Verlässliche Echtzeitsysteme

Übungen zur Vorlesung

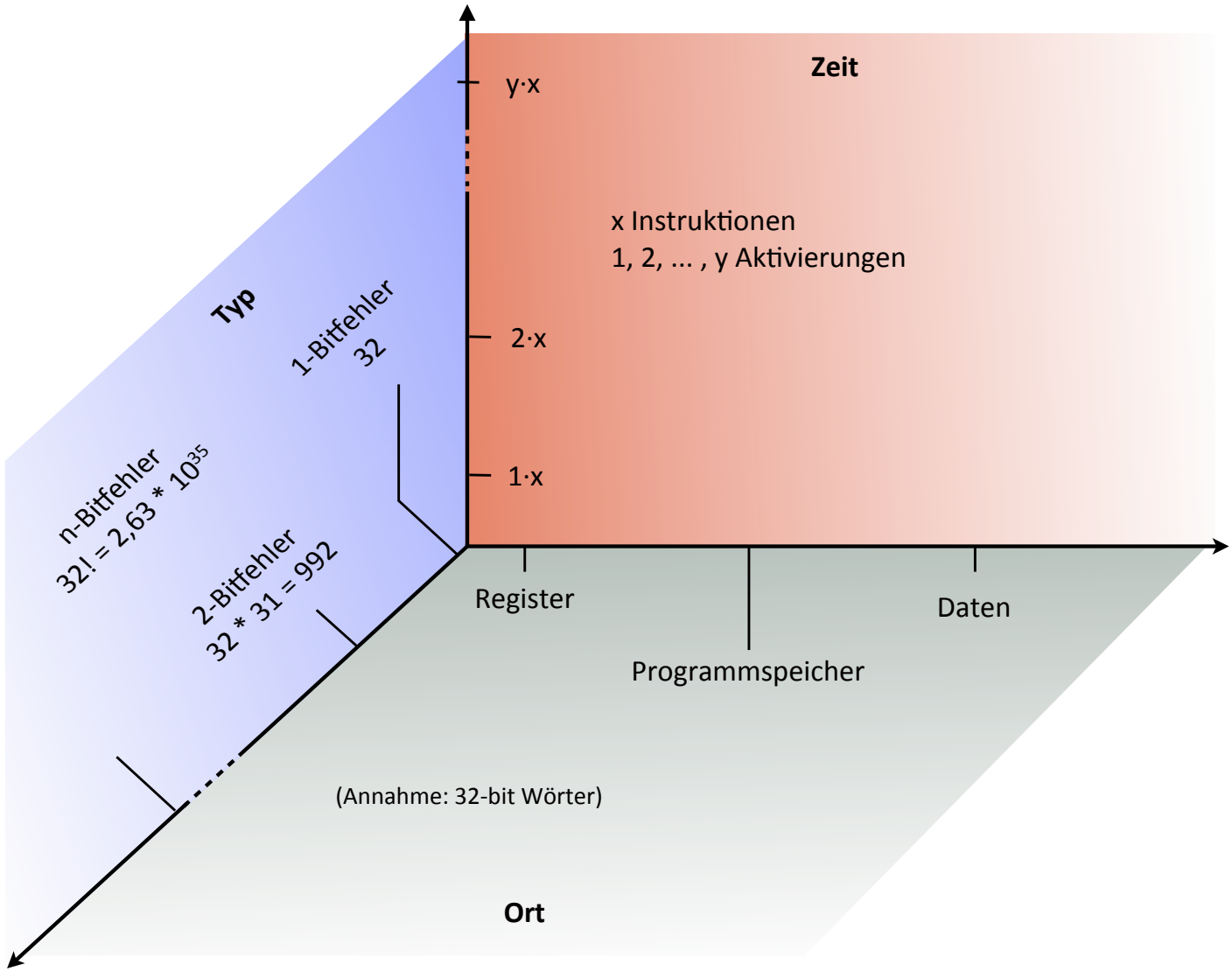
Arbeiten mit Binärdateien und Assembler

Phillip Raffeck, Florian Schmaus, Simon Schuster

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
https://www4.cs.fau.de

Sommersemester 2020

Fehlerraum


Zeit
y·x
x Instruktionen
1, 2, ... , y Aktivierungen
2·x
1·x
Typ
1-Bitfehler
32
n-Bitfehler
32! = 2,63 * 10^35
2-Bitfehler
32 * 31 = 992
Register
Programmspeicher
Daten
(Annahme: 32-bit Wörter)
Ort

C-Code vs. Assembler-Code

C-Code

```
int a;
int b = 1;
const int c = 2;
void main() {
  static int s = 3;
  int x, y;
  char* p =
       malloc(100);
}
```

Assembler-Code

```
4004f0 <main>:
  4004f0: push   %rbp
  4004f1: mov    %rsp,%rbp
  4004f4: sub    $0x10,%rsp
  4004f8: movabs $0x64,%rdi
  400502: callq  4003e0 <malloc@plt>
  400507: mov    %rax,-0x10(%rbp)
  40050b: add    $0x10,%rsp
  40050f: pop    %rbp
  400510: retq
```

Wo können Datenfehler auftreten?

C-Code vs. Assembler-Code

C-Code

```
int a;
int b = 1;
const int c = 2;
void main() {
  static int s = 3;
  int x, y;
  char* p =
       malloc(100);
}
```

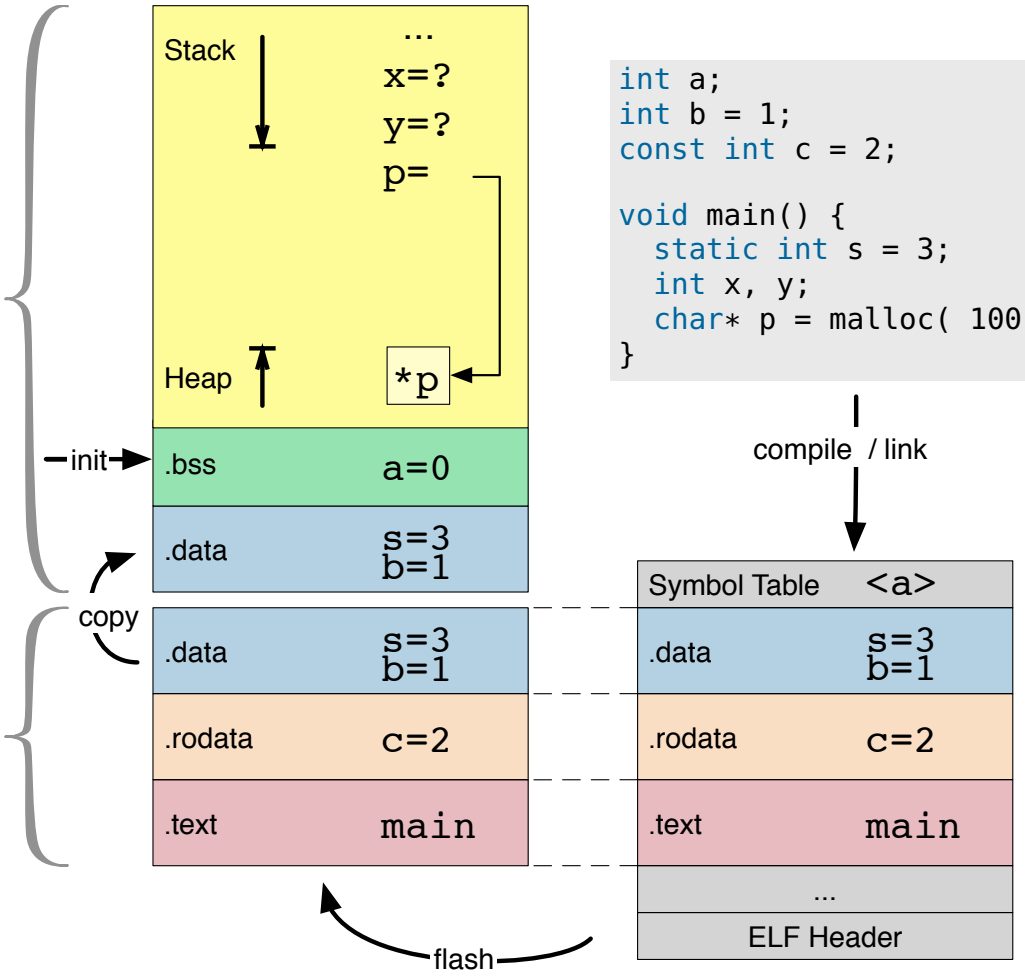
Assembler-Code

```
4004f0 <main>:
  4004f0: push   %rbp
  4004f1: mov    %rsp,%rbp
  4004f4: sub    $0x10,%rsp
  4004f8: movabs $0x64,%rdi
  400502: callq  4003e0 <malloc@plt>
  400507: mov    %rax,-0x10(%rbp)
  40050b: add    $0x10,%rsp
  40050f: pop    %rbp
  400510: retq
```

Wo können Datenfehler auftreten?

1. RAM: `-0x10(%rbp)`
   - ⇝ Stack, globale Daten, Heap, (Programmcode)
2. Allgemeine CPU-Register: `%rsp`
3. Sonstige CPU-Register: `%rip`, `%rflags`

Speicherorganisation und Sektionsgrößen




```c
int a;                          // a: global, uninitialized
int b = 1;                      // b: global, initialized
const int c = 2;                // c: global, const

void main() {
  static int s = 3;             // s: local, static, initialized
  int x, y;                     // x: local, auto; y: local, auto
  char* p = malloc( 100 );      // p: local, auto; *p: heap (100 byte)
}
```

Quellprogramm

μ-Controller

ELF-Binary

```
$ size --format=berkeley a.out
text data bss  dec hex filename
2592    8   4 2604 a2c    a.out
```

Sektionsgrößen in Byte,
„`text`“ umfasst hier: `.text` + `.rodata`

Umgang mit Assembly-Code I

nm: Ausgabe der Symboltabelle

```
1  0000000000004028 D var_initialized
2  ...
3  0000000000201028 B __bss_start
4  0000000000201028 b var_uninitialized
5  ...
6  000000000000061a T main
7  0000000000000580 t register_tm_clones
8  0000000000000510 T _start
9  ...
10 000000000000066d t int max<int,0u>(int,int)
11 000000000000067c t int max<int,1u>(int,int)
```

- Nützliche Optionen
  - `-C, --demangle`: Dekodieren der C++-Namensmangelung:
    `_Z3maxIiLj0EET_S0_S0_` ⇒ `int max<int, 0u>(int, int)`
  - `-S, --print-size`: Ausgabe der Symbolgrößen
    `000000000000061 a 00000028 T main`

Umgang mit Assembly-Code II

objdump: Ausgabe von Informationen über Objektdateien

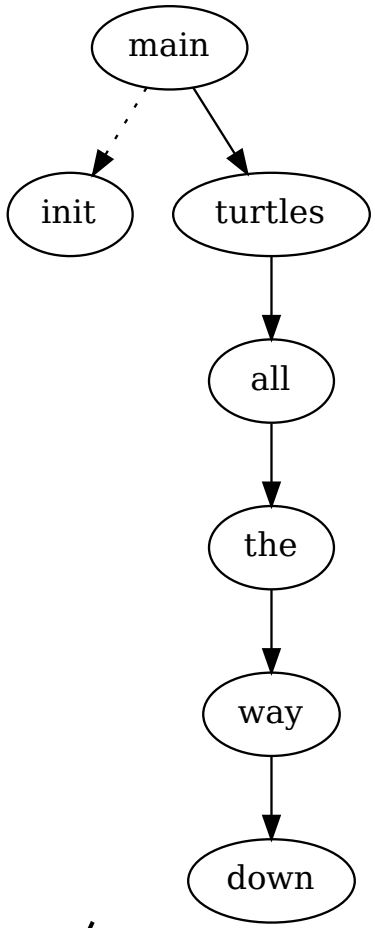
```
1  int main(){
2   4007cd:   55                    push   %rbp
3   4007ce:   48 89 e5              mov    %rsp,%rbp
4   4007d1:   48 83 ec 10           sub    $0x10,%rsp
5  int a = max<int>(23U, 42);
6   4007d5:   be 2a 00 00 00        mov    $0x2a,%esi
7   4007da:   bf 17 00 00 00        mov    $0x17,%edi
8   4007df:   e8 04 01 00 00        callq
9                                   4008e8 <_Z3maxIiET_S0_S0_>
10  4007e4:   89 45 fc              mov    %eax,-0x4(%rbp)
11 std::cout << a << "\n";
12  4007e7:   8b 45 fc              mov    -0x4(%rbp),%eax
13  ...
```

- Nützliche Optionen
  - `-S`: Ausgabe von Quell-Code im Assembly-Code (Debug-Symbole notwendig)
  - `-D`: alle Sektionen disassemblieren

Hinweise zur Aufgabenbearbeitung

Analyse der Toleranzmaßnahmen

- normalerweise: Fehlerinjektion
- dieses Semester: Fehlerraumanalyse
  - Einfluss des Speichers
  - Einfluss der Laufzeit



- Aufrufgraph

  `make callgraph_{baseline,tmr,ean}`
- ELF

  `build/{baseline,tmr,ean}.elf`
- Objektdateien

  `build/CMakeFiles/{baseline,tmr,ean}.dir/src/*.o`