Übung zu Betriebssysteme

Interruptsynchronisation

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

Interrupts verändern (potenziell) den Zustand des Systems

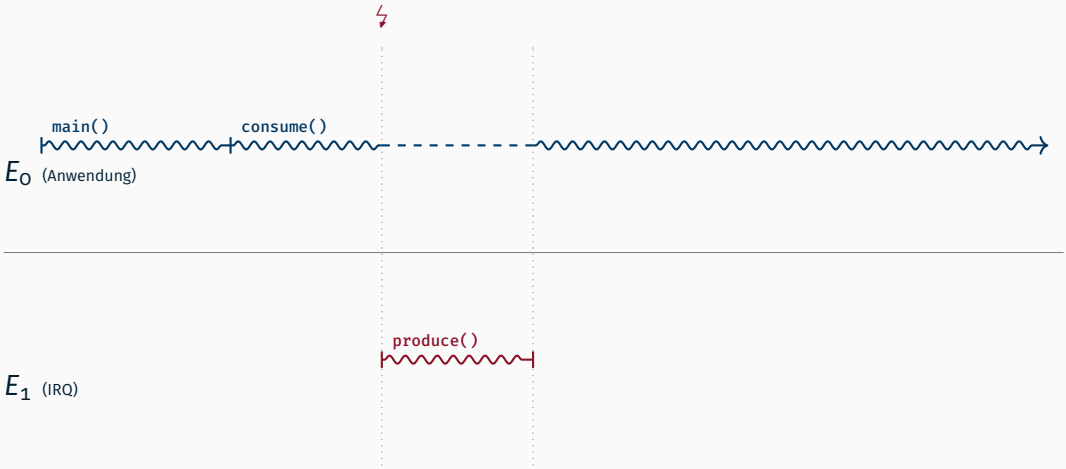
```
main(){
  while(1){
    // ...
    consume();
    // ...
  }
}
```

```
interrupt_handler(){

  produce();

}
```

Ohne Synchronisation



Ohne Synchronisation

```
int buf[SIZE];
int pos = 0;

void produce(int data) {
    if (pos < SIZE)
        buf[pos++] = data;
}

int consume() {
    return pos > 0 ? buf[--pos] : -1;
}
```

Ohne Synchronisation

Lost Update möglich!

Bewährtes Hausmittel: Mutex

```
void produce(int data) {
    mutex.lock();
    if (pos < SIZE)
        buf[pos++] = data;
    mutex.unlock();
}

int consume() {
    mutex.lock();
    int r = pos > 0 ? buf[--pos] : -1;
    mutex.unlock();
    return r;
}
```

Bewährtes Hausmittel: Mutex

Verklemmt sich!

Weiche Synchronisation

```
void produce(int data) {
    if (pos < SIZE)
        buf[pos++] = data;
}

int consume() {
    int x, r = -1;
    if (pos > 0)
        do {
            x = pos;
            r = buf[x];
        } while(!CAS(&pos, x, x-1));
    return r;
}
```

Weiche Synchronisation

Funktioniert!

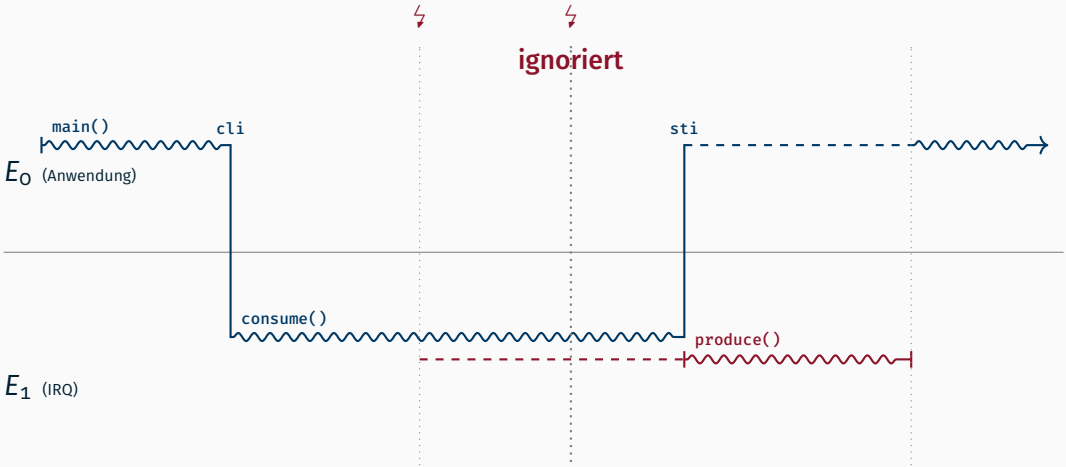
Weiche Synchronisation

Optimistischer Ansatz

- \+ keine Interruptsperre
- \+ kann sehr effizient sein
- – kein generischer Ansatz
- – kann sehr kompliziert werden
- – fehleranfällig

Betriebssystemarchitekt sollte Treiber- und Anwendungsentwicklern entgegenkommen ➔ für Erfolg des Betriebsystems entscheidend

Harte Synchronisation



Harte Synchronisation

Pessimistischer Ansatz

- \+ einfach
- \+ funktioniert immer
- – Verzögerung von IRQs
  ➔ hohe Latenz, ggf. Verlust von Interrupts
- – blockiert pauschal alle IRQs

Prolog/Epilog-Modell

Idee

Aufspalten der IRQ-Behandlung in

Prolog erledigt das Nötigste auf E_1

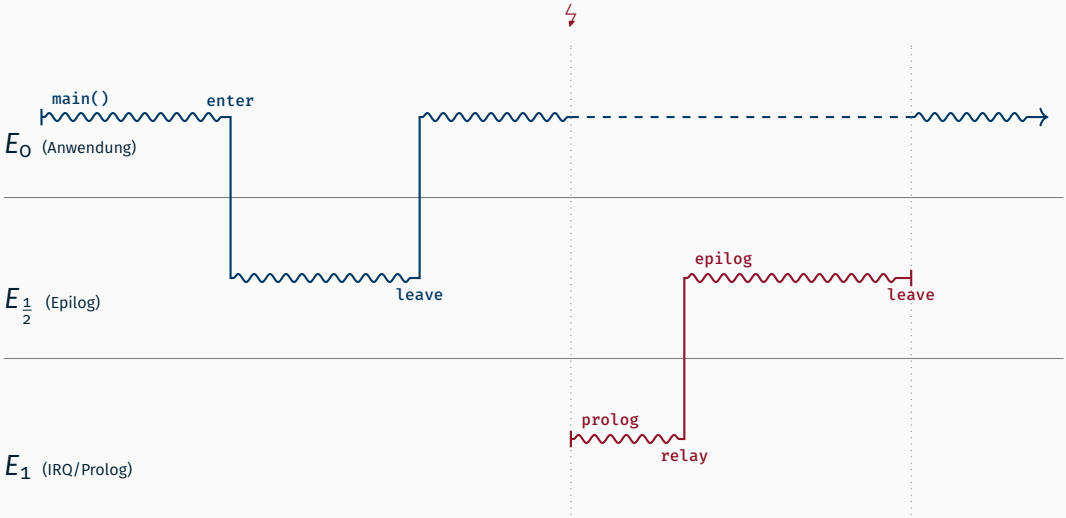
Epilog läuft auf neuer Ebene $\frac{1}{2}$ und übernimmt Synchronisation somit Ebene `1` / IRQs wieder frei

Operationen

- höhere Ebene betreten: `cli`, `enter`
- höhere Ebene verlassen: `sti`, `leave`
- niedrigere Ebene unterbrechen: **IRQ-Leitung**, `relay`

bei **harter Synchronisation** und **Prolog/Epilog-Modell**

Prolog/Epilog-Modell



Prolog/Epilog-Modell

Kombinierter Ansatz

- \+ einfaches Programmiermodell
  (für Anwendungsentwickler)
- \+ geringer Interruptverlust
- – Ebene $\frac{1}{2}$ ist zusätzlicher Overhead
- – etwas mehr Arbeit für den Betriebssystemarchitekten

→ **guter Kompromiss**

Umsetzung

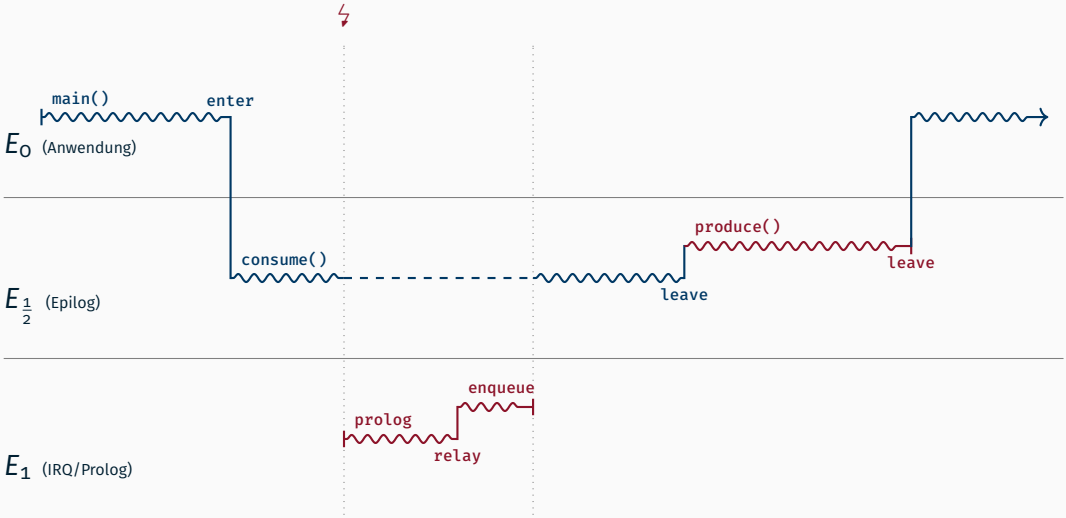
```
main(){
  while(1){
    enter();
    consume();
    leave();
  }
}
```

```
epilog(){

  // ...
  produce();
  // ...

}
```

Umsetzung


main()
enter
E_0 (Anwendung)
consume()
produce()
leave
$E_{\frac{1}{2}}$ (Epilog)
leave
enqueue
prolog
relay
E_1 (IRQ/Prolog)