Compiling and Linking

Praktikum „C-Programmierung“

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

Wissenschaftliches Rechnen
Fachbereich Informatik
Universität Hamburg

2019-01-07

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Compiler Toolchain: Eine Übersicht

- `binutils`
- GCC: GNU Compiler Collection
  - C Library
  - Runtime
- Linux Kernel Headers

Neben GCC gibt es noch diverse andere Toolchains teilweise Open Source, teilweise proprietär. Beispiele: LLVM/Clang, Intel (C/C++, Fortran), Cray (C/C++, Fortran), IBM (C/C++), PGI, NVIDEA (CUDA LLVM), ARM (GCC or LLVM based), usw.

Weitere Informationen zu GCC vs. LLVM z.B. unter: `https://clang.llvm.org/comparison.html`

binutils

Als Teil der GNU Software project eine Sammlung von "binary tools". Insbesondere:

```
1 as – Assembler, Erzeugt Object–Dateien aus architektur–spezifischer Textform
2 ld – Linker, Verbindet Object–Dateien zu Shared–Library/Executable/Object–Datei
```

```
1  addr2line  – Converts addresses into filenames and line numbers.
2  ar         – A utility for creating, modifying and extracting from archives.
3  c++filt    – Filter to demangle encoded C++ symbols.
4  dlltool    – Creates files for building and using DLLs.
5  gold       – A new, faster, ELF only linker, still in beta test.
6  gprof      – Displays profiling information.
7  nlmconv    – Converts object code into an NLM.
8  nm         – Lists symbols from object files.
9  objcopy    – Copies and translates object files.
10 objdump    – Displays information from object files.
11 ranlib     – Generates an index to the contents of an archive.
12 readelf    – Displays information from any ELF format object file.
13 size       – Lists the section sizes of an object or archive file.
14 strings    – Lists printable strings from files.
15 strip      – Discards symbols.
```

Siehe auch: https://www.gnu.org/software/binutils/

GNU Compiler Collection

GCC ist die Compiler Collection mit sog. Frontends zu verschiedenen Sprachen:

- C, C++, Objective-C, Fortran, Ada, Go, and D, (sogar Java bis GCC7/2016)

Dazu bietet GCC sog. Backends für viele (70+) Architekturen/Plattformen.

Bestandteile von GCC:

```
1  cc1, cc1plus
2      Die eigentlichen Compiler, Erzeugen lediglich Assembly Code
3
4  gcc, g++
5      Interfaces zu den Compiler, aber auch integration von binutils, as und ld
6
7  Target libraries, libgcc, libstdc++, libfortran
8      Verschiedene Runtimes und haeufig gebrauchte Funktionen
9
10 Headerfiles fuer die Standard C++ Library
```

Siehe auch: `https://www.gnu.org/software/gcc/`

Linux Kernel Headers and GCC

- Das Betriebsystem abstrahiert das System für Programme. Dazu muss es verschiedene Schnittstellen anbieten sog. System-Calls.
- Beim Linux Kernel gibt es dazu die Userspace-API die in etwa 700 Header-Dateien (Linux 4.8) die Schnittstellen definiert.
- Im Source-Tree finden sich die Header unter `include/uapi`

  Siehe auch: `https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/include/uapi?h=v4.20`

Die vom Linux Kernel bereitgestellten Header vertragen sich gut mit dem GCC Projekt, so dass GCC in den eigens kompilierten Runtimes automatisch die richtigen Syscalls finden kann.

- Die Header für die installierte Runtime (aber auch Dritt-Bibliotheken) befinden sich bei den vielen Distributionen unter `/usr/include/`

Application Binary Interface (ABI)

Das Application Binary Interface (ABI) definiert wie auf Datenstrukturen und Routinen in der Maschinenrepresentationen zugegriffen werden kann:

- Register Dateistruktur, Stack Organisation, Speicherlayout
- Größen, Aufbau und Alignments von Basistypen
- Aufruf-Konventionen: wie Argumente und Rückgabewerte übergeben werden
- Wie Systemaufrufe auszuführen sind
- Die Struktur der Object-Dateien

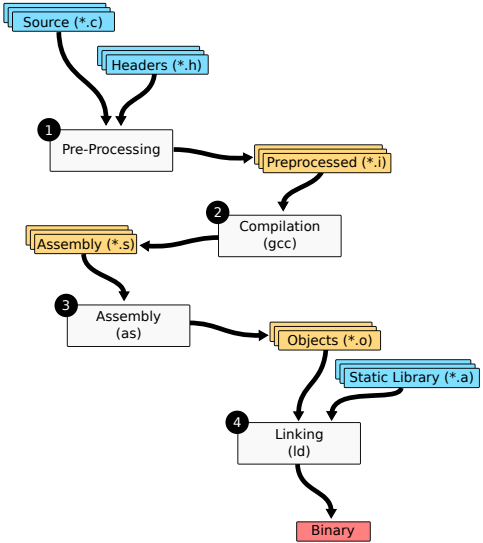
Die Linux ABI is weitesgehend rückwärtskompatibel:

- Daher, ältere Linux-Header i.d.R. weiterhin benutzbar:
  z.B. ein 3.4 Header funktioniert mit einem 4.5 Kernel
- Neue Header (insbesondere mit neuen Features) mit einem alten Kernel zu benutzen führt meistens zu Problemen

C Library

Die C Library stellt z.B. die von POSIX definierten Standardfunktionen bereit. Es gibt diverse Implementationen der C Library jeweils mit unterschiedlichen Schwerpunkten:

- glibc
- uClibc-ng
- musl
- bionic (Android)
- newlib, dietlib, klibc (for very minimal systems)



1. `cpp hello.c > hello.i`
2. `gcc -S hello.i`
3. `as -o hello.o hello.s`
4. `ld -o hello hello.o -lc ...`

Konsultiert `gcc -v -o hello hello.c` für Details ;)

Compiler Flags

GCC kommt mit vielen Optionen und Einstellungen die i.d.R. über sog. Compiler-Flags gesteuert werden.

```
1 -Wl  werden an den linker (ld) weitergereicht (siehe "man ld")
```

Empfohlene Flags

```
# Sicherheit
-D_FORTIFY_SOURCE=2              Laufzeit Overflow Erkennung
-fpie -Wl,-pie                   Address Space Layout Randomization (ASLR)
-fstack-clash-protection         Increased reliability of stack overflow detection
-fstack-protector                Overflow Erkennug via Canary (variants: all, strong) (RHEL6+)
-mcet -fcf-protection            Control flow integrity protection (future)
# Optimierung
-O2                              Recommended optimizations
-pipe                            Compile time optimization (avoid temporary files)
# Linker
-WL,-z,defs                      Detect and reject unterlinking
-Wl,-z,now                       Disable lazy binding (RHEL7+)
-Wl,-z,relro                     Read-only segments ofter relation (RHEL6+)
# Fehlerbehandlung
-fasynchronous-unwind-tables     Increased reliability of backtraces
-fexceptions                     Enable table-based thread cancellation
# Object Structure / Introspection
-fpic -shared                    No text relocations for shared libraries
-fplugin=annobin                 Inquire about hardening options, ABI compatability
# Debugging Informationen
-g                               Add debuggin information and labels
-grecord-gcc-switches            Compilerflags Metadata als debugging info
# Warnungen und Hinweise
-Wall                            Recommended compiler warnings
-Werror=format-security          Reject potentially unsafe format strings
-Werror=implicit-function-declaration   Reject missing function prototypes
```

Siehe auch: https://developers.redhat.com/blog/2018/03/21/compiler-and-linker-flags-gcc/

-fstack-protector

```
1 void fun() {
2     char *buf = alloca(0x100);
3     /* Don't allow gcc to optimise away the buf */
4     asm volatile("" :: "m" (buf));
5 }
```

Siehe auch: https://idea.popcount.org/2013-08-15-fortify_source/

-fstack-protector

```
1  08048404 <fun>:
2  push   %ebp               ; prologue
3  mov    %esp,%ebp
4
5  sub    $0x128,%esp        ; reserve 0x128B on the stack
6  lea    0xf(%esp),%eax     ; eax = esp + 0xf
7  and    $0xfffffff0,%eax   ; align eax
8  mov    %eax,-0xc(%ebp)    ; save eax in the stack frame
9
10 leave                     ; epilogue
11 ret
```

Siehe auch: `https://idea.popcount.org/2013-08-15-fortify_source/`

-fstack-protector

```
08048464 <fun>:
push   %ebp                ; prologue
mov    %esp,%ebp

sub    $0x128,%esp         ; reserve 0x128B on the stack

mov    %gs:0x14,%eax       ; load stack canary using gs
mov    %eax,-0xc(%ebp)     ; save it in the stack frame
xor    %eax,%eax           ; clear the register

lea    0xf(%esp),%eax      ; eax = esp + 0xf
and    $0xfffffff0,%eax    ; align eax
mov    %eax,-0x10(%ebp)    ; save eax in the stack frame

mov    -0xc(%ebp),%eax     ; load canary
xor    %gs:0x14,%eax       ; compare against one in gs
je     8048493 <fun+0x2f>
call   8048340 <__stack_chk_fail@plt>

leave                      ; epilogue
ret
```

Siehe auch: https://idea.popcount.org/2013-08-15-fortify_source/

-D_FORTIFY_SOURCE=2

```
1 void fun(char *s) {
2     char buf[0x100];
3     strcpy(buf, s);  // Though you should prefer strncpy anyways! ;)
4     /* Don't allow gcc to optimise away the buf */
5     asm volatile("" :: "m" (buf));
6 }
```

Siehe auch: https://idea.popcount.org/2013-08-15-fortify_source/

-D_FORTIFY_SOURCE=2

```
08048450 <fun>:
push   %ebp                ; prologue
mov    %esp,%ebp

sub    $0x118,%esp         ; reserve 0x118B on the stack
mov    0x8(%ebp),%eax      ; load parameter  s  to eax
mov    %eax,0x4(%esp)      ; save parameter for strcpy
lea    -0x108(%ebp),%eax   ; count  buf  in eax
mov    %eax,(%esp)         ; save parameter for strcpy
call   8048320 <strcpy@plt>

leave                      ; epilogue
ret
```

Siehe auch: https://idea.popcount.org/2013-08-15-fortify_source/

-D_FORTIFY_SOURCE=2

```
08048470 <fun>:
push   %ebp               ; prologue
mov    %esp,%ebp

sub    $0x118,%esp        ; reserve 0x118B on the stack
movl   $0x100,0x8(%esp)   ; save value 0x100 as parameter
mov    0x8(%ebp),%eax     ; load parameter  s  to eax
mov    %eax,0x4(%esp)     ; save parameter for strcpy
lea    -0x108(%ebp),%eax  ; count  buf  in eax
mov    %eax,(%esp)        ; save parameter for strcpy
call   8048370 <__strcpy_chk@plt>

leave                      ; epilogue
ret
```

Siehe auch: https://idea.popcount.org/2013-08-15-fortify_source/

Beispiel: -D_FORTIFY_SOURCE

```c
/* Copyright (C) 1991–2018 Free Software Foundation, Inc.
This file is part of the GNU C Library.
The GNU C Library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.
The GNU C Library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.
You should have received a copy of the GNU Lesser General Public
License along with the GNU C Library; if not, see
<http://www.gnu.org/licenses/>.  */

#include <stddef.h>
#include <string.h>
#include <memcopy.h>

#undef strcpy

/* Copy SRC to DEST with checking of destination buffer overflow.  */
char * __strcpy_chk (char *dest, const char *src, size_t destlen)  {
    size_t len = strlen (src);
    if (len >= destlen)
        __chk_fail ();
    return memcpy (dest, src, len + 1);
}
```

Siehe auch: http://sourceware.org/git/?p=glibc.git;a=blob_plain;f=debug/strcpy_chk.c;hb=HEAD