Fachhochschule Wiesbaden
Fachbereich Informatik
Prof. Dr. Reinhold Kröger

Übersicht zur Programmentwicklung unter UNIX

Auf den folgenden Seiten werden die wesentlichen Aspekte und üblichen Techniken zur C/C++-Programmentwicklung unter UNIX zusammengestellt. Ziel dieser Zusammenstellung ist es, Studierenden eine für praktische Zwecke brauchbare Einführung in diese Thematik mit sehr beschränktem Umfang bereitstellen zu können. Nicht Vollständigkeit und Tiefe der Darstellung werden angestrebt, sondern eine Reduktion der Einarbeitungszeit, etwa im Rahmen praktischer Übungen zu Betriebssysteme oder zur Durchführung eines Projekts im Bereich Verteilte Systeme. Insofern ersetzt diese Zusammenstellung kein Lehrbuch oder man pages, sondern ist eher als Ergänzung hierzu anzusehen, um sicherzustellen, daß Studierende schnell erste Erfolgserlebnisse sehen und vielleicht darauf begründet, Interesse an einer eigenverantwortlichen Vertiefung gewinnen. Zugrunde liegt die Beobachtung, daß viele Studierende sich in der UNIX-Umgebung anfangs schwer tun. Auch wenn UNIX in seinen Anfängen gerade zur besonderen Unterstützung der Programmentwicklung entstanden ist, sind viele Studierende mit PC-basierten, integrierten Entwicklungsumgebungen "aufgewachsen". Sie vermissen die graphische Oberfläche, einen komfortablen Editor, "App Wizards", usw. Auf der anderen Seite ist auch zu beobachten, daß die Kenntnis der elementaren Zusammenhänge aufgrund der komfortablen Werkzeuge z.T. zu wünschen übrig läßt.

Die folgenden Lehrbücher und andere Quellen, die an der FH Wiesbaden verfügbar sind, sollten ebenfalls zum Thema herangezogen werden:

[1] Gulbins, J.; Obermayr, K.:*UNIX System V.4 - Begriffe, Konzepte, Kommandos, Schnittstellen*
4. überarbeitete Auflage, Springer-Verlag, 1995
Kap. 9, S. 549-602

[2] RRZN: *UNIX, Eine Einführung*
RRZN Uni Hannover, 6. Auflage, 1992
Kap. 8, S. 135-151

[3] Stevens., W.R.: *Advanced Programming in the UNIX Environment*
Addison-Wesley, 1992

[4] UNIX Manual Pages.

Besonders sei auf die preiswerte UNIX-Broschüre des RRZN verwiesen, die zum Selbstkostenpreis vom Fachbereich zur Verfügung gestellt wurde und wird.

Die wesentlichen Hilfsmittel für die Programmentwicklung sind generell:

- Editoren zur Erstellung und Bearbeitung von Quellprogrammen,
- Präprozessoren zur textuellen Bearbeitung von Quellprogrammen,
- Compiler oder Interpreter zu deren Übersetzung bzw. unmittelbaren Ausführung,
- Binder (linker) für das Zusammenbinden von Moduln zu größeren Einheiten und das letzliche Erstellen von ausführfähigen Programmen,
- Werkzeuge für die Analyse von Quellprogrammen,
- Testwerkzeuge (Debugger),
- Werkzeuge für die Bewertung der Leistungsfähigkeit von Programmen in Ausführung,
- Werkzeuge für die Verwaltung von Programmsystemen während ihrer gesamten Lebensdauer (Projektmanagement, Versionsverwaltung, Bibliotheksverwaltung).

UNIX enthält für alle diese Aufgaben grundlegende Werkzeuge, wenn auch in den letzten Jahren Werkzeuge zur Programmentwicklung zunehmend als eigenständige, optionale Produkte vertrieben werden. Daneben sind viele Werkzeuge hoher Qualität frei verfügbar, wie etwa die sehr anerkannten GNU-Werkzeuge. In Tabelle 1 sind übliche Werkzeuge zusammengestellt.

Editoren	vi, ed, sed, emacs, joe, ...
Präprozessor	cpp
Compiler	cc, gcc, g++, ... für alle Sprachen
Binder	ld
Analyse	lint
Test	adb, xdb, dbx, gdb, ...
Leistungsbewertung	time, prof, ...
Management	make, ar, sccs, rcs, c2man, ...

Tabelle 1: Tabellarische Übersicht der Werkzeuge.

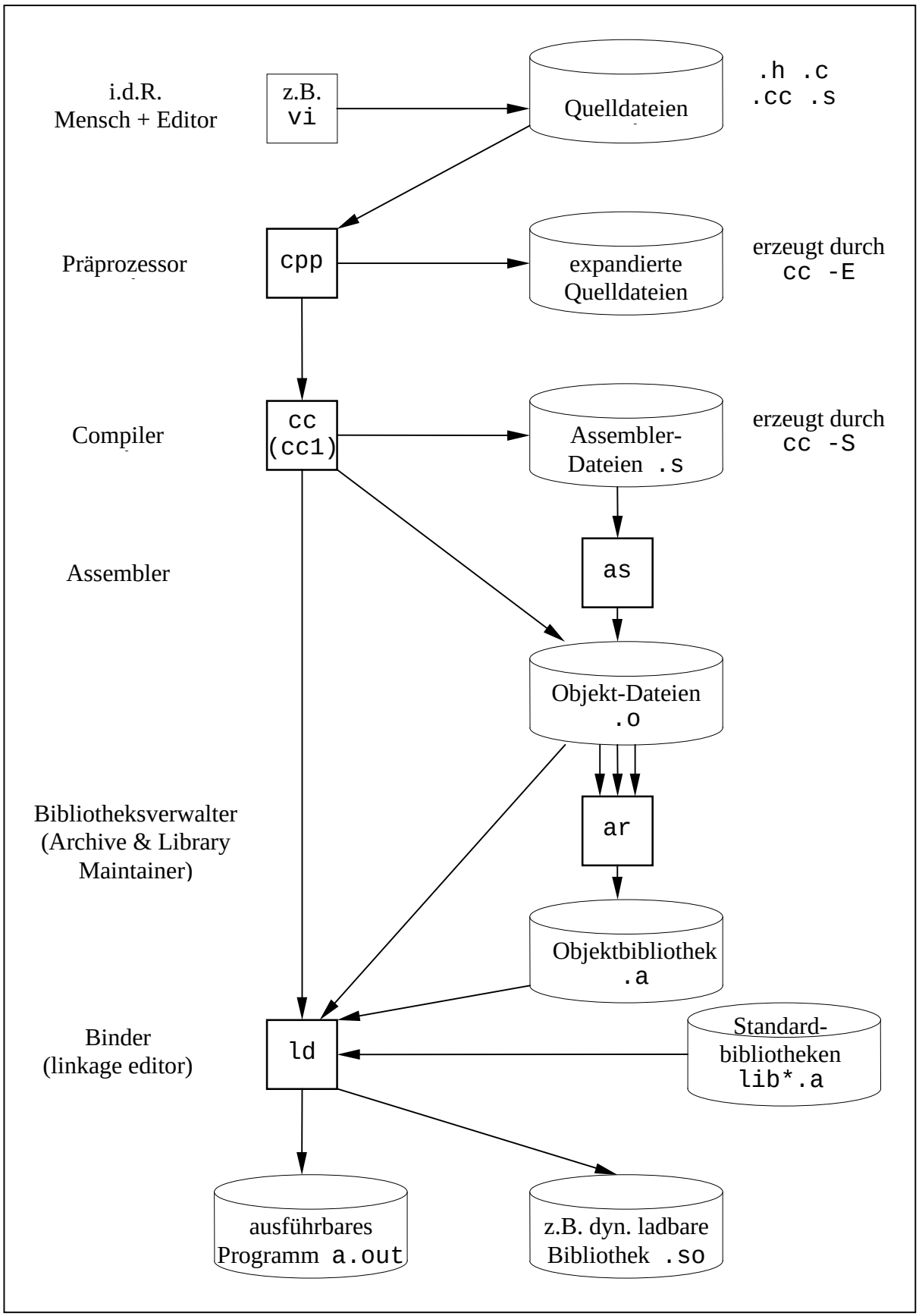

Bild 1: Werkzeuge zur Programmerzeugung im Zusammenhang.

Anhand von Bild 1 soll das Zusammenwirken der wesentlichen Werkzeuge bei der C/C++-Programmerzeugung im Zusammenhang aufgezeigt werden. Auf alle anderen Aspekte wird hier nicht eingegangen.

Die Erstellung eines Quellprogramms mit Hilfe eines Editors wird hier nicht weiter betrachtet. Für UNIX-Systeme sind Compiler für nahezu alle gängigen Programmiersprachen verfügbar. Im weiteren soll die C/C++-Programmentwicklung im Vordergrund stehen.

Ein Programmsystem für die Lösung eines etwas komplexeren Problems, als ein Studierender es üblicherweise in den ersten Semestern bearbeitet, wird in Teilsysteme gegliedert, denen jeweils ein Teilbaum des gesamten Quellcodes (Source-Trees) des Programmsystems entspricht (vgl. Softwaretechnik). Ein Teilsystem kann sich wiederum in Untersysteme gliedern, so daß ein mehrstufiger Baum zustande kommt. Auf der untersten Ebene wird für die Lösung eines Teilsystems eine Menge von Quellcode-Moduln vorgesehen, die i.d.R. jeweils in einer Datei abgelegt sind. Zur Isolierung von Konstantendefinitionen, Makros, Datenstruktur-Definitionen und Prototypen, die häufig in mehreren Programmdateien verwendet werden müssen, werden sogenannte Include-Dateien oder Header-Dateien verwendet, deren Namen üblicherweise mit dem Suffix `.h` enden. Die Endungen der Quelldateien kennzeichnen die verwendete Programmiersprache: C-Dateien benutzen `.c`, C++-Dateien verwenden `.cc`, `.cpp` oder `.C`, Pascal benutzt `.p` usw.

C-Compiler (Übersetzer), wie etwa `cc` oder der GNU C-Compiler `gcc` (üblicherweise in `/usr/local/bin`), sind mehrstufige Compiler-Systeme und durchlaufen im Default-Fall alle Stufen bis zur Erzeugung eines ausführbaren Programms. Die Ausführung der verschiedenen verketteten Werkzeuge wird mit der Option `-v` (verbose) angezeigt (Beispiel: `gcc -v gesamt.c`). Die Compiler-Systeme `cc` und `gcc` besitzen zahlreiche weitere Optionen, vgl. `man cc, man gcc`. Die Verarbeitungsstufen werden im folgenden skizziert.

Zunächst wird ein Präprozessor zur textuellen Manipulation der Quelldateien vor der eigentlichen Compilierung eingesetzt. Dieser hat den Namen `cpp` und wird üblicherweise im Verzeichnis `/lib` bzw. `/usr/lib` gehalten. Er wird auch von vielen anderen Compilern zur Vorverarbeitung genutzt und kann auch oft mit Vorteil vom normalen Anwender für Textmanipulationsaufgaben eingesetzt werden. Der Präprozessor wird beim Aufruf des Compilers unsichtbar für den Benutzer ausgeführt. Die Funktionen des Präprozessors werden in den Quelldateien durch sogenannte Präprozessor-Direktiven festgelegt. Die häufigsten Nutzungen geschehen im Rahmen des Einschleusens von Include-Dateien (z.B. zur Deklaration

von Strukturen oder Funktionen, s.o.) durch `#include`-Anweisungen, zur Definition von Konstanten mittels `#define`-Anweisungen, zur Expansion von Makroaufrufen sowie zur bedingten Übersetzung. Durch z.B. `#ifdef` *name*-`#else`-`#endif`-Konstrukte werden hierbei unterschiedliche Teile des Quellprogramms, abhängig von der Existenz des Namens *name* zum Zeitpunkt des Präprozessorlaufs, Teil des zu erzeugenden Programms. Beispielsweise kann dies zur bedingten Generierung von Debug-Code oder von Architektur- oder Betriebssystem-abhängigen Code-Varianten eingesetzt werden. Bei Bedarf kann der durch den Präprozessor erzeugte Text eines Quellprogramms durch eine Compiler-Option in eine Datei ausgegeben werden (z.B. `cc -E`). Daneben bestehen viele andere Optionen zur Steuerung des Präprozessors (vgl. `man cpp`, [1] S. 555 ff).

An die Präprozessor-Verarbeitung schließt sich der eigentliche Übersetzungsvorgang durch den Compiler (`cc`-Stufe `cc1`) an. Die expandierte Quelldatei wird in eine sogenannte Objektdatei übersetzt. Objektdateien besitzen das Suffix `.o`. Bei Bedarf kann der durch den Compiler erzeugte Assembler-Code in einer Datei ausgegeben werden (Option `-S` erzeugt <*name*>`.s`) (Hier können Sie sich mal ansehen, was der Compiler so aus Ihren C-Konstrukten generiert!). Diese Datei kann dann mit dem Assembler `as` händisch in die gleiche Objektdatei assembliert werden. Oft bedient sich auch der Compiler zur Maschinencode-Generierung selbst des Assemblers. UNIX-basierte Compiler erlauben von je her die getrennte Übersetzung einzelner Moduln (bzw. Dateien). Aus einer einzelnen Quelldatei <*name*>`.c` entsteht unter Verwendung der `cc`-Option `-c` die zugehörige Objektdatei <*name*>`.o`. Dies bringt enorme Vorteile bei größeren Programmsystemen, da nicht alle Programmkomponenten neu übersetzt werden müssen, wenn z.B. nur in einer einzigen Quellcode-Datei eine Modifikation vorgenommen wurde. (Die getrennte Übersetzbarkeit wird z.B. vom `make`-Werkzeug zur automatischen Generierung eines Programmsystems ausgenutzt).

Objektdateien haben ein wohldefiniertes, relativ komplexes Format, das hier nur angedeutet werden kann. In der Vergangenheit wurden verschiedene solcher Objekt-Formate in UNIX-Systemen verwendet. Das älteste ist das sogenannte a.out-Format; COFF (Common Object File Format) wurde für UNIX System V eingeführt; heute wird auf vielen Systemen das jüngste, sogenannte ELF-Format (Extensible Linking Format) verwendet. Unabhängig von dem tatsächlichen Format nennt man häufig jedes Objektdatei-Format auch a.out-Format, da die default-Ausgabedatei nach einem Bindevorgang den Namen a.out besitzt. Informationen zum Objectfile-Format kann man z.B. mit `man a.out` erhalten.

Ausgehend von einem Header ist eine Objektdatei in Abschnitte unterteilt. Ein Abschnitt enthält neben dem eigentlichen generierten Maschinencode (program "text") und den durch diesen Modul definierten Daten (program "data", wobei initialisierte und nicht initialisierte Daten unterschieden werden) Symboltabellen, Relokations-Informationen, Zeilennummern-zuordnungen zum Quellprogramm, etc.. Die Symboltabellen enthalten Verzeichnisse der im Programm verwendeten Symbole. Insbesondere existiert konzeptionell eine External Symbol Table, die die durch den Compilationslauf nicht auflösbaren, externen Symbole sowie die von dem Modul aus exportierten Symbole (die von anderen Moduln referiert werden können) beschreibt. Für jedes Symbol werden über die verschiedenen Tabellen insgesamt zahlreiche Attribute festgehalten, insbesondere seine Lage im Modul, seine Speicherklasse, usw.. Die Tabelleninformation wird insbesondere auch von Debuggern verwendet, die auf diese Weise die Variablen, Prozeduren usw. im Programm wiederfinden. Einblick in diese Information einer Objektdatei kann man mittels des Dienstprogrammes `nm` (auf einigen Systemen auch mittels `objdump`) erhalten.

Der Binder (linkage editor, `ld`) erzeugt aus einer Menge von Objektdateien eine einzige neue Objektdatei. Diese Datei hat prinzipiell die Eigenschaften wie jede andere Objektdatei, im Normalfall soll sie jedoch als ausführbare Objektdatei (executable object file) verwendet werden. Für diesen Fall wird ein zusätzliches, systemdefiniertes Objektmodul (Loader-Modul) hinzugebunden. Während der Ausführung des Binders werden die "Text" (Code)-Teile sowie die Datenteile der zu bindenden Module zusammengefaßt sowie, und das ist die eigentliche Aufgabe des Binders, die externen Referenzen zwischen den Moduln der Menge soweit wie möglich aufgelöst, so daß für die entstehende Objektdatei wiederum eine Symboltabelle übrigbleibt, die ausschließlich (in äußeren Moduln definierte) externe Symbole und exportierte Symbole dieser Objektdatei enthält. Alle (internen) symbolischen Namensbezüge zwischen den Moduln der Menge werden durch relative Adressen bezogen auf die Text- und Datenteile der neuen Objektdatei ersetzt. Die Startadresse des ersten angegebenen Moduls definiert per default die Startadresse des gebundenen Moduls.

Darüberhinaus kann der Binder sogenannte Objektbibliotheken (d.h. Sammlungen von Objektdateien, die i.d.R. in einem funktionalen Zusammenhang stehen) nach Symboldefinitionen durchsuchen und die ensprechenden Objektmoduln in die zu erzeugende Objektdatei einbeziehen. Namen von Objektbibliotheken besitzen den Suffix `.a` (archive). Neben vom Betriebssystem bereitgestellten, zu Compilern oder Standardanwendungen gehörenden

Standardbibliotheken (z.B. `libc.a, libm.a, libX11.a`), die sich in wohldefinierten Verzeichnissen im Dateisystem befinden (`/lib`, `/usr/lib`, `/usr/local/lib`), können eigene Bibliotheken verwendet werden. Zu beachten ist, daß Bibliotheken in der angegebenen Reihenfolge und nur einmal durchsucht werden, so daß die Angabe der Reihenfolge wesentlich ist.

Heutige Binder unter UNIX erlauben i.d.R. auch die Erzeugung und Verwendung von sogenannten "Shared Libraries", das sind Bibliotheken, die nicht zum Zeitpunkt des `ld`-Laufs (statisch) eingebunden werden, sondern erst zum Ladezeitpunkt des Programms (oder im Falle des voll dynamischen Bindens zur Laufzeit des Programms) und dann von mehreren Programmen zur Laufzeit gemeinsam genutzt werden können ("sharing"). Die Namen von Shared Libraries besitzen üblicherweise den Suffix `.so` (shared object). Shared Libraries erlauben kleinere "ausführbare" Programme (da die Bibliotheksmodule noch nicht Bestandteil der ausführbaren Objektdatei geworden sind) und eine sinnvollere Speichernutzung zur Laufzeit (durch die gemeinsame Nutzung durch mehrere Programme). Da aber in diesem Fall beim Laden eines Programms noch externe Referenzen zu solchen Bibliotheken aufgelöst werden müssen, erfordert dies einen besonders befähigten Lader (Linking Loader, z.B. `ld.so`).

Zur Erstellung und Verwaltung von Bibliotheken existiert ein Bibliotheksverwalter namens `ar` (Archive and Library Maintainer). Er erlaubt die Erstellung einer Bibliothek aus einer Menge von Objektdateien, das Hinzufügen, Ersetzen und Löschen einzelner Objektmodule sowie die Extraktion einer weiterverwendbaren Objektdatei aus einer Bibliothek (vgl. `man ar`).