Übung zu Betriebssysteme

Kontextwechsel

Wintersemester 2021/22

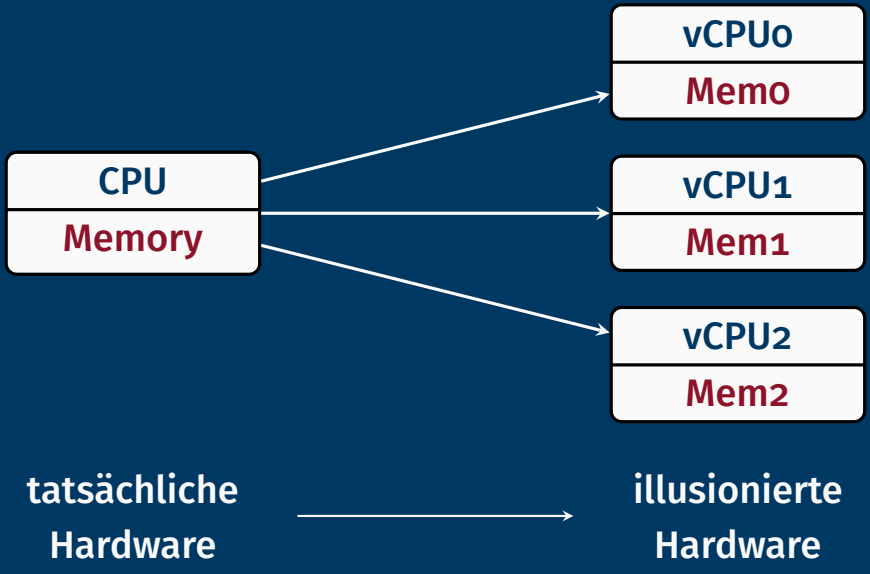
Bernhard Heinloth

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg

Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT


CPU
Memory
vCPU0
Mem0
vCPU1
Mem1
vCPU2
Mem2
tatsächliche Hardware
illusionierte Hardware

- Speicher virtualisieren ist einfach:

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

- CPU virtualisieren
  - nicht teilbar im Ort
  - aber teilbar in der Zeit

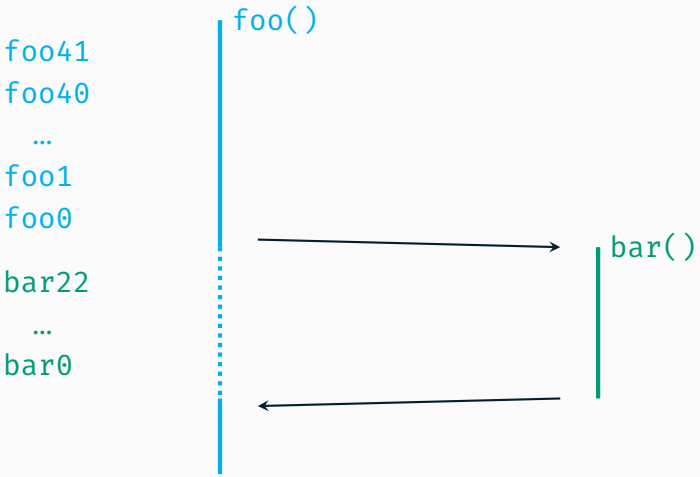
Koroutinen

Einseitiger Aufruf

```
void foo(){
  int f = 42;

  while (f--){
    kout << "foo"
         << f
         << endl;

  }
  bar();
}
```

```
void bar(){
  int b = 23;

  while (b--){
    kout << "bar"
         << b
         << endl;

  }

}
```

Einseitiger Aufruf



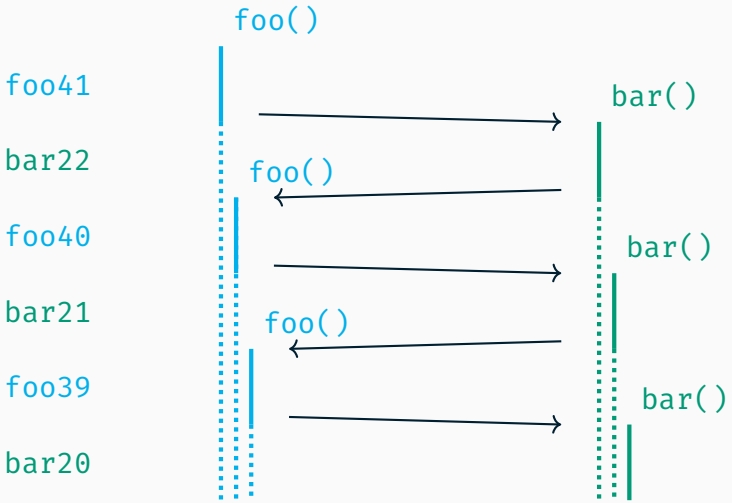
Einseitiger Aufruf

Gegenseitiger Aufruf

```
void foo(){
  static int f = 42;

  while (f--){
    kout << "foo"
         << f
         << endl;
    bar();
  }

}
```

```
void bar(){
  static int b = 23;

  while (b--){
    kout << "bar"
         << b
         << endl;
    foo();
  }

}
```

Gegenseitiger Aufruf



Gegenseitiger Aufruf

hoher Speicherverbrauch & (ggf.) Endlosrekursion

Umschaltung

Kontrollflusszustand sichern & laden

- Stack
- Register

} Kontext

- Umschaltefunktion

```
context_switch(StackPointer * current, StackPointer * next)
```

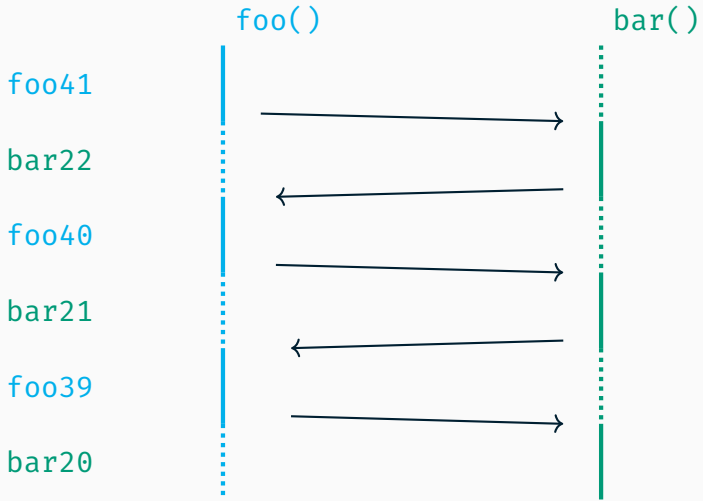
Wechselt vom aktuellen Kontext `current` nach `next`

Umschaltung

```
void foo(){
  int f = 42;

  while (f--){
    kout << "foo"
         << f
         << endl;
    context_switch(
        &stack_foo,
        &stack_bar
      );
  }
}
```

```
void bar(){
  int b = 23;

  while (b--){
    kout << "bar"
         << b
         << endl;
    context_switch(
        &stack_bar,
        &stack_foo
      );
  }
}
```

Umschaltung



Umschaltung

✓ Genau was wir wollen

Umsetzung des Kontextwechsel

Umschaltung im Detail

Kontrollflusszustand sichern & laden

Notwendige **Schritte** in `context_switch`:

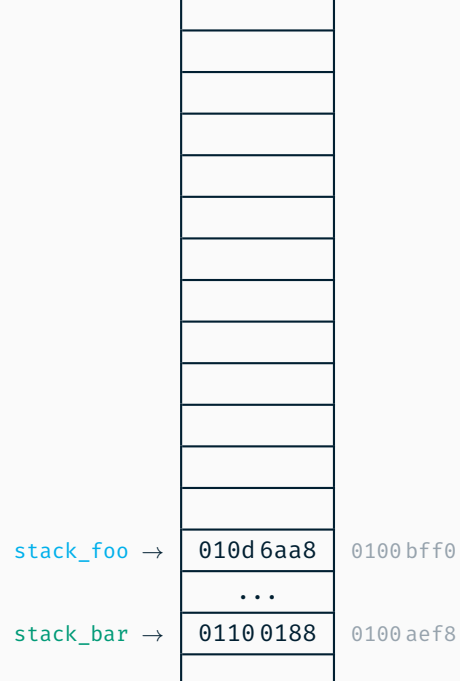
1. Register sichern (*via Stack*)
2. Stackpointer (`rsp`) sichern
3. Stackpointer (`rsp`) laden
4. Register wiederherstellen

Ziel: Instruktionszeiger `rip` ändern

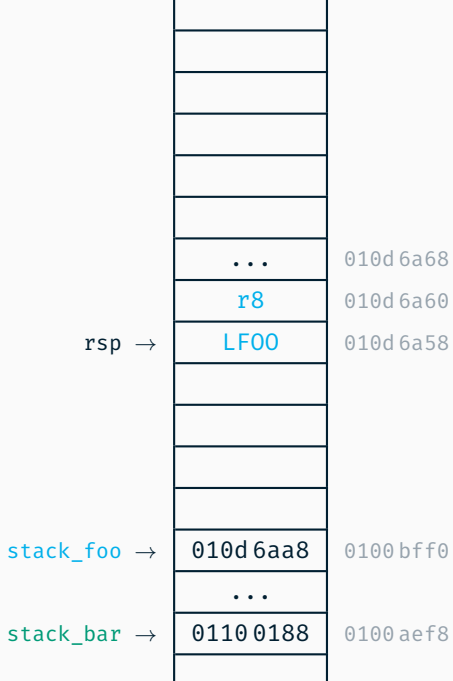
Problem: Register `rip` kann nicht direkt beschrieben werden

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

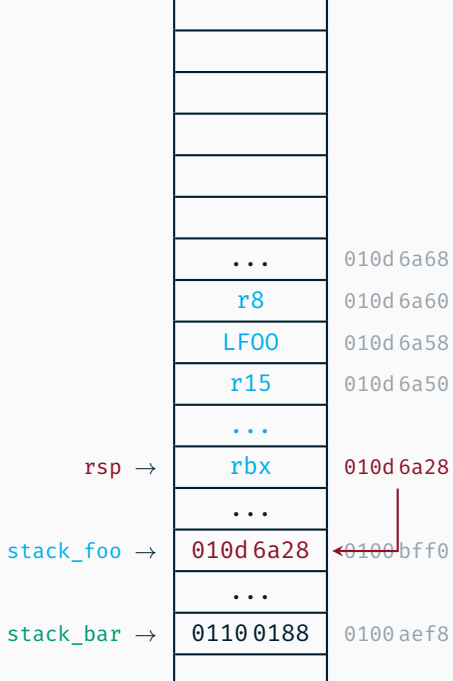
```
// Global sichtbar:

struct StackPointer {
  void *kernel;

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

StackPointer stack_foo;

StackPointer stack_bar;

// Sinnvoll initialisieren
// (mehr dazu später...)
```



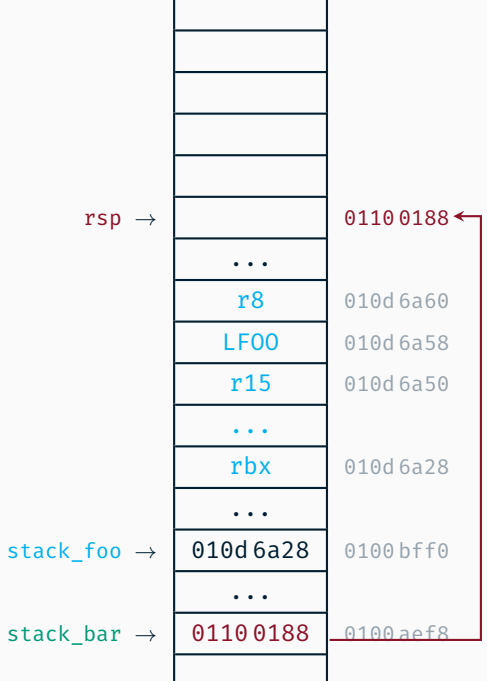
```
void foo(){
  ...

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

  context_switch(
      &stack_foo,
      &stack_bar
    );
LFOO:

  ...
```



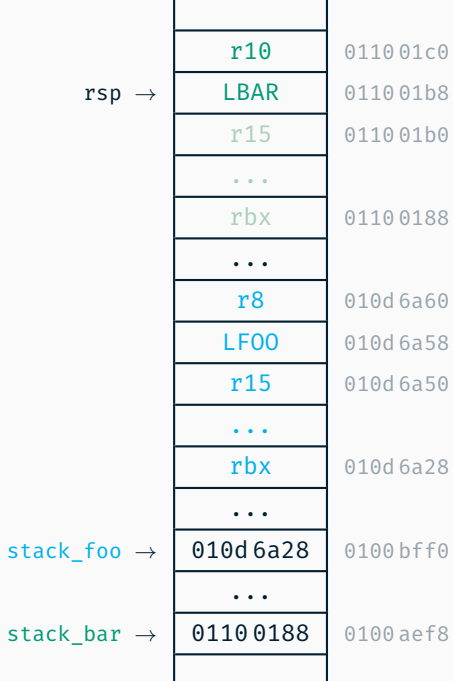
```
context_switch:

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

  ;; Stackzeiger von
  ;; foo sichern

  ret
```



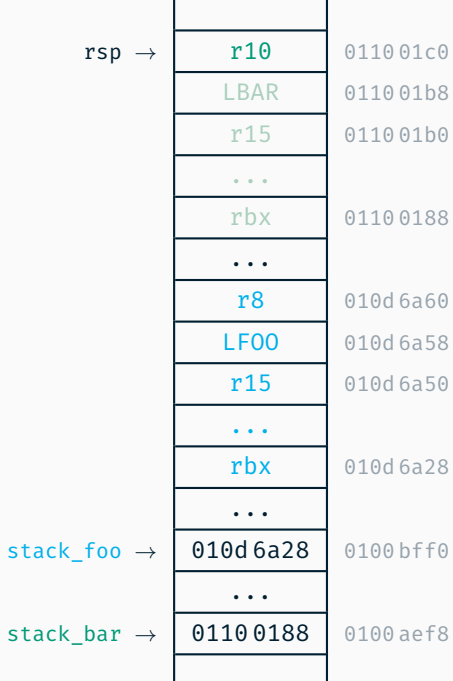
```
context_switch:

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

  ;; Stackzeiger von
  ;; foo sichern

  ;; Stackzeiger von
  ;; bar laden




  ret
```



```
context_switch:

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

  ;; Stackzeiger von
  ;; foo sichern

  ;; Stackzeiger von
  ;; bar laden

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

  ret
```

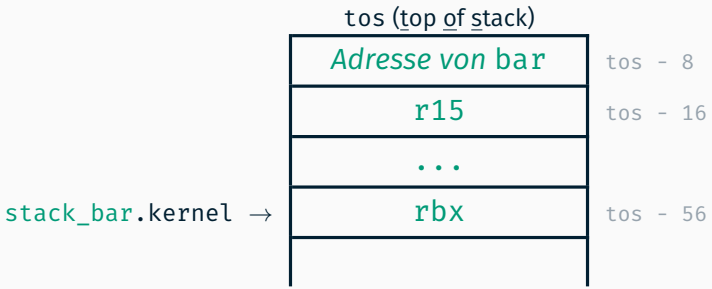


```
void bar(){
  ...

  context_switch(
      &stack_bar,
      &stack_foo
    );
LBAR:

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

  ...
```



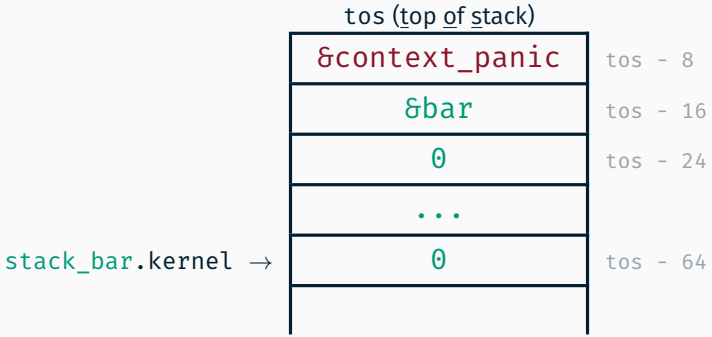
Stack aufsetzen

Stack für Einsprung in Koroutine `void bar()`



Stack aufsetzen (Robust)

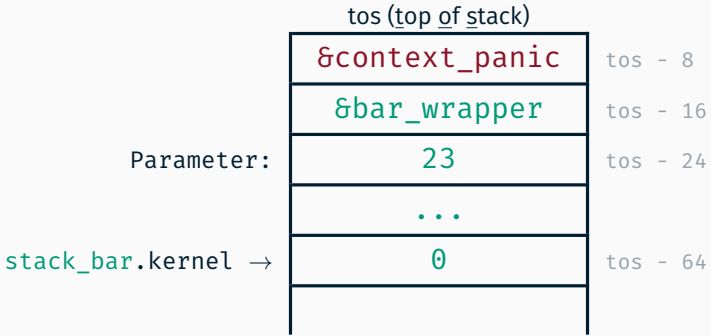
```
void context_panic() {
  kernelpanic("Application should not return!");
}
```



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`)



Koroutine (erstmalig) starten

Was wird für jede Koroutine gebraucht?

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

Wie rufe ich die aller erste Koroutine auf?

- `context_switch` mit Dummy-Stackzeiger als 1. Parameter