

Proseminar im Sommersemester 2008: Linux für Umsteiger

C/C++-Programmentwicklung unter Linux

Elvira Khisamutdinova

Universität Heidelberg

8. Juli 2008

Betreuer: Julian Martin Kunkel, Olga Mordvinova

Gliederung

- Vorbereitung
- Kompilieren/Ausführen
- Tools
 - make-Programm
 - ddd-Debugger
- Entwicklungsumgebungen
 - KDevelop
 - Eclipse
- Fazit
- Quellen

Hier ist meine Gliederung:

Zuerst kommt die Vorbereitung: ich werde erklären warum man mit Programmieren überhaupt anfängt, wie sich Programmiersprachen C und C++ unterscheiden und welche Entwicklungswerkzeuge man braucht; dann erzähle ich, wie man Programme unter Linux kompiliert und ausführt, dann wie das make-Programm und der Debugger funktionieren, dann wie man den Programmierprozess einfacher und schöner mit Entwicklungsumgebungen KDevelop und Eclipse machen kann. Und zum Schluss kommen Zusammenfassung und meine Quellen.

Bevor ich anfangen möchte ich euch darauf aufmerksam machen, dass mein Vortrag keine Einführung in die Programmierung mit C und C++ ist. Selbstverständlich werde ich versuchen, einige Begriffe möglichst einfach zu erklären: dabei werden kleine C/C++ Beispielprogramme sehr hilfreich. In der ersten Linie wird aber die Programmierung selbst stehen und zwar unter Linux.

Gliederung

✓Vorbereitung

- Kompilieren/Ausführen
- Tools
 - make-Programm
 - ddd-Debugger
- Entwicklungsumgebungen
 - KDevelop
 - Eclipse
- Fazit
- Quellen

Motivation

- keine benötigte Anwendung
- keine zufriedenstellende Anwendung
- Vereinfachung bestimmter Aufgaben
- Fun

Motivation:

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 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.

Wieso sich dann mit Programmierung beschäftigen? Ein paar Programmierkenntnisse helfen in vielen Situationen, z.B.: es gibt keine Anwendung, die man bräuchte oder die existierende stellt nicht zufrieden oder man will eine bestimmte Aufgabe vereinfachen, um stumpfsinnige Arbeit zu vermeiden. Und Programmieren ist natürlich Fun - Bergsteigen, Schifahren oder Zeichnen gefallen den Leuten, die sich dafür interessieren. Es gibt aber auch andere, die den erhabenen Einblick in "eine innere Welt, wo niemand jemals zuvor gewesen ist", nicht verpassen wollen.

Unter Linux stehen mehrere Programmiersprachen zur Verfügung. Warum es sich lohnt, sich mit C oder C++ zu beschäftigen, erkläre ich gleich.

Kurzvorstellung

- ✓ C:
mächtig, flexible, portabel, "Hauptsprache" unter Linux
Beispiel: LinuxKernel
- ✓ C++:
Obermenge von C, Objektorientierung
Beispiel: KDE

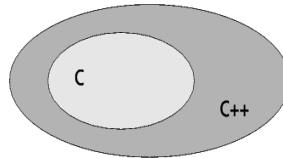


Abb.1: C++ - Obermenge von C

Bemerkung: keine Einstiegssprachen

Die Programmiersprache C ist mächtig und flexible: ihr Ersatzbereich ist unbeschränkt.

C ist portabel: ein Programm, geschrieben für ein bestimmtes Computersystem, läuft auch auf einem anderen Computersystem.

C ist die "Hauptsprache" unter Linux: weil der Linuxkernel in C geschrieben ist und ein Großteil der zu einer typischen Linux-Distribution gehörenden Programme auch.

Und was ist mit C++?

C++ ist eine Obermenge von C, das heißt, sie hat alle Syntaxeigenschaften von C, aber noch einige z.B. Objektorientierung. ("Objektorientierte Programmierung" befasst sich mit Programmen, die Klassen verwenden. Wir können uns ein Objekt als eine selbständige Instanz vorstellen, die ihre eigenen Daten verwaltet und ihre eigenen Funktionen besitzt.) C++ ist auf vielen Plattformen inkl. Linux sehr häufig anzutreffen. Beispiel unter Linux ist KDE, welche komplett in C++ mit Hilfe der Bibliothek Qt geschrieben ist. Man schreibt viele kommerzielle Programme in C++.

Eine kleine Bemerkung: bitte, Vorsicht, C und C++ sind keine Einstiegssprachen, keine der beiden ist eine gute Wahl, um sie als erstes zu lernen. Dafür ist z.B. Python sehr gut geeignet. Eine Einführung in die Sprache hatten wir vor einigen Wochen.

Entwicklungswerkzeuge

- Text-Editor (vim, emacs, etc)
- Compiler (GCC)
- make-Programm
- Debugger

Ubuntu: Paket **<build-essential>**

Um mit Programmieren unter Linux anzufangen, braucht man einige Programmierwerkzeuge:

- einen Text-Editor (z.B. vim oder emacs. Emacs ist für das Programmieren ganz besonders geeignet, weil er so viele Erweiterungen erhält, dass man zum Programmieren eigentlich nichts anderes mehr benötigt);
- einen Compiler (der Standard-Compiler unter Linux ist GCC. Das ist GNU Compiler Collection (wurde in C geschrieben, "Collection", weil er mehrere Programmiersprachen bietet) und
- solche hilfreiche tools wie z.B. das make-Programm und den Debugger (für Fehlersuche).

Bei Ubuntu sind alle diese Programme in einem Paket **<build-essential>** zusammengefasst. Später kann man vielleicht weitere Bibliotheken oder Developer-Pakete brauchen aber für den Anfang sollte das einmal reichen.

Gliederung

- ✓ Vorbereitung
- ✓ Kompilieren/Ausführen
 - Tools
 - make-Programm
 - ddd-Debugger
 - Entwicklungsumgebungen
 - KDevelop
 - Eclipse
 - Fazit
 - Quellen

Kompilieren/Ausführen

1. eine Datei mit dem Quellcode erstellen
\$ emacs erstes_programm.cpp
2. den Quellcode kompilieren
\$ g++ erstes_programm.cpp
3. das Programm ausführen
\$./a.out

Anmerkung:

C: ersetze g++ durch gcc

Wie entwickelt man ein Programm?

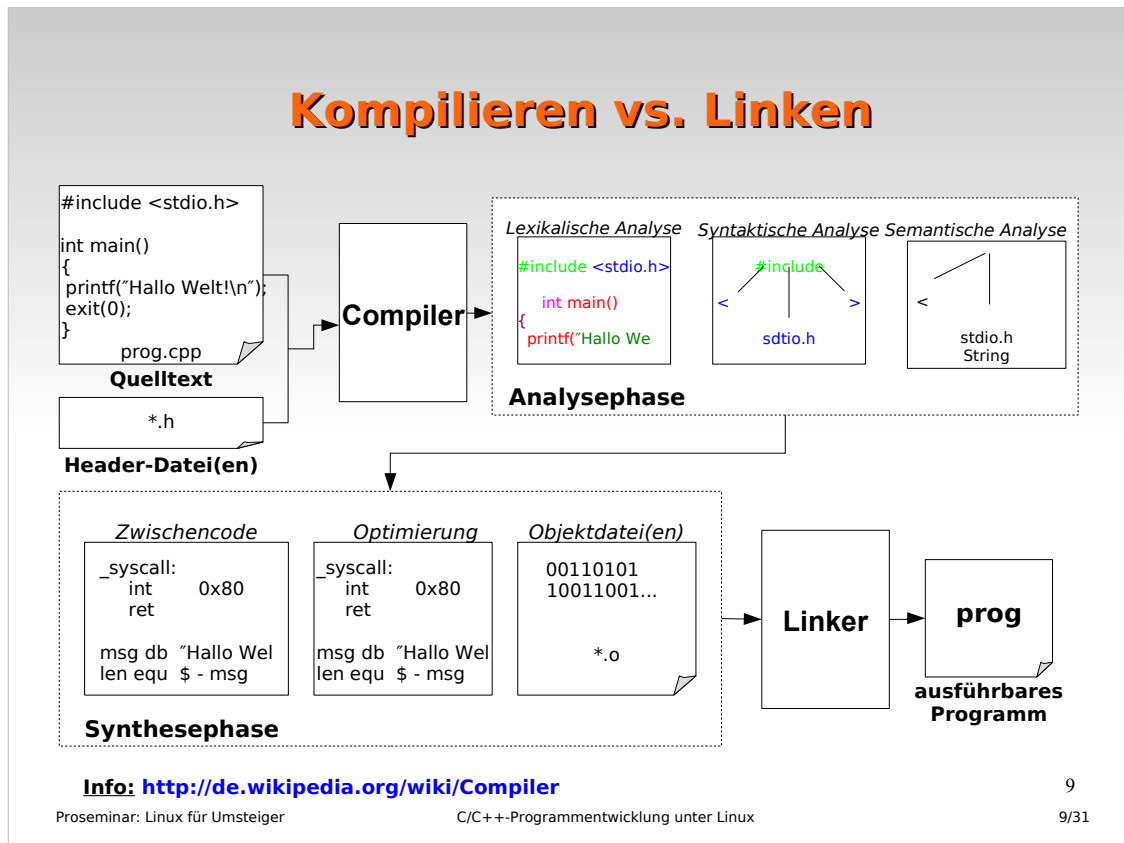
1) Zuerst erstellt man mit einem Text-Editor eine Datei mit dem Quellcode. (Quellcode ist was wir bei unseren Programmen selbst eintippen, also, Text eines Computerprogrammes).

2) Dann übersetzen wir diesen Quellcode mit einem Compiler in Maschinencode. Wenn es keine Fehler gibt, wird die Datei gelinkt und eine ausführbare Datei namens a.out erzeugt.

3) Und zum Schluss führt man das Programm aus, um festzustellen, ob es wie geplant funktioniert.

Bei C-Programmen muss man g++ durch gcc ersetzen.

Kompilieren vs. Linken



Diese Folie zeigt die Aufgaben vom Compiler und Linker.

Moderne Compiler werden in verschiedene Phasen gegliedert. Im Wesentlichen lassen sich zwei Phasen unterscheiden: Analysephase und Synthesephase. Während der Analysephase werden die Quelltexte und Header-Dateien analysiert, strukturiert und auf Fehler geprüft und während der Synthesephase wird der Zwischencode erstellt und optimiert und schließlich Objektdateien erzeugt.

Ein weiterer Arbeitsschritt, der häufig automatisch mit ausgeführt wird, heißt das Linken und wird vom Linker gesteuert. Der Linker verbindet einzelne Programmmodule zu einem ausführbaren Programm. Die meisten Programme enthalten Bestandteile oder Module, die in anderen Programmen Verwendung finden können. Mehrere kompilierte Module mit Funktionen (so genannte Objektdateien) können zu Funktionsbibliotheken (Programmbibliotheken) zusammengefasst werden. Der Code wird vom Linker zum Hauptprogramm hinzugefügt, falls die entsprechende Funktion benötigt wird. Um ein Programmmodul in einem anderen Programm verwenden zu können, müssen die symbolischen Adressen der Funktionen und Variablen des Moduls in Speicheradressen umgewandelt werden. Genau diese Aufgabe übernimmt der Linker.

Name der Ausgabedatei

```
$ g++ erstes_programm.cpp -o erstes_programm  
$ ./erstes_programm
```

Mehrere Quelltextdateien

```
$ g++ datei1.ccp datei2.cpp /gen/datei3.cpp -o prog  
$ ./prog
```

Standardmäßig wird immer der Name "a.out" für die Ausgabedatei verwendet. Das kann man jedoch über den Parameter "-o" ändern.

Wenn ein Programm aus mehreren Quelltextdateien besteht, kann man auch mehrere hintereinander angeben.

```
$ g++ datei1.ccp datei2.cpp /gen/datei3.ccp -o prog
```

Also besteht das ausführbare Programm "prog" aus drei Dateien: datei1.ccp und datei2.ccp, die sich im gleichen Verzeichnis befinden und aus der Datei datei3.ccp, die sich im Verzeichnis /gen befindet.

Wichtige Parameter für gcc

Option	Bedeutung
-o	Bestimmt den Namen der Ausgabedatei (Standard: „a.out“)
-c	Der Quellcode wird nur kompiliert, aber noch nicht gelinkt
-g	Hinzufügen von Informationen für den Debugger
-Wall	Ausgabe von Warnungen bei "unschönem" Code
-v	Gibt die Kommandos und Schritte, die der Compiler ausführt (Präprozessor, Compiler, Linker, etc.)

Der GNU-Compiler kennt viele anderen Optionen. Hier sind die Wichtigsten.

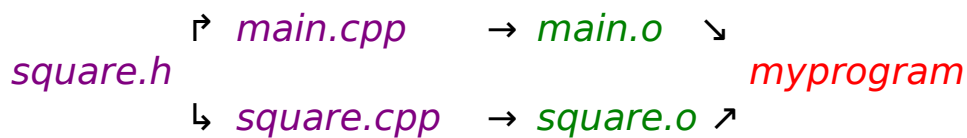
Gliederung

- ✓ Vorbereitung
- ✓ Kompilieren/Ausführen
- ✓ Tools
 - make-Programm
 - ddd-Debugger
- Entwicklungsumgebungen
 - KDevelop
 - Eclipse
- Fazit
- Quellen

Make-Programm

Projekt: main.cpp, square.cpp, square.h
Ziel: myprogram

Abhängigkeiten:



Es gibt viele Programmierertools, die die Arbeit des Programmierers erleichtern. Ein der wichtigsten ist make-Programm. Es hilft in Programmen aus mehreren Quelltextdateien. Wenn man an einer davon etwas ändert, muss man dann alle anderen neu kompilieren? Nein, muss man nicht. "Make" kompiliert nur die geänderte Datei.

Beispiel: wir haben ein Projekt mit main-Funktion main.cpp, Definition der Funktion square.cpp und ihrem Prototyp square.h.

Um myprogram zu erstellen, braucht man main.o und square.o.
Um main.o zu erstellen, braucht man main.cpp und square.h.
Um square.o zu erstellen, braucht man square.cpp und square.h.

Also, sehen wir die Abhängigkeitsliste:
Das Ziel - myprogram, seine Abhängigkeiten - main.o und square.o, usw.

Das alles sollten wir nun als Makefile-Datei zusammenfassen.

Wichtige Regeln

- eine Regel, Definition oder Befehl – pro Zeile
- `<Ziel>:_<Abhängigkeit>`
- `TAB $ <Befehl>`
- `#` Kommentar

Wichtig sind einige Regeln:

- 1) Pro Zeile steht immer nur eine Regel, Definition oder Befehl
- 2) Das Ziel steht vor dem Doppelpunkt, dann kommt mindestens ein Leerzeichen und dann die Abhängigkeit
- 3) Man sollte angeben, durch welchen Befehl das Ziel aus der Quelle erzeugt werden kann. Zeilen mit ausführbaren Befehlen muss man mit einem Tabulatorzeichen einrücken
- 4) `"#"` Doppelkreuz – für Kommentare

Makefile

```
#makefile-Datei für myprogram
#main.o square.o -> myprogram
myprogram: main.o square.o
    $ g++ main.o square.o -o myprogram
#main.cpp square.h -> main.o
main.o: main.cpp square.h
    $ g++ main.cpp -c
#sqr.cpp square.h -> square.o
sqr.o: square.cpp square.h
    $ g++ square.cpp -c
#lösche alle Objektdateien ($ make clean)
clean:
    $ rm -f *.o
```

Hier ist die fertige Makefile-Datei.

Um die Makefile-Datei zu verwenden, muss man den Befehl "make" eingeben. Das Ergebnis ist die ausführbare Datei "myprogram". Wenn wir jetzt etwas an einer Datei ändern, müssen wir nicht jede einzelne Datei neu kompilieren, es ist genug, den Befehl "make" eingeben, um die neue Version des Programms auszuführen.

Debugger (Fehlersuche)

"bug" - Laufzeit-Fehler

- gdb (konsolenbasiert)
- ddd (graphisch)

Hinzufügen von Info für Debugger

```
$ g++ bug_program.cpp -g -o bug_program
```

Aufruf des Debuggers

```
$ ddd bug_program
```

Nächstes Tool ist Debugger.

Wahrscheinlich habt ihr auch die Erfahrung gemacht, dass eure Programme beim ersten Mal oft nicht funktionieren. Tatsächlich ist es selten, ein nicht triviales C/C++-Programm zu schreiben, das beim ersten Mal, als man versucht, es auszuführen, keine Fehler enthielt.

Es gibt zwei Arten von Fehlern:

- Compilerzeit-Fehler, das sind Fehler, die der Compiler findet, sie sind relativ leicht zu beheben und
- Laufzeit-Fehler, diese Fehler kann der Compiler nicht finden, er kompiliert das Programm, es läuft aber leider nicht wie der Programmierer es geplant hat. Solche Fehler nennt man auch "Bugs" und sie sind schwieriger zu finden...

Natürlich kann man aufhören zu programmieren. Es gibt aber eine Alternative: man kann die Fehler mit dem Debugger finden und beheben.

Unter anderen stehen unter Linux zwei beliebte Debugger: "gdb" und "ddd" zur Verfügung.

"gdb" ist ein konsolenbasierter Debugger und "ddd" - sein graphischer Frontend.

"ddd"-Optionen

- View -> Execution Window - Konsolenfenster
 - Break - einen Haltepunkt setzen
 - Run - das Programm ausführen
 - Next - nächste Anweisung
- Step - in eine Funktion hineinspringen

"ddd"-Optionen:

"Break" - Haltepunkte (*engl.* breakpoints):

Der Debugger hält das Programm an, wenn die Ausführung auf einen Haltepunkt stößt. Also kann man nur einen Teil des Programms ausführen, um festzustellen, wo genau das Problem auftritt.

"Next" - Nächster Teil (Einzelschrittausführung):

Mit "Next" kann man das Programm zeilenweise ausführen, um jeden einzelnen Ausgabebefehl zu kontrollieren. Der Punkt der Ausführung wird nur zum nächsten ausführbaren Befehl verschoben. Dabei werden z.B. Deklarationen übersprungen, weil eine Deklaration keine Anweisung ist und nicht ausgeführt wird. Eine Deklaration reserviert einfach Speicherplatz für eine Variable.

"Step" - Überspringen (*engl.* stepping over):

Wenn das Programm innerhalb einer Funktion abstürzt, enthält die Funktion entweder einen Fehler oder die Argumente, die an die Funktion übergeben wurden, sind nicht korrekt. "Next" behandelt einen Funktionsaufruf wie eine einzelne Anweisung. Eine Funktion besteht jedoch selbst aus einer Reihe von C/C++-Befehlen. Deswegen muß man in die Funktion hineinspringen und jeden Befehl innerhalb der Funktion ausführen, um besser zu sehen, was passiert. Diese Funktionalität wird durch die "Step"-Anweisung des Debuggers zur Verfügung gestellt.

Gliederung

- ✓ Vorbereitung
- ✓ Kompilieren/Ausführen
- ✓ Tools
 - make-Programm
 - ddd-Debugger
- ✓ Entwicklungsumgebungen
 - KDevelop
 - Eclipse
- Fazit
- Quellen

Entwicklungsumgebungen

(IDEs - Integrated Development Environments)

Merkmale:

- 4 in 1
 - Editor -> **Syntaxeinfärbung**
 - erzeugte Programme -> sofort gestartet
 - makefiles -> automatisch erzeugt
- Codestellen mit Fehlern -> automatisch angezeigt
 - "Breaks" und "Steps" -> im Editor aktiviert
- alle Dateien eines Projekts -> in einem Dialog
 - Versionsverwaltung...

Unter Linux ist ja schön, dass man alles so automatisieren kann wie man es will, z.B. von der Kommandozeile aus kompilieren usw. Außerdem, wenn man das "von Hand" macht, lernt man besser, was so gemacht werden muss. Es kann aber trotzdem für einige Entwickler etwas mühsam sein, immer mit den Kommandozeilentools zu arbeiten. Bequemer und produktiver wird die Programmierung mit integrierten Entwicklungsumgebungen - Integrated Development Environments oder kurz IDEs.

Diese verfügen alle über folgende Merkmale:

- 4 in 1 ("four in one"): Editor, Compiler, Make-Programm und Debugger sind unter ein einheitliches User Interface gepackt
- Der Editor hat Syntaxeinfärbung
- Die erzeugten Programme können sofort aus der IDE gestartet werden
- Makefiles werden automatisch erzeugt
- Codestellen, an denen der Compiler Fehler findet, werden automatisch angezeigt
- Bei der Fehlersuche mit dem Debugger werden die gerade durchlaufenen Quelltexte im Editor angezeigt; Haltepunkte und Einzelschritte können dort aktiviert werden.
- Die Dateien, die zu einem Projekt gehören, werden über einen Dialog festgelegt.
- Die Versionsverwaltung ist in die IDE eingebunden, so dass die Dateien über den Editor aus- und eingecheckt werden können.

Ich will euch heute zwei IDEs vorstellen, die alle diese Kriterien erfüllen (und noch mehr): das sind KDevelop und Eclipse.

KDevelop

Vorteile:

- Auto-Vervollständigung
- Assistenten für Projekte, Klassen, Dateien
- Doxygen, CVS
- C, C++, Java, Fortran, Pascal, Perl, Python...
- externe Programme in Menüs
- Plugins
- **Qt-Designer**
- große Projekte...

Nachteile:

- nicht einfach für Anfänger

KDevelop ist die integrierte Entwicklungsumgebung des KDE Projekts.

KDevelop bietet:

- Auto-Vervollständigung (z.B., wenn eine Klasse einige Methoden besitzt, kann man beim Code-Eintippen aus denen auswählen);
- Assistenten für das Erstellen neuer Projekte, Klassen oder einzelner Dateien. Sie helfen beim Erstellen: was will man erstellen? "Simple Hello World Program" oder "Simple KDE Application"? Wie heißt der Entwickler? Wie lautet seine/ihre e-mail Adresse? Welche Lizenz möchte er/sie für sein/ihr Program? usw;
- Doxygen - das Dokumentationssystem, damit kann man fundierte Documentation für API-Funktionen (Application and Programming Interface) erzeugen;
- CVS - Versionskontrollsystem: es bewahrt den ganzen Quellcode in einem Platz, der "Repository" (Lager) heißt, so, dass jeder den Code eines bestimmten Zeitpunktes sehen kann, sehr praktisch, weil der Code nicht blockiert sein muss, was die parallele Arbeit von Entwicklern ermöglicht;
- eine Vielzahl von Programmiersprachen (C, C++, Java, Fortran, Pascal, Perl, Python, etc.) - man kann mehrsprachige Applikationen entwickeln;
- läßt sich durch Plugins erweitern;
- externe Programme lassen sich in die Menüs einbinden;
- Qt-Designer. Qt ist eine Klassenbibliothek für die Entwicklung von grafischen Benutzeroberflächen (Die Abkürzung Qt wird offiziell wie das englische Wort "cute" [kju:t] ausgesprochen. Dieses Wort soll die Ansicht der Entwickler ausdrücken, dass der Quelltext süß, hübsch aber auch pffiffig sein sollte).

KDevelop eignet sich durch seinen Funktionsumfang und seine Erweiterbarkeit für große Projekte, macht es aber Anfängern schwer einen Zugang zum gesamten Funktionsumfang der Entwicklungsumgebung zu finden.

KDevelop

Installation:

✓ **Ubuntu:**

Hinzufügen/Entfernen KDevelop

✓ **Pakete:**

<automake> (>1.6)

<autoconf> (>2.5)

<build-essential>

<kdevelop-doc>

Installation: Unter Ubuntu genügt es, mit Hinzufügen/Entfernen KDevelop zu installieren. Nach der Installation kann es sich herausstellen, dass wir auf die Option "Run automake & friends and configure" nicht zugreifen können. Dies liegt daran, dass KDevelop bei Automake eine Version höher als 1.6, bei Autoconf höher als 2.5 benötigt, um richtig funktionieren zu können. Dieses Problem kann man allerdings einfach beheben, indem man entsprechende Pakete nachinstalliert. Wenn das Paket <build-essential> installiert ist, kann man sich dann über automake und autoconf keine Sorgen machen. Was machen diese Programme? automake erstellt automatisch makefiles und autoconf braucht man, um die Generierung automatisch an die lokale Hard- und Softwareumgebung anzupassen. Es empfiehlt sich auch die Dokumentation zu installieren, da sie uns beim Schreiben einer Applikation hilft - selbst wenn die Applikation einfach ist.

KDevelop Oberfläche

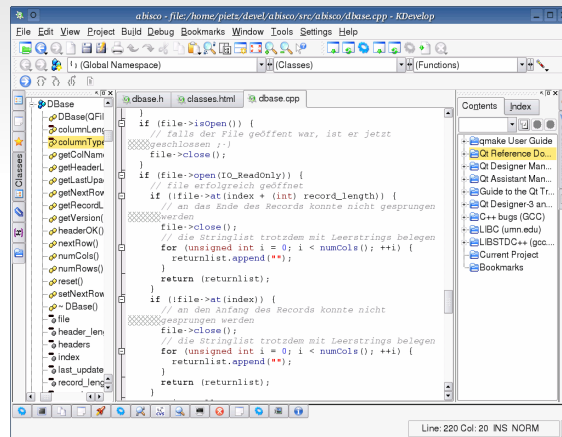


Abb.4 Oberfläche von KDevelop

Das ist die Oberfläche von KDevelop. Auf der linken Seite kann man zwischen verschiedenen Projektdateien navigieren. Im Hauptfenster ist der Editor. Auf der rechten Seite - der Hilfebrower. Mittels Registerkarten kann zwischen den Ansichten gewechselt werden. Am unteren Teil des Fensters kann man die Ausgabe der verschiedenen externen Programmen wie make-Programm, Debugger und sonstige I/O Operationen ablesen.

KDevelop

Hello World!

Simple KDE Application

Projekt erzeugen:

Project->New Project->C++ ->KDE->Simple KDE Application

Projekt übersetzen:

Build->Run automake & friends

Build->Run Configure

Build->Build Project

Build->Install (as root user)

Starten:

Build->Execute Program



Abb.5 Die einfache KDE Anwendung

Info: <http://women.kde.org/articles/tutorials/kdevelop3/de/index.html>

Eclipse

eine universelle Plattform

- IBM-Geschenk
- Konsortium: IBM, Borland, Intel, Nokia, HP, SAP AG
- verschiedene Programmierparadigmen und Technologien
- Plugin-Architektur

Weitere Merkmale:

Resourcenverwaltung, Aufgabenverwaltung
Problembehandlung, Hilfesystem, Assistenten
Codevervollständigung, Refactoring, lokale Historie
intelligente Korrekturvorschläge, RCP...

Die nächste IDE ist eine der neuesten und viel beachteten IDEs unter Linux - Eclipse. Ihre erste Version wurde im Jahr 2001 freigegeben. Angekündigt wurde Eclipse von IBM als ein 40-Millionen-Dollar-Geschenk an die Open-Source-Gemeinde. Also steht jetzt der gesamte Code von Eclipse als Quellcode zur Verfügung und kann leicht eingesehen werden. Das Eclipse-Konsortium besteht aus über 150 Mitglieder: IBM, Borland, Intel, Nokia, Hewlett Packard, SAP AG (Microsoft ist nicht dabei).

Was genau ist Eclipse?

Die Idee von Eclipse besteht darin, dass heutige Entwicklungsprojekte aus verschiedenen Programmierparadigmen und Technologien wie z.B. C++, Java, Python, HTML oder XML auskommen. Deswegen möchte Eclipse eine universelle Plattform sein, die durch Plugins viele andere Werkzeuge einbinden kann. Weitere Merkmale sind: Resourcenverwaltung, Aufgabenverwaltung, Problembehandlung, Hilfesystem, verschiedene Assistenten, Codevervollständigung, Refactoring (Funktion für die Umformung von Programmen); das Führen einer lokalen Historie, die eine Rückkehr zu früheren Codeversionen erlaubt; intelligente Korrekturvorschläge; Rich Client Platform (RCP) - generische Plattform für eine weite Klasse von Anwendungen... Und das ist nur die Spitze des Eisbergs. Durch sein sehr offenes Plattformkonzept ist Eclipse eine sehr interessante IDE und eignet sich für kleine bis mittelgroße Projekte.

Eclipse

Installation:

- ✓ Java Runtime Environment
Paket: <ubuntu-restricted-extras>
Info: <http://wiki.ubuntuusers.de/Java>
- ✓ Eclipse IDE for C/C++ Developers
<http://www.eclipse.org/downloads/>

Die Installation von Eclipse ist nicht schwierig. Unter Ubuntu braucht man Java Runtime Environment. Dieses sorgt dafür, dass in Java geschriebene Programme auf jeder Plattform ausgeführt werden können. Normalerweise reicht es das Paket <ubuntu-restricted-extras> zu installieren. Info findet man hier: <http://wiki.ubuntuusers.de/Java>. Und dann sollte man das Ganze von dieser Seite herunterzuladen:

Eclipse IDE for C/C++ Developers
<http://www.eclipse.org/downloads/>

Das kann unter /opt ausgepackt werden und ist sofort lauffähig.

Eclipse Oberfläche

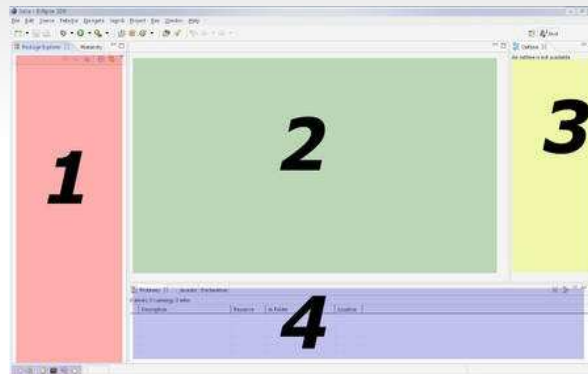


Abb.6: Eclipse Workbench

Das ist das Hauptfenster von Eclipse, das man "Workbench" nennt. Der Workbench ist die Oberfläche von der man alles regeln kann. Wie der Workbench aussieht, kann man selber festlegen. Standardmässig gibt es folgende Aufteilung:

- Im ersten Bereich findet man einen Überblick seiner/ihrer Projekte. Man kann zwischen den Projekten navigieren, Dateien öffnen, neue Dateien oder Ordner in den Projekten erstellen, etc.
- Im zweiten Bereich ist der Editor. Wenn man den Sourcecode schreibt, wird er farbig hervorgehoben. Wenn Eclipse Fehler erkennt, markiert er sie und schlägt in machen Fällen Lösungen vor.
- Im dritten Bereich findet man die Outline: Wissenswertes über die im Moment geöffnete Datei. z.B. bei einer Klasse sieht man hier die Attribute und Methoden. Durch einen Klick auf die Methode kommt man an die Stelle im Quelltext wo sie implementiert ist.
- Im vierten Bereich werden Ausgabe und Fehler aufgezeigt.

Die einzelnen kleinen Fenster, die Funktionen bereitstellen, werden als Views bezeichnet. Durch Anklicken sind sie aktiviert, durch Doppelklick kann man sie maximieren und minimieren.

Die Anordnung der momentanen Tools und den Umfang der Tools bezeichnet man als Perspektive.

Eclipse

Hello World C++ Project

Projekt erzeugen:

File -> New -> Project -> C++ Project -> Executable ->
Hello World C++ Project -> Finish

Projekt übersetzen:

Project -> Build Project

Starten:

Run -> Run as -> 1 Local C/C++ Application

Gliederung

- ✓ Vorbereitung
- ✓ Kompilieren/Ausführen
- ✓ Tools
 - make-Programm
 - ddd-Debugger
- ✓ Entwicklungsumgebungen
 - KDevelop
 - Eclipse
- ✓ Fazit
 - Quellen

Fazit

Was macht Linux fürs Programmieren so attraktiv?

- **komplette Programmier-Umgebung**
Compiler, Interpreter, fast alle gängigen Programmiersprachen
Bibliotheken, Debugging- und Automationstools, Editoren
- **programmiererfreundliche interne Struktur**
nicht künstlich verschleiertes System:
"Ich werde dafür sorgen, dass du verstehst wie ich funktioniere."

Zusammenfassung.

Gerade eben haben wir gesehen wie man unter Linux programmieren kann. Am Anfang kann das alles kompliziert vorkommen aber wenn man einige Programme geschrieben hat, macht man viele Sachen automatisch und arbeitet schnell und effizient. Mein Ziel war zu zeigen was Linux fürs Programmieren so attraktiv macht. Ich hoffe, dass ihr jetzt auf diese Frage die Antwort gefunden habt: um in die Programmierung einzusteigen, ist Linux ganz besonders gut geeignet, denn:

1) Linux kommt mit einer kompletten Programmier-Umgebung: es gibt Compiler und Interpreter für fast alle gängigen Programmiersprachen, Bibliotheken, Debugging- und Automationstools, Editoren wie Sand am Meer. Linux macht die Entwicklung von Software so leicht wie möglich.

2) Linux ist 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."

Zum Schluss möchte ich sagen, dass das Ganze ein Fass ohne Boden ist... Ich konnte hier nur eine sehr kleine Einführung geben. Bei Interesse und Bedarf kann man viel mehr über das Thema erfahren. Ich hoffe, es hat euch trotzdem was gebracht und ihr etwas Spaß bei der Sache hattet.

Hier sind meine Quellen. Vielen Dank für die Aufmerksamkeit und "Happy Hacking!"

Gliederung

- ✓ Vorbereitung
- ✓ Kompilieren/Ausführen
- ✓ Tools
 - make-Programm
 - ddd-Debugger
- ✓ Entwicklungsumgebungen
 - KDevelop
 - Eclipse
- ✓ Fazit
- ✓ Quellen

Quellen

- Programmieren unter Linux:
<http://www.cpp-entwicklung.de>
<http://www.galileocomputing.de/openbook/linux/>
- Kompilieren/Linken:
<http://de.wikipedia.org/wiki/Compiler>
http://de.wikipedia.org/wiki/Binder_%28Computerprogramm%29
- KDevelop:
<http://www.kdevelop.org>
<http://women.kde.org/articles/tutorials/kdevelop3/de/index.html>
<http://www.pro-linux.de/berichte/appfokus/kdevelop/kdevelop.htm>
- Eclipse Workshop und Tutorial:
<http://www.admin-wissen.de/eigene-tutorials/programmierung/>
- Onlinekurse für C und C++ (en/de)
<http://www.c-plusplus.de/cms/index.php>

Meine Empfehlung für den Anfang wäre das Buch "cpp-entwicklung", das man entweder bestellen oder online lesen kann. Das Besondere an diesem Buch ist, dass man hier nicht nur etwas über die Programmiersprache C++, sondern auch viel über die Werkzeuge (Compiler, Make-Programm, Debugger, etc.) lernt, die für die Entwicklung unter Linux zur Verfügung stehen. Der Autor zeigt, wie man unter Linux ähnlich bequem wie unter Windows entwickeln kann - ein wenig anders aber dafür bekommt man fast alles, was man braucht, so gut wie umsonst. Es geht hier aber nicht um das Geld, sondern um die Freiheit und Zusammenarbeit.

Danke für die Aufmerksamkeit
und viel Erfolg
beim C/C++-Programmieren
unter Linux!