Modulare Programmierung

Praktikum „C-Programmierung“

Eugen Betke, Nathanael Hübbe,
Michael Kuhn, Jakob Lüttgau, Jannek Squar

Wissenschaftliches Rechnen
Fachbereich Informatik
Universität Hamburg

2019-01-14

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Motivation

Kapselung zusammengehörigen Codes

- Aufteilung der Programmlogik
- Header definiert Schnittstelle
- Besserer Überblick
- Vereinfacht testen und debuggen
- Vereinfacht Code-Änderungen
- *Don't repeat yourself* $\rightarrow$ Wiederverwendung bestehenden (Fremd-)Codes

Beschreibung

- Inhalt:
  - Funktionsdeklarationen
  - Konstanten/Daten
  - Makros
  - (Funktionsbody)
- Inklusionsaufforderung
  - `#include <myheader.h>`
  - `#include "myheader.h"`
  - Übergabe neuen Suchpfades über `-I<pfad>`
- File-Endung `.h`

- Guard verhindert mehrfache Inklusion

Guard

```
#ifndef MYHEADER_H
#define MYHEADER_H
int foo = 42;
bar();
#endif
```

Verwendung

```c
#include <stdio.h>

int main()
{
    printf("Hello World");
    return 0;
}
```

Bedingte Inklusion

```
1 #ifdef _OPENMP
2     #include <omp.h>
3     int threadCount = omp_get_num_procs();
4 #else
5     int threadCount = 1;
6 #endif
```

C-Standard-Header

<assert.h>
<complex.h>
<ctype.h>
<errno.h>
<fenv.h>
<float.h>
<inttypes.h>
<iso646.h>
<limits.h>
<locale.h>
<math.h>
<setjmp.h>
<signal.h>
<stdalign.h>
<stdarg.h>
<stdatomic.h>
<stdbool.h>
<stddef.h>
<stdint.h>
<stdio.h>
<stdlib.h>
<stdnoreturn.h>
<string.h>
<tgmath.h>
<threads.h>
<time.h>
<uchar.h>
<wchar.h>
<wctype.h>

Weitere Infos: [cSt, ISO]

Header-Implementation

- Implementation vieler Standard-C-Funktionen in `libc`
  - Standardmäßig eingebunden bei gcc
- Implementation mancher Standard-Header in eigene Lib ausgelagert
  - Bsp: `math.h`
  - explizite Angabe der Lib
  - Fehlende Lib resultiert in Linkerfehler

- Container für kompilierten Code und Daten
  - → Keine Neu-Kompilierung von Abhängigkeiten notwendig
- Arten von Programm-Libraries:
  - Static
  - Shared

Static [HOWc]

- Archiv von Objektdateien
- Linker fügt Code in Binary ein
- Vereinfachte Portierung auf anderes System
- File-Endung `.a`
- Geringfügig schnellere Laufzeit als shared Library

Verwendung

```
$ gcc -c myLib.c
$ gcc -c main.c
$ ar rcs libmyLib.a myLib.o
$ gcc -o main.x main.o -L. -lmyLib
```

Shared [HOWb]

- Linker fügt Symbole ein
- Einbindung bei Programmstart über Loader
- Dynamische Einbindung zur Laufzeit möglich ([HOWa])
- Kleinere Binary, kürzere Build-Time
- Deutliche Vereinfachung von Code-Updates
- Namenskonvention: *lib*LibraryName.so(.*VERSION*)

Verwendung

```
$ gcc -c -fPIC myLib.c
$ gcc -shared -o libmyLib.so myLib.o
$ gcc -o main.x helloWorld.c
    ↪ -lmyLib -L.
```

Suchreihenfolge

Suche nach Library bzgl. einer Abhängigkeit:

1. `rpath`
2. `LD_LIBRARY_PATH`
3. `runpath`
4. `/etc/ld.so.conf`
5. `Standard-Systempfade`

Userdefinierte Suchpfade

RPath:

- Optionale Einträge in ELF-Sektion `.dynamic`
- Linker schreibt ELF-Einträge

`LD_LIBRARY_PATH`:

- `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<neuer Suchpfad>`
- `LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<neuer Suchpfad> ./app`
- Explizite Angabe einer Library im Suchpfad über `LD_PRELOAD=<library.so>`

Runpath:

- Auswertung von `runpath` nach `LD_LIBRARY_PATH`
  → Überschreibung potentieller Treffer in `runpath` durch User möglich
- Inkonsistentes Verhalten bei verschiedenen Distributionen möglich (vgl. [cka])

RPath/Runpath setzen

- RPath (relativ zu aktuellem Verzeichnis)
  - `$ gcc -o main.x main.o -lmyLib -L. -Wl,-rpath,.`
- RPath (relativ zu Binary)
  - `$ gcc -o main.x main.o -lmyLib -L. -Wl,-rpath,"\$ORIGIN"`
- Runpath
  - `$ gcc -o main.x main.o -lmyLib -L.`
    `↪ -Wl,-rpath,.,--enable-new-dtags`
- *Hinweis:* `LD_LIBRARY_PATH` sowie rpath/runpath mit `$ORIGIN` werden bei gesetztem setuid o.ä. ignoriert

Standard-Pfade (Fedora29))

Header:

```
1 /usr/lib/gcc/x86_64-redhat-linux/8/include
2  /usr/local/include
3  /usr/include
```

Libraries:

```
1 /usr/lib/gcc/x86_64-redhat-linux/8
2 /usr/lib
3 /lib
4 /usr/lib64
5 /lib64
```

Ausgabe der Standardpfade beim Kompilieren mit `gcc -v ...` bzw. `gcc -H ...`

Kommando-Sammlung

ldd:	*Rekursive* Anzeige aller benötigten shared Libraries
nm:	Auflistung aller Symbole eines Objekt-Files
objdump:	Auslesen von Informationen eines Objekt-Files
readelf:	Auslesen von Informationen eines ELF-Objektes
ldconfig:	Aktualisieren des System-Caches von Pfaden mit Suchpfaden bzgl. Libraries

The end is near

- Präprozessor inkludiert Header
- Linker löst Aufrufe externer Funktionen auf
- Übergabe eines Header-Suchpfades über `-I<pfad>`
- Übergabe eines Library-Suchpfades über `-L<pfad>`
- Einbindung einer externen Library über `-l<libname>`
- Binary muss Pfad zu shared Libs kennen (`rpath`, `LD_LIBRARY_PATH`)
- Mehr: `http://tldp.org/HOWTO/Program-Library-HOWTO/`

Quellen I

[cka] ckamm. Rpath and runpath. http://blog.qt.io/blog/2011/10/28/rpath-and-runpath/.

[cSt] C standard library header files. https://en.cppreference.com/w/c/header.

[HOWa] Program Library HOWTO. Shared libraries. http://tldp.org/HOWTO/Program-Library-HOWTO/dl-libraries.html.

[HOWb] Program Library HOWTO. Shared libraries. http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html.

[HOWc] Program Library HOWTO. Static libraries. http://tldp.org/HOWTO/Program-Library-HOWTO/static-libraries.html.

Quellen II

[ISO] ISO International Standard ISO/IEC. Programming languages — c [working draft]. http://www.open-std.org/jtc1/sc22/wg14/www/abq/c17_updated_proposed_fdis.pdf.