

Übung zu Betriebssystembau

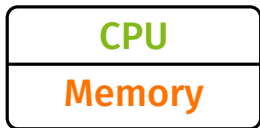
Kontextwechsel

09. Dezember 2025

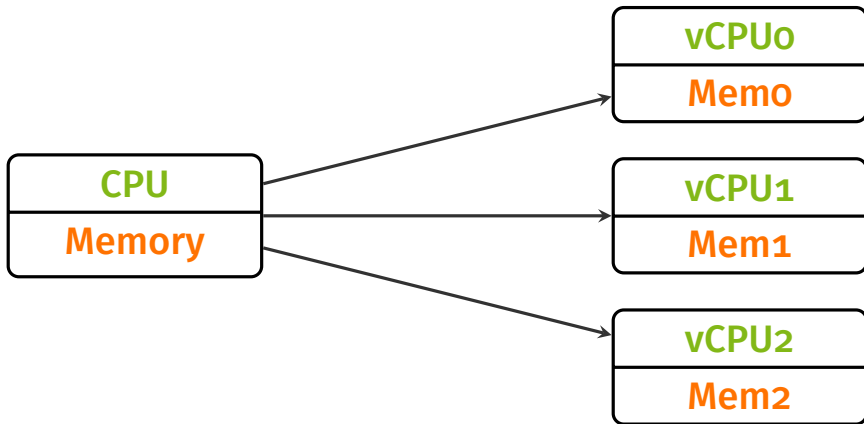
Alexander Krause

Arbeitsgruppe Systemsoftware
Technische Universität Dortmund

(Mit Material vom Lehrstuhl 4 der FAU)



tatsächliche
Hardware



tatsächliche
Hardware



illusionierte
Hardware

■ Speicher virtualisieren

■ CPU virtualisieren

- nicht teilbar im Ort
- aber teilbar in der Zeit

■ Speicher virtualisieren ist einfach:

```
#define MEMSIZE 100  
char Mem0[MEMSIZE];  
char Mem1[MEMSIZE];  
char Mem2[MEMSIZE];
```

Memory

...

Memo

Mem1

Mem2

...

■ CPU virtualisieren

- nicht teilbar im Ort
- aber teilbar in der Zeit

Motivation: mehr Aktivitätsträger als CPUs

```
1 void foo(){
2     int f = 42;
3
4     while (f--){
5         kout << "foo"
6             << f
7             << endl;
8
9     }
10
11 }
```

```
1 void bar(){
2     int b = 23;
3
4     while (b--){
5         kout << "bar"
6             << b
7             << endl;
8
9     }
10
11 }
```



Einseitiger Aufruf

```
1 void foo(){
2     int f = 42;
3
4     while (f--){
5         kout << "foo"
6             << f
7             << endl;
8
9     }
10    bar();
11 }
```

```
1 void bar(){
2     int b = 23;
3
4     while (b--){
5         kout << "bar"
6             << b
7             << endl;
8
9     }
10
11 }
```



foo41

foo40

...

foo1

foo0



foo41

foo40

...

foo1

foo0

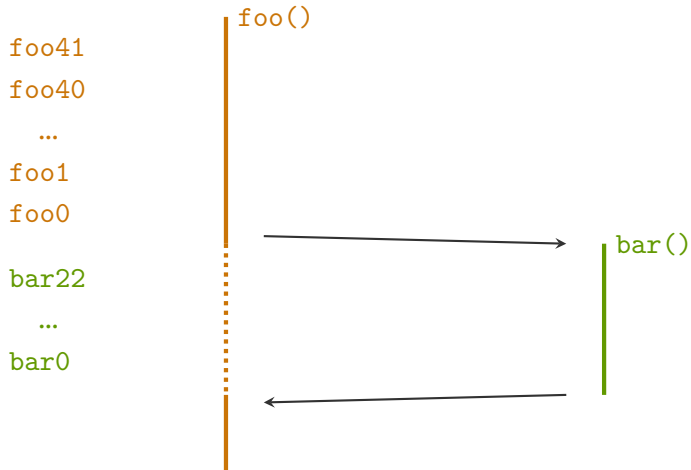
bar22

...

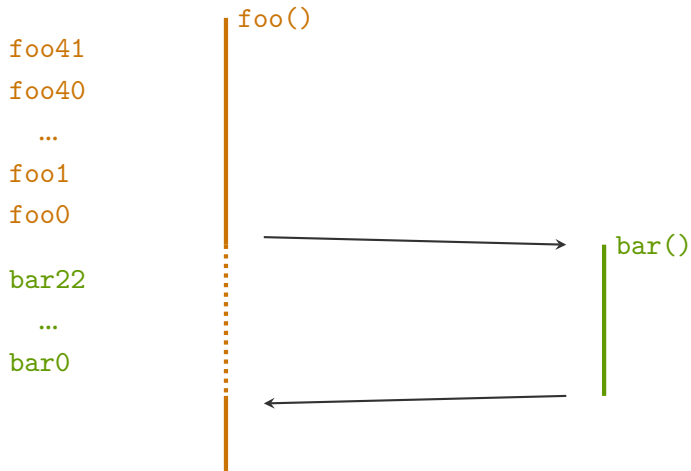
bar0



Einseitiger Aufruf



Einseitiger Aufruf



⚡ nicht parallel



Gegenseitiger Aufruf

```
1 void foo(){
2     static int f = 42;
3
4     while (f--){
5         kout << "foo"
6             << f
7             << endl;
8         bar();
9     }
10
11 }
```

```
1 void bar(){
2     static int b = 23;
3
4     while (b--){
5         kout << "bar"
6             << b
7             << endl;
8         foo();
9     }
10
11 }
```



foo41

bar22

foo40

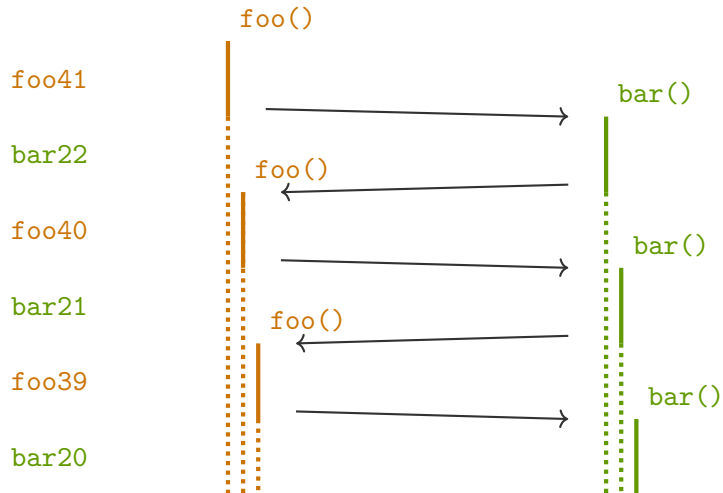
bar21

foo39

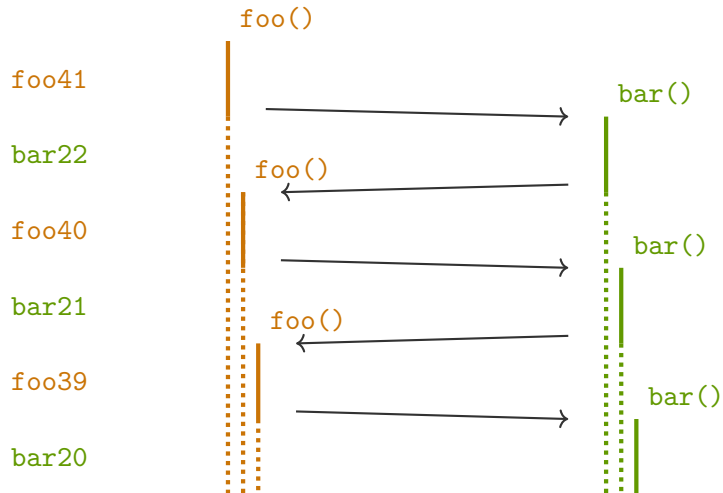
bar20



Gegenseitiger Aufruf



Gegenseitiger Aufruf



⚡ hoher Speicherverbrauch & (ggf.) Endlosrekursion



Umschaltung



Umschaltung



Kontrollflusszustand **sichern** & **laden**





Kontrollflusszustand **sichern** & **laden**

- Stack
- Register

Umschaltung



Kontrollflusszustand **sichern** & **laden**

- Stack
 - Register
- } Kontext



Umschaltung



Kontrollflusszustand **sichern** & **laden**

- Stack
 - Register
 - Umschaltefunktion
- } Kontext

Umschaltung



Kontrollflusszustand **sichern** & **laden**

- Stack
 - Register
 - Umschaltefunktion
- } Kontext



Umschaltung

```
1 void foo(){
2     int f = 42;
3
4     while (f--){
5         kout << "foo"
6             << f
7             << endl;
8         context_switch(
9             &stack_foo,
10            &stack_bar
11        );
12    }
13 }
```

```
1 void bar(){
2     int b = 23;
3
4     while (b--){
5         kout << "bar"
6             << b
7             << endl;
8         context_switch(
9             &stack_bar,
10            &stack_foo
11        );
12    }
13 }
```



foo41

bar22

foo40

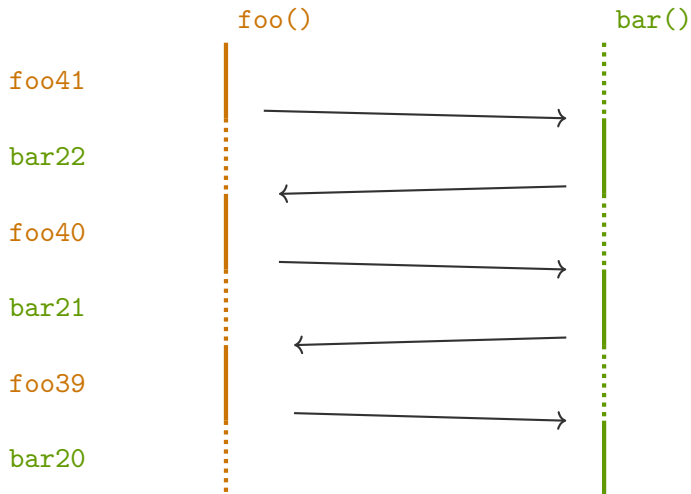
bar21

foo39

bar20

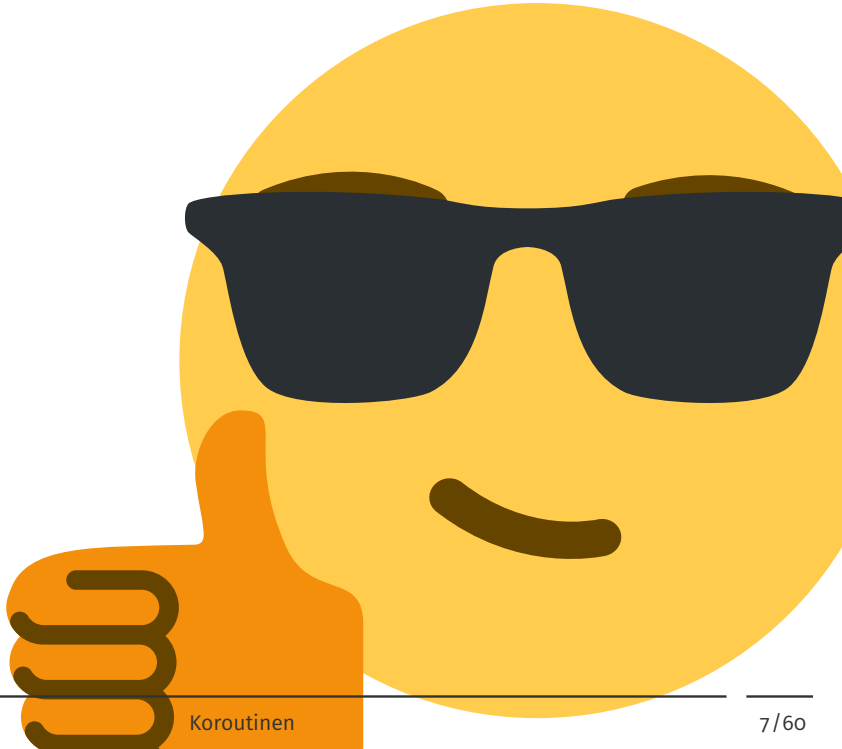


Umschaltung



Genau was wir wollen.

Bild: twemoji (modifiziert)



Ablauf einer Übersetzung

```
1 #include "user/app1/appl.h"
2 #include "device/textstream.h"
3 #include "interrupt/guarded.h"
4 #include "machine/core.h"
5
6 extern TextStream kout;
7
8 void Application::action() {
9     unsigned core = Core::getID();
10    for (unsigned n = 0; ; ++n) {
11        Guarded section;
12        kout.setPos(0, id);
13        kout << n;
14        kout.flush();
15    }
16 }
```



Ablauf einer Übersetzung

```
1 #include "user/app1/appl.h"
2 #include "device/textstream.h"
3 #include "interrupt/guarded.h"
4 #include "machine/core.h"
5
6 extern TextStream kout;
7
8 void Application::action() {
9     unsigned core = Core::getID();
10    for (unsigned n = 0; ; ++n) {
11        Guarded section;
12        kout.setPos(0, id);
13        kout << n;
14        kout.flush();
15    }
16 }
```

höhere Programmiersprache (3. Generation)



Ablauf einer Übersetzung

```
1 #include "user/app1/appl.h"
2 #include "device/textstream.h"
3 #include "interrupt/guarded.h"
4 #include "machine/core.h"
5
6 extern TextStream kout;
7
8 void Application::action() {
9     unsigned core = Core::getID();
10    for (unsigned n = 0; ; ++n) {
11        Guarded section;
12        kout.setPos(0, id);
13        kout << n;
14        kout.flush();
15    }
16 }
```

höhere Programmiersprache (3. Generation)



Ablauf einer Übersetzung

```
1 #include "user/app1/appl.h"
2 #include "device/textstream.h"
3 #include "interrupt/guarded.h"
4 #include "machine/core.h"
5
6 extern TextStream kout;
7
8 void Application::action() {
9     unsigned core = Core::getID();
10    for (unsigned n = 0; ; ++n) {
11        Guarded section;
12        kout.setPos(0, id);
13        kout << n;
14        kout.flush();
15    }
16 }
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

```
f3 0f 1e fa 55 53 31 db
48 83 ec 08 e8 4f d1 ff
ff 89 c5 0f 1f 44 00 00
e8 e3 c3 ff ff 89 ea 31
f6 bf 84 45 02 01 e8 e5
d2 ff ff 89 de bf 20 45
02 01 83 c3 01 e8 c6 fd
ff ff bf 20 45 02 01 e8
1c bf ff ff e8 37 c4 ff
ff eb cd
```

höhere Programmiersprache
(3. Generation)



Ablauf einer Übersetzung

```
1 #include "user/app1/appl.h"
2 #include "device/textstream.h"
3 #include "interrupt/guarded.h"
4 #include "machine/core.h"
5
6 extern TextStream kout;
7
8 void Application::action() {
9     unsigned core = Core::getID();
10    for (unsigned n = 0; ; ++n) {
11        Guarded section;
12        kout.setPos(0, id);
13        kout << n;
14        kout.flush();
15    }
16 }
```

höhere Programmiersprache
(3. Generation)

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

```
f3 0f 1e fa 55 53 31 db
48 83 ec 08 e8 4f d1 ff
ff 89 c5 0f 1f 44 00 00
e8 e3 c3 ff ff 89 ea 31
f6 bf 84 45 02 01 e8 e5
d2 ff ff 89 de bf 20 45
02 01 83 c3 01 e8 c6 fd
ff ff bf 20 45 02 01 e8
1c bf ff ff e8 37 c4 ff
ff eb cd
```

Maschinensprache
(1. Generation)

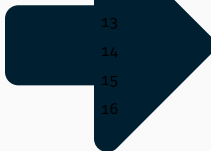


Ablauf einer Übersetzung

```
1 #include "user/app1/appl.h"
2 #include "device/textstream.h"
3 #include "interrupt/guarded.h"
4 #include "machine/core.h"
5
6 extern TextStream kout;
7
8 void Application::action() {
9     unsigned core = Core::getID();
10    for (unsigned n = 0; ; ++n) {
11        Guarded section;
12        kout.setPos(0, id);
13        kout << n;
14        kout.flush();
15    }
16 }
```

höhere Programmiersprache
(3. Generation)

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
```



```
f3 0f 1e fa 55 53 31 db
48 83 ec 08 e8 4f d1 ff
ff 89 c5 0f 1f 44 00 00
e8 e3 c3 ff ff 89 ea 31
f6 bf 84 45 02 01 e8 e5
d2 ff ff 89 de bf 20 45
02 01 83 c3 01 e8 c6 fd
ff ff bf 20 45 02 01 e8
1c bf ff ff e8 37 c4 ff
ff eb cd
```

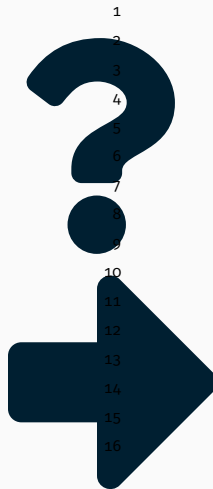
Maschinensprache
(1. Generation)



Ablauf einer Übersetzung

```
1 #include "user/app1/appl.h"
2 #include "device/textstream.h"
3 #include "interrupt/guarded.h"
4 #include "machine/core.h"
5
6 extern TextStream kout;
7
8 void Application::action() {
9     unsigned core = Core::getID();
10    for (unsigned n = 0; ; ++n) {
11        Guarded section;
12        kout.setPos(0, id);
13        kout << n;
14        kout.flush();
15    }
16 }
```

höhere Programmiersprache
(3. Generation)



```
f3 0f 1e fa 55 53 31 db
48 83 ec 08 e8 4f d1 ff
ff 89 c5 0f 1f 44 00 00
e8 e3 c3 ff ff 89 ea 31
f6 bf 84 45 02 01 e8 e5
d2 ff ff 89 de bf 20 45
02 01 83 c3 01 e8 c6 fd
ff ff bf 20 45 02 01 e8
1c bf ff ff e8 37 c4 ff
ff eb cd
```

Maschinensprache
(1. Generation)



Ablauf einer Übersetzung

```
1 #include "user/app1/appl.h"
2 #include "device/textstream.h"
3 #include "interrupt/guarded.h"
4 #include "machine/core.h"
5
6 extern TextStream kout;
7
8 void Application::action() {
9     unsigned core = Core::getID();
10    for (unsigned n = 0; ; ++n) {
11        Guarded section;
12        kout.setPos(0, id);
13        kout << n;
14        kout.flush();
15    }
16 }
```



Vererbung auflösen

```
1 class TextStream : public OutputStream, public TextWindow {
2     // ...
3
4 } kout;
5
6 void Application::action() {
7     unsigned n = 0;
8     unsigned core;
9     core = Core::getID();
10    while(true) {
11        Guard::enter();
12        kout.setPos(0, id);
13        unsigned i = n
14        n += 1;
15        kout.operator<<(i);
16        kout.flush();
17        Guard::leave();
18    }
19 }
```



Methoden zu Funktionen „umwandeln“

```
1 class TextStream : public OutputStream, public TextWindow {
2     // ...
3
4 } kout;
5
6 void Application::action(Application * this) {
7     unsigned n = 0;
8     unsigned core;
9     core = Core::getID();
10    while(true) {
11        Guard::enter();
12        TextWindow::setPos(&kout, 0, id);
13        unsigned i = n
14        n += 1;
15        OutputStream::operator<<(&kout, i);
16        TextStream::flush(&kout);
17        Guard::leave();
18    }
19 }
```



Objekte zu Basisklassen

```
1 class TextStream {
2     class OutputStream { ... };
3     class TextWindow { ... };
4 } kout;
5
6 void Application::action(Application * this) {
7     unsigned n = 0;
8     unsigned core;
9     core = Core::getID();
10    while(true) {
11        Guard::enter();
12        TextWindow::setPos(&(kout.TextWindow), 0, id);
13        unsigned i = n
14        n += 1;
15        OutputStream::operator<<(&(kout.OutputStream), i);
16        TextStream::flush(&(kout.OutputStream));
17        Guard::leave();
18    }
19 }
```



Offsets brechnen und einsetzen

```
1 class TextStream {
2     class OutputStream { ... }; // offset 0
3     class TextWindow { ... };   // offset 64
4 } kout;
5
6 void Application::action(Application * this) {
7     unsigned n = 0;
8     unsigned core;
9     core = Core::getID();
10    while(true) {
11        Guard::enter();
12        TextWindow::setPos(&kout+64, 0, id);
13        unsigned i = n
14        n += 1;
15        OutputStream::operator<<(&kout+0, i);
16        TextStream::flush(&kout+0);
17        Guard::leave();
18    }
19 }
```



name mangling in C++

```
1 class TextStream {
2     class OutputStream { ... }; // offset 0
3     class TextWindow { ... };   // offset 64
4 } kout;
5
6 void _ZN11Application6actionEv(Application * this) {
7     unsigned n = 0;
8     unsigned core;
9     core = _ZN4Core5getIDEv();
10    while(true) {
11        _ZN5Guard5enterEv();
12        _ZN10TextWindow6setPosEjj(&kout+64, 0, id);
13        unsigned i = n
14        n += 1;
15        _ZN12OutputStream1sEj(&kout, i);
16        _ZN10TextStream5flushEv(&kout);
17        _ZN5Guard5leaveEv();
18    }
19 }
```



```
1      void _ZN11Application6actionEv(Application * this) {
2
3
4
5          unsigned n = 0
6          unsigned core;
7          core = _ZN4Core5getIDEv();
8
9          while(true) {
10             _ZN5Guard5enterEv();
11
12
13
14             _ZN10TextWindow6setPosEjj(&kout+64, 0, core)
15             unsigned i = n
16
17             n += 1;
18             _ZN12OutputStreamlsEj(&kout, i);
19
20             _ZN10TextStream5flushEv(&kout);
21             _ZN5Guard5leaveEv();
22         }
23     }
```



```
1      _ZN11Application6actionEv:
2      endbr64
3      push    rbp
4      push    rbx
5      xor     ebx, ebx
6      sub     rsp, 0x8
7      call    _ZN4Core5getIDEv
8      mov     ebp, eax
9  LOOP:
10     call    _ZN5Guard5enterEv
11     mov     edx, ebp
12     xor     esi, esi
13     mov     edi, kout+64
14     call    _ZN10TextWindow6setPosEjj
15     mov     esi, ebx
16     mov     edi, kout
17     add     ebx, 0x1
18     call    _ZN12OutputStream5Ej
19     mov     edi, kout
20     call    _ZN10TextStream5flushEv
21     call    _ZN5Guard5leaveEv
22     jmp     LOOP
23
```

```
1      _ZN11Application6actionEv:  
2      endbr64  
3      push    rbp  
4      push    rbx  
5      xor     ebx, ebx  
6      sub     rsp, 0x8  
7      call    _ZN4Core5getIdEv  
8      mov     ebp, eax  
9  LOOP:  
10     call    _ZN5Guard5enterEv  
11     mov     edx, ebp  
12     xor     esi, esi  
13     mov     edi, kout+64  
14     call    _ZN10TextWindow6setPosEjj  
15     mov     esi, ebx  
16     mov     edi, kout  
17     add     ebx, 0x1  
18     call    _ZN12OutputStream5Ej  
19     mov     edi, kout  
20     call    _ZN10TextStream5flushEv  
21     call    _ZN5Guard5leaveEv  
22     jmp     LOOP  
23
```

Assembler

- Sprache der 2. Gen.
(maschinenorientiert)
- **Mnemonics** statt Bytes
- 1:1-Übersetzung
- für x64-Architektur
Dokumentation im
Intel-Manual, Vol. 2



Intel® 64 and IA-32 Architectures
Software Developer's Manual

Volume 2 (2A, 2B, 2C & 2D):
Instruction Set Reference, A-Z

NOTE: The Intel 64 and IA-32 Architectures Software Developer's Manual consists of four volumes: Basic Architecture, Order Number 253665; Instruction Set Reference A-Z, Order Number 325383; System Programming Guide, Order Number 314518-04; Model-Specific Registers, Order Number 315592. Refer to all four volumes when evaluating your design needs.



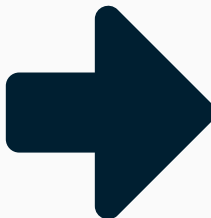
```
1 0100ba80 <_ZN11Application6actionEv>:
2 100ba80:     endbr64
3 100ba84:     push    rbp
4 100ba85:     push    rbx
5 100ba86:     xor     ebx, ebx
6 100ba88:     sub     rsp, 0x8
7 100ba8c:     call    1008be0 <_ZN4Core5getIDEv>
8 100ba91:     mov     ebp, eax
9 100ba93:     nop     DWORD PTR [rax+rax*1+0x0]
10 100ba98:     call    1007e80 <_ZN5Guard5enterEv>
11 100ba9d:     mov     edx, ebp
12 100ba9f:     xor     esi, esi
13 100baa1:     mov     edi, 0x1024584 <kout+64>
14 100baa6:     call    1008d90 <_ZN10TextWindow6setPosEjj>
15 100baab:     mov     esi, ebx
16 100baad:     mov     edi, 0x1024520 <kout>
17 100bab2:     add     ebx, 0x1
18 100bab5:     call    100b880 <_ZN12OutputStreamlsEj>
19 100baba:     mov     edi, 0x1024520 <kout>
20 100babf:     call    10079e0 <_ZN10TextStream5flushEv>
21 100bac4:     call    1007f00 <_ZN5Guard5leaveEv>
22 100bac9:     jmp     100ba98 <_ZN11Application6actionEv+0x18>
```

```
1 0100ba80 <_ZN11Application6actionEv>:
2 100ba80:     endbr64
3 100ba84:     push    rbp
4 100ba85:     push    rbx
5 100ba86:     xor     ebx, ebx
6 100ba88:     sub     rsp, 0x8
7 100ba8c:     call    1008be0
8 100ba91:     mov     ebp, eax
9 100ba93:     nop     DWORD PTR [rax+rax*1+0x0]
10 100ba98:     call    1007e80
11 100ba9d:     mov     edx, ebp
12 100ba9f:     xor     esi, esi
13 100baa1:     mov     edi, 0x1024584
14 100baa6:     call    1008d90
15 100baab:     mov     esi, ebx
16 100baad:     mov     edi, 0x1024520
17 100bab2:     add     ebx, 0x1
18 100bab5:     call    100b880
19 100baba:     mov     edi, 0x1024520
20 100babf:     call    10079e0
21 100bac4:     call    1007f00
22 100bac9:     jmp     100ba98
```



Maschinencode

```
1 0100ba80 <_ZN11Application6actionEv>:  
2 100ba80: endbr64  
3 100ba84: push rbp  
4 100ba85: push rbx  
5 100ba86: xor ebx, ebx  
6 100ba88: sub rsp, 0x8  
7 100ba8c: call 1008be0  
8 100ba91: mov ebp, eax  
9 100ba93: nop DWORD PTR [rax+rax*1+0x0]  
10 100ba98: call 1007e80  
11 100ba9d: mov edx, ebp  
12 100ba9f: xor esi, esi  
13 100baa1: mov edi, 0x1024584  
14 100baa6: call 1008d90  
15 100baab: mov esi, ebx  
16 100baad: mov edi, 0x1024520  
17 100bab2: add ebx, 0x1  
18 100bab5: call 100b880  
19 100baba: mov edi, 0x1024520  
20 100babf: call 10079e0  
21 100bac4: call 1007f00  
22 100bac9: jmp 100ba98
```



```
1  
2 f3 0f 1e fa  
3 55  
4 53  
5 31 db  
6 48 83 ec 08  
7 e8 4f d1 ff ff  
8 89 c5  
9 0f 1f 44 00 00  
10 e8 e3 c3 ff ff  
11 89 ea  
12 31 f6  
13 bf 84 45 02 01  
14 e8 e5 d2 ff ff  
15 89 de  
16 bf 20 45 02 01  
17 83 c3 01  
18 e8 c6 fd ff ff  
19 bf 20 45 02 01  
20 e8 1c bf ff ff  
21 e8 37 c4 ff ff  
22 eb cd
```





Syntaxunterschiede bei x86-Assembler

Intel:

```
mov edi, 0x17
```

(Ziel, Quelle)

```
objdump -M intel
```

Standard bei `nasm` (Netwide Assembler)

AT&T:

```
mov $0x17, %edi
```

(Quelle, Ziel)

Standard bei `objdump`

und `GCC inline asm`

Berechnungen auf Registern,



Long Mode Register



Berechnungen auf Registern, Zugriff mittels `mov`-**Instruktion**



Berechnungen auf Registern, Zugriff mittels `mov`-Instruktion

Register $\leftarrow$ **Konstante** `mov rax, 42`



Berechnungen auf Registern, Zugriff mittels `mov`-Instruktion

Register $\leftarrow$ **Konstante** `mov rax, 42`

Register $\leftarrow$ **Register** `mov rax, rcx`



Berechnungen auf Registern, Zugriff mittels `mov`-Instruktion

Register $\leftarrow$ Konstante `mov rax, 42`

Register $\leftarrow$ Register `mov rax, rcx`

Register $\leftrightarrow$ Speicher `mov rax, [0xb8000]`

`mov [0xb8000 + 4], rax`

`mov rax, [rcx + rdx]`

`mov rax, [rsp]`

~~`mov [0xb8000], [0xb8002]`~~



Besonderheiten bei `mov`



Besonderheiten bei `mov`



`mov eax, XX` `eax` wird gesetzt, obere Hälfte **genullt**



Besonderheiten bei `mov`



`mov eax, XX` `eax` wird gesetzt, obere Hälfte **genullt**

`mov ax, XX` `ax` wird gesetzt, **Rest unverändert**

`mov al, XX` `al` wird gesetzt, **Rest unverändert**

Falls Rest gesetzt werden muss: `movzx`, `movsx`

Details zur Performance bei partiellem Schreiben: → [StackOverflow](#)



Stack $\leftarrow$ **Register** `push rax`





Zur Erinnerung – für den Stack auf x64 gilt

- Der Stack „wächst“ von *oben* (hohe Adresse) nach *unten* (niedrige Adresse)
- Der Stackzeiger (`rsp`) zeigt auf das zuletzt hinzugefügte Datum
 - `push X` verringert zuerst den Wert von `rsp` um 8 Byte und legt dann `X` an die resultierende Adresse
 - `pop X` liest den Inhalt des Speichers, auf das `rsp` zeigt, in `X` und erhöht anschließend den Wert von `rsp` um 8 Byte.



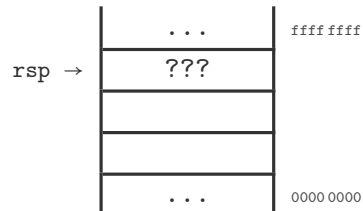
Zur Erinnerung – für den Stack auf x64 gilt

- Der Stack „wächst“ von *oben* (hohe Adresse) nach *unten* (niedrige Adresse)
- Der Stackzeiger (`rsp`) zeigt auf das zuletzt hinzugefügte Datum
 - `push X` verringert zuerst den Wert von `rsp` um 8 Byte und legt dann `X` an die resultierende Adresse
 - `pop X` liest den Inhalt des Speichers, auf das `rsp` zeigt, in `X` und erhöht anschließend den Wert von `rsp` um 8 Byte.
- Bei Funktionsaufrufen muss der Stack an 16 Byte ausgerichtet sein

Stack $\leftarrow$ **Register** `push rax`



Stack $\leftarrow$ **Register** `push rax`

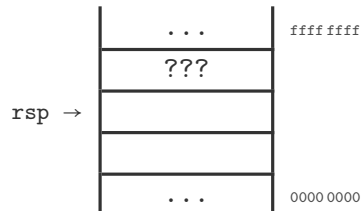


Operationen mit dem Stack

Stack $\leftarrow$ **Register**

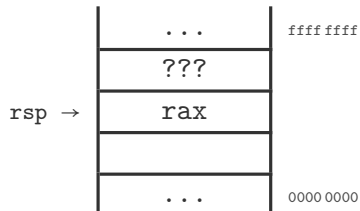
```
push rax
```

```
sub rsp, 8
```



Stack $\leftarrow$ **Register**

```
push rax  
sub rsp, 8  
mov [rsp], rax
```



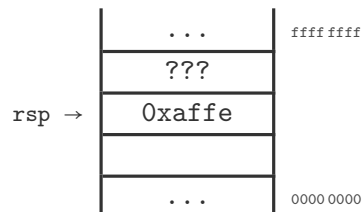
Stack ← Register

```
mov eax, 0xaffe
```

```
push rax
```

```
sub rsp, 8
```

```
mov [rsp], rax
```



Operationen mit dem Stack

Stack ← Register

```
mov eax, 0xaffe
```

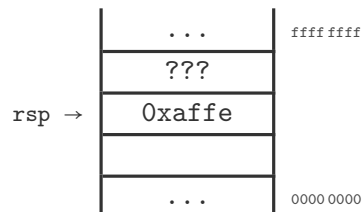
```
push rax
```

```
sub rsp, 8
```

```
mov [rsp], rax
```

Register ← Stack

```
pop rcx
```



Operationen mit dem Stack

Stack ← Register

```
mov eax, 0xaffe
```

```
push rax
```

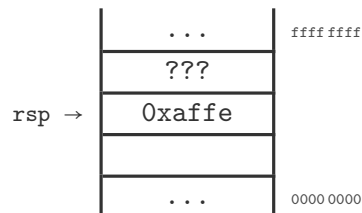
```
sub rsp, 8
```

```
mov [rsp], rax
```

Register ← Stack

```
pop rcx
```

```
mov rcx, [rsp]
```



Operationen mit dem Stack

Stack ← Register

```
mov eax, 0xaffe
```

```
push rax
```

```
sub rsp, 8
```

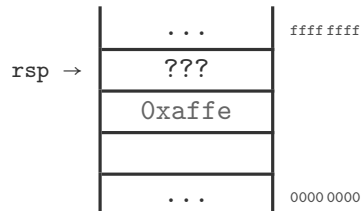
```
mov [rsp], rax
```

Register ← Stack

```
pop rcx
```

```
mov rcx, [rsp]
```

```
add rsp, 8
```



Addition/Subtraktion	<code>add rax, 4</code>	<code>x += 4;</code>
	<code>sub QWORD [rax], rcx</code>	<code>*x -= rcx;</code>
	<code>inc rax</code>	<code>x++;</code>
	<code>dec rax</code>	<code>x--;</code>
Bit-Operationen	<code>and rax, 0xff</code>	<code>x &= 0xff;</code>
	<code>or rax, [rcx]</code>	<code>x = *rcx;</code>
	<code>xor rax, 0xaa</code>	<code>x ^= 0xaa;</code>
	<code>not rax</code>	<code>x = !x;</code>
	<code>neg rcx</code>	<code>x = ~x;</code>
	<code>shl rax, 1</code>	<code>x <<= 1;</code>
	<code>shr rax, 1</code>	<code>x >>= 1;</code>



Sprungmarke

LABEL:

<...>

Einfache Sprünge

jmp LABEL

Bedingte Sprünge

je/jne/jg/jge/jl/jle/jz LABEL

Letztere sind abhängig von den Flags im Statusregister (rflags)

Sprungmarke

LABEL:
<...>

Einfache Sprünge

jmp LABEL

Bedingte Sprünge

je/jne/jg/jge/jl/jle/jz LABEL

Letztere sind abhängig von den Flags im Statusregister (rflags)

Mathematische Operationen und Vergleiche setzen *carry*, *zero* & *sign flag* (Bit 0, 6 & 7 im Statusregister):

Vergleiche

cmp rax, rdx

cmp rax, 0x10

≅ sub rax, 0x10 (aber rax unverändert)

Funktionsaufruf

`call function`

`push <next RIP>`

`jmp function`

Rückkehr aus Funktion

`ret`

`pop rip`

Rückkehr aus Interrupt

`iretq`

Stellt Hardware-Kontext (`rip`, `cs`, `rflags`) wieder her

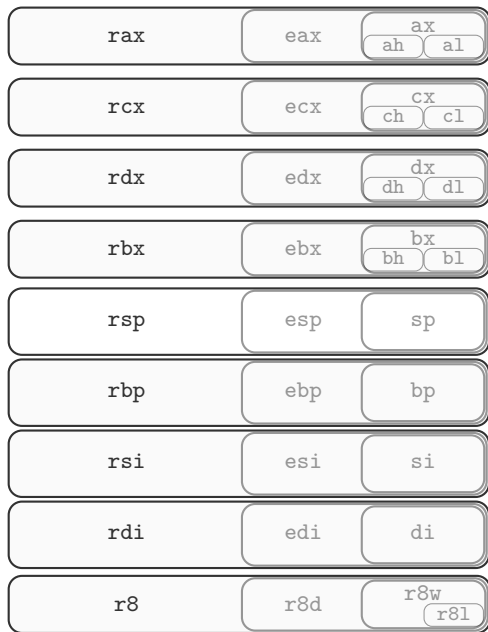


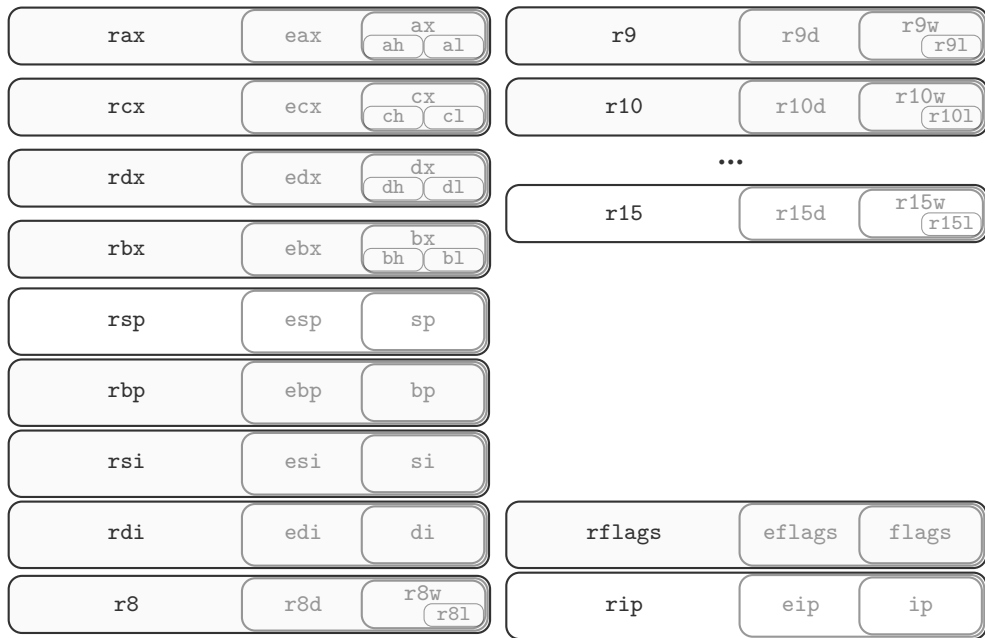
Kontextsicherung gemäß Konvention

```
1 void baz(){
2     ...
3
4     // flüchtige Register
5     // sichern
6
7     func();
8
9     // flüchtige Register
10    // wiederherstellen
11
12    ...
13 }
```

```
1 void func(){
2     // nicht-flüchtige
3     // Register
4     // sichern
5
6     ...
7
8     // nicht-flüchtige
9     // Register
10    // wiederherstellen
11
12    return;
13 }
```







rax	eax	ax ah al	r9	r9d	r9w r9l
rcx	ecx	cx ch cl	r10	r10d	r10w r10l
rdx	edx	dx dh dl	r11	r11d	r11w r11l
rbx	ebx	bx bh bl	r12	r12d	r12w r12l
rsp	esp	sp	r13	r13d	r13w r13l
rbp	ebp	bp	r14	r14d	r14w r14l
rsi	esi	si	r15	r15d	r15w r15l
rdi	edi	di	rflags	eflags	flags
r8	r8d	r8w r8l	rip	eip	ip



rax	eax	ax ah al	r9	r9d	r9w r9l
rcx	ecx	cx ch cl	r10	r10d	r10w r10l
rdx	edx	dx dh dl	r11	r11d	r11w r11l
rbx	ebx	bx bh bl	r12	r12d	r12w r12l
rsp	esp	sp	r13	r13d	r13w r13l
rbp	ebp	bp	r14	r14d	r14w r14l
rsi	esi	si	r15	r15d	r15w r15l
rdi	edi	di	rflags	eflags	flags
r8	r8d	r8w r8l	rip	eip	ip

flüchtige (*scratch* / *caller-save*) Register



rax	eax	ax ah al	r9	r9d	r9w r9l
rcx	ecx	cx ch cl	r10	r10d	r10w r10l
rdx	edx	dx dh dl	r11	r11d	r11w r11l
rbx	ebx	bx bh bl	r12	r12d	r12w r12l
rsp	esp	sp	r13	r13d	r13w r13l
rbp	ebp	bp	r14	r14d	r14w r14l
rsi	esi	si	r15	r15d	r15w r15l
rdi	edi	di	rflags	eflags	flags
r8	r8d	r8w r8l	rip	eip	ip

flüchtige (scratch / caller-save) Register

nicht-flüchtig (non-scratch / callee-save)



Wieso nicht gleich die
flüchtigen Register beim
Funktionsaufruf nutzen?

Bild: twemoji







Rückgabewert

4. Parameter

3. Parameter

2. Parameter

1. Parameter

5. Parameter



```
int i = func(23, 42);
```



Parameterübergabe gemäß Konvention

```
int i = func(23, 42);
```

```
; ggf. Register Sicherung  
push r9  
; 2. Parameter in Register rsi  
mov esi, 0x2a  
; 1. Parameter in Register rdi  
mov edi, 0x17  
; Funktionsaufruf  
call func
```



Parameterübergabe gemäß Konvention

```
int i = func(23, 42);
```

```
; ggf. Register Sicherung
push r9
; 2. Parameter in Register rsi
mov esi, 0x2a
; 1. Parameter in Register rdi
mov edi, 0x17
; Funktionsaufruf
call func
; pushed implizit die
; Rücksprungadresse
L1:
```

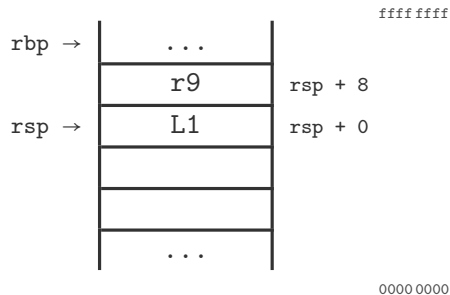


Parameterübergabe gemäß Konvention

```
int i = func(23, 42);
```

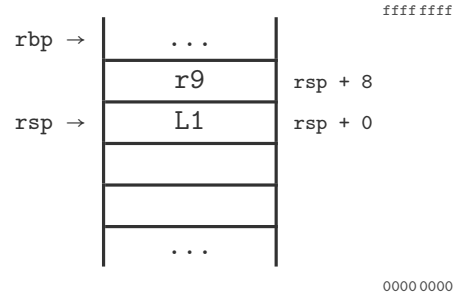
```
; ggf. Register Sicherung
push r9
; 2. Parameter in Register rsi
mov esi, 0x2a
; 1. Parameter in Register rdi
mov edi, 0x17
; Funktionsaufruf
call func
; pushed implizit die
; Rücksprungadresse
L1:
```

Stack beim Funktionsaufruf:



Parameterübergabe gemäß Konvention

func:

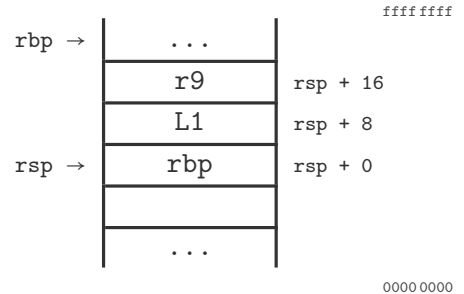


Parameterübergabe gemäß Konvention

```
func:
```

```
; alten Rahmenzeiger sichern
```

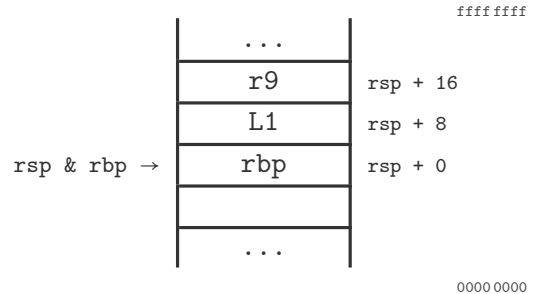
```
push rbp
```



Parameterübergabe gemäß Konvention

func:

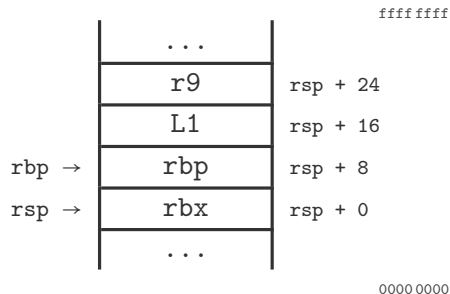
```
; alten Rahmenzeiger sichern  
push rbp  
; aktuellen Rahmenzeiger setzen  
mov rbp, rsp
```



Parameterübergabe gemäß Konvention

func:

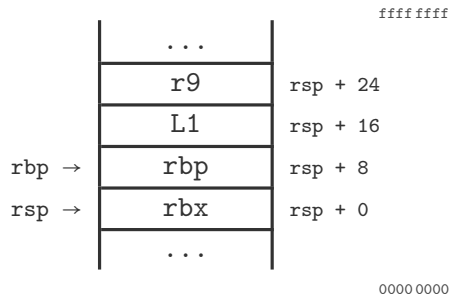
```
; alten Rahmenzeiger sichern  
push rbp  
; aktuellen Rahmenzeiger setzen  
mov rbp, rsp  
; ggf. weitere Register sichern  
push rbx
```



Parameterübergabe gemäß Konvention

func:

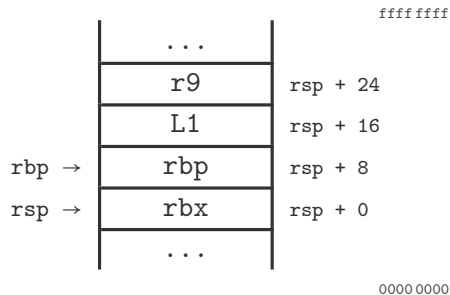
```
; alten Rahmenzeiger sichern  
push rbp  
; aktuellen Rahmenzeiger setzen  
mov rbp, rsp  
; ggf. weitere Register sichern  
push rbx  
  
...
```



Parameterübergabe gemäß Konvention

func:

```
; alten Rahmenzeiger sichern  
push rbp  
; aktuellen Rahmenzeiger setzen  
mov rbp, rsp  
; ggf. weitere Register sichern  
push rbx  
  
...  
  
; Rückgabewert nach rax  
mov eax, 0xd
```



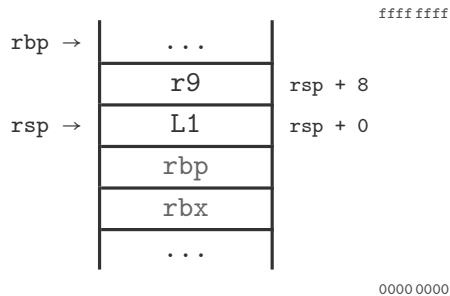
Parameterübergabe gemäß Konvention

func:

```
; alten Rahmenzeiger sichern
push rbp
; aktuellen Rahmenzeiger setzen
mov rbp, rsp
; ggf. weitere Register sichern
push rbx

...

; Rückgabewert nach rax
mov eax, 0xd
; Register wiederherstellen
pop rbx
pop rbp
```



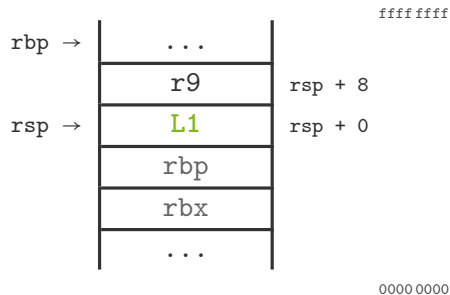
Parameterübergabe gemäß Konvention

func:

```
; alten Rahmenzeiger sichern
push rbp
; aktuellen Rahmenzeiger setzen
mov rbp, rsp
; ggf. weitere Register sichern
push rbx

...

; Rückgabewert nach rax
mov eax, 0xd
; Register wiederherstellen
pop rbx
pop rbp
; Rücksprung
ret
```

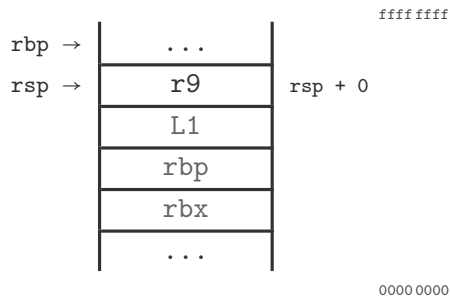


Parameterübergabe gemäß Konvention

```
int i = func(23, 42);
```

```
; ggf. Register Sicherung
push r9
; 2. Parameter in Register rsi
mov esi, 0x2a
; 1. Parameter in Register rdi
mov edi, 0x17
; Funktionsaufruf
call func
; pushed implizit die
; Rücksprungsadresse
L1:
```

Stack nach Funktionsaufruf:

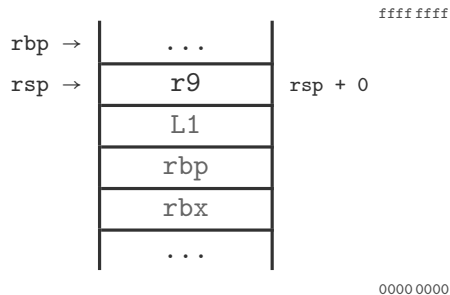


Parameterübergabe gemäß Konvention

```
int i = func(23, 42);
```

```
; ggf. Register Sicherung
push r9
; 2. Parameter in Register rsi
mov esi, 0x2a
; 1. Parameter in Register rdi
mov edi, 0x17
; Funktionsaufruf
call func
; pushed implizit die
; Rücksprungsadresse
L1:
; Rückgabewert in rax
mov rsi, rax
```

Stack nach Funktionsaufruf:

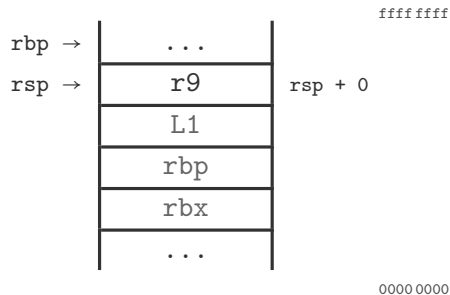


Parameterübergabe gemäß Konvention

```
int i = func(23, 42);
```

Stack nach Funktionsaufruf:

```
; ggf. Register Sicherung
push r9
; 2. Parameter in Register rsi
mov esi, 0x2a
; 1. Parameter in Register rdi
mov edi, 0x17
; Funktionsaufruf
call func
; pushed implizit die
; Rücksprungsadresse
L1:
; Rückgabewert in rax
mov rsi, rax
; ggf. Register wiederherstellen
pop r9
```



Umschaltung

```
1 void foo(){
2     int f = 42;
3
4     while (f--){
5         kout << "foo"
6             << f
7             << endl;
8         context_switch(
9             &stack_foo,
10            &stack_bar
11        );
12    }
13 }
```

```
1 void bar(){
2     int b = 23;
3
4     while (b--){
5         kout << "bar"
6             << b
7             << endl;
8         context_switch(
9             &stack_bar,
10            &stack_foo
11        );
12    }
13 }
```



Umschaltung im Detail



Kontrollflusszustand **sichern** & **laden**

Notwendige **Schritte** in `context_switch`:

Umschaltung im Detail



Kontrollflusszustand **sichern** & **laden**

Notwendige **Schritte** in `context_switch`:

1. **Register** sichern



Kontrollflusszustand **sichern** & **laden**

Notwendige **Schritte** in `context_switch`:

1. **Register** sichern (*via Stack*)



Kontrollflusszustand **sichern** & **laden**

Notwendige **Schritte** in `context_switch`:

1. **Register** sichern (*via Stack*)
2. **Stackpointer** (`rsp`) sichern



Kontrollflusszustand **sichern** & **laden**

Notwendige **Schritte** in `context_switch`:

1. **Register** sichern (via *Stack*)
2. **Stackpointer** (`rsp`) sichern
3. **Stackpointer** (`rsp`) laden



Kontrollflusszustand **sichern** & **laden**

Notwendige **Schritte** in `context_switch`:

1. **Register** sichern (via Stack)
2. **Stackpointer** (`rsp`) sichern
3. **Stackpointer** (`rsp`) laden
4. **Register** wiederherstellen

```
// Global sichtbar:
```

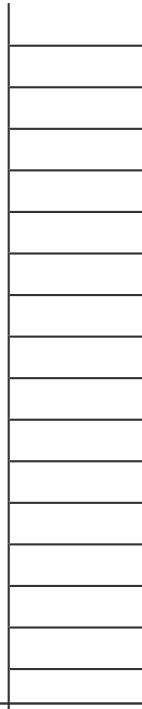
```
struct StackPointer {
```

```
    void *kernel;
```

```
    // Später auch User-
```

```
    // Stackpointer (in BST)
```

```
};
```



```
// Global sichtbar:
```

```
struct StackPointer {  
    void *kernel;
```

```
  
    // Später auch User-  
    // Stackpointer (in BST)  
};
```

```
StackPointer stack_foo;
```

stack_foo →

0100 bff0



```
// Global sichtbar:

struct StackPointer {
    void *kernel;

    // Später auch User-
    // Stackpointer (in BST)
};

StackPointer stack_foo;

StackPointer stack_bar;
```



A vertical stack of 15 rectangular boxes representing memory frames. The stack grows downwards. The bottom-most box is labeled '0100 aef8' on its right side. The box immediately above it is labeled '...' on its right side. The box above that is labeled '0100 bff0' on its right side. To the left of the stack, the label 'stack_foo' is aligned with the '0100 bff0' box, and 'stack_bar' is aligned with the '...' box. Arrows point from these labels to their respective boxes.

stack_foo → 0100 bff0

...

stack_bar → 0100 aef8




```
void foo(){
```

```
...
```

```
context_switch(  
    &stack_foo,  
    &stack_bar  
);
```

```
...
```

rsp →

...

010d6a68

stack_foo →

010d6aa8

0100bff0

...

stack_bar →

01100188

0100aef8

```
void foo(){
```

```
...
```

```
// Sicherung der  
// flüchtigen Register  
// von Kontext foo
```

```
context_switch(  
    &stack_foo,  
    &stack_bar  
);
```

```
...
```

rsp →

...

010d6a68

stack_foo →

010d6aa8

0100bff0

...

stack_bar →

01100188

0100aef8



```
void foo(){
```

```
...
```

```
// Sicherung der  
// flüchtigen Register  
// von Kontext foo  
// durch Übersetzer
```

```
context_switch(  
    &stack_foo,  
    &stack_bar  
);
```

```
...
```

rsp →

...

010d6a68

stack_foo →

010d6aa8

0100bff0

...

stack_bar →

01100188

0100aef8



```
void foo(){
```

```
...
```

```
// Sicherung der  
// flüchtigen Register  
// von Kontext foo  
// durch Übersetzer  
// (z.B. r8)
```

```
context_switch(  
    &stack_foo,  
    &stack_bar  
);
```

```
...
```

rsp →

...

010d6a68

r8

010d6a60

stack_foo →

010d6aa8

0100bff0

...

stack_bar →

01100188

0100aef8



```
void foo(){
```

```
...
```

```
// Sicherung der  
// flüchtigen Register  
// von Kontext foo  
// durch Übersetzer  
// (z.B. r8)
```

```
context_switch(  
    &stack_foo,  
    &stack_bar  
);
```

```
...
```

rsp →

...

010d6a68

r8

010d6a60

stack_foo →

010d6aa8

0100bff0

...

stack_bar →

01100188

0100aef8



```
void foo(){
```

```
...
```

```
// Sicherung der  
// flüchtigen Register  
// von Kontext foo  
// durch Übersetzer  
// (z.B. r8)
```

```
context_switch(  
    &stack_foo,  
    &stack_bar  
);
```

```
LF00:
```

```
...
```

rsp →

...

010d6a68

r8

010d6a60

LF00

010d6a58

stack_foo →

010d6aa8

0100bff0

...

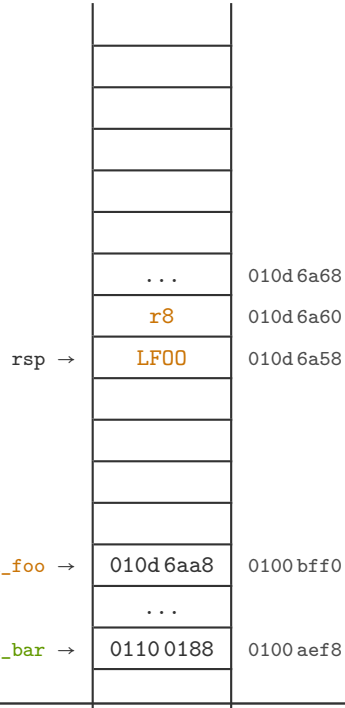
stack_bar →

01100188

0100aef8



context_switch:



ret

stack_foo →

stack_bar →



context_switch:

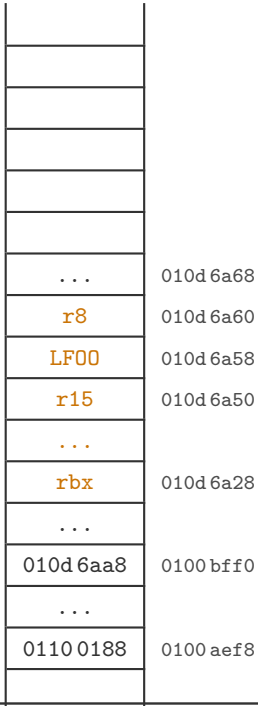
```
;; nicht-flüchtige  
;; Register von  
;; foo sichern
```

ret

stack_foo →

stack_bar →

rsp →



context_switch:

```
;; nicht-flüchtige  
;; Register von  
;; foo sichern
```

```
;; Stackzeiger von  
;; foo sichern
```

ret



context_switch:

```
;; nicht-flüchtige  
;; Register von  
;; foo sichern
```

```
;; Stackzeiger von  
;; foo sichern
```

```
;; Stackzeiger von  
;; bar laden
```

ret

rsp →

0110 0188 ←

...

r8

010d 6a60

LF00

010d 6a58

r15

010d 6a50

...

rbx

010d 6a28

...

stack_foo →

010d 6a28

0100 bff0

...

stack_bar →

0110 0188

0100 aef8

context_switch:

;; nicht-flüchtige
;; Register von
;; foo sichern

;; Stackzeiger von
;; foo sichern

;; Stackzeiger von
;; bar laden

ret

rsp →

stack_foo →

stack_bar →

r10 0110 01c0

LBAR 0110 01b8

r15 0110 01b0

...

rbx 0110 0188

...

r8 010d 6a60

LF00 010d 6a58

r15 010d 6a50

...

rbx 010d 6a28

...

010d 6a28 0100 bff0

...

0110 0188 0100 aef8



context_switch:

```
;; nicht-flüchtige  
;; Register von  
;; foo sichern
```

```
;; Stackzeiger von  
;; foo sichern
```

```
;; Stackzeiger von  
;; bar laden
```

```
;; nicht-flüchtige  
;; Register von  
;; bar laden
```

ret

rsp →

r10 0110 01c0

LBAR 0110 01b8

r15 0110 01b0

...

rbx 0110 0188

...

r8 010d 6a60

LF00 010d 6a58

r15 010d 6a50

...

rbx 010d 6a28

...

stack_foo → 010d 6a28 0100 bff0

...

stack_bar → 0110 0188 0100 aef8

...



context_switch:

```
;; nicht-flüchtige  
;; Register von  
;; foo sichern
```

```
;; Stackzeiger von  
;; foo sichern
```

```
;; Stackzeiger von  
;; bar laden
```

```
;; nicht-flüchtige  
;; Register von  
;; bar laden
```

ret

rsp →

r10 0110 01c0

LBAR 0110 01b8

r15 0110 01b0

...

rbx 0110 0188

...

r8 010d 6a60

LF00 010d 6a58

r15 010d 6a50

...

rbx 010d 6a28

...

stack_foo → 010d 6a28 0100 bff0

...

stack_bar → 0110 0188 0100 aef8

...



```
void bar(){
```

```
...
```

```
context_switch(  
    &stack_bar,  
    &stack_foo  
);
```

```
LBAR:
```

```
...
```

rsp →

r10

0110 01c0

LBAR

0110 01b8

r15

0110 01b0

...

rbx

0110 0188

...

r8

010d 6a60

LF00

010d 6a58

r15

010d 6a50

...

rbx

010d 6a28

...

stack_foo →

010d 6a28

0100 bff0

...

stack_bar →

0110 0188

0100 aef8



```
void bar(){
```

```
...
```

```
context_switch(  
    &stack_bar,  
    &stack_foo  
);
```

```
LBAR:
```

```
// Wiederherstellen der  
// flüchtigen Register  
// von Kontext bar  
// durch Übersetzer  
// (z.B. r10)
```

```
...
```

rsp →

r10

0110 01c0

LBAR

0110 01b8

r15

0110 01b0

...

rbx

0110 0188

...

r8

010d 6a60

LF00

010d 6a58

r15

010d 6a50

...

rbx

010d 6a28

...

stack_foo →

010d 6a28

0100 bff0

...

stack_bar →

0110 0188

0100 aef8



Koroutine (erstmalig) starten

Wie rufe ich die Koroutine erstmalig auf?

Ziel: Instruktionszeiger `rip` ändern



Koroutine (erstmalig) starten

Wie rufe ich die Koroutine erstmalig auf?

Ziel: Instruktionszeiger `rip` ändern

Problem: Register `rip` kann nicht direkt beschrieben werden



Koroutine (erstmalig) starten

Wie rufe ich die Koroutine erstmalig auf?

Ziel: Instruktionszeiger `rip` ändern

Problem: Register `rip` kann nicht direkt beschrieben werden

Lösung: Zieladresse auf Stack und `ret` ändert `rip`



Koroutine (erstmalig) starten

Wie rufe ich die Koroutine erstmalig auf?

Ziel: Instruktionszeiger `rip` ändern

Problem: Register `rip` kann nicht direkt beschrieben werden

Lösung: Zieladresse auf Stack und `ret` ändert `rip`

→ `context_switch` mit entsprechenden Stack



Koroutine (erstmalig) starten

Wie rufe ich die Koroutine erstmalig auf?

Ziel: Instruktionszeiger `rip` ändern

Problem: Register `rip` kann nicht direkt beschrieben werden

Lösung: Zieladresse auf Stack und `ret` ändert `rip`

→ `context_switch` mit entsprechenden Stack

Wie rufe ich die aller erste Koroutine auf?



Koroutine (erstmalig) starten

Wie rufe ich die Koroutine erstmalig auf?

Ziel: Instruktionszeiger `rip` ändern

Problem: Register `rip` kann nicht direkt beschrieben werden

Lösung: Zieladresse auf Stack und `ret` ändert `rip`

→ `context_switch` mit entsprechenden Stack

Wie rufe ich die aller erste Koroutine auf?

- `context_go` mit nur einem Parameter



Koroutine (erstmalig) starten

Wie rufe ich die Koroutine erstmalig auf?

Ziel: Instruktionszeiger `rip` ändern

Problem: Register `rip` kann nicht direkt beschrieben werden

Lösung: Zieladresse auf Stack und `ret` ändert `rip`

→ `context_switch` mit entsprechenden Stack

Wie rufe ich die aller erste Koroutine auf?

- `context_go` mit nur einem Parameter

Was wird für jede Koroutine gebraucht?



Koroutine (erstmalig) starten

Wie rufe ich die Koroutine erstmalig auf?

Ziel: Instruktionszeiger `rip` ändern

Problem: Register `rip` kann nicht direkt beschrieben werden

Lösung: Zieladresse auf Stack und `ret` ändert `rip`

→ `context_switch` mit entsprechenden Stack

Wie rufe ich die aller erste Koroutine auf?

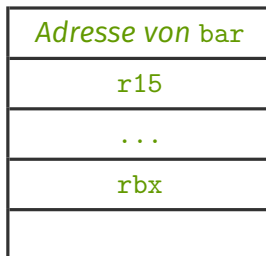
- `context_go` mit nur einem Parameter

Was wird für jede Koroutine gebraucht?

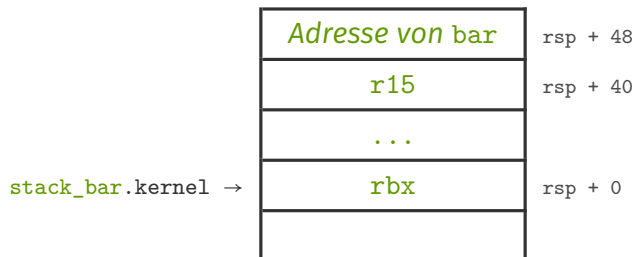
- eigener, präparierter Stack
- Speicher für Stackzeiger (`struct StackPointer`)



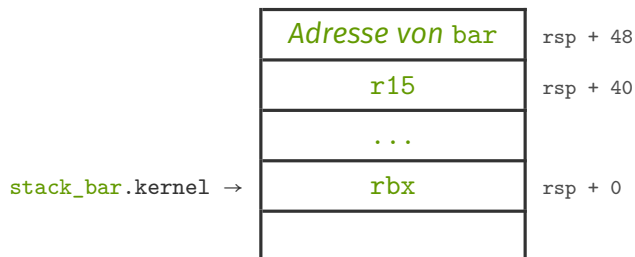
Stack für Einsprung in Koroutine `void bar()`



Stack für Einsprung in Koroutine `void bar()`

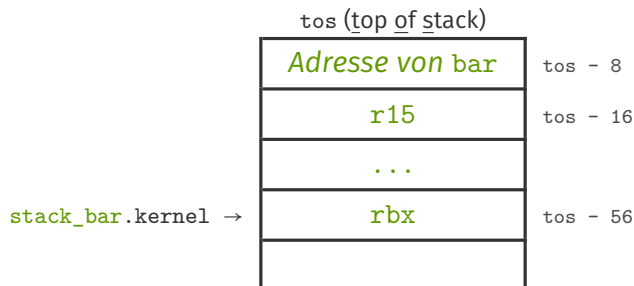


Stack für Einsprung in Koroutine `void bar()`



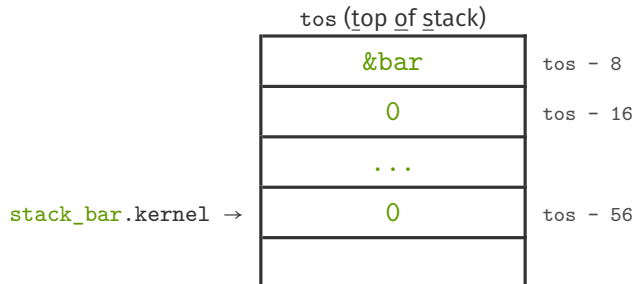
```
1 #define STACKSIZE 4096
2 char stackBar[STACKSIZE];
3 void* tos = stackFoo + STACKSIZE;
```

Stack für Einsprung in Koroutine `void bar()`



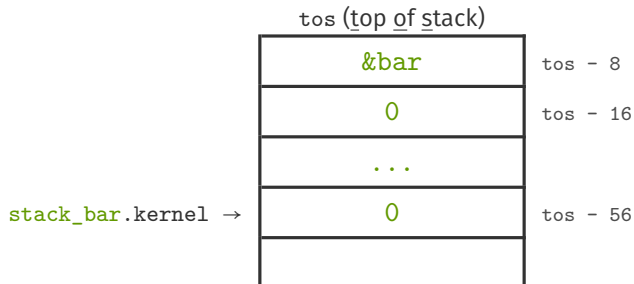
```
1 #define STACKSIZE 4096
2 char stackBar[STACKSIZE];
3 void* tos = stackFoo + STACKSIZE;
```

Stack für Einsprung in Koroutine `void bar()`



Stack aufsetzen

Stack für Einsprung in Koroutine `void bar()`

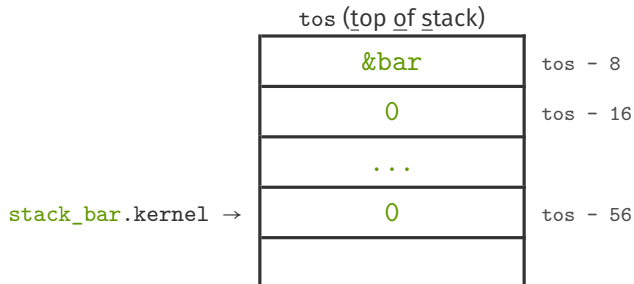


Was passiert nun bei `context_switch`?



Stack aufsetzen

Stack für Einsprung in Koroutine `void bar()`



Was passiert, wenn die Koroutine `bar` abgearbeitet ist?



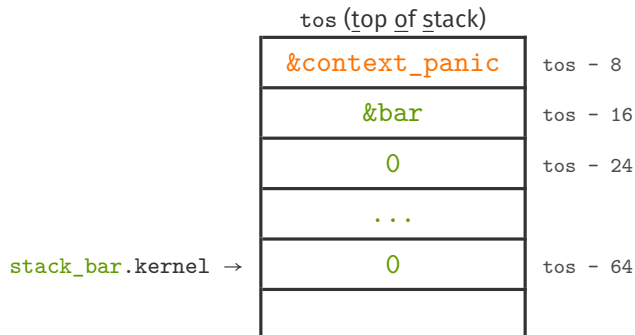
Stack aufsetzen (Robust)

```
1 void panic() {  
2     kernelpanic("Application should not return!1!!11");  
3 }
```



Stack aufsetzen (Robust)

```
1 void panic() {  
2     kernelpanic("Application should not return!1!!11");  
3 }
```



Stack aufsetzen mit Funktionsparameter

Stack für Einsprung in Koroutine `void bar(int i)`



Stack aufsetzen mit Funktionsparameter

Stack für Einsprung in Koroutine `void bar(int i)` unter Verwendung einer zusätzlichen Hilfsfunktion – z.B. `bar_wrapper`



Stack aufsetzen mit Funktionsparameter

Stack für Einsprung in Koroutine `void bar(int i)` unter Verwendung einer zusätzlichen Hilfsfunktion – z.B. `bar_wrapper`

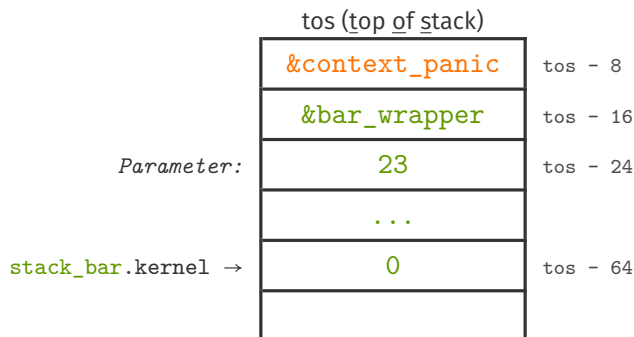
- liest Parameter aus nicht-flüchtigem Register (`r15`)
- schreibt Wert in flüchtiges Parameterregister (`rdi`)
- springt in eigentliche Funktion (`bar`)



Stack aufsetzen mit Funktionsparameter

Stack für Einsprung in Koroutine `void bar(int i)` unter Verwendung einer zusätzlichen Hilfsfunktion – z.B. `bar_wrapper`

- liest Parameter aus nicht-flüchtigem Register (r15)
- schreibt Wert in flüchtiges Parameterregister (rdi)
- springt in eigentliche Funktion (`bar`)



```
1 class Thread {  
2     StackPointer sp;  
3     public:  
4     Thread();  
5     virtual void action() = 0;  
6 };
```



```
1 class Thread {  
2     StackPointer sp;  
3     public:  
4     Thread();  
5     virtual void action() = 0;  
6 };
```

Adresse einer virtuellen Member-Funktion nicht (einfach) ermittelbar

```
1 class Thread {  
2     StackPointer sp;  
3     public:  
4     Thread();  
5     virtual void action() = 0;  
6 };
```

Adresse einer virtuellen Member-Funktion nicht (einfach) ermittelbar

```
1 Thread * x = &app;  
2 x->action(); // Foo::action oder Bar::action ?
```

```
7 class Foo : public Thread {  
8     void action() { ... }  
9 };
```

```
1 class Bar : public Thread {  
2     void action() { ... }  
3 };
```



```
1 class Thread {  
2     StackPointer sp;  
3     public:  
4     Thread();  
5     virtual void action() = 0;  
6 };
```

Adresse einer virtuellen Member-Funktion nicht (einfach) ermittelbar

```
1 void kickoff(Thread* t){  
2     t->action();  
3 }
```

```
1 void * prepareContext(void * tos,  
2                       void (*kickoff)(void*),  
3                       void * param);
```

Präpariert einen Stack für den ersten Einsprung

```
1 void * prepareContext(void * tos,  
2                       void (*kickoff)(void*),  
3                       void * param);
```

Präpariert einen Stack für den ersten Einsprung

- statisch reservierten Speicher `tos` als Stack aufsetzen

```
1 void * prepareContext(void * tos,  
2                       void (*kickoff)(void*),  
3                       void * param);
```

Präpariert einen Stack für den ersten Einsprung

- statisch reservierten Speicher `tos` als Stack aufsetzen
- nach dem Einsprung soll `kickoff` mit `param` aufgerufen werden

```
1 void * prepareContext(void * tos,  
2                       void (*kickoff)(void*),  
3                       void * param);
```

Präpariert einen Stack für den ersten Einsprung

- statisch reservierten Speicher `tos` als Stack aufsetzen
- nach dem Einsprung soll `kickoff` mit `param` aufgerufen werden
- Stackpointer `kernel` in Thread entsprechend setzen

```
1 void * prepareContext(void * tos,  
2                       void (*kickoff)(void*),  
3                       void * param);
```

Präpariert einen Stack für den ersten Einsprung

- statisch reservierten Speicher `tos` als Stack aufsetzen
- nach dem Einsprung soll `kickoff` mit `param` aufgerufen werden
- Stackpointer `kernel` in Thread entsprechend setzen
- in C++ (statt Assembler)



```
1 void * prepareContext(void * tos,  
2                       void (*kickoff)(void*),  
3                       void * param);
```

Präpariert einen Stack für den ersten Einsprung

- statisch reservierten Speicher `tos` als Stack aufsetzen
- nach dem Einsprung soll `kickoff` mit `param` aufgerufen werden
- Stackpointer `kernel` in Thread entsprechend setzen
- in C++ (statt Assembler)
- Pointerarithmetik ist hilfreich



```
1 #define STACKSIZE 4096
2 char stackBar[STACKSIZE];
3 char stackFoo[STACKSIZE];
```



```

1 #define STACKSIZE 4096
2 char stackBar[STACKSIZE];
3 char stackFoo[STACKSIZE];

```

	...	ffff ffff
stackBar[1] →		0102 6021
stackBar[0] →		0102 6020
stackFoo[4095] →		0102 601f
stackFoo[4094] →		0102 601e
stackFoo[4093] →		0102 601d
stackFoo[4092] →		0102 601c
stackFoo[4091] →		0102 601b
stackFoo[4090] →		0102 601a
stackFoo[4089] →		0102 6019
stackFoo[4088] →		0102 6018
stackFoo[4087] →		0102 6017
stackFoo[4086] →		0102 6016
stackFoo[4085] →		0102 6015
stackFoo[4084] →		0102 6014
stackFoo[4083] →		0102 6013
stackFoo[4082] →		0102 6012
stackFoo[4081] →		0102 6011
stackFoo[4080] →		0102 6010
stackFoo[4079] →		0102 600f
	...	
stackFoo[1] →		0102 5021
stackFoo[0] →		0102 5020



```

1 #define STACKSIZE 4096
2 char stackBar[STACKSIZE];
3 char stackFoo[STACKSIZE];

4 void* tos = stackFoo + STACKSIZE;
5 void** rsp = (void**) tos;

```

		...	ffff ffff
	stackBar[1] →		0102 6021
	stackBar[0] →		0102 6020
	stackFoo[4095] →		0102 601f
	stackFoo[4094] →		0102 601e
	stackFoo[4093] →		0102 601d
	stackFoo[4092] →		0102 601c
	stackFoo[4091] →		0102 601b
	stackFoo[4090] →		0102 601a
	stackFoo[4089] →		0102 6019
	stackFoo[4088] →		0102 6018
	stackFoo[4087] →		0102 6017
	stackFoo[4086] →		0102 6016
	stackFoo[4085] →		0102 6015
	stackFoo[4084] →		0102 6014
	stackFoo[4083] →		0102 6013
	stackFoo[4082] →		0102 6012
	stackFoo[4081] →		0102 6011
	stackFoo[4080] →		0102 6010
	stackFoo[4079] →		0102 600f
		...	
	stackFoo[1] →		0102 5021
	stackFoo[0] →		0102 5020



```

1 #define STACKSIZE 4096
2 char stackBar[STACKSIZE];
3 char stackFoo[STACKSIZE];

4 void* tos = stackFoo + STACKSIZE;
5 void** rsp = (void**) tos;

6 rsp--;
7 // rsp = &(stackFoo[4088])
8 *rsp = (void*) 0xdeadbeef0badf00d;

```

	...	ffff ffff
	stackBar[1] →	0102 6021
	stackBar[0] →	0102 6020
	stackFoo[4095] →	de 0102 601f
	stackFoo[4094] →	ad 0102 601e
	stackFoo[4093] →	be 0102 601d
	stackFoo[4092] →	ef 0102 601c
	stackFoo[4091] →	0b 0102 601b
	stackFoo[4090] →	ad 0102 601a
	stackFoo[4089] →	f0 0102 6019
	stackFoo[4088] →	0d 0102 6018
	stackFoo[4087] →	0102 6017
	stackFoo[4086] →	0102 6016
	stackFoo[4085] →	0102 6015
	stackFoo[4084] →	0102 6014
	stackFoo[4083] →	0102 6013
	stackFoo[4082] →	0102 6012
	stackFoo[4081] →	0102 6011
	stackFoo[4080] →	0102 6010
	stackFoo[4079] →	0102 600f
	...	
	stackFoo[1] →	0102 5021
	stackFoo[0] →	0102 5020



```

1 #define STACKSIZE 4096
2 char stackBar[STACKSIZE];
3 char stackFoo[STACKSIZE];

4 void* tos = stackFoo + STACKSIZE;
5 void** rsp = (void**) tos;

6 rsp--;
7 // rsp = &(stackFoo[4088])
8 *rsp = (void*) 0xdeadbeef0badf00d;

10 rsp--;
11 // rsp = &(stackFoo[4080])
12 extern Thread foo;
13 // &foo = 0x102 4280
14 *rsp = (void*) &foo;

```

	...	ffff ffff
stackBar[1] →		0102 6021
stackBar[0] →		0102 6020
stackFoo[4095] →	de	0102 601f
stackFoo[4094] →	ad	0102 601e
stackFoo[4093] →	be	0102 601d
stackFoo[4092] →	ef	0102 601c
stackFoo[4091] →	0b	0102 601b
stackFoo[4090] →	ad	0102 601a
stackFoo[4089] →	f0	0102 6019
stackFoo[4088] →	0d	0102 6018
stackFoo[4087] →	00	0102 6017
stackFoo[4086] →	00	0102 6016
stackFoo[4085] →	00	0102 6015
stackFoo[4084] →	00	0102 6014
stackFoo[4083] →	01	0102 6013
stackFoo[4082] →	02	0102 6012
stackFoo[4081] →	42	0102 6011
stackFoo[4080] →	80	0102 6010
stackFoo[4079] →		0102 600f
	...	
stackFoo[1] →		0102 5021
stackFoo[0] →		0102 5020



```

1 #define STACKSIZE 4096
2 char stackBar[STACKSIZE];
3 char stackFoo[STACKSIZE];

4 void* tos = stackFoo + STACKSIZE;
5 void** rsp = (void**) tos;

6 rsp--;
7 // rsp = &(stackFoo[4088])
8 *rsp = (void*) 0xdeadbeef0badf00d;

10 rsp--;
11 // rsp = &(stackFoo[4080])
12 extern Thread foo;
13 // &foo = 0x102 4280
14 *rsp = (void*) &foo;

16 // ...

```

	...	ffff ffff
	stackBar[1] →	0102 6021
	stackBar[0] →	0102 6020
	stackFoo[4095] →	de 0102 601f
	stackFoo[4094] →	ad 0102 601e
	stackFoo[4093] →	be 0102 601d
	stackFoo[4092] →	ef 0102 601c
	stackFoo[4091] →	0b 0102 601b
	stackFoo[4090] →	ad 0102 601a
	stackFoo[4089] →	f0 0102 6019
	stackFoo[4088] →	0d 0102 6018
	stackFoo[4087] →	00 0102 6017
	stackFoo[4086] →	00 0102 6016
	stackFoo[4085] →	00 0102 6015
	stackFoo[4084] →	00 0102 6014
	stackFoo[4083] →	01 0102 6013
	stackFoo[4082] →	02 0102 6012
	stackFoo[4081] →	42 0102 6011
	stackFoo[4080] →	80 0102 6010
	stackFoo[4079] →	0102 600f
	...	
	stackFoo[1] →	0102 5021
	stackFoo[0] →	0102 5020





Stapelüberlauf

- Die Stacks sind nur 4K groß
 - Die Stacks liegen (oft) hintereinander
 - Das gilt auch für die initialen Stacks beim Systemstart
- Mechanismus zum Erkennen von Überlaufen hilfreich („Stack Canary“)

```

1 #define STACKSIZE 4096
2 char stackBar[STACKSIZE];
3 char stackFoo[STACKSIZE];

4 void* tos = stackFoo + STACKSIZE;
5 void** rsp = (void**) tos;

6 rsp--;
7 // rsp = &(stackFoo[4088])
8 *rsp = (void*) 0xdeadbeef0badf00d;

10 rsp--;
11 // rsp = &(stackFoo[4080])
12 extern Thread foo;
13 // &foo = 0x102 4280
14 *rsp = (void*) &foo;

16 // ...

```

	...	ffff ffff
stackBar[1] →		0102 6021
stackBar[0] →		0102 6020
stackFoo[4095] →	de	0102 601f
stackFoo[4094] →	ad	0102 601e
stackFoo[4093] →	be	0102 601d
stackFoo[4092] →	ef	0102 601c
stackFoo[4091] →	0b	0102 601b
stackFoo[4090] →	ad	0102 601a
stackFoo[4089] →	f0	0102 6019
stackFoo[4088] →	0d	0102 6018
stackFoo[4087] →	00	0102 6017
stackFoo[4086] →	00	0102 6016
stackFoo[4085] →	00	0102 6015
stackFoo[4084] →	00	0102 6014
stackFoo[4083] →	01	0102 6013
stackFoo[4082] →	02	0102 6012
stackFoo[4081] →	42	0102 6011
stackFoo[4080] →	80	0102 6010
stackFoo[4079] →		0102 600f
	...	
stackFoo[1] →		0102 5021
stackFoo[0] →		0102 5020



```

1 #define STACKSIZE 4096
2 char stackBar[STACKSIZE];
3 char stackFoo[STACKSIZE];

4 void* tos = stackFoo + STACKSIZE;
5 void** rsp = (void**) tos;

6 rsp--;
7 // rsp = &(stackFoo[4088])
8 *rsp = (void*) 0xdeadbeef0badf00d;

10 rsp--;
11 // rsp = &(stackFoo[4080])
12 extern Thread foo;
13 // &foo = 0x102 4280
14 *rsp = (void*) &foo;

16 // ...

```

	...	ffff ffff
stackBar[1] →	aa	0102 6021
stackBar[0] →	55	0102 6020
stackFoo[4095] →	de	0102 601f
stackFoo[4094] →	ad	0102 601e
stackFoo[4093] →	be	0102 601d
stackFoo[4092] →	ef	0102 601c
stackFoo[4091] →	0b	0102 601b
stackFoo[4090] →	ad	0102 601a
stackFoo[4089] →	f0	0102 6019
stackFoo[4088] →	0d	0102 6018
stackFoo[4087] →	00	0102 6017
stackFoo[4086] →	00	0102 6016
stackFoo[4085] →	00	0102 6015
stackFoo[4084] →	00	0102 6014
stackFoo[4083] →	01	0102 6013
stackFoo[4082] →	02	0102 6012
stackFoo[4081] →	42	0102 6011
stackFoo[4080] →	80	0102 6010
stackFoo[4079] →		0102 600f
	...	
stackFoo[1] →	aa	0102 5021
stackFoo[0] →	55	0102 5020



Zentrale Verwaltung der Koroutinen (Threads)

```
1 class Scheduler {  
2     Queue<Thread> readylist;  
3     public:  
4         void ready(Thread *);  
5         void schedule();  
6         void resume();  
7         void exit();  
8         void kill(Thread *);  
9 };
```



```
1 // Thread AppFoo
2 void AppFoo::action(){
3     while (1){
4         kout << "foo" << endl;
5         @scheduler.resume();@
6     }
7 }
```

```
1 // Thread AppBar
2 void AppBar::action(){
3     while (1){
4         kout << "bar" << endl;
5         @scheduler.resume();@
6     }
7 }
```



```
1 // Thread AppFoo
2 void AppFoo::action(){
3     while (1){
4         kout << "foo" << endl;
5         @scheduler.resume();@
6     }
7 }
```

```
1 // Thread AppBar
2 void AppBar::action(){
3     while (1){
4         kout << "bar" << endl;
5         @scheduler.resume();@
6     }
7 }
```

```
1 // main.cc
2 AppFoo appfoo;
3 AppBar appbar;
4 int main (){
5     // ...
6     @scheduler.ready(&appfoo);@
7     @scheduler.ready(&appbar);@
8     @scheduler.schedule();@
9 }
```



```
main()
```

```
// ...
```

```
scheduler.ready(&appfoo)
```

```
scheduler.ready(&appbar)
```

```
scheduler.schedule()
```

```
StackPointer dummy;
```

```
context_switch(dummy, appfoo.sp)
```

AppFoo

```
kickoff(&appfoo)
```

```
appfoo->action()
```

```
kout << "foo" << endl
```

```
scheduler.resume()
```

```
context_switch(appfoo.sp, appbar.sp)
```

AppBar

```
kickoff(&appbar)
```

```
appbar->action()
```

```
kout << "bar" << endl
```

```
scheduler.resume()
```

```
context_switch(appbar.sp, appfoo.sp)
```

```
kout << "foo" << endl
```

```
scheduler.resume()
```

```
context_switch(appfoo.sp, appbar.sp)
```

```
kout << "bar" << endl
```

```
scheduler.resume()
```

```
context_switch(appbar.sp, appfoo.sp)
```

foo

bar

foo

bar

E_0 (Anwendung)

$E_{\frac{1}{2}}$ (Epilog)

E_1 (IRQ/Prolog)





$E_{\frac{1}{2}}$ (Epilog)

E_1 (IRQ/Prolog)



OOSTuBS immer synchrone Aufrufe
→ Konsistenz gesichert



OOSTuBS immer synchrone Aufrufe

→ Konsistenz gesichert

MPStuBS Synchronisation notwendig

OOSTuBS immer synchrone Aufrufe

→ Konsistenz gesichert

MPStuBS Synchronisation notwendig

- Scheduling auf der Epilogebe

main()

//...

guard.enter()

scheduler.schedule()

StackPointer dummy;

context_switch(dummy, appfoo.sp)

AppFoo

kickoff(&appfoo)

guard.leave()

appfoo->action()

kout << "foo" << endl

guard.enter()

scheduler.resume()

context_switch(appfoo.sp, appbar.sp)

AppBar

kickoff(&appbar)

guard.leave()

appbar->action()

kout << "bar" << endl

guard.enter()

scheduler.resume()

context_switch(appbar.sp, appfoo.sp)

guard.leave()

kout << "foo" << endl

guard.enter()

scheduler.resume()

context_switch(appfoo.sp, appbar.sp)

guard.leave()

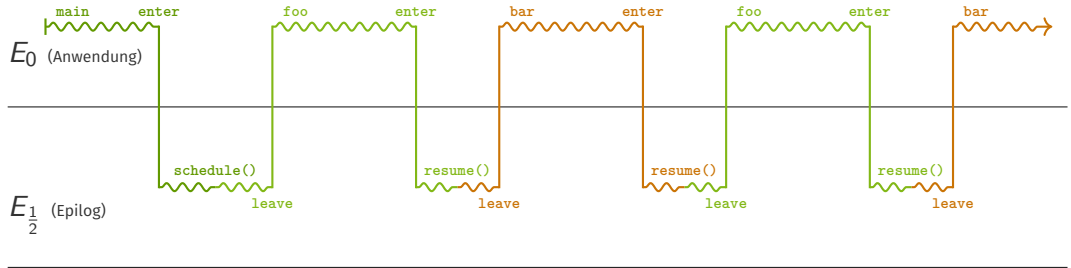
kout << "bar" << endl

foo

bar

foo

bar



E_1 (IRQ/Prolog)



Fragen?

Abgabe der Aufgabe
bis Mittwoch, den 07. Januar

Bild: twemoji

