen. Die Plattform wird regelmäßig weiterentwickelt, und die genutzten Bussysteme und Prozessoren werden aktualisiert. Die meisten verkauften »Computer für den persönlichen Bedarf« gehören zu dieser Kategorie – trotz modernerer Systeme wie Tablets.

Als Betriebssystem des IBM PC und kompatibler Systeme kam zunächst *MS-DOS* des damals völlig unbekannten Herstellers *Microsoft* zum Einsatz. Die Legenden, wie es zu dieser ungewöhnlichen Entscheidung kam, sind zahlreich. Angeblich soll Gary Kildall, der Entwickler des für 8-Bit-Computer häufig eingesetzten Betriebssystems CP/M, die verantwortlichen IBM-Manager brüskiert haben, sodass diese eher notgedrungen mit dem 26-jährigen Bill Gates über die Lieferung eines Betriebssystems verhandelten, das dieser über einen Subunternehmer innerhalb kurzer Zeit zur Verfügung stellen konnte.

Nachdem die grafische Oberfläche des Apple Macintosh ziemliches Aufsehen erregte, begann man auch bei Microsoft, eine grafische Oberfläche unter dem Namen *Windows* zu

entwickeln. Diese wurde zunächst als »Aufsatz« für MS-DOS konzipiert; die Version 1.0 erblickte 1985 das Licht der Welt. Weitere Versionen folgten rasch. Gleichzeitig entwickelte Microsoft eine neue Betriebssystemarchitektur unter dem Namen *Windows NT*, die 1993 vorgestellt wurde. Es ist gewissermaßen der Urvater des gegenwärtig aktuellen *Windows 11*. MS-DOS erwies sich schon in den 1990er-Jahren als nicht mehr zeitgemäß, es endete 1994 mit der Version 6.22. Die auf MS-DOS basierende Windows-Entwicklungslinie endete 2006 mit *Windows ME*.

Darüber hinaus gibt es für den PC ein weiteres wichtiges Betriebssystem, das 1991 als Hobbyprojekt des finnischen Informatikstudenten Linux Torvalds entstand: *Linux*. Im Prinzip handelt es sich um eine Neuimplementierung von UNIX, das als Gemeinschaftsprojekt von über den gesamten Erdball verteilten Freiwilligen, aber auch durch Beiträge aus der Industrie entstand und noch entsteht. Im Gegensatz zu den bislang erwähnten *proprietären* Betriebssystemen, deren Quellcode im Allgemeinen den Herstellern gehört und geheim ist, steht der Quellcode von Linux unter einer sogenannten *Open-Source*-Lizenz, er kann von jedem eingesehen und nach Gutdünken verändert werden. Linux wurde ursprünglich für die PC-Architektur auf Basis des 80386-Prozessors entwickelt, mittlerweile gibt es jedoch Portierungen auf viele andere Architekturen.

Es gibt sogar noch eine weitere quelloffene, UNIX-basierte Betriebssystemfamilie für den PC (und nicht nur ihn): das Trio *FreeBSD*, *OpenBSD* und *NetBSD*. Das Akronym BSD steht hierbei für *Berkeley Software Distribution*, ein Begriff aus der Historie von UNIX. In vielen Aspekten ähneln diese Betriebssysteme Linux. An dieser Stelle soll nicht näher auf sie eingegangen werden.

Mobile Rechner

Die PC-Architektur dominierte den Markt persönlicher Computer für lange Zeit. Dies änderte sich mit der Entwicklung mobiler Rechner, die mit dem Mobiltelefon ihren Anfang nahm. Betriebssystemtechnisch gab es dabei bemerkenswerte Dualitäten zur Entwicklung von »stationären« Computern.

Die erste Generation von Mobiltelefonen besaß wiederum kein Betriebssystem. Der eingebaute Mikroprozessor übernahm die Kommunikation mit dem Nutzer sowie die Codierung und Dekodierung der Sprachdaten. Die Software war in Form einer Firmware realisiert. Jeder Hersteller kochte hierbei sein eigenes Süppchen.

Bald kam man auf die Idee, wenn nun schon einmal eine CPU im Gerät verbaut war, diese auch für weitere »konventionelle« Datenverwaltungsaufgaben, wie die persönliche Termin- und Kontaktverwaltung, die Erfassung von Notizen und die Aufzeichnung und Wiedergabe von Audio- und Videoinformationen, zu nutzen: Das *Smartphone* entstand. Der resultierende deutlich höhere Entwicklungsaufwand für die Software rechtfertigte wiederum die Abstraktion eines Betriebssystems. Etwa ab 1997 erschienen in rascher Folge Geräte, die auf dem Betriebssystem *Symbian OS* und Prozessoren der ARM-Architektur basierten. Symbian OS besitzt eine wechselvolle Geschichte. Die mit Symbian ausgestatteten Geräte litten an verschiedenen »Kinderkrankheiten«, nicht zuletzt war die Nutzung mobiler Daten durch das Nichtvorhandensein entsprechender Dienste und Tarife sehr teuer.

Dies änderte sich mit der Einführung des Apple *iPhone* 2007, das auf dem Betriebssystem *iOS* basiert und seinerseits eine modifizierte UNIX-Implementierung, den sogenannte XNU-Kernel, nutzt. Kurze Zeit danach stellte Google mit *Android* eine zweite Betriebssystembasis für mobile Geräte vor. Android wiederum basiert auf dem bereits erwähnten Linux. Beide Betriebssysteme, iOS und Android, sind mittlerweile für weitere mobile Geräte, wie die etwa ab 2010 entwickelten Tablet-Computer, verfügbar.

Damit ist der kurze Rundgang über 60 Jahre Betriebssystem-Entwicklung zunächst komplett. Er ist stark subjektiv gefärbt und erhebt natürlich auch keinen Anspruch auf Vollständigkeit. Insbesondere haben wir den Bereich der eingebetteten Systeme bislang ausgeblendet.

Kurz gefasst stellt sich die gegenwärtige Situation folgendermaßen dar: Desktop-PCs oder Notebooks mit Intel- (oder kompatiblem) Prozessor werden abgesehen von einer Minderheit, die eine BSD-Variante oder Linux nutzen, mit einer Version von Windows betrieben. Rechner der Firma Apple bilden eine eigenen Kosmos: Mobile Geräte nutzen iOS, die Notebooks und Desktop-Geräte laufen unter OS X. Rechner, die Daten als Server im Inter- oder Intranet zur Verfügung stellen, nutzen sehr häufig Linux oder BSD, seltener Windows Server. Mobile Geräte, die nicht von Apple stammen, setzen fast ausschließlich Android (und damit Linux) ein.

Aufgaben eines Betriebssystems

Falls Sie den Abschnitt über die Historie der Betriebssysteme nicht übersprungen haben, wissen Sie bereits, dass ein Betriebssystem gewissermaßen als Mittler zwischen der Hardware und den Applikationsprogrammen fungiert. Im Folgenden soll diese recht allgemeine Aussage genauer ausgeführt werden. Dabei werden Sie erfahren, warum ein Betriebssystem für den Betrieb eines Rechensystems essenziell ist.

Bereitstellung von Abstraktionen und Diensten

Ein Computer benötigt aus Hardwaresicht mindestens zwei Komponenten: ein »Gehirn« in Form eines Mikroprozessors (*Central Processing Unit*, CPU), der seinerseits Befehle über ihm zur Verfügung gestellten Daten ausführt. Zur kurzfristigen Speicherung dieser Daten benötigt das System den Hauptspeicher (*Random Access Memory*, RAM). Will man die Daten längerfristig aufbewahren, dann wird ein Massenspeicher wie die Festplatte notwendig.

Darüber hinaus sind sogenannte Peripheriegeräte nötig, damit der Nutzer oder andere Computer mit dem Computer selbst in Verbindung treten können, also Bildschirm, Tastatur, Maus, Drucker, Netzwerkgeräte und so weiter.

Die Kommunikation mit und zwischen diesen Geräten ist sehr einfach und trotzdem sehr mühsam, denn alle Beteiligten verstehen nur zwei Zustände: 0 und 1 (davon aber ungeheure Massen). Alles, was in die CPU oder in die Festplatte hinein- oder aus ihr herauskommt, sind sehr lange Ströme von Nullen und Einsen. Ein JPEG-Bild, wie es Kameras zu Tausenden liefern, besteht aus einem endlichen, aber sehr langen Bitstrom aus Nullen und Einsen.

Damit wir Menschen mit Rechensystemen interagieren und kommunizieren können, benötigen wir *Abstraktionen*, die diese Bitströme in etwas für unseren Verstand Fassbares verwandeln. Das gerade angesprochene JPEG-Bild ist eine solche Abstraktion, allerdings eine anwendungsorientierte.

Stellen Sie sich vor, Sie wollen als Nutzer ein Bild vom Massen- in den Hauptspeicher transferieren, um es beispielsweise zu bearbeiten. Haben Sie kein Betriebssystem mit geeigneten Abstraktionen zur Verfügung, dann müssen Sie ungefähr das Folgende tun:

Sie müssen zunächst eine Liste mit einigen Hunderttausend (oder Millionen) Einträgen konsultieren, in der vermerkt ist, welche Blöcke der Festplatte das Bild enthalten. Danach müssten Sie eine Reihe sogenannter SATA-Kommandos an die Festplatte schicken, die diese anweisen, die entsprechenden Blöcke bereitzustellen. Danach müssten Sie in einer weiteren riesigen Liste nachsehen, ob es im Hauptspeicher einen Bereich geeigneter Größe gibt, und diesen für das Bild reservieren (also der Liste einen entsprechenden Eintrag hinzufügen). Schließlich, das ist sicherlich der einfachste Teil, müssten Sie die Daten aus dem Speicher der Festplatte in den Hauptspeicher kopieren. Umständlich, nicht wahr?

Interessanterweise sind die gerade angesprochenen SATA-Kommandos ebenfalls Abstraktionen, deren Funktionalität zu früheren Zeiten durch das Betriebssystem hätten erbracht werden müssen, die aber mittlerweile durch die Festplatte selbst bereitgestellt werden, die ihrerseits wiederum ein spezielles Rechensystem, (mit eigenem Prozessor, Speicher, Programm) konstituiert.

Ein Betriebssystem nimmt Ihnen den allergrößten Teil dieser Arbeit ab, indem es die Abstraktion »Datei« zur Verfügung stellt.

Es gibt natürlich viele weitere Abstraktionen, die Sie bei der Lektüre des Buches kennenlernen werden. Manche sind Ihnen intuitiv längst vertraut, wie die bereits angesprochene Datei oder das »Verzeichnis«, von manchen haben Sie vielleicht schon einmal gehört, etwa vom »Prozess« oder von einem »Thread«, und von manchen lesen Sie vielleicht zum ersten Mal: »Semaphore« oder »Spinlock«.

Natürlich nützen Ihnen Abstraktionen allein gar nichts. Sie müssen mit ihnen hantieren, arbeiten können. Dafür wiederum stellt ein Betriebssystem Dienste in Form von sogenannten *Systemrufen* zur Verfügung. Für die oben angesprochene Aufgabe, ein Bild in einen Hauptspeicherbereich zu laden, bietet Ihnen das Betriebssystem UNIX die Dienste `open()`, `read()` und `mmap()` an.

Koordinierung paralleler Abläufe

In Rechensystemen geschieht sehr viel gleichzeitig. Dies betrifft sowohl den Kontext der Applikationsprogramme als auch das Betriebssystem selbst. Beim Mehrnutzerbetrieb leuchtet dies unmittelbar ein. Ist die Aussage jedoch auch haltbar für ein Einprozessorsystem, das vielleicht gar keinen interaktiven Nutzerbetrieb kennt, sagen wir für eine mikroprozessorgesteuerte Waschmaschine?

Nun, es ist sicherlich ein Unterschied, ob das System 24 oder eine CPU besitzt und ob einige Dutzend oder gar kein einziger Nutzer mit dem System arbeiten. Nichtsdestotrotz gibt es

selbst bei den klitzekleinsten Rechnern Gleichzeitigkeit in Form von externen Ereignissen. Prozessoren besitzen sogenannte Interrupteingänge, die externe Geräte zur Signalisierung nutzen, indem sie einen kurzen elektrischen Impuls senden. Beispielsweise könnte es möglich sein, dass die Bedienelemente der Kaffeemaschine jeweils an einem Interrupteingang angeschlossen sind. Damit kann es wieder Gleichzeitigkeit geben: Die CPU führt Code aus, sie könnte zum Beispiel gerade eine Statusmeldung auf dem Display ausgeben. Der Nutzer drückt eine Taste, und nun muss die Kaffeemaschine mit dieser Form der Gleichzeitigkeit (Bedienhandlung und Meldungsausgabe) klarkommen. Soll sie zunächst die Ausgabe der Meldung fortsetzen oder lieber sofort auf die gedrückte Taste reagieren? Was ist, wenn der Nutzer zwei Tasten gleichzeitig drückt und somit zwei Interrupts generiert werden?

Zugegebenermaßen ist das Beispiel hinreichend simpel, und der Systemdesigner wird sicher für alle möglichen Formen der Parallelität eine für die Maschine und den Menschen akzeptable Lösung finden. Es soll aber nur die Aussage untersetzen, dass in Rechensystemen viele Dinge gleichzeitig geschehen und es im Allgemeinen Aufgabe des Betriebssystems ist, diese Gleichzeitigkeit zu koordinieren. In Kapitel 7 erlernen Sie dafür geeignete Techniken und Algorithmen.

Ressourcenverwaltung inklusive Protokollierung

Aus Sicht des Nutzers eines Rechensystems besteht dieses eigentlich aus Aktivitäten und Ressourcen. Sie werden im Kapitel 5 »Action! Aktivitäten, Prozesse und all das« genauer erfahren, was es mit diesen beiden Kategorien auf sich hat. An dieser Stelle soll es reichen, sich Aktivitäten als »Programme in Ausführung« vorzustellen. Sehr häufig findet man in der Literatur auch den Begriff *Prozess*.

Der Speicher, egal, ob temporär oder permanent, die CPUs, sämtliche Peripheriegeräte, jedoch auch viele sogenannte *logische Abstraktionen* (das sind Dinge im Rechner, die keine physische Repräsentation haben, die Sie also weder betrachten noch anfassen können), wie Dateien, Verzeichnisse oder ein (digital abgelegter) Film, werden als *Ressourcen* bezeichnet, die durch die Aktivitäten benutzt werden können. Damit sich die Aktivitäten nicht ins Gehege kommen, also sich gegenseitig Ressourcen streitig machen, verweigern oder sich ungerechtfertigte Vorteile bei der Ressourcenzuteilung verschaffen, sorgt das Betriebssystem für die Verwaltung der Ressourcen. Dazu gehört auch, die Nutzung aller Ressourcen ordentlich zu protokollieren, um Fairness zu gewährleisten oder Informationen für den Fall einer Überlastung zu sammeln.

Basis für Schutz und Sicherheit

Last but not least muss das Betriebssystem dem Nutzer Mechanismen zur Absicherung des Rechensystems bieten. Besondere Bedeutung kommt dabei dem sogenannten virtuellen Speicher zu, den Sie im Kapitel 9 gründlich kennenlernen werden. Ebenso zählen dazu die Verwirklichung eines Mehrnutzerkonzepts, die sichere Ablage von Passworthashes, ein sicherer Bootprozess und vieles andere mehr. Diese Funktionalität ist klassischer Bestandteil des Betriebssystems, denn es hat sich gezeigt, dass die Implementation von Sicherheitsmechanismen in der Applikationsschicht ohne solide Basis im Betriebssystem keinen zuverlässigen Schutz bieten kann.

Fassen wir zusammen: Betriebssysteme haben die folgenden vier Aufgaben:

- ✔ Verwirklichung von Abstraktionen und Diensten für die Arbeit mit dem System
- ✔ Koordination aller parallelen Abläufe
- ✔ Verwaltung aller Ressourcen eines Systems
- ✔ Realisierung einer Basis für die Systemsicherheit

Perspektiven auf Betriebssysteme

Um den Fokus auf Betriebssysteme noch ein bisschen zu weiten, sollen im Folgenden kurz einige Perspektiven auf Betriebssysteme skizziert werden. Wer interessiert sich für Betriebssysteme und aus welchem Grund?

Nun, beginnen wir diesen Rundgang mit dem gewöhnlichen *Anwender* oder *Nutzer*. Jeder, der mit Rechensystemen arbeitet, kommt auf die ein oder andere Weise mit dem Betriebssystem in Berührung, sei es durch die normale Interaktion, egal, ob text- oder grafikorientiert (oder beides!), sei es bei der Installation des Systems. Entscheidend für den Nutzer ist daher häufig die Benutzeroberfläche (*User Interface, UI*), während die »inneren Werte« zunächst weniger von Interesse sind, solange alles einigermaßen funktioniert.

Während der private Nutzer meist das System selbst aufspielt, konfiguriert und wartet, erfolgt diese Tätigkeit im professionellen Umfeld meist über *Administratoren*, also Spezialisten, die für den Betrieb größerer IT-Infrastrukturen ausgebildet und angestellt sind. Ihre Perspektive unterscheidet sich von der des gewöhnlichen Nutzers; im Vordergrund stehen Aspekte wie effiziente Interaktionsmöglichkeiten, die Stabilität der Systeme, die Automatisierbarkeit von sich wiederholenden Bedienoperationen oder die Verfügbarkeit von Werkzeugen zur Überwachung und Wartung größerer Rechnerbestände.

Bei den *Softwareentwicklern* kann man grob zwei Kategorien unterscheiden. Entwickeln sie Dienstprogramme (*Applikationen*, im Jargon mobiler Systeme meist *Apps* genannt) für ein bestimmtes Betriebssystem, dann müssen sie Kenntnisse über dessen Dienste besitzen und darüber, wie eine bestimmte Funktionalität mit den Diensten eines (oder mehrerer) Betriebssysteme effizient erbracht werden kann. Entsteht eine Komponente *für* ein Betriebssystem, beispielsweise ein *Treiber* (eine Komponente, die es dem Betriebssystem ermöglicht, mit einem bestimmten Hardwaregerät zu kommunizieren), dann sind zusätzliche umfangreiche Kenntnisse über die Interna des jeweiligen Systems notwendig. Diese sind oftmals sehr komplex!

Ein *Angreifer*, landläufig auch gern fälschlich als »Hacker« bezeichnet, wiederum benötigt diese exzellenten Kenntnisse über Interna von Betriebssystemen ebenso. Ihn interessiert meist, welche Komponenten eines Betriebssystems Schwachstellen aufweisen, um diese auszunutzen und sich beispielsweise unberechtigt Zugang zum System zu verschaffen. Jedoch ist nicht jeder Angreifer potenziell ein Feind: Sogenannte *Penetration Tester* tun genau das Gleiche, bieten es jedoch als Dienstleistung an, um die Widerstandsfähigkeit der Systeme ihrer Kunden gegen Angriffe zu erhöhen.

IT-Forensiker wiederum sind bestrebt, die nach einem erfolgreichen Angriff vorhandenen Spuren innerhalb des Systems, ob im Haupt- oder auf dem Massenspeicher, auszuwerten, und dem Angreifer auf die Schliche zu kommen, die Systeme zu säubern und mögliche Verletzungen der Datenintegrität zu beheben. Dies tun sie auch bei der Suche nach Beweismitteln auf den Rechnern Tatverdächtiger. Daher kennen sie sich besonders mit den Eigenarten der zahlreichen Dateisysteme und den Möglichkeiten der Wiederherstellung absichtlich und versehentlich gelöschter Daten aus, was im Übrigen sogenannte *Datenretter* ebenfalls als Dienstleistung anbieten.

Der Blick von *Betriebssystem-Wissenschaftlern* ist wiederum verschieden zu den bislang aufgeführten Kategorien. Diese versuchen, ganz unterschiedliche Aspekte von Betriebssystemen zu optimieren. So gibt es Informatiker, die über bessere Nutzerschnittstellen nachdenken, genauso wie andere versuchen, neue Abstraktionen zu finden oder die Leistung oder die Sicherheit bestehender Systeme und zugehöriger Werkzeuge zu verbessern. Sie stehen häufig in direkter Konkurrenz zu den bereits erwähnten Angreifern.

Nicht vergessen werden sollten *Computerarchäologen*. Ihr Interesse gilt insbesondere der Analyse und allgemein dem Betrieb historischer Rechensysteme, die natürlich historische Betriebssysteme benutzen. Zwei Anwendungsbeispiele sind die Restauration alter Datenbestände und das Zugänglichmachen von historischer Software, die als erhaltenswertes Kulturgut angesehen wird (Computerspiele!). Sie bedienen sich dabei Techniken der Virtualisierung und der Emulation, da die Verfügbarkeit der originalen Rechensysteme durch Verschleiß und Ausfall von Hardwarekomponenten kontinuierlich abnimmt.

Zu guter Letzt soll noch die spezifische Perspektive des *Lehrers* erwähnt sein. Sein Fokus (und auch der dieses Buches) liegt auf der Vermittlung von Prinzipien, Verfahren und Algorithmen, die bei Entwurf, Konstruktion, Analyse und Optimierung von Betriebssystemen angewandt werden. Dazu benötigt er (Lehr-)Betriebssysteme, die die enorme Komplexität »richtiger« Betriebssysteme vereinfachen und die insbesondere einen niedrigschwelligen Zugang zu den »Innereien« des Systems erlauben. Das Betriebssystem MINIX [25] des Informatikers Andrew S. Tanenbaum ist ein Beispiel dafür. Sie werden diesem Namen im Laufe der Lektüre noch einige Male begegnen.

Klassifizierung von Betriebssystemen

Nachdem Sie verschiedene Sichtweisen auf Betriebssysteme kennengelernt haben, ist es nun an der Zeit, einmal einen Blick auf die Fülle an existierenden Betriebssystemen zu werfen. Eine einfache Auflistung, die keinen Anspruch auf Vollständigkeit erhebt, finden Sie beispielsweise in der Wikipedia unter dem URL `https://de.wikipedia.org/wiki/Liste_von_Betriebssystemen`.

Um einen besseren Überblick über die schier unübersehbare Menge von Betrachtungsgegenständen (hier: Betriebssystemen) zu erhalten, ist es nützlich, diese zu *klassifizieren*, also nach bestimmten Kriterien in Kategorien einzuteilen. Dies soll im folgenden Abschnitt geschehen.

Klassifizierung nach Einsatzzweck

Ein unmittelbar einleuchtendes Kriterium ist die Frage, welches Problem oder welche Problemklasse mittels des betrachteten Rechensystems gelöst werden soll. In erster Näherung kann man hier *Universal-* von *Spezialbetriebssystemen* unterscheiden. Universalbetriebssysteme sind, wie der Name nahelegt, für eine große Menge verschiedener Problemkreise oder Rechensysteme geeignet, während Spezialbetriebssysteme üblicherweise sehr genau für einen einzigen Einsatzzweck optimiert wurden.

Etwas klarer wird die Klassifizierung, wenn konkretere Kategorien genutzt werden.

Desktop-Betriebssysteme, also Betriebssysteme, die auf Rechensystemen für die Büroarbeit laufen, sind beispielsweise viele Windows-Varianten, MacOS für Systeme der Firma Apple sowie Linux und die historischen Vertreter MS-DOS und CP/M.

Typische Betriebssysteme, die für *Server* jeglicher Art genutzt werden, sind Linux, die BSD-Varianten, viele weitere UNIX-Derivate sowie Windows Server.

Eine besonders große und heterogene Kategorie sind sogenannte *eingebettete Systeme*, also Rechner, die Teil eines größeren Gesamtsystems und als typische Computer häufig gar nicht zu erkennen sind. Darunter fallen industrielle Steuerungen, Rechner in Autos, Flugzeugen und Schiffen (der sogenannte *Automotive*-Bereich), aber auch *Consumer Electronics*, wie Fernseher, Haushaltgeräte und Ähnliches. Exemplarisch sollen die Betriebssysteme Linux, FreeRTOS, OSEK und VxWorks genannt sein; es gibt sehr viele weitere Vertreter.

Eng mit dieser Kategorie verwandt sind die sogenannten *Echtzeitbetriebssysteme*. Diese Systeme zeichnen sich dadurch aus, dass sie die Dauer bestimmter Operationen, beispielsweise Reaktionszeiten auf externe Ereignisse, garantieren können. Dies erfordert im Vergleich zu universellen Vertretern des Genres umfangreiche Modifikationen. Die bereits erwähnten FreeRTOS, OSEK und VxWorks sind ebenso Vertreter dieser Kategorie.

Wie bereits im historischen Abriss erwähnt wurde, bildeten und bilden die Mainframe-Computer eine eigene Welt, was die hier genutzten Betriebssysteme unterstreichen. Systeme von IBM nutzten das bereits erwähnte OS/360, später OS/390 und nun z/OS; jedoch begegnet uns erneut das bereits häufig referenzierte Linux. Andere Hersteller wie Unisys oder Fujitsu nutzen ebenfalls eigene Betriebssysteme.

Man kann also also resümieren, dass es auf der einen Seite Betriebssysteme gibt, die für spezielle Einsatzzwecke konzipiert sind. Andererseits gibt es universell einsetzbare Betriebssysteme, die man in mehreren Systemkategorien finden kann.

Andere Klassifikationskriterien

Es gibt viele weitere Kriterien, nach denen sich Betriebssysteme einteilen lassen. Dazu zählen:

- ✔ **Unterstützte Hardwareplattform.** Manche Betriebssysteme sind für verhältnismäßig viele verschiedene Prozessorarchitekturen verfügbar. Meist wurde das Betriebssystem zunächst für eine einzige Plattform entwickelt und später auf andere übertragen (*portiert*). Als Beispiel für diese Art Betriebssystem kann wiederum Linux dienen.

Andere Betriebssysteme sind nur für eine einzige Architektur verfügbar. MS-DOS ist beispielsweise nur für die Intel-Prozessorarchitektur verfügbar, allerdings wiederum für mehrere konkrete Prozessortypen ab dem Intel 8086.

- ✔ **Möglichkeit, Nutzer zu unterscheiden.** Gibt es die Abstraktion *Nutzer* (*User*), so können verschiedene Benutzer mit dem System arbeiten. Dies muss nicht notwendigerweise gleichzeitig erfolgen. Sind mehrere Nutzer möglich, dann spricht man vom *Mehrnutzerbetriebssystem* (*Multi-User Operating System*) andernfalls vom *Einnutzerbetriebssystem* (*Single-User Operating System*). Das Betriebssystem Android beispielsweise wurde ab Version 4.2 mit der Mehrnutzerfähigkeit ausgestattet, bis dahin konnte es pro Gerät nur einen einzigen Nutzer geben. MS-DOS ist klassisch nur für einen einzigen Nutzer, alle modernen Desktop- und Server-Betriebssysteme sind für mehrere Nutzer ausgelegt.

- ✔ **Anzahl gleichzeitig möglicher unabhängiger Aktivitäten.** Betriebssysteme, die nur eine einzige Aktivität zu einem Zeitpunkt abarbeiten können, werden *Singletasking-Betriebssysteme* genannt (hier gibt es keine adäquate deutsche Bezeichnung). Kann der Nutzer mehrere Aktivitäten gleichzeitig etablieren (also beispielsweise die Textverarbeitung, ein Spiel und den Videoplayer hintereinander starten), ohne die anderen Aktivitäten jeweils zuvor beenden zu müssen, dann spricht man von einem *Multitasking-Betriebssystem*.

  Bitte beachten Sie, dass ein System, das nur einen einzigen Prozessor oder Core aufweist, prinzipiell zu einem Zeitpunkt nur eine Aktivität ausführen kann, da nur ein einziger Registersatz zur Verfügung steht. Ein Multitasking-Betriebssystem erlaubt es trotzdem, mehrere Aktivitäten zu etablieren; die Illusion der Gleichzeitigkeit kommt durch die enorme Arbeitsgeschwindigkeit und das schnelle Umschalten zwischen den Aktivitäten zustande. Diesen Vorgang werden Sie im Kapitel 6 »Planen von Aktivitäten (Scheduling)« näher kennenlernen. Multi-User-Systeme bedingen nicht unbedingt Multitasking, denn es kommt darauf an, ob mehrere Nutzer hintereinander oder gleichzeitig arbeiten können.

- ✔ **Kommunikation mit der Umwelt.** Systeme, mit denen die Nutzer in einen Dialog treten können, werden *interaktive Systeme* genannt. Es gibt jedoch auch Systeme, bei denen diese Arbeitsweise nicht im Vordergrund steht, beispielsweise Clusterrechner, die aus vielen Tausenden Einzelrechnern bestehen. Die Anschaffung und der Betrieb solcher Systeme sind extrem teuer, sodass man es sich nicht leisten kann, wertvolle Rechenzeit durch (aus Sicht des Rechners) extrem langsame interaktive Kommunikation mit dem Nutzer zu verschwenden.

Typische interaktive Systeme, wie das Notebook, auf dem der Autor dieses Buch schreibt, tun genau das: Sie warten fast ihre gesamte Betriebszeit auf die Eingaben des Nutzers. Sehr schnelle Schreiber schaffen ungefähr 500 Anschläge pro Minute. Das benutzte Notebook besitzt eine Taktfrequenz von 3.6 GHz und verfügt über vier Rechenkerne. In einer Minute könnte es ungefähr $4 \cdot 60 \cdot 3.6 \cdot 10^9 \approx 864$ Milliarden Instruktionen ausführen. Um ein paar Tastenanschläge (es sind deutlich weniger als 500) entgegenzunehmen, die Symbole auf dem Bildschirm darzustellen und den Cursor neu zu positionieren, benötigt es vielleicht eine oder zwei Millionen Instruktionen. Da der Rechner sonst nichts zu tun hat, »verwartet« er die Zeit der restlichen Instruktionen.

Teure Parallelrechner besitzen häufig extra Eingaberechner, die die einzelnen Aufgaben (»Jobs«), die das System ausführen soll, in eine Warteschlange einordnen. Sobald ein Job komplettiert wurde, übernimmt ein weiterer Rechner die Ergebnisse und legt diese auf einem Massenspeicher ab, während der Eingaberechner den nächsten Job aus der Warteschlange entnimmt und auf dem Parallelrechner startet. Ein solches System wird *Jobsystem* oder *Stapelsystem* genannt. Insbesondere historische lochkartenbasierte Systeme benutzten diese Form der Ein- und Ausgabe.

Eine dritte Ausprägung dieser Kategorie sind Betriebssysteme für *autonome* Systeme. Mit diesen ist ebenfalls kein direkter Dialogbetrieb vorgesehen, sondern diese arbeiten weitestgehend selbstständig, also autonom. Klassische eingebettete Systeme im Automotive- oder Raumfahrt-Bereich sind dafür Beispiele.

- ✔ **Verteilung.** Ein letztes Kriterium unterscheidt, ob das Betriebssystem für einen einzelnen Rechner konzipiert ist oder ob eine Menge an Rechnern, die über ein Netzwerk verbunden sind, gemeinsam an Aufgaben arbeiten. Im ersten Fall spricht man von einem lokalen (Betriebs-)System, im letzteren Fall von einem *verteilten Betriebssystem*. Dieses abstrahiert von den vielen individuellen Rechnern, die zusammen den Parallel- oder Clusterrechner konstituieren. Ein verteiltes Betriebssystem gestattet es also, ein aus vielen Einzelrechnern bestehendes System als homogene Einheit wahrzunehmen.

  Sie werden in diesem Buch ausschließlich Aspekte lokaler Betriebssysteme kennenlernen, diese sind schließlich kompliziert genug! Das Teilgebiet der verteilten Algorithmen und Betriebssysteme würde ein weiteres Buch füllen, das aber infolge der Komplexität dieser Systeme den Rahmen »…für Dummies« sprengen würde.

Damit soll es der Klassifiziererei und Kategorisierei genug sein. Sie besitzen nun einen groben Überblick über verschiedene Betriebssysteme und können diese in verschiedene »Schubladen« einsortieren. Bitte achten Sie darauf, dass Betriebssysteme einer kontinuierlichen Weiterentwicklung unterliegen und somit ab und zu von einer Schublade in eine andere schlüpfen können. Beispielsweise wurde aus dem ursprünglich für Server- und Desktop-Systeme konzipierten Linux zusätzlich eines der wichtigsten Betriebssysteme für eingebettete Systeme.

Lizenzierungsaspekte

Als Nächstes ist es notwendig, einige juristische Aspekte zu klären. Es sind aber die einzigen im Buch, versprochen!

Betriebssysteme sind Software, und Software unterliegt wie alle geistigen Werke dem Urheberrecht. Wenn Sie Betriebssysteme einsetzen wollen, dann müssen Sie zunächst klären, ob und unter welchen Bedingungen Sie dies dürfen. Was Sie mit dem Betriebssystem dürfen und was nicht, ist in aller Regel in der dazugehörigen Lizenz niedergeschrieben.

Offen oder geschlossen?

Wenn man Software entwickelt, so kann man dies eigentlich auf zwei verschiedene Arten tun. Zum einen kann man Software entwickeln und ausschließlich den entstehenden Programmcode, also das ausführbare Programm, kommerziell vertreiben, damit Gewinn erwirtschaften, der wiederum in die Weiterentwicklung der Software (oder in neue Software) investiert werden kann. Dies ist die traditionelle Vorgehensweise von Softwarefirmen. Für den Kaufpreis erhält der Kunde ein begrenztes Nutzungsrecht an der Software in Form einer sogenannten *proprietären Lizenz*, die meist recht restriktiv ausfällt und die Analyse, die Modifikation und jegliche Weitergabe der Software verbietet. Der zugrunde liegende Quellcode bleibt Eigentum und Geheimnis der Entwickler, denn aus dessen Kenntnis könnte ein Konkurrent vielleicht eine ähnliche oder identische Software herstellen und diese wiederum billiger verkaufen. Er könnte auf diese Weise ja den gesamten Entwicklungsaufwand sparen und den Quellcode einfach kopieren. Das Prinzip wird daher häufig auch *Closed Source* genannt. Die Firma Microsoft vertreibt ihre Betriebssysteme seit jeher ziemlich erfolgreich auf diese Weise.

Es gibt jedoch noch eine zweite Herangehensweise. Viele Entwickler sehen den Code nicht in erster Linie als unbedingt zu schützendes Betriebsgeheimnis, sondern als Einladung an andere Entwickler, diesen herunterzuladen, zu analysieren, potenzielle Fehler aufzuspüren und so bei der Verbesserung der Software mitzuarbeiten. In Abgrenzung zur Closed Source wird dieses Modell *Open Source* genannt. Hersteller oder Entwickler stellen zur Software den Quellcode zur Verfügung, damit die Nutzer diesen bei Bedarf einsehen können.

Diese »Mitwirkung« des Nutzers kann sogar noch weitergehen. Häufig (aber nicht immer) ist er ausdrücklich eingeladen, die Software selbst zu verbessern, zu modifizieren und natürlich dann auch weiterzuverteilen. Diese Form der Mitwirkung bietet dem Nutzer ein Höchstmaß an Freiheit im Umgang mit der Software, daher wird diese Form gern *freie Software* genannt. Die ersten Varianten von UNIX entstanden auf diese Weise und (viel) später auch Linux. Open Source ist also streng genommen ein Teilaspekt freier Software; trotzdem werden beide Kategorien häufig (fälschlich) synonym gebraucht.

Es bleibt die Frage zu klären, wie Open-Source- und Freie-Software-Entwickler ihre Brötchen verdienen; denn wenn es jedem erlaubt ist, die Software zu kopieren, ergibt es kaum Sinn, diese für teures Geld anzubieten. Nun, zum einen sind rund um Open-Source-Software vielerlei Dienstleistungen zu erbringen; Systeme müssen häufig adaptiert, gewartet und weiterentwickelt werden. Zum anderen beschäftigen viele kommerzielle Hard- und Softwarefirmen Entwickler, die Open-Source-Software entwickeln. Die Entwicklung des Linux-Kernels beispielsweise liegt in der Hand vieler Programmierer, die von großen Unternehmen wie IBM oder Amazon oder AMD beschäftigt werden.

Bei der Erörterung der Vor- und Nachteile freier respektive proprietärer Software ist im Allgemeinen Vorsicht geboten. Entsprechende Diskussionen werden häufig sehr leidenschaftlich geführt und enden nicht selten unsachlich mit Beschimpfungen oder Nazi-Vergleichen.

GNU General Public License (GPL)

Zum Abschluss dieses Abschnittes werfen wir einen Blick auf die mit Abstand populärste Lizenz für freie Software, die sogenannte *GNU General Public License* (GPL). Der Linux-Kern und sehr viele der ein komplettes Linux-System umfassenden Werkzeuge, wie Editoren, Compiler, Linker und Debugger, unterliegen dieser Lizenz.

Richard M. Stallman entwarf sie 1989, um den Gedanken der freien Software zu popularisieren. Ein wichtiges Prinzip ist das sogenannte *Copyleft* (ein Wortspiel mit dem Begriff »Copyright« – Urheberrecht), das vorschreibt, dass Kopien und Modifikationen der unter GPL lizenzierten Software wiederum unter der GPL stehen müssen. Damit wird verhindert, dass ein geschäftstüchtiger Programmierer die Software kopiert, leicht modifiziert und dann unter eine proprietäre Lizenz stellt. Einmal als frei deklarierte Software bleibt damit frei.

In Kurzform enthält die GPL die folgenden Regeln:

- ✔ Die Software darf für beliebige Zwecke verwendet werden.
- ✔ Die Software darf beliebig modifiziert werden.
- ✔ Die Software darf in beliebiger Form weitergegeben werden, kostenlos oder kostenpflichtig. Der Quelltext (auch eigener Modifikationen) ist stets mitzuliefern.
- ✔ Diese Regeln sind unabänderlich.

Das oben erwähnte *Copyleft* weist auch Nachteile auf. So ist es beispielsweise unmöglich, Code, der unter GPL steht, mit kommerziellem Code zu vermischen. Dies wird häufig als Hemmnis für den Einsatz freier Software im Unternehmenskontext aufgefasst, was wiederum zur Schaffung weiterer, weniger restriktiver Lizenzen wie der GNU Lesser General Public License (LGPL) und der BSD-Lizenz führte.

GNU

Falls Sie sich fragen, was es mit dem »GNU« im Titel der Lizenz auf sich hat: GNU ist der Name des Projektes, das sich die Schaffung eines komplett freien Betriebssystems, das ebenfalls »GNU« heißt, zum Ziel gesetzt hat. Der bereits erwähnte Richard M. Stallman gründete es 1984. Gleichzeitig ist GNU ein sogenanntes rekursives Akronym, es steht für »GNU's not UNIX.«. Rekursiv bedeutet, dass sich die Definition selbst aufruft; es handelt sich um einen typischen Informatikerwitz.

Sehr viele wichtige Basiswerkzeuge wurden durch GNU entwickelt, so die *GNU Compiler Collection* (GCC), der *GNU Debugger (GDB)* und die *GNU Toolchain*. Allerdings ist es GNU bislang nicht gelungen, einen funktionsfähigen Betriebssystemkern zu entwickeln, weshalb man einfach den Linux-Kernel nutzte und das resultierende System »GNU/Linux« taufte.

Lesevorschläge

Wenn Sie mehr über die Geschichte der PC-Rechentechnik und speziell über die Historie von Betriebssystemen erfahren möchten, sollten Sie die folgenden Vorschläge in Betracht ziehen:

- ✔ Gerard O'Regan: *A Brief History of Computing*. Springer Verlag, 2012
- ✔ Fred P. Brooks: *The Mythical Man-Month*. Addison-Wesley, 1995
- ✔ Brian Bagnall: *Volkscomputer*. Aufstieg und Fall des Computer-Pioniers Commodore und die Geburt der PC-Industrie, Gameplan, 2011
- ✔ Stephen Manes und Paul Andrews: *Gates. Wie der Microsoft-Chef die PC-Industrie revolutionierte und zum reichsten Mann Amerikas wurde*. Addison-Wesley, 1993.
- ✔ Brian Kernighan: *UNIX. A History and a Memoir*. Kindle Direct Publishing, 2020
- ✔ Linus Torvalds, David Diamond: *Just for Fun: Wie ein Freak die Computerwelt revolutionierte*. dtv, 2002

Übungsaufgaben

1. Ordnen Sie Windows 3.1, MS-DOS, FreeRTOS, Android 12, iOS und ein in diesem Buch bislang nicht erwähntes (denken Sie an den Wikipedia-Artikel) Betriebssystem den einzelnen Kategorien zu!
2. Finden Sie heraus, worin sich die Open-Source-Lizenzen MIT, BSD und GPL unterscheiden.