1 Programmieren unter Linux

In diesem ersten Kapitel werden Sie entdecken, was das Besondere an Linux ist, welchen Einfluss sein Vorbild Unix ausübt und wie der Gedanke von freier Software zu einer Massenbewegung geworden ist. Genauer werden wir dabei folgende Punkte untersuchen:

- Was ist eigentlich Unix?
- Ist Linux ein Unix? (Seite 10)
- Welche Ziele hat das GNU-Projekt? (Seite 12)
- Wie sieht die Programmentwicklung unter Unix aus? (Seite 15)

1.1 Das Unix-Betriebssystem

Für die meisten Leute ist heutzutage der erste Computer, mit dem sie in Kontakt kommen, ein PC, auf dem sich in aller Regel ein vorinstalliertes Windows-Betriebssystem von Microsoft befindet. Diese Geräte sind mittlerweile so weit verbreitet, dass sich manche schon gar nicht mehr vorstellen können, dass es auch noch andere Möglichkeiten gibt, einen Computer zu betreiben.

1.1.1 Die Unix-Familie

Im wissenschaftlich-technischen Bereich, wo immer schon größere Rechenleistung gefragt war, die der PC bis vor kurzem nicht erbringen konnte, setzt man seit langem so genannte *Workstations* ein. Auf diesen mit speziellen Prozessoren ausgestatteten Rechnern laufen Betriebssysteme, die es erlauben, dass mehrere Benutzer gleichzeitig darauf arbeiten und mehrere Programme nebeneinander laufen können. Die meisten dieser Betriebssysteme gehören dabei zur *Unix-Familie*.

Ursprung bei AT&T

Der Ausgangspunkt war eine Entwicklung beim US-Telefonkonzern AT&T in den siebziger Jahren. Die erste Version von Unix war noch in Assembler geschrieben; es wurde aber schon bald in die eigens dafür entworfene Hochsprache C portiert. In der Folgezeit übernahmen verschiedene

Hersteller das Unix-Konzept und entwickelten es für ihre eigene Hardware weiter [DAEMISCH 2001]. Heute sind die Unix-Varianten zwar in ihren Grundkonzepten kompatibel, so dass Programme von einem so genannten Derivat zum anderen relativ leicht zu übertragen sind, sie unterscheiden sich jedoch sowohl in vielerlei tieferen Details als auch (wegen meist eigener Window-Manager) im Erscheinungsbild für den Benutzer.

Das Warenzeichen UNIX

Genau genommen ist UNIX heute ein eingetragenes Warenzeichen der Open Group (einem Industriekonsortium, dem alle namhaften Firmen angehören, die in diesem Bereich tätig sind). Ein Betriebssystem darf sich demnach »UNIX« nennen, wenn es eine von dieser Organisation genau festgelegte Spezifikation erfüllt. Da deren Verifikation jedoch ein sehr teurer Prozess ist, gibt es nicht besonders viele Systeme, die dieses Kriterium erfüllen — nicht einmal alle kommerziellen. Im allgemeinen Sprachgebrauch hat sich daher eingebürgert, ein Betriebssystem »Unix« zu nennen, wenn es auf der ursprünglichen AT&T-Entwicklung fußt.

Es gibt neben einer Reihe kommerzieller Varianten wie Solaris (Sun Microsystems), AIX (IBM), HP-UX (Hewlett Packard) oder Irix (Silicon Graphics) auch einige frei erhältliche wie FreeBSD, NetBSD oder eben Linux.

1.1.2 Besondere Eigenschaften von Unix

Unix verfügt über einige Eigenschaften, die es auf PCs mit Microsoft-Betriebssystemen überhaupt nicht, nur in vereinfachter Form oder erst seit kurzem gibt. Der Umgang mit Unix erfordert daher sowohl vom Benutzer als auch vom Programmentwickler eine etwas andere Arbeitsweise als mit Windows. Da ich davon ausgehe, dass Sie bereits Linux auf Ihrem Rechner installiert haben, werden Sie vermutlich auch schon ein Gefühl dafür bekommen haben, was ich mit »andere Arbeitsweise« meine.

Multitasking

Zunächst arbeitet Unix mit so genanntem *Multitasking*. Kein Programm kann dabei normalerweise alle Systemressourcen wie Speicherplatz, Rechenleistung oder Netzwerkdurchsatz für sich alleine in Anspruch nehmen. Es laufen stets eine ganze Reihe von Programmen im Hintergrund, denen das System abwechselnd, aber gleichmäßig Zugriff auf die Ressourcen erlaubt. Wenn Sie in einer Shell etwa den Befehl `ps aux` eingeben, werden Sie weit über zwanzig so genannte *Prozesse* sehen, die auf Ihrem Rechner momentan (fast) gleichzeitig ablaufen. Jeder Prozess läuft dabei in einem eigenen Bereich und beeinflusst die anderen kaum.

Multiuser

Ein weiteres wesentliches Merkmal von Unix ist die Fähigkeit, dass mehrere Benutzer gleichzeitig auf dem Computer arbeiten. Man bezeichnet dies auch als *Multiuser-Konzept*. (Natürlich darf man sich das nicht so vorstellen, dass vier Leute mit vier Tastaturen vor einem Bildschirm sitzen und jeder in irgendeiner Ecke etwas eingibt; mit »gleichzeitiger Benutzung« ist

vielmehr der Zugriff über ein Rechnernetz gemeint.) Diese Fähigkeit hat eine Reihe von Konsequenzen:

- *Jede Datei gehört einem Benutzer*: Es muss sichergestellt sein, dass ein Benutzer nicht die Dateien eines anderen überschreiben oder löschen kann. (Aus Datenschutzgründen sollte sogar das Lesen verhindert werden können.) Diese Anforderungen erfüllt das Unix-Dateisystem, indem es zu jeder Datei den Namen des Benutzers speichert, der sie angelegt hat. Da zusätzlich Benutzer zu Gruppen zusammengefasst sind, kann man außerdem für jede Datei angeben, ob nur der Benutzer oder seine Gruppe oder jedermann sie lesen, überschreiben beziehungsweise ausführen darf.
- *Auch Prozesse sind an Benutzer gebunden:* Nicht nur die Dateien, sondern auch jeder einzelne Prozess (beispielsweise jedes laufende Programm) ist eindeutig einem Benutzer zugeordnet, so dass ein Saboteur nicht so einfach Prozesse anderer Benutzer beeinflussen oder beenden kann.
- *Konfigurationsinformationen stets benutzerspezifisch:* Anwendungen dürfen nicht, wie etwa unter Windows zum Teil üblich, ihre Konfigurationsinformationen an einem zentralen Platz abspeichern, denn jeder Benutzer könnte ja unterschiedliche Konfigurationen wünschen. Jeder Anwender verfügt daher über einen eigenen Datenbereich, das so genannte Home-Verzeichnis. Applikationen hinterlegen ihre Einstellungen unter Unix somit im Allgemeinen in diesem Verzeichnis (oder sie legen sich dort ein Unterverzeichnis an).

Sie sollten diese Eigenschaften von Unix nicht nur im Hinterkopf behalten, weil wir zusammen gleich damit arbeiten wollen. Später wollen Sie ja auch selbst Programme unter Unix entwickeln, die natürlich auch diesen Randbedingungen gehorchen müssen. Beispielsweise sollten Sie stets vermeiden, dass Ihre Programme durch Leerlaufaktivitäten (etwa beim Warten auf Eingaben) mehr Ressourcen für sich in Anspruch nehmen, als eigentlich notwendig wären.

1.1.3 Die Werkzeug-Philosophie

Tradition der kleinen Hilfsprogramme

Wie Sie vielleicht auch schon gemerkt haben, arbeitet man unter Unix traditionell sehr viel intensiver mit der Kommandozeile (der Shell) als unter DOS oder gar Windows. Daher gibt es unter Unix auch eine große Zahl kleiner Hilfsprogramme, die eine eng begrenzte Aufgabe effektiv erledigen. Dazu gehören Dateibetrachter wie *more*, Zeileneditoren wie *sed* und *awk*, Suchprogramme wie *find* und *grep* sowie Hilfesysteme wie *man*

und *info*. (Diese Liste ließe sich noch einige Zeit fortsetzen...) Jedes Werkzeug soll nur genau eine Aufgabe übernehmen, diese dann aber so gut wie möglich beherrschen.

Für den Neuling ist die Vielfalt dieser Utilities oft verwirrend, hat doch jedes seine eigenen Parameter, eventuell sogar einige Nebeneffekte. Geübte Unix-Anwender schwören indessen auf ihre Werkzeuge, da sie damit nach eigenen Angaben sehr viel effektiver arbeiten können. Ich nutze zwar auch hin und wieder einige dieser Werkzeuge, bin aber der Ansicht, dass Linux auch für alle verwendbar und nützlich sein sollte, die nicht die Zeit (oder die Lust) haben, sich ein Jahr mit der Shell und allen Tools vertraut zu machen. In diesem Buch werden Sie daher nur ein paar wenige, aber essenzielle Werkzeuge beschrieben finden; ansonsten will ich versuchen, Sie an verschiedene integrierte Lösungen mit grafischen Benutzeroberflächen heranzuführen, die es mittlerweile in respektabler Form auch unter Linux gibt. Wenn diese integrierten Umgebungen Teile ihrer Funktionalität aus dem Aufruf der »kleinen« Werkzeuge beziehen, soll uns das dann auch nicht weiter stören.

1.2 Linux

In welchem Verhältnis steht nun Linux zu Unix? Dazu muss man den Begriff »Linux« erst einmal etwas genauer erklären.

1.2.1 Der Kernel

Geschichte von Linux

Ursprünglich bezeichnet Linux nur den Betriebssystemkern (Englisch *kernel*), der 1989 vom mittlerweile berühmten Finnen Linus Torvalds und vielen anderen entwickelt wurde und seitdem fortlaufend verbessert wird. Dieser Kernel war am Anfang inspiriert von einem Lehrbetriebssystem namens Minix (das seinerseits die Funktionalität von Unix Version 7 bot), wurde aber auf vollkommen eigener Codebasis erstellt und verwendet weder Programmzeilen der AT&T-Entwicklung noch von Minix. Durch den intensiven Austausch der Ideen und Beiträge Hunderter Entwickler über das Internet wurde der Linux-Kernel zu dem eigenständigen, modernen und leistungsfähigen Unix, mit dem Sie heute arbeiten. Eine Verifizierung im Sinne des Unix-Warenzeichens von X/Open ist damit natürlich nicht verbunden, da die Linux-Gemeinde sicher nicht die Gebühren aufbringen könnte (und wollte!) und auch kein kommerzielles Interesse hinter Linux steht.

Linux-Systeme heute

Wenn heute jemand davon spricht, er habe Linux auf seinem Computer installiert, so meint er damit im Allgemeinen wesentlich mehr. Ein Linux-System enthält neben dem Kernel noch eine große Zahl verschiedener Hilfs- und Anwendungsprogramme, die ebenfalls kostenlos erhältlich sind und die zumeist aus dem GNU-Projekt der Free Software Foundation stammen (siehe Abschnitt 1.3, Seite 12).

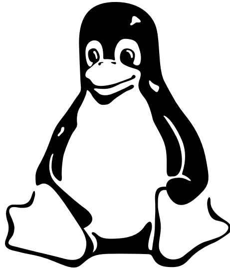

Abbildung 1.1
Das Maskottchen von Linux ist seit Version 2.0 der Pinguin Tux, das Lieblingstier von Linus Torvalds.

1.2.2 Distributionen

Der Linux-Kernel ist sehr flexibel und arbeitet auf mehreren Prozessor-Architekturen, wo er einen Großteil der vorhandenen Hardware unterstützt. Der Preis für diese Vielseitigkeit ist ein hoher Konfigurationsaufwand, der Laien oft überfordert und selbst Fachleute zuweilen die Stirn runzeln lässt. Allein aus dem Source-Code ein lauffähiges System zu erstellen, ist eine Aufgabe für Experten. Aus dieser Erkenntnis haben sich eine Reihe von Firmen gegründet, die so genannte »Distributionen« von Linux auf CD-ROM anbieten. Der Käufer erhält damit einen lauffähigen Kernel, ein Installations- und Konfigurationsprogramm sowie eine Menge freie und einige kommerzielle Software (deren Umfang oft für die Preisunterschiede zwischen den einzelnen Distributionen verantwortlich ist). In Deutschland bekannte Hersteller sind beispielsweise neben den Marktführern SuSE und Red Hat auch Corel, Caldera, Mandrake oder Debian (wobei es natürlich auch einige andere attraktive gibt). Auch Sie werden vermutlich Ihr Linux-System aus einer solchen Distribution installiert haben.

Viele der Werkzeuge, die ich Ihnen in diesem Buch vorstellen werde, sind bereits in allen Distributionen enthalten, etwa der GNU-C++-Compiler oder das `make`-Utility. Einige werden Sie dagegen in keiner Distribution finden. Daher ist es am besten, wenn Sie entweder die Programme von der beiliegenden CD-ROM installieren oder sich die jeweils neueste Version von den an den entsprechenden Stellen angegebenen Adressen im Internet holen.

1.2.3 Kompatibilität und Portabilität

Fast alle Unix-Software auch für Linux

Wir haben oben festgestellt, dass Linux im Grunde auch eine Unix-Variante ist. Das hat zur Folge, dass fast alle Programme, die für Unix entwickelt wurden, auch unter Linux übersetzt und gestartet werden können. (Genau genommen vereint Linux verschiedene Elemente von zwei unterschiedlichen Unix-Entwicklungslinien, nämlich System V und BSD.) Damit konnten die Linux-Benutzer bereits in der Frühzeit auf ein großes Reservoir an Software zurückgreifen.

Weiter- bzw. Neuentwicklung oft unter Linux

Es ist auch umgekehrt nicht besonders schwierig, unter Linux geschriebene Programme auf andere Unix-Varianten zu portieren. Die große Popularität, die Linux mittlerweile hat, und die preisgünstige Hardware, auf dem es läuft, haben sogar dazu geführt, dass die Weiter- beziehungsweise Neuentwicklung vieler Werkzeuge unmittelbar unter Linux stattfindet und sie erst anschließend auf andere Unix-Plattformen übertragen werden. Sie können also als Linux-Benutzer davon ausgehen, fast immer die aktuellsten Versionen der Applikationen und Tools zur Verfügung zu haben.

Portablität ist natürlich stets ein relativer Begriff. Fast alle Beispielprogramme, die ich Ihnen im Laufe dieses Buches vorstellen werde, können Sie genauso gut unter Windows oder MacOS übersetzen und zum Laufen bringen. Die Programmiersprachen C und C++ sind durch ihre Standardisierung hochportabel. Die Einschränkungen, die in den letzten Absätzen angeklungen sind, beziehen sich daher weitgehend auf direkte Systemzugriffe, also beispielsweise Netzwerk- oder Grafik-Programmierung.

1.3 Kommerzielle und freie Software

Von Anfang an war Software ein Wirtschaftsgut. Wenn man sich ansieht, welche Größe und welchen Umsatz Firmen wie Microsoft erreichen können, deren oft einziges Geschäftsfeld die Erstellung von Software ist, kann man ermessen, dass es dabei um eine Menge Geld geht. Im Unix-Bereich ist das nicht sehr viel anders. Von den Anfängen abgesehen, haben fast alle Hersteller von Unix-Betriebssystemen eine Gebühr für die Überlassung ihrer Software verlangt. Da sich bereits die Hardware, also die Workstations, in einem hohen Preissegment befanden, war es offenbar für die Hersteller von Unix-Applikationen nahe liegend, auch für ihre Produkte ein gehobenes Preisniveau anzusetzen. (Auch heute noch kosten Anwendungen, die sowohl für Windows als auch für ein Unix verfügbar sind, unter Letzterem meist deutlich mehr.)

1.3.1 Das GNU-Projekt

Einer der Ersten, die davon überzeugt waren, dass sich auch freie, also für den Anwender kostenlose Software qualitativ hochwertig erstellen lässt, war Richard Stallman, der Autor des Emacs-Editors (siehe Abschnitt 5.2, Seite 348). Er war einer der Initiatoren des GNU-Projektes, das am Massachusetts Institute of Technology (MIT) in Cambridge, Massachusetts, seinen Ausgangspunkt nahm und das sich das ehrgeizige Ziel gesetzt hatte, ein freies Unix zu schaffen. GNU ist dabei eine rekursiv definierte Abkürzung für »GNU´s Not Unix«.

Die Bestrebungen der GNU-Gemeinde wurden mittlerweile von Linux überholt; damit hat sich im Grunde auch ihr Ziel erfüllt, wenn auch etwas anders als anfangs gedacht. Großen Verdienst an der Nutzbarkeit und damit

Abbildung 1.2 Das Gnu ist das Symbol für das gleichnamige Projekt.

an der Verbreitung von Linux hat das GNU-Projekt aber trotzdem. In ihm wurden (und werden) viele wesentliche Werkzeuge erstellt, die kennzeichnend und notwendig für ein Unix sind (siehe Seite 9).

GNU-Werkzeuge

Dazu gehören:

- die Shell `bash`,
- die C- und C++-Compiler `gcc` beziehungsweise `g++`,
- der Debugger `gdb`,
- das Utility `GNUmake` zur Organisation der Erzeugung von Programmen aus Quelltext,
- der Editor `emacs`
- und viele andere mehr.

Somit kann man sagen, dass der Kernel von Linus Torvalds und die Tools von GNU die beiden Urzellen sind, aus dem sich das gesamte Linux-System entwickelt hat.

Das Maskottchen

Als Symbol für das GNU-Projekt wurde übrigens auch das Bild eines Gnu gewählt (Abbildung 1.2). Wie es im Kommentar auf der GNU-Homepage dazu heißt, hat dieses »den typischen Bart und intelligent aussehende gebogene Hörner. Er oder sie scheint angesichts der bereits vollbrachten Arbeit zufrieden zu lächeln, während sein Blick jedoch noch in die Ferne schweift.«

GNU im Internet

Die Adressen der GNU-Seiten im Internet können Sie Tabelle 1.1 entnehmen. Neben den eigentlichen Sites sind auch die Adressen einiger Mirrors im deutschsprachigen Raum angegeben. Da diese täglich aktualisiert werden, sollten Sie am besten den Ihnen am nächsten gelegenen Server verwenden.

Tabelle 1.1 *Die Seiten des GNU-Projekts im Internet*

GNU´s not Unix	
Homepage:	
Original	www.gnu.org
Mirror Deutschland	org.gnu.de
Mirror Österreich	gd.tuwien.ac.at:8060
FTP:	
Original	ftp.gnu.org
Mirror Deutschland	ftp.informatik.rwth-aachen.de/pub/gnu
Mirror Österreich	ftp.univie.ac.at/packages/gnu
Mirror Schweiz	ftp.eunet.ch/mirrors4/gnu

1.3.2 Die GNU General Public License

Das Besondere an den GNU-Werkzeugen ebenso wie am Linux-Kernel ist, dass es sich dabei um freie Software handelt. Dahinter steckt die Idee, dass jeder diese Software kostenlos benutzen, ja sie sogar verändern kann. Man bezahlt nicht mit Geld, sondern mit seiner Arbeitskraft, durch die man zur Verbesserung und Weiterentwicklung beiträgt (s. a. [WIELAND 2000]).

Die Free Software Foundation

Um dieser Idee Nachdruck zu verleihen und sie auf eine breitere Grundlage zu stellen, gründete Richard Stallman eine Organisation namens *Free Software Foundation* (FSF). Diese gibt die GNU General Public License (*GPL*) heraus, unter deren Bedingungen sämtliche GNU-Software, der Linux-Kernel und viele andere Programme stehen. Es handelt sich dabei um ein ausgefeiltes Vertragswerk, das bis ins Detail die Kommerzialisierung von ehemals freier Software zu verhindern versucht.

Gebühr für Distribution erlaubt

Wer freie Software verbreitet, darf für diese Dienstleistung auch eine Gebühr verlangen, gewährt seinen Kunden aber auch alle Rechte daran, die er selbst hat. Er muss zudem sicherstellen, dass der Benutzer auf Wunsch auch den Quellcode erhält, damit er diesen nach seinen Bedürfnissen abändern kann. Auch eine originalgetreue Kopie der GPL muss immer mitgeliefert werden, damit alle Empfänger über ihre Rechte im Klaren sind.

Neue Software wieder unter GPL

Eine wichtige Einschränkung für Autoren ist, dass jeder, der Teile des Codes eines unter GPL stehenden Programms in einer neuen Software verwenden will, seine Produkte ebenfalls wieder unter die GPL stellen, also frei verfügbar machen muss. Dieser Virus-Effekt hat sehr viel zur Verbreitung und zum Erfolg der freien Software beigetragen.

LGPL für Bibliotheken

Für Bibliotheken (auch für die Systembibliotheken von Linux) ist diese Vorschrift äußerst restriktiv. Denn eigentlich muss ja jedes Programm Code für Systemaufrufe enthalten. Daher gibt es für diese Zwecke eine abgemilderte Form der Lizenzierung, die Lesser General Public License *(LGPL)*. Solange Sie keine Modifikationen an den Bibliotheken vornehmen, dürfen Sie diese zu Ihrer Software dazulinken und dennoch Geld dafür verlangen.

Auf der beiliegenden CD-ROM finden Sie beispielsweise den GNU-Compiler und den Debugger, deren Archive Kopien des GPL-Textes enthalten.

1.3.3 Der Erfolg freier Software

Vorteil: intensiverer und vielseitigerer Test

Die GPL trägt ganz wesentlich zum Erfolg und zur Robustheit von Linux und seinen Werkzeugen bei. Weil alle Nutzer Zugriff auf den Quellcode haben, können sie selbst nach den Ursachen von Abstürzen und Fehlverhalten suchen. Die meisten melden ihre Verbesserungsvorschläge sehr schnell per E-Mail an die federführenden Autoren zurück. Auf diese Weise wird ein Softwarepaket in kurzer Zeit viel intensiver und vielseitiger getestet, als dies ein kommerzieller Hersteller erreichen könnte.

Entwickler arbeiten unentgeltlich

Die Kommunikation über das Internet hat es möglich gemacht, dass selbst Entwicklungsvorhaben, die ehemals als so groß eingestuft wurden, dass nur ein hierarchisch organisiertes und streng planendes Unternehmen sie bewältigen könnte — wie Betriebssysteme, Benutzeroberflächen oder Büro-Softwarepakete —, von einer Gruppe Freiwilliger, die über die ganze Welt verstreut ist, realisiert werden. Denn auch das ist das Ungewöhnliche an Software unter der GPL: Da sie frei ist, erhalten auch die Entwickler *keinen Lohn* dafür. Alle Beteiligten bringen ihre Arbeitsleistung unentgeltlich ein. Die meisten sehen darin eine Art Tauschgeschäft, erhalten sie doch eine große Menge qualitativ hochwertiger Programme zum Nulltarif (einmal abgesehen von den Download-Kosten . . .). Im Gegenzug arbeiten sie dann an dem einen oder anderen Projekt mit, um sich erkenntlich zu zeigen und die Idee der freien Software weiter zu fördern.

Ihr Beitrag

Wenn Sie möchten, können auch Sie mit den Kenntnissen, die Sie sich nach Durcharbeiten dieses Buches erworben haben, Ihren Beitrag zur Verbesserung vorhandener oder zur Erstellung neuer Werkzeuge und Anwendungsprogramme leisten.

1.4 Programmentwicklung in Unix

Von DOS und Windows kennen Sie sicherlich den Unterschied zwischen Programmen und Skripten:

- Ausführbare Programme liegen in maschinenlesbarer Form vor und haben die Endung .exe.
- Skripten sind auch für uns lesbar und haben die Endung .bat. Sie bestehen aus einer Folge von Anweisungen, die der Rechner schrittweise abarbeiten soll.

Unter Unix gibt es diese Programmtypen zwar auch, sie sind jedoch nicht so leicht zu erkennen. Denn so etwas wie standardisierte Namenserwei-

terungen existiert hier nicht. Im Unix-Dateisystem dient ein entsprechendes Attribut dazu, eine Datei als »ausführbar« zu kennzeichnen, wobei die Shell, aus der das Programm gestartet wird, selbst den Unterschied zwischen »echten« Programmen und Skripten erkennt. Für den Benutzer ist also am Namen nicht ersichtlich, was er da gerade lostritt.

1.4.1 Wichtige Begriffe

Für das Verständnis des weiteren Textes brauchen wir noch ein paar Begriffe, damit Sie wissen, von was ich rede. Wenn Sie schon Programmiererfahrung haben, sind Ihnen diese sicher geläufig, so dass Sie diesen Abschnitt überspringen können.

Compiler

Bei den Programmiersprachen C, C++, Fortran, Pascal und so weiter handelt es sich um Hochsprachen, die durch ein spezielles Programm in Maschinencode übersetzt werden, aus dem schließlich eine ausführbare Datei entsteht. Ein solches Programm nennt man *Compiler*. Dieser ist also das zentrale und entscheidende Werkzeug, wenn Sie selbst Programme schreiben wollen. Bei der Umwandlung in maschinenlesbare Form, die man als *Übersetzen* oder Kompilieren bezeichnet, überprüft er Ihre Programmtexte auch daraufhin, ob diese im Einklang mit den Regeln der Programmiersprache (der *Syntax*) stehen. Dabei findet der Compiler nicht nur grobe Fehler, sondern weist Sie auch durch Warnungen auf mögliche Inkonsistenzen und Mehrdeutigkeiten hin. Es ist immer ein guter Stil, die Warnungen des Compilers ernst zu nehmen und die Programme nach Möglichkeit so lange zu bearbeiten, bis keine Warnungen mehr auftreten.

Dateiarten

Quelltext

Was Sie bei Ihren Programmen selbst eintippen, ist der so genannte *Quelltext* (*source code*). Quelltextdateien kennzeichnet man mit einer Endung, die auf die Programmiersprache hinweist. Bei C ist dies einfach .c; bei C++ verwendet man unter anderem .cc, .cxx, .cpp und .C.

Header-Dateien

Wie Sie vielleicht schon wissen, sind bei C und C++ die Schnittstellen von Unterprogrammen vom übrigen Code getrennt. Sie stehen oft in so genannten *Header-Dateien* (auch *Include-Dateien* genannt), die man mit Endungen wie .h, .hh, .hxx oder .hpp kennzeichnet. (Nach dem neuen C++-Standard haben die Header-Dateien der C++-Standardbibliothek gar keine Endungen mehr.) Bei der objektorientierten Programmierung gilt als Faustregel: Für jede Klasse verwendet man eine Quell- und eine Header-Datei.

Objektcode

Aus diesem Quelltext, der aus einer oder mehreren Dateien bestehen kann, erzeugt der Compiler entweder gleich das ausführbare Programm

oder eine Zwischenstufe, den Objektcode, aus dem er dann später das Programm aufbauen kann. Diese haben unter Microsoft-Betriebssystemen die Endung .obj, unter Unix allerdings schlicht .o.

Linker

Wenn der Compiler mit dem Übersetzen der Quelltextdateien fertig ist, baut er das ausführbare Programm aus den Objektcodes zusammen und berücksichtigt auch die Aufrufe von Systemfunktionen, etwa zur Ein- und Ausgabe. Diese Aufgabe übergibt er im Allgemeinen einem darauf spezialisierten Programm, dem *Linker*. Demgemäß spricht man von diesem Vorgang auch als »linken«.

Bibliotheken

Manchmal möchte man auch mehrere Unterprogramme in übersetzter Form zusammenfassen, die allein zwar noch kein eigenes Programm bilden, die aber für verschiedene Programme nützlich sein könnten. Solche Codesammlungen bezeichnet man als *Bibliotheken* (engl. *libraries*).

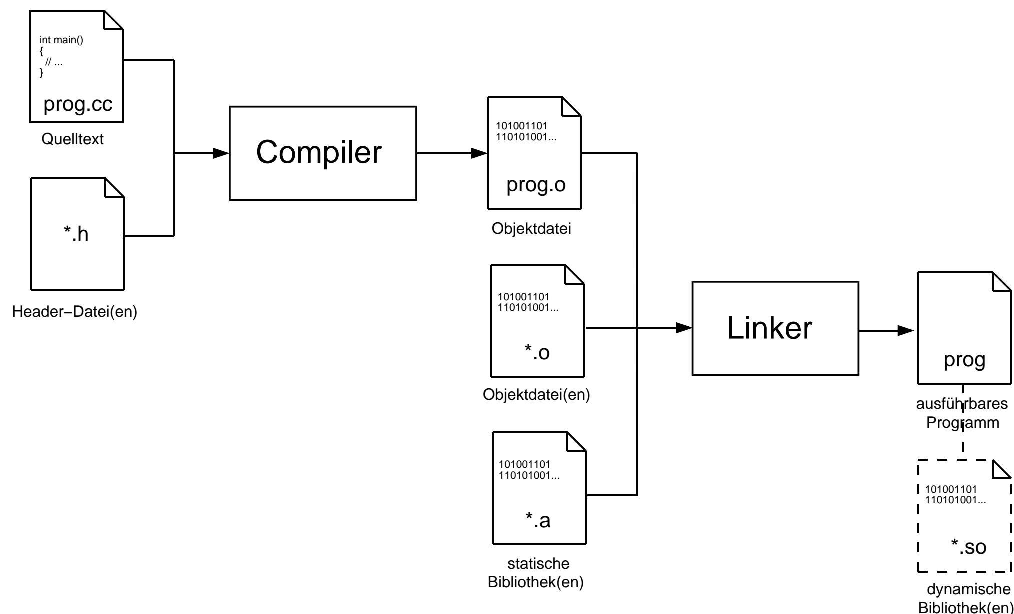

Abbildung 1.3
Compiler und Linker erstellen die ausführbare Programmdatei.

Bibliotheken gibt es übrigens in zwei Varianten:

- als *statische Bibliotheken*, die die Endung .a tragen und zur »Compile-Zeit« (wie man sagt) zur ausführbaren Programmdatei dazugelinkt werden, und

- als *dynamische Bibliotheken*, die erst zur Laufzeit des Programms hinzugezogen werden (entsprechend einer DLL unter Windows). Diese tragen die Endung .so (für »shared objects«).

Wenn Sie eigene Bibliotheken verwenden, geben Sie dies dem Linker bekannt, der dann die notwendigen Bibliotheksabschnitte in Ihr ausführbares Programm kopiert. Systembibliotheken werden im Allgemeinen automatisch eingebunden.

Abbildung 1.3 veranschaulicht nochmals die Zusammenhänge. Beachten Sie dabei, dass für ein einfaches Programm bereits eine einzige Quelldatei genügt. Header-Dateien und weitere Objektdateien benötigen Sie erst bei größeren Projekten, die Sie aus mehreren Dateien zusammensetzen. Dynamische Bibliotheken sind optional und daher gestrichelt gezeichnet.

Beispiel

In Abschnitt 6.1 ab Seite 372 werden wir uns beispielsweise der Frage widmen, wie ein Programm zur Erinnerung an Geburtstage von Freunden gepflegt werden kann und wie dessen Dateien bei Bedarf zu übersetzen sind. Im Spiel sind dabei folgende Dateien:

- Quelltextdateien: date.cc für das Datum, birthlist.cc für eine Geburtstagsliste und birthday.cc als Hauptprogramm
- Header-Dateien: date.h enthält die Deklarationen zum Datum, birthlist.h enthält die der Geburtstagsliste
- Objektcode: Jede der drei Quelltextdateien wird zu einer Objektdatei kompiliert, also date.o, birthlist.o und birthday.o
- Bibliotheken: Der Linker bindet selbstständig die nötigen Systembibliotheken hinzu, beispielsweise libc.a
- Ausführbares Programm: birthcontrol

1.4.2 Systemdateien zur Entwicklung

Wenn Sie schon mehr Erfahrung mit Linux (oder einem anderen Unix) haben, kennen Sie sicherlich die Konventionen, in welchen Verzeichnissen welche Dateien und Dateitypen abgespeichert und aufbewahrt werden. Für einen Entwickler kann es von Vorteil sein, ein wenig darüber zu wissen, welche Dateien das System bereitstellt und wo er diese finden kann.

Programme

Ausführbare Programme finden Sie üblicherweise in Unterverzeichnissen mit dem Namen bin, unter anderem

- /usr/bin mit den meisten der Programme, die vom System zur Verfügung gestellt werden, darunter oft auch Compiler, Linker und andere Werkzeuge für Entwickler.

- In /bin befindet sich eine kleine Sammlung der elementarsten Systemprogramme (wie Shells, *ls* oder *ps*).
- /usr/local/bin enthält solche Programme, die nicht zum Standard im engeren Sinne gehören beziehungsweise nur für diesen Computer einzeln hinzugefügt wurden. Die Unterscheidung zwischen /usr/bin und /usr/local/bin wird oft bei lokalen Netzen mit mehreren Rechnern gemacht, um eine einheitliche (und damit besser zu pflegende) Konfiguration aller Geräte zu erreichen; Programme, die nicht auf allen, sondern nur auf einem Computer installiert sind, legt man dann unter /usr/local/bin ab. Bei Einzelplatzrechnern ist die Trennung meist weit weniger streng, so dass hier oft /usr/local/bin gleich ganz leer bleibt.
- In usr/X11/bin finden Sie Programme mit grafischer Benutzeroberfläche, die unter dem X-Window-System laufen. Auch die ausführbaren Dateien zum Starten und Betreiben dieser Oberfläche sind meist hier abgelegt.

Bibliotheken

Die Standardbibliotheken des Systems stehen unter /lib und /usr/lib, etwa libm.a für mathematische Routinen oder libpthread.a für die Parallelisierung von Programmen. Ähnlich wie bei den Programmen verwendet man manchmal auch /usr/local/lib. Bibliotheken für die X-Window-Programmierung finden sich in /usr/X11/lib.

Header-Dateien

Die gerade erwähnten Standardbibliotheken des Systems sind üblicherweise in C geschrieben. Um Funktionen aus diesen in C- oder C++-Programmen zu nutzen, brauchen Sie auch die zugehörigen Header-Dateien mit den Definitionen der Schnittstellen. Im Verzeichnis /usr/include und dessen Unterverzeichnissen finden Sie die meisten davon.

Wie Sie vielleicht wissen, liefert unter MS-Windows im Allgemeinen erst die Entwicklungsumgebung des Compilers die Header-Dateien des Windows-API (der Programmierschnittstelle) mit. Dagegen sind unter Unix die entsprechenden Dateien normalerweise auf jedem System installiert — unabhängig vom gewählten Compiler und auch unabhängig davon, ob Sie Ihren Rechner überhaupt zum Programmieren nutzen wollen oder nicht.

Das genannte Verzeichnis kann auch Dateien für spezielle Bibliotheken in Unterverzeichnissen enthalten, etwa

- /usr/include/sys für betriebssystemnahe Routinen,
- /usr/include/X11 für die Bibliotheken zur X-Window-Programmierung oder

- /usr/include/g++ für die GNU-Implementierung der C++-Standard-bibliothek.

Manche Systemwerkzeuge, die Sie auf Ihrem Rechner installieren, hinterlegen die Header-Dateien zu ihren Bibliotheken auch im Verzeichnis /usr/local/include.

Wir werden uns natürlich noch sehr ausführlich mit den Bibliotheken, ihren Funktionen und ihrer Verwendung beschäftigen. Mit diesen Überblicksinformationen fällt es Ihnen indessen sicherlich leichter sich zu orientieren, wo Sie welche Arten von Dateien finden können.

1.5 Übungsfragen

1. Wann darf sich ein Betriebssystem ´UNIX´ nennen?
2. Nennen und erklären Sie zwei besondere Eigenschaften von Unix?
3. Was versteht man unter einer ´Linux-Distribution´? Was enthält eine solche?
4. Aus welchen beiden Entwicklungslinien ist das Linux-Gesamtsystem entstanden?
5. Was ist die Idee der ´freien Software´? Nennen Sie drei kennzeichnende Eigenschaften freier Software.
6. Wozu benötigt man Header-Dateien? In welchen Verzeichnissen finden Sie die Header-Dateien für die Systembibliotheken?