SelfLinux-0.10.0

Programmierung unter Linux

Autor: Florian Fredegar (*florian.haftmann@stud.tum.de*)
Formatierung: Axel Gross (*axelgross@web.de*)

Inhaltsverzeichnis

1 Muss man programmieren können, um mit Linux zu arbeiten?

Hier lautet die Antwort definitiv: Nein. Die Zeiten, in denen es notwendig war, sich zum Allround-Programmierer fortzubilden, um Linux überhaupt zum Laufen zu bringen, haben wir hinter uns. Natürlich benötigt man nach wie vor gewisse Computerkenntnisse, um Linux zu administrieren, aber wenn mal alles läuft (oder von jemand anderem zum Laufen gebracht wurde), ist Linux nicht schwerer zu handhaben als andere Betriebssysteme, die gern mit ihrer Benutzerfreundlichkeit hausieren gehen.

2 Wieso sich dann mit Programmierung beschäftigen?

Programmieren ist Fun - natürlich, Bergsteigen, Schifahren oder Zeichnen gefällt den Leuten, die sich dafür interessieren, ganz hervorragend. Hingegen gibt es andere Zeitgenossen, die damit gar nichts anzufangen wissen. So verhält es sich auch mit dem Programmieren. Dennoch sollte man dies nicht zum Anlass nehmen, nicht herauszufinden, zu welche Sorte Mensch in Bezug aufs Programmieren man denn gehört - sonst verpasst man den erhabenen Einblick in "eine innere Welt, wo niemand jemals zuvor gewesen ist".

Ein paar Programmierkenntnisse helfen in vielen Situationen, stumpfsinnige Arbeit zu vermeiden. Beispiel: man hat eine Seite aus dem WWW heruntergeladen, die man gerne ausdrucken und ablegen möchte; leider stellt sich beim Drucken heraus, dass die kleinste Schriftart etwas zu fusselig geworden ist. Mann kann jetzt "von Hand" bei jedem zu kleinen Absatz die Schriftgröße erhöhen - ab mittlerer Dokumentlänge keine Tätigkeit, die einen zum Jauchzen bringt; aber mit der geeigneten Programmiersprache ist in Minutenschnelle ein Skript geschrieben, dass die stumpfsinnige Arbeit im Nu verrichtet. "Die Faulen arbeiten, die Intelligenten optimieren".

Programmieren erweitert die Möglichkeiten, das Gerät Computer besser zu beherrschen; man bekommt ein Gespür dafür, was geht und was nicht und traut sich plötzlich Dinge in Angriff zu nehmen, die einem vor kurzen noch nicht lösbar erschienen sind.

Je mehr man in die Programmierung eintaucht, desto besser versteht man die Funktionsweise des Rechners und des Betriebssystems. Zu den schönsten Momenten eines Programmierenden gehören jene, wo ein bislang undurchschaubares Phänomen sich auflöst in eine elegante Gedankenkonstruktion.

Linux ist nicht vom Himmel gefallen; Linux wurde uns gebracht von einer großen Gemeinde an Freiwilligen, die unentgeltlich Schwerstarbeit leisten. In vielen zufriedenen Benutzern schlummert daher der Wunsch, der Entwicklergemeinde etwas zurückgeben zu können - und die beste Möglichkeit hierfür ist, aktiv zur Weiterentwicklung von Linux beizutragen.

Um ein altes Vorurteil auszuräumen: um programmieren zu können, ist es nicht nötig, die Mathematik gut zu kennen. Natürlich muss man sich eine gewisse Form des Denkens antrainieren, die auch beim Verständnis der Mathematik hilfreich ist, aber diese Fähigkeit muss sich nicht unbedingt in mathematischer Begabung äußern. Wer z. B. gerne Kreuzworträtsel oder Denksportaufgaben löst, ein analytisches Ohr für Musik besitzt oder gut Schach spielt, beherrscht dieses Denken höchstwahrscheinlich auch, während z. B. die Fähigkeit, diskrete Faltungssummationen über unendliche nichtkonvergente Reihen auszurechnen über das Programmiertalent noch nicht viel verrät.

Was aber nicht verschwiegen werden soll: ohne grundlegende Kenntnisse der englischen Sprache geht so gut wie nichts; natürlich gibt es Einführungsbücher in allen möglichen Sprachen, aber bei der Komplexität heutiger Entwicklungssysteme führt kein Weg an Dokumentationen vorbei, die nunmal nur in Englisch vorliegen. Man sollte in der Lage sein, diese so flüssig zu lesen, dass es nicht sonderlich anstrengt, zu verstehen, was drin steht.

3 Was für unterschiedliche Arten an Programmiersprachen gibt es?

Programmiersprachen lassen sich nach unterschiedlichen Kategorien klassifizieren. Zuerst einmal gibt es Sprachen, die zwar oft als Programmiersprachen bezeichnet werden, aber in eigentlichem Sinne keine darstellen: dazu gehören z. B. die Seitenbeschreibungssprache PostScript, die Auszeichnungssprache HTML des World Wide Web oder die Datenbank-Anfragesprache SQL. Daneben gibt es die echten Programmiersprachen, mit denen man dem Computer tatsächlich etwas "befehlen" kann.

Daneben gibt es die Unterscheidung in proprietäre und offene Programmiersprachen; Programmiersprachen heißen i. A. offen, wenn:

* sie in allgemein zugänglicher Form standardisiert sind
* eine freie Implementierung existiert
* für mehrere Betriebssysteme Implementierungen existieren
* sie nicht von einem elitären Zirkel, sondern einer offenen Community gepflegt und fortentwickelt werden.

Sonst heißen sie proprietär; welchen Typ Sprachen der Linuxer bevorzugt, ist nach dem oben gesagten nicht schwer zu erraten...

Etliche Programmiersprachen sind nur für einen bestimmten Zweck optimiert, für den sie sich dann hervorragend eignen. Dafür taugen sie gar nichts, wenn man mit ihnen irgendwas anderes machen will (Nischensprachen). Im Gegensatz dazu sind die Allzwecksprachen prinzipiell für alle Probleme geeignet, auch wenn sie sie unterschiedlich gut oder elegant bewältigen.

Ein Computer, "so wie er ist", versteht genau genommen nur eine Programmiersprache: die Maschinensprache seines Prozessors (auch Assembler genannt). Um Programme in anderen Sprache überhaupt zum Laufen zu bringen, muss man einen der folgenden Wege einschlagen:

* Ein Compiler übersetzt vor der Ausführung das gesamte Programm in die Maschinensprache; das hat u. a. den Vorteil, dass damit eine recht schnelle Ausführung erzielt wird. Nachteil ist, dass das Compilieren selbst sehr zeitaufwändig sein kann und gewisse Programmiertechniken, die dann und wann sinnvoll sind (z. B. sich selbst ändernde Programme), hier nicht eingesetzt werden können.
* Ein Interpreter arbeitet Stück für Stück die Anweisungen der Programmiersprache ab; das geht i. A. auf Kosten der Geschwindigkeit, vereinfacht aber u. U. die Entwicklung von Programmen.
* Um die Vorteile beider Welten vereinen zu können, gibt es auch noch folgende Möglichkeit: die Programmiersprache wird von einem Compiler in eine Zwischensprache (Bytecode) übersetzt, die dann von einem Interpreter ("virutal machine") hochperformant ausgeführt werden kann.

Um Programmierprobleme zu beschreiben, gibt es unterschiedliche Paradigmen. Die wichtigsten sind:

* das funktionale Paradigma: man beschreibt, wie verschiedene Funktionen ihre Funktionswerte aus den Eingabewerten bestimmen und beschreibt das Problem als geeignete Komposition mehrerer Funktionen.
* das imperative Paradigma: man gibt dem Computer Anweisungen, wie er das Problem schrittweise lösen kann.
* das objektorientierte Paradigma: man modelliert das Problem mit unterschiedlichen Objekten, die untereinander über wohldefinierte Schnittstellen kommunizieren.

Dies ist nur eine grobe Definition. Um zu erkennen, was dies tatsächlich bedeutet, muss man erst etwas Programmiererfahrung sammeln. Darüberhinaus gibt es andere Ansätze in Nischensprachen (z.B. PROLOG: logisches Paradigma)

In der theoretischen Informatik ist noch der Begriff der "Turing-Vollständigkeit" bedeutsam; eine Sprache heißt turing- vollständig, wenn man jedes Problem, das berechenbar ist, mit ihr lösen kann. Eine Allzweck-Sprache ist immer turing-vollständig, es gibt aber auch turing-vollständige Sprachen, die, obgleich sie alles berechnen können, für die Praxis nicht taugen, da sie entweder so kompliziert sind, dass sie keiner beherrschen kann, oder so simpel, dass für eine einfache Addition seitenlange Anweisungsschritte nötig sind...

Weitere Unterscheidungsmöglichkeiten ergeben sich nach dem Einsatzbereich der Programme:

* Daemonen (auch: Services, Dienste) sind Programme, die ständig im Hintergrund laufen sollen und dort irgendwelche Sachen erledigen, z.B. Webseiten ausliefern, Mail weiterleiten, den Drucker steuern usw. Der eigentliche Computerbenutzer kommt mit ihnen so gut wie gar nicht in Berührung. Da für solche Dämonen ein Höchstmaß an Stabilität und Performance sowie eine enge Interaktion mit dem Betriebssystem angestrebt wird, sind sie meist in C geschrieben, es gibt aber auch Ausnahmen.
* Viele Programme sind dazu gedacht, von der Textkonsole aus gestartet zu werden, dabei ggf. Argumente und Funktionen entgegenzunehmen und mit einer Ausgabe zu antworten. Dazu gehören unter anderem die grundlegenden Befehle (cp, ls, rm, ...), Shell-Scripte oder Programme in anderen Skriptsprachen, insbesondere die "Einweg-Scripte", die nur für eine einmalig zu erledigende Aufgabe geschrieben und anschließend nicht mehr verwendet werden.
* Die meisten Endbenutzer sind es gewohnt, dass ein Programm über eine gefällige grafische Benutzeroberfläche (GUI - Graphical User Interface) zu bedienen ist. Die meisten Standardsprachen verfügen über Bibliotheken oder Schnittstellen, mit denen sich solche GUI-Programme realisieren lassen können.
* Seit dem Durchbruch des World Wide Web gewinnt die Gattung der webbasierten Programme an Bedeutung: diese nehmen ihre Eingaben über ein Formular im Webbrowser entgegen und liefern als Antwort eine HTML-Seite, die im Browser angezeigt wird. Charakteristisch ist, dass der Benutzer und der Rechner, auf dem das Programm läuft, voneinander entfernt sind. Dies ist die klassische Domäne von Skriptsprachen wie Perl, Python oder PHP.
* Ein bisschen aus dem Rahmen fallen die so genannten "Sandbox-Programme". Diese sind gar nicht dazu gedacht, direkt vom Betriebssystem heraus gestartet zu werden, sondern machen nur in einer speziellen Umgebung ("Sandbox") Sinn. Für Programme in der Sprache PROLOG ist das der PROLOG-Interpreter, für JavaScript der Browser usw.

4 Was macht Linux fürs Programmieren so attraktiv?

Um in die Programmierung einzusteigen, ist Linux ganz besonders gut geeignet:

* Linux kommt mit einer kompletten Programmier-Umgebung: neben der Systemsprache C und den unterschiedlichen Shells gibt es Compiler und Interpreter für fast alle gängigen Programmiersprachen, Bibliotheken, Debugging- und Automationstools und Editoren wie Sand am Meer. Nichts muss zeitraubend oder kompliziert nachinstalliert werden.
* Linux ist als UNIX-ähnliches System in seiner internen Struktur programmiererfreundlich angelegt - man kann von Haus aus "in das System hineinschauen", es wird nicht künstlich verschleiert, wie die einzelnen Komponenten zusammenwirken. Linux ist das Gegenteil eines Systems, das sagt: "Ich glaube nicht, dass du verstehen kannst, wie ich funktioniere. Ich werde dafür sorgen, dass du mir hilflos ausgeliefert bleibst und ich dir wie ein höheres, überlegenes Wesen vorkomme, mit dem du nur über mysteriöse Rituale kommunizieren kannst."

5 Wo anfangen?

Hierzu sei *Eric S. Raymond* zitiert, der den Weg zum Hacker (wem das zu verwegen klingt, der ersetze sinngemäß: Meisterprogrammierer) in seinem Text How To Become A Hacker wie folgt skizziert:

> Wenn man noch keine Computersprache kennt, empfehle ich, mit Python anzufangen. Es hat ein sauberes Design, ist gut dokumentiert und recht entgegenkommend für Anfänger: eine gute Einstiegssprache, aber deswegen noch lange kein Spielzeug. Python ist sehr mächtig, flexibel und gut geeignet für große Projekte. Ich habe eine eingehendere Untersuchung über Python verfasst.

Gute Tutorials gibt es auf der Python-Website oder hier.

Java ist auch eine gute Sprache, um damit programmieren zu lernen. Sie ist schwieriger als Python, aber liefert schnelleren Code. Nach meiner Meinung eine hervorragende zweite Sprache. Zu Java gibt es ein komplettes deutsches Tutorial in Buchform mit dem Titel "Java ist auch eine Insel" im Internet.

Aber Vorsicht! Man kann nicht die Fähigkeiten eines Hackers oder auch nur eines Programmierers erreichen, wenn man nur eine oder zwei Sprachen kennt - vielmehr muss man lernen, über Programmierprobleme in einer recht allgemeinen Weise zu denken, unabhängig von einer Sprache. Um ein richtiger Hacker zu werden, muss man es so weit bringen, dass man eine neue Sprache innerhalb von Tagen lernen kann, indem man das, was im Handbuch steht in Beziehung bringt zu dem, was man bereits weiß. Das bedeutet, dass man etliche grundverschiedene Sprachen lernen sollte.

Um richtig ernsthaft zu programmieren, muss man C lernen, die zentrale Sprache von UNIX. C++ ist nah verwandt zu C; wenn man das eine kennt, wird es nicht schwer fallen, das andere zu lernen. Keine der beiden Sprachen jedoch ist eine gute Wahl, um sie als erstes zu lernen. Tatsächlich kann man produktiver arbeiten, je mehr man C vermeidet.

C ist sehr effizient, und sehr schonend mit den Ressourcen des Rechners. Unglücklicherweise erreicht C diese Effizient, in dem es einem abverlangt, einen großen Packen Low-Level-Management (z. B. Speicherverwaltung) von Hand zu erledigen; dieser ganze Low-Level-Code ist komplex und fehlerträchtig, und wird viel Zeit fürs Debuggen (die Fehlersuche) verschlingen. Mit heutigen Maschinen, so schnell wie sie sind, ist das ungünstig - es ist besser, eine Sprache zu verwenden, die etwas verschwenderischer mit der Rechenzeit umgeht, aber die Zeit des Programmierers schont. Daher: Python.

Zu anderen Sprachen mit besonderer Wichtigkeit für Hacker gehören Perl und LISP. Perl ist lernenswert aus praktischen Gründen: es ist weit verbreitet bei aktiven Webseiten und bei der Systemadministration, so dass man Perl verstehen lernen sollte, auch wenn man nie damit was schreibt. Viele Leute verwenden Perl für den Zweck, für den ich Python vorschlage: Vermeidung von C, wenn die Programmieraufgaben nicht die Effizienz von C verlangen. Es wird sich als nötig erweisen, ihren Code zu verstehen.

LISP ist aus einem anderen Grund lernenswert - die beeindruckende Erfahrung der Erkenntnis, wenn man es schließlich verstanden hat. Diese Erfahrung macht einen zu einem besseren Programmierer für den ganzen Rest seines Lebens, selbst wenn man LISP selbst kaum einsetzt.

In der Tat ist es das Beste, alle diese fünf Programmiersprachen zu lernen. Python, Java, C/C++, Perl, und LISP. Nicht nur, dass sie die wichtigsten Hackersprachen sind, sie verkörpern auch sehr unterschiedliche Herangehensweisen ans Programmieren, und jede einzelne bringt unschätzbare Erfahrungen.

Ich kann an dieser Stelle keine vollständige Instruktion geben, wie man Programmieren lernt - es ist eine komplexe Fertigkeit. Aber ich kann behaupten, dass es Bücher und Kurse nicht tun werden (viele, vielleicht sogar die meisten der besten Hacker sind Autodidakten). Man kann einzelne Features einer Sprache aus Büchern

lernen, aber das geistige Rüstzeug, das dieses Wissen in lebendige Fertigkeit umsetzt, kann nur durch Übung und Lernen erworben werden. Was zu tun ist, ist (a) Code zu lesen und (b) Code zu schreiben.

Programmieren lernen, ist wie eine natürliche Sprache beherrschen lernen. Das Beste ist, ein bisschen was zu lesen, was von Meistern der Disziplin geschrieben wurde, ein bisschen was selbst zu schreiben, sehr viel mehr zu lesen, ein bisschen mehr zu schreiben, sehr viel mehr zu lesen, etwas mehr zu schreiben... und das wiederholen, bis die eigenen Schriftwerke die Kraft und Anmut bekommen, die man in den Vorbildern erkennt.

Guten Code zu finden ist in der Vergangenheit schwierig gewesen, da nur wenige große Programme im Quellcode vorlagen, den angehende Hacker lesen oder damit herumspielen hätten können. Das hat sich gewaltig verändert; Open-Source-Software, Programmierwerkzeuge und Betriebssysteme, ausschließlich von Hackern aufgebaut, sind nun allgemein verfügbar."

Das ist natürlich nur eine ungefähre Richtschnur. Man sollte im Zweifelsfall einfach der eigenen Nase folgen und soviel davon lernen wie man braucht, um die Arbeit, die man erledigen will, gut erledigen zu können. Das wichtigste aber: nicht entmutigen lassen! Es ist bekanntlich noch kein Meister vom Himmel gefallen. Mit dem Programmieren verhält es sich im gewissen Sinne wie mit Schach: die Regeln sind einfach, aber um wirklich Schach zu spielen, braucht es viel Übung.

In diesem Sinne: Happy Hacking!