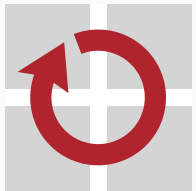
Dynamischer ELF Loader

ELF Unterstützung für OctoPOS - Erfahrungen und Ergebnisse

02.03.2020

Vanessa Hack, Dorothea Ehrl

Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Vorgehen

1. Interpretieren und Parsen der Informationen aus ELF Dateien
2. Geeignetes Tool dazu schreiben (readelf) bis 6. März

Vorgehen

1. Interpretieren und Parsen der Informationen aus ELF Dateien
2. Geeignetes Tool dazu schreiben (readelf) bis 6. März
   bis 3. März

Vorgehen

1. Interpretieren und Parsen der Informationen aus ELF Dateien
2. Geeignetes Tool dazu schreiben (readelf) bis 6. März
   bis 3. März
3. Mappen der Sections in den Speicher bis 10. März

Vorgehen

1. Interpretieren und Parsen der Informationen aus ELF Dateien
2. Geeignetes Tool dazu schreiben (readelf) bis 6. März
   bis 3. März
3. Mappen der Sections in den Speicher bis 10. März
   bis 5. März

Vorgehen

1. Interpretieren und Parsen der Informationen aus ELF Dateien
2. Geeignetes Tool dazu schreiben (readelf) bis 6. März
   bis 3. März
3. Mappen der Sections in den Speicher bis 10. März
   bis 5. März
4. Auflösen der Symbole und Adressen anpassen bis 12. März

Vorgehen

1. Interpretieren und Parsen der Informationen aus ELF Dateien
2. Geeignetes Tool dazu schreiben (readelf) bis 6. März
   bis 3. März
3. Mappen der Sections in den Speicher bis 10. März
   bis 5. März
4. Auflösen der Symbole und Adressen anpassen bis 12. März
   bis 10. März

Vorgehen

1. Interpretieren und Parsen der Informationen aus ELF Dateien
2. Geeignetes Tool dazu schreiben (readelf) bis 6. März
   bis 3. März
3. Mappen der Sections in den Speicher bis 10. März
   bis 5. März
4. Auflösen der Symbole und Adressen anpassen bis 12. März
   bis 10. März
5. Auflösen der Symbole aus externen Bibliotheken (wenn Zeit bleibt)
6. Corona!

Testprogramm

```
extern "C" void hello_main_ilet(claim_t claim) {

    printf("Hello␣World\n");

    shutdown(1);
}
```

positionsunabhängiger Code: alle Adressen sind relativ zu einer Ursprungsadresse

Arbeitsablauf - Readelf

ELF Header

```
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            other OS/ABI than Unix/Linux - GNU
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x240
  Start of program headers:          52 (bytes into file)
  Start of section headers:          4840 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         8
  Size of section headers:           40 (bytes)
  Number of section headers:         18
  Section header string table index: 17
```

- Inhalt der ELF Header Struktur ausgeben

Arbeitsablauf - Readelf

Programm Header bzw. Segmente

```
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x00000034 0x00000034 0x00100 0x00100 R   0x4
  INTERP         0x000134 0x00000134 0x00000134 0x00013 0x00013 R   0x1
  LOAD           0x000000 0x00000000 0x00000000 0x00270 0x00270 R E 0x1000
  LOAD           0x000f58 0x00001f58 0x00001f58 0x000ac 0x000ac RW  0x1000
  DYNAMIC        0x000f58 0x00001f58 0x00001f58 0x00098 0x00098 RW  0x4
  NOTE           0x000148 0x00000148 0x00000148 0x00024 0x00024 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x10
  GNU_RELRO      0x000f58 0x00001f58 0x00001f58 0x000a8 0x000a8 R   0x1
```

- aus Struktur vom ELF Header kann man sich die Strukturen der Programm Header herleiten
- über diese iterieren und ausgeben

Arbeitsablauf - Mapping

```
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x00000034 0x00000034 0x00100 0x00100 R   0x4
  INTERP         0x000134 0x00000134 0x00000134 0x00013 0x00013 R   0x1
  LOAD           0x000000 0x00000000 0x00000000 0x00270 0x00270 R E 0x1000
  LOAD           0x000f58 0x00001f58 0x00001f58 0x000ac 0x000ac RW  0x1000
  DYNAMIC        0x000f58 0x00001f58 0x00001f58 0x00098 0x00098 RW  0x4
  NOTE           0x000148 0x00000148 0x00000148 0x00024 0x00024 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x10
  GNU_RELRO      0x000f58 0x00001f58 0x00001f58 0x000a8 0x000a8 R   0x1
```

Arbeitsablauf - Mapping

```
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x00000034 0x00000034 0x00100 0x00100 R   0x4
  INTERP         0x000134 0x00000134 0x00000134 0x00013 0x00013 R   0x1
  LOAD           0x000000 0x00000000 0x00000000 0x00270 0x00270 R E 0x1000
  LOAD           0x000f58 0x00001f58 0x00001f58 0x000ac 0x000ac RW  0x1000
  DYNAMIC        0x000f58 0x00001f58 0x00001f58 0x00098 0x00098 RW  0x4
  NOTE           0x000148 0x00000148 0x00000148 0x00024 0x00024 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x10
  GNU_RELRO      0x000f58 0x00001f58 0x00001f58 0x000a8 0x000a8 R   0x1
```

- Speicher reservieren (mem_alloc)
- LOAD Segmente zu reserviertem Speicher kopieren

Arbeitsablauf - Mapping

```
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x00000034 0x00000034 0x00100 0x00100 R   0x4
  INTERP         0x000134 0x00000134 0x00000134 0x00013 0x00013 R   0x1
  LOAD           0x000000 0x00000000 0x00000000 0x00270 0x00270 R E 0x1000
  LOAD           0x000f58 0x00001f58 0x00001f58 0x000ac 0x000ac RW  0x1000
  DYNAMIC        0x000f58 0x00001f58 0x00001f58 0x00098 0x00098 RW  0x4
  NOTE           0x000148 0x00000148 0x00000148 0x00024 0x00024 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x10
  GNU_RELRO      0x000f58 0x00001f58 0x00001f58 0x000a8 0x000a8 R   0x1
```

- Speicher reservieren (mem_alloc)
- LOAD Segmente zu reserviertem Speicher kopieren

Achtung!

- Platz für nicht explizit angelegte Teile z.B. bss lassen

Arbeitsablauf - Mapping

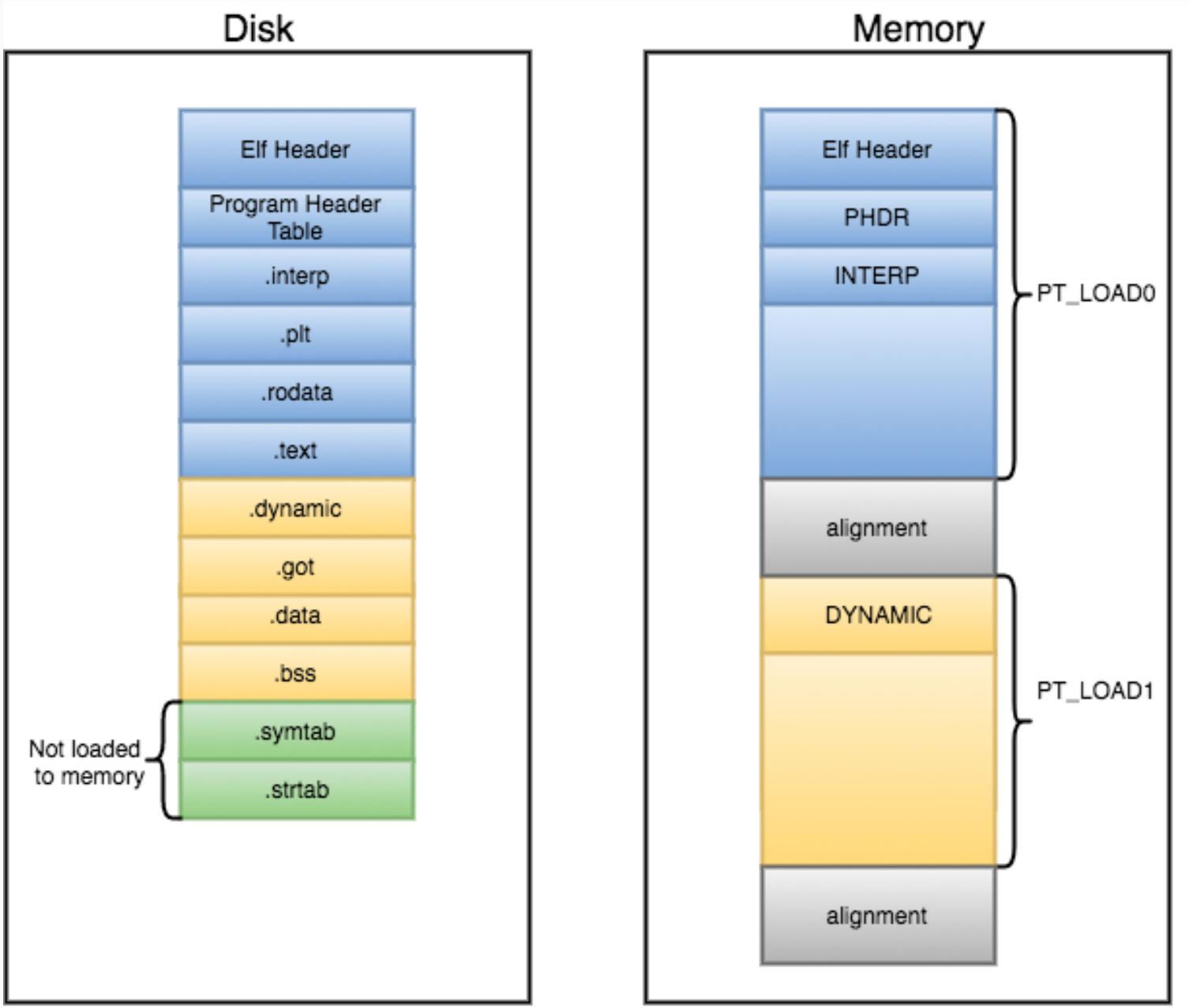
```
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x00000034 0x00000034 0x00100 0x00100 R   0x4
  INTERP         0x000134 0x00000134 0x00000134 0x00013 0x00013 R   0x1
  LOAD           0x000000 0x00000000 0x00000000 0x00270 0x00270 R E 0x1000
  LOAD           0x000f58 0x00001f58 0x00001f58 0x000ac 0x000ac RW  0x1000
  DYNAMIC        0x000f58 0x00001f58 0x00001f58 0x00098 0x00098 RW  0x4
  NOTE           0x000148 0x00000148 0x00000148 0x00024 0x00024 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x10
  GNU_RELRO      0x000f58 0x00001f58 0x00001f58 0x000a8 0x000a8 R   0x1
```

- Speicher reservieren (mem_alloc)
- LOAD Segmente zu reserviertem Speicher kopieren

Achtung!

- Platz für nicht explizit angelegte Teile z.B. bss lassen
- Speicherreservierung abhängig von virtuellen Speicher machen

Arbeitsablauf - Mapping



Arbeitsablauf - Entrypoint

- ELF Header gibt einen positionsunabhängigen Entrypoint an
- dieser wird mit Anfangsadresse des beim Mapping angelegten Speicherbereichs addiert

→ Entrypoint innerhalb des gemappten Speichers

- Anspringen des Entrypoints
- Ausführen der Datei

Arbeitsablauf - Entrypoint

- ELF Header gibt einen positionsunabhängigen Entrypoint an
- dieser wird mit Anfangsadresse des beim Mapping angelegten Speicherbereichs addiert

→ Entrypoint innerhalb des gemappten Speichers

- Anspringen des Entrypoints
- Ausführen der Datei

Erfolg!

Ausführen von ELF Binaries ohne Funktionsaufrufe ist möglich!

Symbolauflösung

Symboltabelle

Enthält benötigte Informationen zur Lokalisierung und Relokation der symbolischen Referenzen eines Programms

- Name: Index in die String Tabelle
- Value: Absoluter Wert oder Adresse, je nach Kontext
  hier: Section Offset des Symbols
- Size, Info, Other ...

String Tabelle:

Index	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
0	\0	n	a	m	e	.	\0	V	a	r
10	i	a	b	l	e	\0	a	b	l	e
20	\0	\0	x	x	\0					

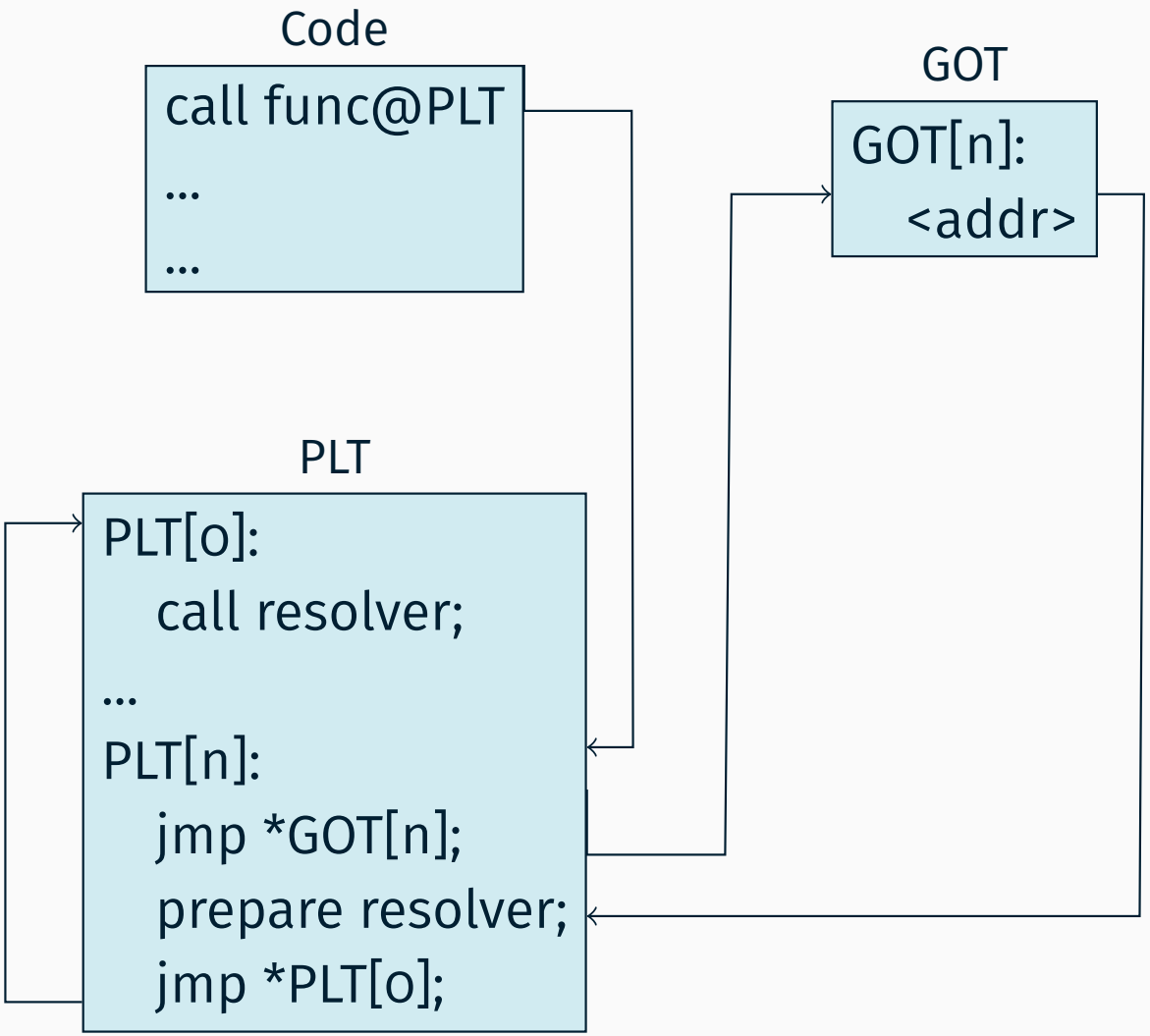
readelf -s

```
Symbol table '.symtab' contains 27 entries:
Num:    Value          Size Type    Bind   Vis      Ndx Name
0:      00000000         0 NOTYPE  LOCAL  DEFAULT  UND
1:      00000134         0 SECTION LOCAL  DEFAULT    1
2:      00000148         0 SECTION LOCAL  DEFAULT    2
3:      0000016c         0 SECTION LOCAL  DEFAULT    3
4:      00000198         0 SECTION LOCAL  DEFAULT    4
5:      000001f8         0 SECTION LOCAL  DEFAULT    5
6:      0000022c         0 SECTION LOCAL  DEFAULT    6
7:      00000240         0 SECTION LOCAL  DEFAULT    7
8:      00000270         0 SECTION LOCAL  DEFAULT    8
9:      0000029b         0 SECTION LOCAL  DEFAULT    9
10:     000002a8         0 SECTION LOCAL  DEFAULT   10
11:     00001f54         0 SECTION LOCAL  DEFAULT   11
12:     00001fec         0 SECTION LOCAL  DEFAULT   12
13:     00002000         0 SECTION LOCAL  DEFAULT   13
14:     00000000         0 SECTION LOCAL  DEFAULT   14
15:     00000000         0 FILE    LOCAL  DEFAULT  ABS     hello.cc
16:     00000000         0 FILE    LOCAL  DEFAULT  ABS
17:     00001f54         0 OBJECT  LOCAL  DEFAULT   11     _DYNAMIC
18:     00001fec         0 OBJECT  LOCAL  DEFAULT   12     _GLOBAL_OFFSET_TABLE_
19:     00000270        39 FUNC    GLOBAL DEFAULT    8     hello_main_ilet
20:     00000000         0 FUNC    GLOBAL DEFAULT  UND     puts
21:     00002000         4 OBJECT  GLOBAL DEFAULT   13     os_agent_agentsystem_no_init
22:     00000297         0 FUNC    GLOBAL HIDDEN     8     __x86.get_pc_thunk.bx
23:     00002004         0 NOTYPE  GLOBAL DEFAULT   13     __bss_start
24:     00000000         0 FUNC    GLOBAL DEFAULT  UND     shutdown
25:     00002004         0 NOTYPE  GLOBAL DEFAULT   13     _edata
26:     00002004         0 NOTYPE  GLOBAL DEFAULT   13     _end
```

Ausgabe der Symboltabelle

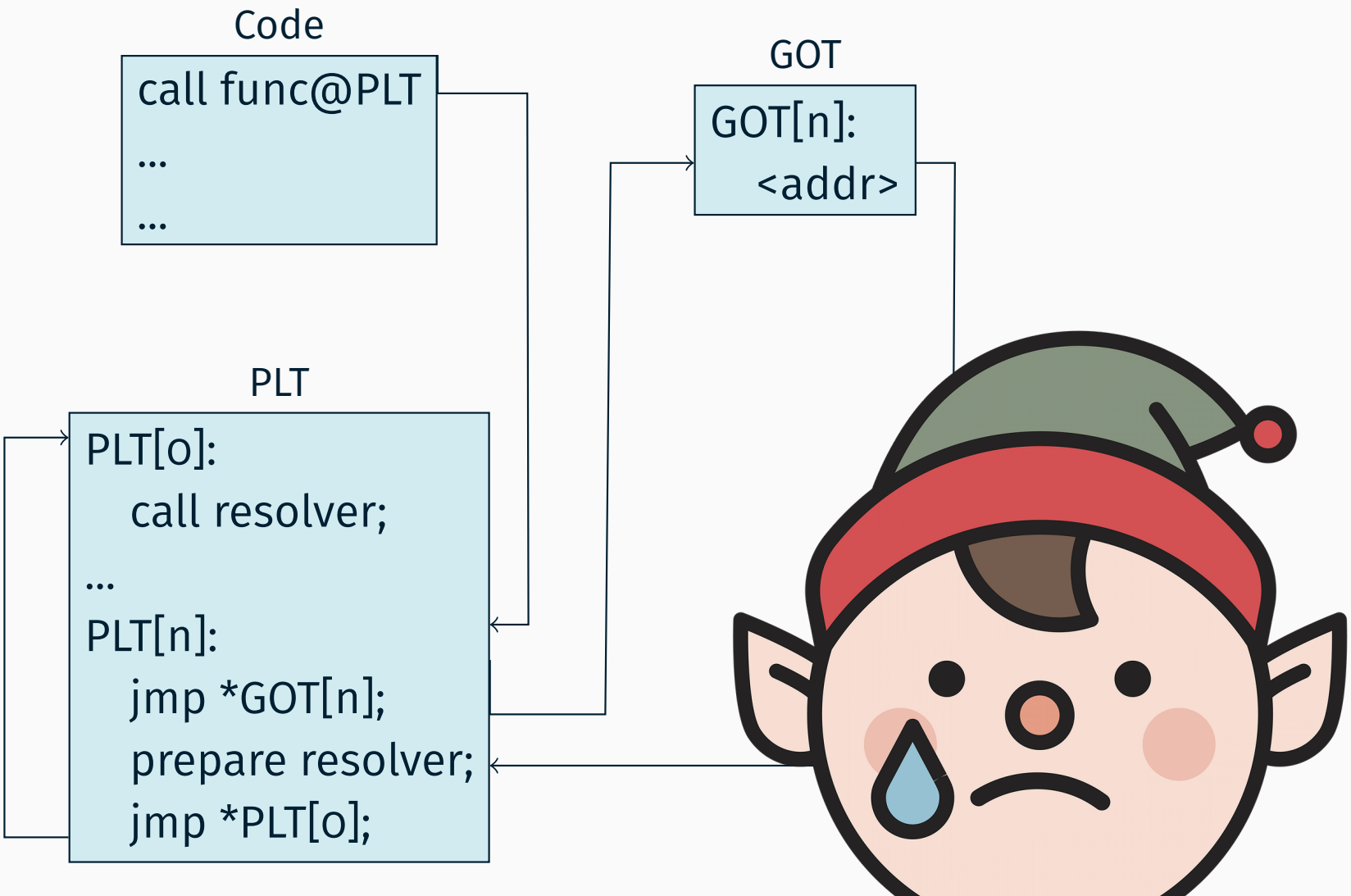
Symbolauflösung - Lazy Loading

Default: **Lazy Loading** - Abhängigkeiten werden erst bei Referenzierung geladen



Symbolauflösung - Lazy Loading

Default: **Lazy Loading** - Abhängigkeiten werden erst bei Referenzierung geladen



Symbolauflösung - Unser Loader

```
20: 00000000     0 FUNC    GLOBAL DEFAULT  UND puts
24: 00000000     0 FUNC    GLOBAL DEFAULT  UND shutdown
```

puts und shutdown Symbole der hello Binärdatei

Symbolauflösung - Unser Loader

```
 20: 00000000     0 FUNC    GLOBAL DEFAULT  UND puts
 24: 00000000     0 FUNC    GLOBAL DEFAULT  UND shutdown
```

puts und shutdown Symbole der hello Binärdatei

```
1351: 080a5611    36 FUNC    GLOBAL DEFAULT    3 puts
2010: 08048290    29 FUNC    GLOBAL DEFAULT    3 shutdown
```

puts und shutdown Symbole der elf-loader Binärdatei

Symbolauflösung - Unser Loader

```
 20: 00000000     0 FUNC    GLOBAL DEFAULT  UND puts
 24: 00000000     0 FUNC    GLOBAL DEFAULT  UND shutdown
```

puts und shutdown Symbole der hello Binärdatei

```
1351: 080a5611    36 FUNC    GLOBAL DEFAULT    3 puts
2010: 08048290    29 FUNC    GLOBAL DEFAULT    3 shutdown
```

puts und shutdown Symbole der elf-loader Binärdatei

Ziel

Auszuführende Binärdatei soll Symbolauflösung mit Hilfe der absoluten Adressen des statisch mit OctoPOS gelinkten elf-loader durchführen.

Symbolauflösung - Unser Loader

```
 20: 00000000     0 FUNC    GLOBAL DEFAULT  UND puts
 24: 00000000     0 FUNC    GLOBAL DEFAULT  UND shutdown
```

puts und shutdown Symbole der hello Binärdatei

```
1351: 080a5611    36 FUNC    GLOBAL DEFAULT    3 puts
2010: 08048290    29 FUNC    GLOBAL DEFAULT    3 shutdown
```

puts und shutdown Symbole der elf-loader Binärdatei

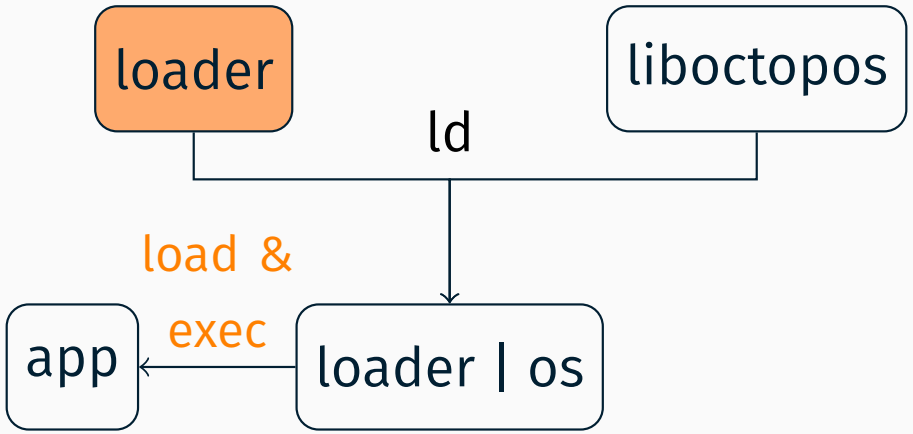
Ziel

Auszuführende Binärdatei soll Symbolauflösung mit Hilfe der absoluten Adressen des statisch mit OctoPOS gelinkten elf-loader durchführen.

```
*dest = reloc_value;
```

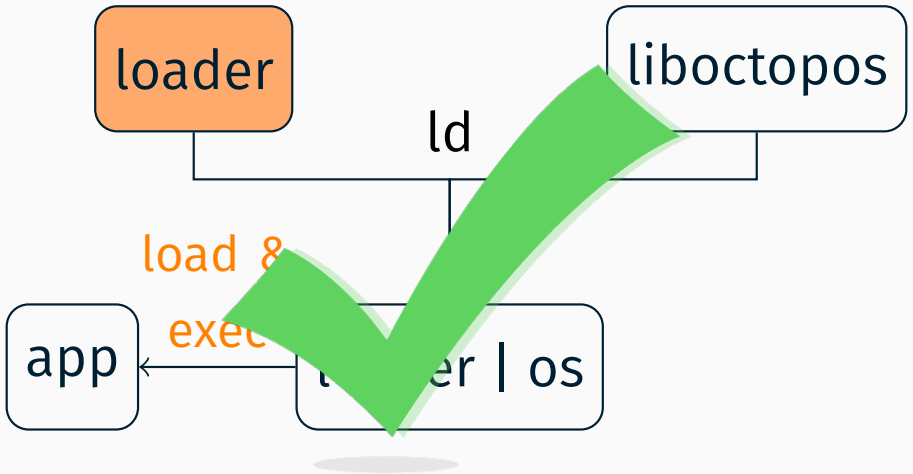
Ergebnis

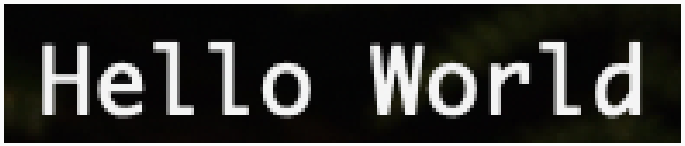
```
$ ./mklib.sh -m i386 elf-loader
$ gcc -m32 -o hello hello.o -nostdlib -L.
  -lelf-loader -Wl,-e,hello_main_ilet -Wl,-z,now
  -fPIC -fPIE -fno-plt
$ ./elf-loader --module elf-loader --module hello
```



Ergebnis

```
$ ./mklib.sh -m i386 elf-loader
$ gcc -m32 -o hello hello.o -nostdlib -L.
  -lelf-loader -Wl,-e,hello_main_ilet -Wl,-z,now
  -fPIC -fPIE -fno-plt
$ ./elf-loader --module elf-loader --module hello
```





Ausblick

Was geht noch nicht?

- Unterstützung von 64 Bit ELFs
- Unterstützung aller Plattformen (x64 native und SPARC LEON)
- Umbau auf C++ und Nutzen von Templates zur Unterscheidung von 32 und 64 Bit
- Symbolauflösung aus externen Bibliotheken
- Ausführung von Code auf anderen Kacheln

Fragen?

Quellen

http://www.skyfree.org/linux/references/ELF_Format.pdf
https://www.technovelty.org/linux/plt-and-got-the-key-to-code-sharing-and-dynamic-libraries.html
https://img.rawpixel.com/s3fs-private/rawpixel_images/website_-content/v767-wan-71_1.png
https://www.pngwave.com/png-clip-art-apqqz
https://eli.thegreenplace.net/images/2011/plt_before.png
https://intezer.com/wp-content/uploads/2018/01/Image5.png