

Übung zu Betriebssystembau

Debuggen mit GDB & GEF

Oktober 2023

Peter Ulbrich & Alexander Lochmann
(Mit Material vom Lehrstuhl 4 der FAU)

Arbeitsgruppe Systemsoftware
Technische Universität Dortmund



Your PC ran into a problem that it couldn't handle, and now it needs to restart.

You can search for the error online: `HAL_INITIALIZATION_FAILED`



Your PC ran into a problem that it couldn't handle, and now it needs to restart.

You can search for the error online: `HAL_INITIALIZATION_FAILED`





Your PC ran into a problem it couldn't handle, and now it needs to restart.

You can search for the error online: `HAL_INITIALIZATION_FAILED`



```

common.mk
kernel
  boot
    sections.ld
    startup.asm
    startup.cc
  compiler.h
  config.h
  debug
    assert.cc
    assert.h
    gdb
      handler.asm
      handler.cc
      init.cc
      protocol.cc
      stub.h
    kernelpanic.h
    null_stream.cc
    null_stream.h
    output.h
  device
    cgastr.cc
    cgastr.h
    console.cc
    console.h
    keyboard.cc
    keyboard.h
    panic.cc
    panic.h
    watch.cc
    watch.h
  guard
    gate.h
    guard.cc
    guard.h
    guardian.cc
    guardian.h
    secure.h
  machine
    acpi.cc
    acpi.h
    apicsystem.cc
    apicsystem.h
    cgastr.cc
    cgastr.h
    cpu.asm
    cpu.h
    gdt.cc
    gdt.h
    ioapic.cc
    ioapic.h
    ioapic_registers.h
    io_port.h
    keyctrl.cc
    keyctrl.h
    keydecoder.cc
    keydecoder.h
    key.h
    lapic.cc
    lapic.h
    lapic_registers.h
    mp_registers.h
    plughbox.cc

```

```

plugbox.h
serial.cc
serial.h
spinlock.h
ticketlock.h
toc.asm
toc.cc
toc.h
toc.inc
main.cc
Makefile
meeting
  bell.cc
  bell.h
  bellringer.cc
  bellringer.h
  messageinfo.h
  semaphore.cc
  semaphore.h
  waitingroom.cc
  waitingroom.h

```

```

memory
  allocator.cc
  allocator.h
  copy.cc
  copy.h
  initmem.cc
  initmem.h
  kernelpage.h
  malloc.cc
  malloc.h
  multiboot.h
  pagecont.cc
  pagecont.h
  dir.cc

```



```

pagefault.h
page.h
pageref.cc
pageref.h
range.h
user_buffer.h
object
  bbuffer.h
  o_stream.cc
  o_stream.h
  queue.h
  queueink.h
  strbuf.cc
  strbuf.h
syscall
  guarded_bell.cc
  guarded_bell.h
  guarded_keyboard.cc
  guarded_keyboard.h
  guarded_scheduler.h
  guarded_semaphore.h
  syscall.cc
  syscall.h
thread
  assassin.cc
  assassin.h
  dispatcher.cc
  dispatcher.h
  idlethread.cc
  idlethread.h
  scheduler.cc
  scheduler.h
  thread.cc
  thread.h
  wakeup.h
types.h
user
  app1
    appl.cc
    appl.h
  app2
    kappl.cc
    kappl.h

```

```

utils
  elf.cc
  elf.h
  libcc.cc
  math.h
  memutil.cc
  memutil.h
  sort.cc
  string.cc
  string.h
  tar.cc
  tar.h

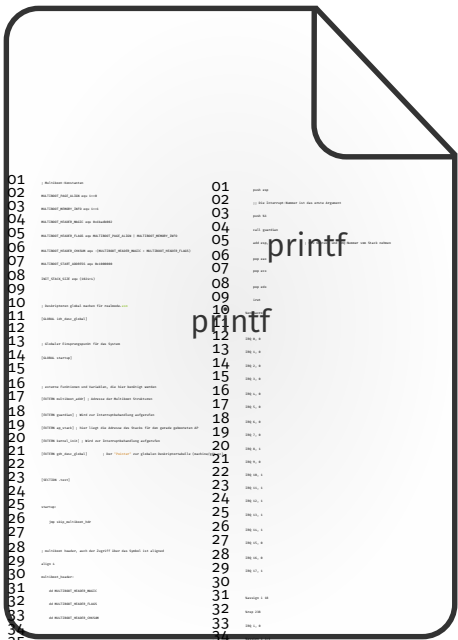
```

```

libsus
  iostream.h
  Makefile
  o_stream.cc
  o_stream.h
  strbuf.cc
  strbuf.h
  syscall_stubs.asm
  syscall_stubs.cc
  types.h
Makefile
README.md
test-stream
  console_out.cc
  console_out.h
  file_out.cc
  file_out.h
  Makefile
  test.cc
user
  app0
    app0.cc
    app0.h
  app1
    app1.cc
    app1.h
  app2
    app2.cc
    app2.h
  app3
    app3.cc
    app3.h
  app4
    app4.cc
    app4.h
  imgbuilder.cc
  init.cc
  Makefile
  Makefile.app
  sections.ld

```

24 directories, 172 files



```

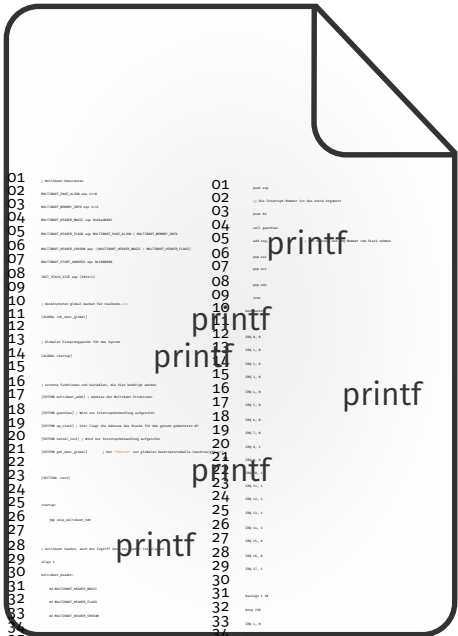
01 // Schreiben Nachrichten
02 #include <stdio.h>
03 #include <stdlib.h>
04 #include <string.h>
05 #include <unistd.h>
06 #include <sys/types.h>
07 #include <sys/stat.h>
08 #include <fcntl.h>
09 #include <sys/wait.h>
10 #include <sys/time.h>
11 #include <sys/resource.h>
12 #include <sys/param.h>
13 #include <sys/mount.h>
14 #include <sys/utsname.h>
15 #include <sys/procfs.h>
16 #include <sys/procfs.h>
17 #include <sys/procfs.h>
18 #include <sys/procfs.h>
19 #include <sys/procfs.h>
20 #include <sys/procfs.h>
21 #include <sys/procfs.h>
22 #include <sys/procfs.h>
23 #include <sys/procfs.h>
24 #include <sys/procfs.h>
25 #include <sys/procfs.h>
26 #include <sys/procfs.h>
27 #include <sys/procfs.h>
28 #include <sys/procfs.h>
29 #include <sys/procfs.h>
30 #include <sys/procfs.h>
31 #include <sys/procfs.h>
32 #include <sys/procfs.h>
33 #include <sys/procfs.h>
34 #include <sys/procfs.h>
35 #include <sys/procfs.h>
36 #include <sys/procfs.h>
37 #include <sys/procfs.h>
38 #include <sys/procfs.h>

```

```

01 main()
02 {
03     printf("Hallo!\n");
04     printf("Hallo!\n");
05     printf("Hallo!\n");
06     printf("Hallo!\n");
07     printf("Hallo!\n");
08     printf("Hallo!\n");
09     printf("Hallo!\n");
10     printf("Hallo!\n");
11     printf("Hallo!\n");
12     printf("Hallo!\n");
13     printf("Hallo!\n");
14     printf("Hallo!\n");
15     printf("Hallo!\n");
16     printf("Hallo!\n");
17     printf("Hallo!\n");
18     printf("Hallo!\n");
19     printf("Hallo!\n");
20     printf("Hallo!\n");
21     printf("Hallo!\n");
22     printf("Hallo!\n");
23     printf("Hallo!\n");
24     printf("Hallo!\n");
25     printf("Hallo!\n");
26     printf("Hallo!\n");
27     printf("Hallo!\n");
28     printf("Hallo!\n");
29     printf("Hallo!\n");
30     printf("Hallo!\n");
31     printf("Hallo!\n");
32     printf("Hallo!\n");
33     printf("Hallo!\n");
34     printf("Hallo!\n");
35     printf("Hallo!\n");
36     printf("Hallo!\n");
37     printf("Hallo!\n");
38     printf("Hallo!\n");

```



```

01 // Drücken Nachrichten
02 #ifndef _msg_001_msg_h_
03 #define _msg_001_msg_h_
04 #include <stdio.h>
05 #include <string.h>
06 #include <stdlib.h>
07 #include <unistd.h>
08 #include <sys/types.h>
09 #include <sys/stat.h>
10 #include <fcntl.h>
11 #include <sys/mman.h>
12 #include <sys/time.h>
13 #include <sys/uio.h>
14 #include <sys/wait.h>
15 #include <sys/resource.h>
16 #include <sys/epoll.h>
17 #include <sys/eventfd.h>
18 #include <sys/signalfd.h>
19 #include <sys/timerfd.h>
20 #include <sys/ptrace.h>
21 #include <sys/auxv.h>
22 #include <sys/procfs.h>
23 #include <sys/procfs.h>
24 #include <sys/procfs.h>
25 #include <sys/procfs.h>
26 #include <sys/procfs.h>
27 #include <sys/procfs.h>
28 #include <sys/procfs.h>
29 #include <sys/procfs.h>
30 #include <sys/procfs.h>
31 #include <sys/procfs.h>
32 #include <sys/procfs.h>
33 #include <sys/procfs.h>
34 #include <sys/procfs.h>
35 #include <sys/procfs.h>
36 #include <sys/procfs.h>
37 #include <sys/procfs.h>
38 #include <sys/procfs.h>

```

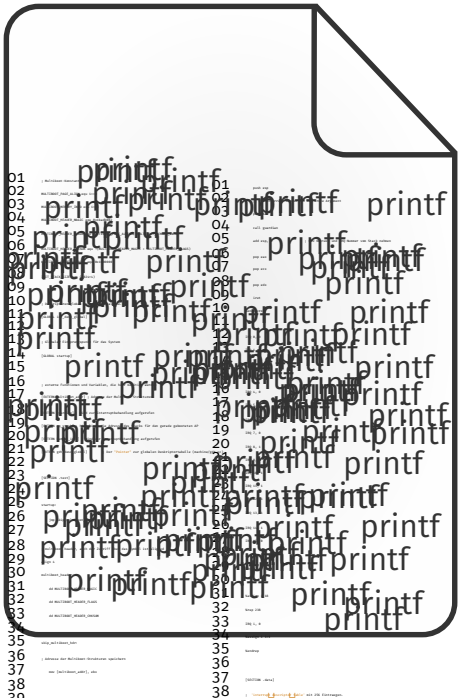
printf

printf

printf

printf

printf



GNU Debugger (GDB)

- + Inspizieren des Systemzustands während das System läuft
- Nur rudimentäres TUI

Entkäfern mit GDB & GEF

GNU Debugger (GDB)

- + Inspizieren des Systemzustands während das System läuft
- Nur rudimentäres TUI

```
(gdb) c
Continuing.

Thread 1 hit Breakpoint 1, guardian (vector=33, context=0x101cf58 <cpu_stack+3912>) at guard/guardian.cc:15
15      Gate* gate = Plugbox::report(vector);
(gdb) █
```

GDB Enhanced Features (GEF)

- + Erweitert GDB um ein brauchbar(er)es Interface
- + Bietet verschiedene Kommandos an – siehe Dokumentation von Gef

```

eax    : 0x0101b520 - 0x00000000 -> 0x00000000
ebx    : 0x01012ba8 - 0x00000000 -> 0x00000000
ecx    : 0x01010870 - 0x8b535657 -> 0x8b535657
edx    : 0x01ffb000 - 0x00113000 -> 0x00000000 + 0x00000000
esp    : 0x0101af1c - 0x0100038b -> 0x5808c483 + 0x5808c483
ebp    : 0x0101af28 - 0x01011b7c -> 0x01001930 + 0x0191c8b8 + 0x00000000 + 0x00000000
esi    : 0x01012ba8 - 0x00000000 -> 0x00000000
edi    : 0x02011200 - 0x02011200
ebp    : 0x01002d40 - 0xe8535657 -> 0xe8535657
*EFlags: [carry parity adjust zero sign trap interrupt direction overflow resume virtualx86 identification]
fs: 0x0008  fs: 0x0010  fs: 0x0010  fs: 0x0010  fs: 0x0010  fs: 0x0010  fs: 0x0010

```

Registerinhalt

```

0x0101af1c +0x0000: 0x0100038b - 0x0000c483 -> 0x0000c483 - esp
0x0101af20 +0x0004: 0x00000021 - 0x00000021
0x0101af24 +0x0008: 0x0101af28 - 0x0101af28 -> 0x00000000 - 0x00000000
0x0101af28 +0x000c: 0x0101af28 - 0x00000000 -> 0x00000000
0x0101af2c +0x0010: 0x0101af28 - 0xe8535657 -> 0xe8535657
0x0101af30 +0x0014: 0x0101af28 - 0x01001930 -> 0x00000000 - 0x00000000
0x0101af34 +0x0018: 0x0101af28 - 0xe8535657 -> 0xe8535657
0x0101af38 +0x001c: 0x00000000 - 0x00000000

```

code:08:32

```

0x1000138: 0x00 0x00 0x00
0x1000139: 0x00 0x00 0x00
0x100013f: 0x00
- 0x1002d40 <guardian+0> push edi
0x1002d41 <guardian+1> push esi
0x1002d42 <guardian+2> push ebx
0x1002d43 <guardian+3> call 0x1010f20 <__x86.get_pc_thunk,b0>
0x1002d48 <guardian+8> add ebx, 0xfe60
0x1002d4e <guardian+14> sub esp, 0x8

```

source:./guard/guardian.cc:19

```

14
15 #include "guard/guard.h"
16 extern Guard guardian;
17
18 extern "C" void guardian(uint32_t vector, irq_context_t context)
- 19 {
20     (void) vector;
21     (void) context;
22     Gate* gate = plugin.report(vector);
23
24     bool wantsEpilogue = gate->prologue();

```

threads

```

[*] Id 1, Name: "", stopped, reason: BREAKPOINT
[*] Id 2, Name: "", stopped, reason: BREAKPOINT
[*] Id 3, Name: "", stopped, reason: BREAKPOINT
[*] Id 4, Name: "", stopped, reason: BREAKPOINT

```

trace

```

[*] 0x1002d40 + guardian(vector=0x21, context=0x101af28)
[*] 0x100038b + irq_entry_31()
[*] 0x1ffb000 + add_32b_PRR [eax]=0x1, 0
[*] 0x1005aa8 + kernel_init()
[*] 0x101b5e0 + add_32b_PRR [eax]=0, 0

```

Thread 1 hit Breakpoint 2, guardian (vector=0x21, context=0x101af28) at ./guard/guardian.cc:19

```

19 {
gef+

```

```

eax : 0x0101b520 - 0x00000000 -> 0x00000000
ebx : 0x01012ba8 - 0x00000000 -> 0x00000000
ecx : 0x01010870 - 0x8b535657 -> 0x8b535657
edx : 0x01ffb000 - 0x00119000 -> 0x00000000 + 0x00000000
esp : 0x0101af1c - 0x0100038b -> 0x5808c483 - 0x5808c483
ebp : 0x0101af28 - 0x0101b520 -> 0x00000000 + 0x00000000
iopl : 0x0101af28 - 0x0101b520 -> 0x00000000 - 0x00000000
edi : 0x01012ba8 - 0x00000000 -> 0x00000000
esi : 0x02011200 - 0x02011200
ebp : 0x01002d40 - 0xe8535657 -> 0xe8535657
eflags: [carry parity adjust zero sign trap interrupt direction overflow resume virtualx86 identification]
fs : 0x0008 fs : 0x0010 fs : 0x0010 fs : 0x0010 fs : 0x0010 fs : 0x0010 fs : 0x0010 fs : 0x0010

```

Registerinhalt

```

0x0101af1c +0x0000: 0x0100038b - 0x5808c483 -> 0x5808c483 - fesp
0x0101af20 +0x0004: 0x00000021 - 0x00000021
0x0101af24 +0x0008: 0x0101af28 - 0x0101b520 -> 0x00000000 + 0x00000000
0x0101af28 +0x000c: 0x0101b520 - 0x00000000 -> 0x00000000
0x0101af2c +0x0010: 0x01010870 - 0x8b535657 -> 0x8b535657
0x0101af30 +0x0014: 0x01ffb000 - 0x00119000 -> 0x00000000 + 0x00000000
0x0101af34 +0x0018: 0x010114dd - 0xe8535657 -> 0xe8535657
0x0101af38 +0x001c: 0x00000008 - 0x00000008

```

stack

Stackinhalt

```

0x1000138: 0x0 0x0
0x1000134: 0x0 0x0
0x1000130: 0x0 0x0
- 0x1002d40: <guardian+0> push edi
0x1002d44: <guardian+1> push esi
0x1002d48: <guardian+2> push ebx
0x1002d4c: <guardian+3> call 0x1010f20 (<__x86.get_pc_thunk.tb>)
0x1002d48: <guardian+5> add ebx, 0xfe60
0x1002d4e: <guardian+14> sub esp, 0x8

```

code:00_32

source:./guard/guardian.cc:13

```

14
15 #include "guard/guard.h"
16 extern Guard guardian;
17
18 extern "C" void guardian(int32_t vector, irq_context_t context)
19 {
20     (void) vector;
21     (void) context;
22     Gate* gate = plugin.report(vector);
23
24     bool wantsEpilogue = gate->prologue();

```

threads

```

[#0] Id 1, Name: "", stopped, reason: BREAKPOINT
[#1] Id 2, Name: "", stopped, reason: BREAKPOINT
[#2] Id 3, Name: "", stopped, reason: BREAKPOINT
[#3] Id 4, Name: "", stopped, reason: BREAKPOINT

```

trace

```

[#0] 0x1002d40 -> guardian(vector=0x21, context=0x101af28)
[#1] 0x100038b - irq_entry_31()
[#2] 0x1ffb000 - add %EBX, 0xFe60(%EBX)
[#3] 0x1005aa8 - kernel_init()
[#4] 0x101b5e0 - add %EBX, 0xFe60(%EBX)

```

Thread 1 hit Breakpoint 2, guardian (vector=0x21, context=0x101af28) at ./guard/guardian.cc:13

```

13 {
gef+

```

```

eax : 0x0101b520 - 0x00000000 -> 0x00000000
ebx : 0x01012ba8 - 0x00000000 -> 0x00000000
ecx : 0x01010870 - 0x8b535657 -> 0x8b535657
edx : 0x01ffb000 - 0x00119000 -> 0x00000000 + 0x00000000
esp : 0x0101af1c - 0x0100038b -> 0x5808c483 - 0x5808c483
ebp : 0x0101af20 - 0x0101b520 -> 0x00000000 + 0x00000000
edi : 0x0101af28 - 0x0101b520 -> 0x00000000 - 0x00000000
esi : 0x0101af2c - 0x000100 -> 0x8b535657 - 0x8b535657
ebp : 0x0101af30 - 0x000104 -> 0x00000000 - 0x00000000
edi : 0x0101af34 - 0x000108 -> 0xe666ff5a - 0xe666ff5a
esi : 0x0101af38 - 0x00010c -> 0x00000008 - 0x00000008
eflags: [carry parity adjust zero sign trap interrupt direction overflow resume virtualx86 identification]
fs: 0x0008 fs: 0x0010 fs: 0x0010 fs: 0x0010 fs: 0x0010 fs: 0x0010 fs: 0x0010 fs: 0x0010

```

Registerinhalt

stack

```

0x0101af1c +0x0000: 0x0100038b - 0x5808c483 -> 0x5808c483 - fesp
0x0101af20 +0x0004: 0x00000021 -> 0x00000021
0x0101af24 +0x0008: 0x0101af28 - 0x0101b520 -> 0x00000000 + 0x00000000
0x0101af28 +0x000c: 0x0101b520 -> 0x00000000 - 0x00000000
0x0101af2c +0x0010: 0x01010870 -> 0x8b535657 - 0x8b535657
0x0101af30 +0x0014: 0x01ffb000 -> 0x00119000 - 0x00000000 + 0x00000000
0x0101af34 +0x0018: 0x010114dd -> 0xe666ff5a - 0xe666ff5a
0x0101af38 +0x001c: 0x00000008 -> 0x00000008

```

Stackinhalt

code:x86:32

```

0x1002f3b xchg ax, ax
0x1002f3d xchg ax, ax
0x1002f3f nop
- 0x1002d40 <guardian+0> push edi
0x1002d41 <guardian+1> push esi
0x1002d42 <guardian+2> push ebx
0x1002d43 <guardian+3> call 0x1010f20 <__x86.get_pc_thunk, bx>
0x1002d48 <guardian+8> add ebx, 0xfe60
0x1002d4e <guardian+14> sub esp, 0x8

```

Stelle im Assembly

source:./guard/guardian.cc:19

```

19
20 #include "guard/guard.h"
21 extern "C" void guardian(void* vector, int* context)
22 {
23     (void) vector;
24     (void) context;
25     Gate* gate = plugin.report(vector);
26     bool wantsEpilogue = gate->prologue();
27 }

```

threads

```

[*] Id 1, Name: "", stopped, reason: BREAKPOINT
[*] Id 2, Name: "", stopped, reason: BREAKPOINT
[*] Id 3, Name: "", stopped, reason: BREAKPOINT
[*] Id 4, Name: "", stopped, reason: BREAKPOINT

```

trace

```

[*] 0x1002d40 + guardian(vector=0x21, context=0x101af28)
[*] 0x100038b + irq_get_irq(31)
[*] 0x1ffb000 + add_EIP_PIR [eax]=0x11_00
[*] 0x1005aa8 + kernel_init()
[*] 0x101b5a0 + add_EIP_PIR [eax]=0x11_00

```

Thread 1 hit Breakpoint 2, guardian (vector=0x21, context=0x101af28) at ./guard/guardian.cc:19

```

19 {
gef+

```

```

eax : 0x0101b520 - 0x00000000 -> 0x00000000
ebx : 0x01012ba8 - 0x00000000 -> 0x00000000
ecx : 0x01010870 - 0x8b535657 -> 0x8b535657
edx : 0x01ffb000 - 0x00119000 -> 0x00000000 + 0x00000000
esp : 0x0101af1c - 0x0100038b -> 0x5808c483 - 0x5808c483
ebp : 0x0101af20 - 0x0101b520 -> 0x00000000 + 0x00000000
edi : 0x0101af28 - 0x0101b520 -> 0x00000000 - 0x00000000
esi : 0x0101af2c - 0x01010870 -> 0x8b535657 - 0x8b535657
edi : 0x02011200 - 0x02011200
ebp : 0x01002d40 - 0xe8535657 -> 0xe8535657
eFlags: [carry parity adjust zero sign trap interrupt direction overflow resume virtualx86 identification]
fs: 0x0008 fs: 0x0010 fs: 0x0010 fs: 0x0010 fs: 0x0010 fs: 0x0010 fs: 0x0010 fs: 0x0010

```

Registerinhalt

```

0x0101af1c +0x0000: 0x0100038b - 0x5808c483 -> 0x5808c483 - fesp
0x0101af20 +0x0004: 0x00000021 -> 0x00000021
0x0101af24 +0x0008: 0x0101af28 - 0x0101b520 -> 0x00000000 + 0x00000000
0x0101af28 +0x000c: 0x0101b520 -> 0x00000000 - 0x00000000
0x0101af2c +0x0010: 0x01010870 -> 0x8b535657 - 0x8b535657
0x0101af30 +0x0014: 0x01ffb000 -> 0x00119000 - 0x00000000 + 0x00000000
0x0101af34 +0x0018: 0x010114dd -> 0xe8535657 - 0xe8535657
0x0101af38 +0x001c: 0x00000008 -> 0x00000008

```

stack

Stackinhalt

```

0x1002f3b xchg ax, ax
0x1002f3d xchg ax, ax
0x1002f3f nop
- 0x1002d40 <guardian+0> push edi
0x1002d41 <guardian+1> push esi
0x1002d42 <guardian+2> push ebx
0x1002d43 <guardian+3> call 0x1010f20 <__x86.get_pc_thunk,bx>
0x1002d48 <guardian+8> add ebx, 0xfe60
0x1002d4e <guardian+14> sub esp, 0x8

```

code:x86:32

Stelle im Assembly

source:./guard/guardian.cc:19

```

14
15 #include "guard/guard.h"
16 extern Guard guard;
17
18 extern "C" void guardian(uint32_t vector, irq_context *context)
- 19 {
20     (void) vector;
21     (void) context;
22     Gate* gate = plugin.report(vector);
23
24     bool wantsEpilogue = gate->prologue();

```

Stelle in C/C++

```

[*0] Id 1, Name: "", stopped, reason: BREAKPOINT
[*1] Id 2, Name: "", stopped, reason: BREAKPOINT
[*2] Id 3, Name: "", stopped, reason: BREAKPOINT
[*3] Id 4, Name: "", stopped, reason: BREAKPOINT

```

threads

```

[*0] 0x1002d40 + guardian(vector=0x21, context=0x101af28)
[*1] 0x100038b + irq_get(31)
[*2] 0x1ffb000 + add EBX, 0xFe60 [eax=0]
[*3] 0x1005aa8 + kernel_init()
[*4] 0x101b5e0 + add EBX, 0x8 [eax=0]

```

trace

```

Thread 1 hit Breakpoint 2, guardian(vector=0x21, context=0x101af28) at ./guard/guardian.cc:19
19 {
gef+

```

```

eax : 0x0101b520 - 0x00000000 - 0x00000000
ebx : 0x01012ba8 - 0x00000000 - 0x00000000
ecx : 0x01010870 - 0x8b535657 - 0x8b535657
edx : 0x01ffb000 - 0x00119000 - 0x00000000 + 0x00000000
esp : 0x0101af1c - 0x0100038b - 0x5808c483 - 0x5808c483
ebp : 0x0101af28 - 0x0101b520 - 0x00000000 + 0x00000000
edi : 0x0101af2c - 0x00000000 - 0x00000000
esi : 0x0101af30 - 0x00000000 - 0x00000000
ebp : 0x0101af34 - 0x00000000 - 0x00000000
edi : 0x02011200 - 0x02011200
ebp : 0x01002d40 - 0xe8535657 - 0xe8535657
eflags: [carry parity adjust zero sign trap interrupt direction overflow resume virtualx86 identification]
fs: 0x0008 fs: 0x0010 fs: 0x0010 fs: 0x0010 fs: 0x0010 fs: 0x0010 fs: 0x0010 fs: 0x0010

```

Registerinhalt

```

0x0101af1c +0x0000: 0x0100038b - 0x5808c483 - 0x5808c483 - fesp
0x0101af20 +0x0004: 0x00000021 - 0x00000021
0x0101af24 +0x0008: 0x0101af28 - 0x0101b520 - 0x00000000 + 0x00000000
0x0101af28 +0x000c: 0x0101b520 - 0x00000000 - 0x00000000
0x0101af2c +0x0010: 0x01010870 - 0x8b535657 - 0x8b535657
0x0101af30 +0x0014: 0x01ffb000 - 0x00119000 - 0x00000000 + 0x00000000
0x0101af34 +0x0018: 0x010114dd - 0xe8535657 - 0xe8535657
0x0101af38 +0x001c: 0x00000008 - 0x00000008

```

Stackinhalt

```

0x100213b xchg ax, ax
0x100213d xchg ax, ax
0x100213f nop
- 0x1002d40 <guardian+0> push edi
0x1002d41 <guardian+1> push esi
0x1002d42 <guardian+2> push ebx
0x1002d43 <guardian+3> call 0x1010f20 <__x86.get_pc_thunk,bx>
0x1002d48 <guardian+8> add ebx, 0xfe60
0x1002d4e <guardian+14> sub esp, 0x8

```

Stelle im Assembly

```

14
15 #include "guard/guard.h"
16 extern Guard guard;
17
18 extern "C" void guardian(uint32_t vector, irq_context *context)
- 19 {
20     (void) vector;
21     (void) context;
22     Gate* gate = plugin.report(vector);
23
24     bool wantsEpilogue = gate->prologue();

```

Stelle in C/C++

```

[#0] Id 1, Name: "", stopped, reason: BREAKPOINT
[#1] Id 2, Name: "", stopped, reason: BREAKPOINT
[#2] Id 3, Name: "", stopped, reason: BREAKPOINT
[#3] Id 4, Name: "", stopped, reason: BREAKPOINT

```

Threads

```

[#0] 0x1002d40 - guardian(vector=0x21, context=0x101af28)
[#1] 0x100038b - irq_get(3)
[#2] 0x1ffb000 - add_printf_fmt_args(0x11, 0)
[#3] 0x1005aa8 - kernel_init()
[#4] 0x101b5e0 - add_printf_fmt_args(1, 0)

```

Thread 1 hit Breakpoint 2, guardian(vector=0x21, context=0x101af28) at ./guard/guardian.cc:19

```

19 {
gef+  █

```

```

eax : 0x0101b520 - 0x00000000 - 0x00000000
ebx : 0x01012ba8 - 0x00000000 - 0x00000000
ecx : 0x01010870 - 0x8b535657 - 0x8b535657
edx : 0x01ffb000 - 0x00119000 - 0x00000000 + 0x00000000
esp : 0x0101af1c - 0x0100038b - 0x5808c483 - 0x5808c483
ebp : 0x0101af20 - 0x0101b520 - 0x00000000 + 0x00000000
edi : 0x0101af28 - 0x0101b520 - 0x00000000 - 0x00000000
esi : 0x0101af2c - 0x01010870 - 0x8b535657 - 0x8b535657
ebp : 0x0101af30 - 0x00119000 - 0x00000000 - 0x00000000
edi : 0x0101af34 - 0x00119000 - 0x00000000 - 0x00000000
esi : 0x0101af38 - 0x00000008 - 0x00000008
eflags: [carry parity adjust zero sign trap interrupt direction overflow resume virtualx86 identification]
fs: 0x0008 fs: 0x0010 fs: 0x0010 fs: 0x0010 fs: 0x0010 fs: 0x0010 fs: 0x0010 fs: 0x0010

```

Registerinhalt

```

0x0101af1c +0x0000: 0x0100038b - 0x5808c483 - 0x5808c483 - fesp
0x0101af20 +0x0004: 0x00000021 - 0x00000021
0x0101af24 +0x0008: 0x0101af28 - 0x0101b520 - 0x00000000 + 0x00000000
0x0101af28 +0x000c: 0x0101b520 - 0x00000000 - 0x00000000
0x0101af2c +0x0010: 0x01010870 - 0x8b535657 - 0x8b535657
0x0101af30 +0x0014: 0x01ffb000 - 0x00119000 - 0x00000000 + 0x00000000
0x0101af34 +0x0018: 0x010114dd - 0xe96bffa5 - 0xe96bffa5
0x0101af38 +0x001c: 0x00000008 - 0x00000008

```

Stackinhalt

```

0x1002f3b xchg ax, ax
0x1002f3d xchg ax, ax
0x1002f3f nop
- 0x1002d40 <guardian+0> push edi
0x1002d41 <guardian+1> push esi
0x1002d42 <guardian+2> push ebx
0x1002d43 <guardian+3> call 0x1010f20 <__x86.get_pc_thunk,bx>
0x1002d48 <guardian+8> add ebx, 0xfe60
0x1002d4e <guardian+14> sub esp, 0x8

```

Stelle im Assembly

```

14
15 #include "guard/guard.h"
16 extern Guard guard;
17
18 extern "C" void guardian(uint32_t vector, irq_context *context)
- 19 {
20     (void) vector;
21     (void) context;
22     Gate* gate = plugin.report(vector);
23
24     bool wantsEpilogue = gate->prologue();

```

Stelle in C/C++

```

[#0] Id 1, Name: "", stopped, reason: BREAKPOINT
[#1] Id 2, Name: "", stopped, reason: BREAKPOINT
[#2] Id 3, Name: "", stopped, reason: BREAKPOINT
[#3] Id 4, Name: "", stopped, reason: BREAKPOINT

```

Threads

```

[#0] 0x1002d40 - guardian(vector=0x21, context=0x101af28)
[#1] 0x100038b - irq_entry_33()
[#2] 0x1ffb000 - add BYTE PTR [eax+0xd1], dl
[#3] 0x1005aa8 - kernel_init()
[#4] 0x101b5e0 - add BYTE PTR [eax], al

```

Backtrace

```

Thread 1 hit Breakpoint 2, guardian(vector=0x21, context=0x101af28) at ./guard/guardian.cc:19
19 {
gef+

```



```

eax : 0x0101b520 - 0x00000000 - 0x00000000
ebx : 0x01012ba8 - 0x00000000 - 0x00000000
ecx : 0x01010870 - 0x8b535657 - 0x8b535657
edx : 0x01ffb000 - 0x00119000 - 0x00000000 + 0x00000000
esp : 0x0101af1c - 0x0100038b - 0x5808c483 - 0x5808c483
ebp : 0x0101af20 - 0x0101b520 - 0x00000000 + 0x00000000
edi : 0x0101af28 - 0x0101b520 - 0x00000000 - 0x00000000
esi : 0x0101af2c - 0x01010870 - 0x8b535657 - 0x8b535657
ebp : 0x0101af30 - 0x0101b520 - 0x00000000 - 0x00000000
edi : 0x0101af34 - 0x010114dd - 0xe96bffa5 - 0xe96bffa5
esi : 0x0101af38 - 0x00000008 - 0x00000008
eflags: [carry parity adjust zero sign trap interrupt direction overflow resume virtualx86 identification]
fs: 0x0008 fs: 0x0010 fs: 0x0010 fs: 0x0010 fs: 0x0010 fs: 0x0010 fs: 0x0010 fs: 0x0010

```

Registerinhalt

```

0x0101af1c +0x0000: 0x0100038b - 0x5808c483 - 0x5808c483 - fesp
0x0101af20 +0x0004: 0x00000021 - 0x00000021
0x0101af24 +0x0008: 0x0101af28 - 0x0101b520 - 0x00000000 + 0x00000000
0x0101af28 +0x000c: 0x0101b520 - 0x00000000 - 0x00000000
0x0101af2c +0x0010: 0x01010870 - 0x8b535657 - 0x8b535657
0x0101af30 +0x0014: 0x01ffb000 - 0x00119000 - 0x00000000 + 0x00000000
0x0101af34 +0x0018: 0x010114dd - 0xe96bffa5 - 0xe96bffa5
0x0101af38 +0x001c: 0x00000008 - 0x00000008

```

Stackinhalt

```

0x1002f3b xchg ax, ax
0x1002f3d xchg ax, ax
0x1002f3f nop
- 0x1002d40 <guardian+0> push edi
0x1002d41 <guardian+1> push esi
0x1002d42 <guardian+2> push ebx
0x1002d43 <guardian+3> call 0x1010f20 <__x86.get_pc_thunk,bx>
0x1002d48 <guardian+8> add ebx, 0xfe60
0x1002d4e <guardian+14> sub esp, 0x8

```

Stelle im Assembly

```

14
15 #include "guard/guard.h"
16 extern Guard guard;
17
18 extern "C" void guardian(uint32_t vector, irq_context *context)
- 19 {
20     (void) vector;
21     (void) context;
22     Gate* gate = plugin.report(vector);
23
24     bool wantsEpilogue = gate->prologue();

```

Stelle in C/C++

```

[#0] Id 1, Name: "", stopped, reason: BREAKPOINT
[#1] Id 2, Name: "", stopped, reason: BREAKPOINT
[#2] Id 3, Name: "", stopped, reason: BREAKPOINT
[#3] Id 4, Name: "", stopped, reason: BREAKPOINT

```

Threads

```

[#0] 0x1002d40 - guardian(vector=0x21, context=0x101af28)
[#1] 0x100038b - irq_entry_33()
[#2] 0x1ffb000 - add BYTE PTR [eax+0xd1], dl
[#3] 0x1005aa8 - kernel_init()
[#4] 0x101b5e0 - add BYTE PTR [eax], al

```

Backtrace

```

Thread 1 hit Breakpoint 2, guardian (vector=0x21, context=0x101af28) at ./guard/guardian.cc:19
19 {
gef- █

```

Eingabezeile

Breakpoints: (gdb) break <location>

Breakpoints

Unterbrechen der Ausführung, sobald eine bestimmte **Codestelle** erreicht wird.

- Funktionsname
 - absolute/relative Codezeile
 - *Adresse
- } optionaler Prefix: Quelldatei

Unterbricht vor dem Ausführen von...

```
(gdb) b main
```

Funktion main

```
(gdb) b main.cc:main
```

... aus main.cc

```
(gdb) b 63
```

Zeile 63 in aktueller Datei

```
(gdb) b main.cc:63
```

Zeile 63 in main.cc

```
(gdb) b +3
```

In 3 Zeilen

```
(gdb) b *0x100a9ca
```

An Adresse 0x100a9ca

Temporäre & Bedingte Breakpoints

Temporäre Breakpoints: `(gdb) tbreak <location>`

Werden nach dem 1. Auslösen entfernt, sonst wie „normale“ Breakpoints.

Temporäre & Bedingte Breakpoints

Temporäre Breakpoints: `(gdb) tbreak <location>`

Werden nach dem 1. Auslösen entfernt, sonst wie „normale“ Breakpoints.

Bedingte Breakpoints: `(gdb) break <location> if <cond>`

Unterbrechung nur falls Bedingung erfüllt ist, z.B:

```
(gdb) break interrupt_handler if vector == 33
```

Unterbricht nur, falls die Funktion `interrupt_handler` aufgrund von Tastatureingabe (Vektor 33) betreten wurde.

Temporäre & Bedingte Breakpoints

Temporäre Breakpoints: `(gdb) tbreak <location>`

Werden nach dem 1. Auslösen entfernt, sonst wie „normale“ Breakpoints.

Bedingte Breakpoints: `(gdb) break <location> if <cond>`

Unterbrechung nur falls Bedingung erfüllt ist, z.B:

```
(gdb) break interrupt_handler if vector == 33
```

Unterbricht nur, falls die Funktion `interrupt_handler` aufgrund von Tastatureingabe (Vektor 33) betreten wurde.



Achtung: Nichttriviale Break- oder Watchpoints werden ohne Hardwareunterstützung umgesetzt → Langsam!

Watchpoints (Data Breakpoints)

Unterbricht wenn Speicherbereich geschrieben (oder gelesen) wird:

watch <location> Schreibzugriff

rwatch <location> Lesezugriff

awatch <location> Schreib- oder Lesezugriff

```
(gdb) watch guard
```

```
(gdb) watch guard if guard.locked == 1
```

Verwalten von Break-/Watchpoints

ignore	<id> <N>	Breakpoint N mal ignorieren
enable	<id> <id> ..	Breakpoints aktivieren
disable	<id> <id> ..	Breakpoints deaktivieren
delete	<id> <id> ..	Breakpoints löschen

Schrittweise Ausführung

`step count` – Nächste Zeile

`stepi count` – Nächste Instruktion

`next count` – Nächste Zeile (ohne Funktionen zu betreten)

`nexti count` – Nächste Instruktion (ohne Funktionen zu betreten)

`until count` – Wiederhole `next` bis zur **textuell** nächsten Zeile

↑
Optional: Anzahl Wiederholungen

`finish` – Bis zum return des aktuellen Stackframes

`advance <location>` – Bis zu <location>

`continue` – Ausführung (zum nächsten Breakpoint) fortsetzen



Der skip Befehl

```
01 unsigned int numCPUs = System::getNumberOfCPUs();  
02 kout << "numCPUs:_" << numCPUs << endl;  
03 ApplicationProcessor::boot();
```

Der skip Befehl

```
01 unsigned int numCPUs = System::getNumberOfCPUs();  
02 kout << "numCPUs:_" << numCPUs << endl;  
03 ApplicationProcessor::boot();
```

Übergehe eine einzelne Funktion:

```
(gdb) skip Guard::enter
```

Übergehe alle Funktionen aus einer Datei:

```
(gdb) skip file object/outputstream.cc
```

Der info Befehl

<code>info locals</code>	Auflistung aller lokalen Variablen
<code>info registers</code>	Auflistung der Registerwerte
<code>info breakpoints</code>	Auflistung der Breakpoints
<code>info threads</code>	Auflistung der Threads
<code>info skip</code>	Auflistung der übersprungenen Funktionen
...	siehe <code>(gdb) info</code>

Ausgabe von Werten in Speicher / Register

```
(gdb) print/<Format> <Ausdruck>
```

```
(gdb) x/<Anzahl><Format> <Ausdruck>
```

Werte für <Format>

x	Ganzzahl (hex)	a	Adresse (hex) + Offset zum Startsymbol
d	Ganzzahl (mit VZ, dezimal)	f	Float
u	Ganzzahl (ohne VZ, dezimal)	i	Instruktion
t	Ganzzahl (binär) (two)		

Bestimmen des Typs eines Symbols

```
ptype <Symbol>
```

Verändern des Zielsystems: (gdb) set

Verändern eines Registers

```
(gdb) set $esp = oxdeadbeef
```

Verändern einer Variable / Speicherbereichs

```
(gdb) set numCPUs = 2
```

```
(gdb) set *((int *) 0x1013fdc) = 42
```

Generell: Optimierungen sind doof fürs Debuggen:

- Inlining von Funktionen
- Elimination von Variablen
- ...

Relevante Compileroptionen

- g Generiere Debuginformationen
- O0 Optimierungen aus
- Og Nur Optimierungen, die das Debuggen nicht stören

Generell: Optimierungen sind doof fürs Debuggen:

- Inlining von Funktionen
- Elimination von Variablen
- ...

Relevante Compileroptionen

- g Generiere Debuginformationen
- O0 Optimierungen aus
- Og Nur Optimierungen, die das Debuggen nicht stören
- O2 Fast alle Optimierungen

Die bittere Wahrheit...

Ihr werdet debuggen müssen

Die bittere Wahrheit...

Ihr werdet debuggen müssen

Anlegen einer eigenen `.gdbinit`:

Die bittere Wahrheit...

Ihr werdet debuggen müssen

Anlegen einer eigenen `.gdbinit`:

- Auf *mars* oder *syllabXX* im Labor:

```
cp /fs/stubs/tools/gdbinit ~/.gdbinit
```

Die bittere Wahrheit...

Ihr werdet debuggen müssen

Anlegen einer eigenen `.gdbinit`:

- Auf *mars* oder *syllabXX* im Labor:

```
cp /fs/stubs/tools/gdbinit ~/.gdbinit
```

- Lokal:

- `scp mars.cs.tu-dortmund.de:/fs/stubs/tools/gdbinit
~/.gdbinit`

- `scp ↵
mars.cs.tu-dortmund.de:/fs/stubs/tools/gdbinit-gef.py
~/gdbinit-gef.py`

- Pfad in `~/gdbinit` anpassen

Die bittere Wahrheit...

Ihr werdet debuggen müssen

- `make qemu-gdb-noopt`
- Profit?
- `make qemu-gdb`



Fragen?



Snippet: Ausgabe von Details zur Exception im Interrupt Handler

```
01 // Optional: Print exceptions on DBG stream to support debugging
02 if (vector < Core::Interrupt::EXCEPTIONS) {
03     dout << "Exception_" << dec << vector;
04     switch (vector) {
05         case 0:  dout << "_(Div_by_0)"; break;
06         case 6:  dout << "_(InvalidOpcode)"; break;
07         case 10: dout << "_(InvalidTSS)"; break;
08         case 13: dout << "_(GeneralProtectionFault)"; break;
09         case 14: dout << "_(PageFault)"; break;
10         default: break;
11     }
12     if (context->error_code != 0) {
13         dout << "_[" << bin << context->error_code << "];"
14     }
15     dout << "_@" << hex << context->ip << flush;
16     dout << endl;
17     Core::die();
18 }
```