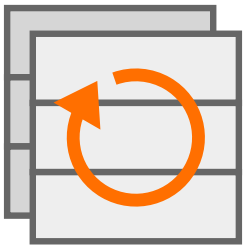


Operating
System
Group

TUHH
Technische Universität Hamburg

BSB – Übung 6: IPC

Yannick Loeck

2022-06-29

Inhalt

TUHH
Technische Universität Hamburg

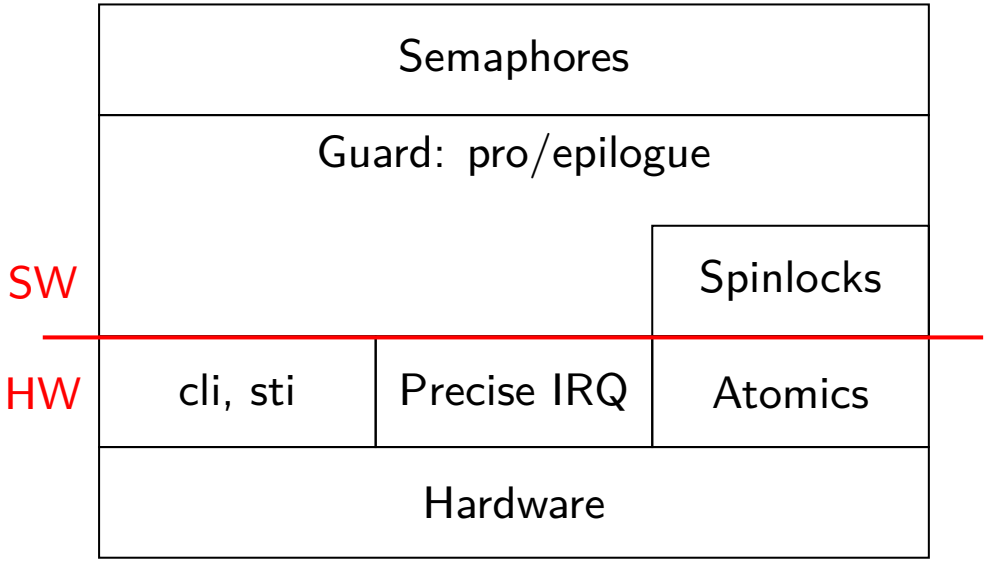
BSB - HÜ6

Threads schlafen lassen und Strom sparen

Worum geht es in der Übung heute:

- Synchronisation mit Semaphoren
- Threads und CPUs schlafen legen

Synchronisation



- Threads sollen jederzeit verdrängbar sein
- Geht nicht mit kritischem Abschnitt
- Synchronisationsobjekt auf E0: Semaphoren
- Semaphoren-Operationen `Guard`-en
- Synchronisation zwischen Ebenen vs. zwischen CPUs

Semaphore

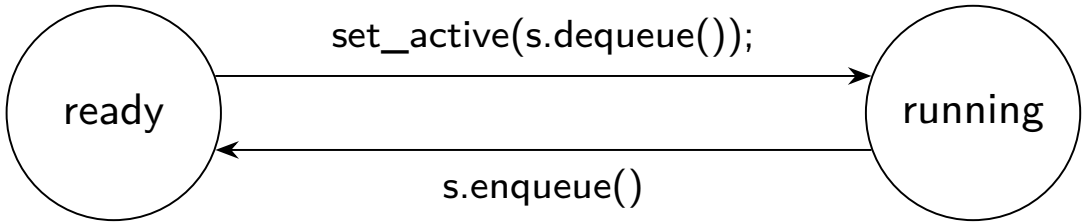
TUHH
Technische Universität Hamburg

- Positiver Integer mit Operationen
- `P()/down()`: *probeer te verlagen* / versuchen zu senken
- `V()/up()`: *verhogen* / erhöhen

Einsatzmöglichkeiten

1. Critical Section, binäre Semaphore
2. Ressourcenverwaltung, zählende Semaphore
3. Signalisierung, zählende Semaphore

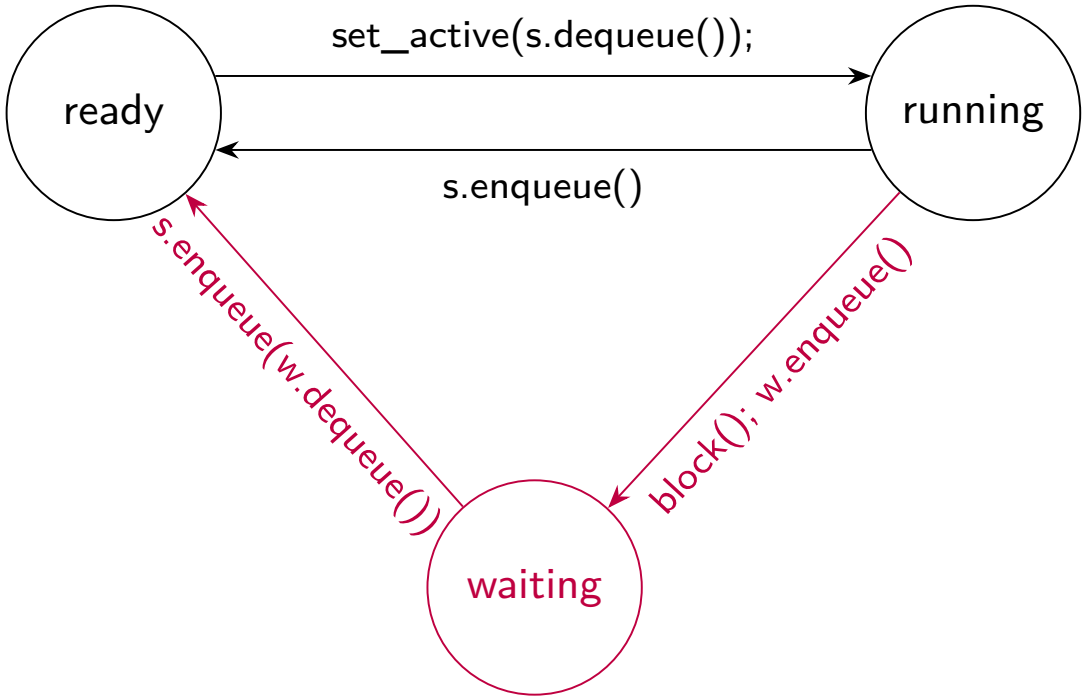
Thread-Zustände



Wir haben bisher 2 Zustände:

- running: Thread in life pointer vom `dispatcher`
- ready: Thread in ready queue vom `scheduler`

Thread-Zustände



Wir haben bisher 2 Zustände:

- running: Thread in life pointer vom `dispatcher`
- ready: Thread in ready queue vom `scheduler`
- waiting: Thread setzt sich selbst wartend

Waitingroom und Semaphore


```
class Waitingroom { Queue<Thread> waiting; }
```

- `Waitingroom` bietet Liste an wartenden Threads
- Thread aus Waitingroom entfernen: in ready!

- `Semaphore` erbt von `Waitingroom`
- Operationen durch `Guard` geschützt
  - `counter++, wakeup`
  - `counter--, block`

Beispiel: Bounded Buffer


```
BoundedBuffer bb(20);
```

```
produce(c) {

    bb.put(c);

}
```

```
consume() {

    char x = bb.get()

    return x;
}
```

- Ziel: Implementierung mit Semaphoren schützen

Beispiel: Bounded Buffer

TUHH
Technische Universität Hamburg

```
BoundedBuffer bb(20);
Semaphore lock(1);
```

```
produce(c) {

    lock.P();
    bb.put(c);
    lock.V();

}
```

```
consume() {

    lock.P();
    char x = bb.get()
    lock.V();

    return x;
}
```

- Ziel: Implementierung mit Semaphoren schützen

1. Buffer schützen: lock für kritischen Abschnitt — binär

Beispiel: Bounded Buffer

TUHH
Technische Universität Hamburg

```
BoundedBuffer bb(20);
Semaphore lock(1);
Semaphore empty(20);
```

```
produce(c) {
     empty.P();
     lock.P();
     bb.put(c);
     lock.V();

}
```

```
consume() {

     lock.P();
     char x = bb.get()
     lock.V();
     empty.V();
     return x;
}
```

- Ziel: Implementierung mit Semaphoren schützen

1. Buffer schützen: lock für kritischen Abschnitt — binär
2. Overflow verhindern: begrenzte Ressource verwalten — zählend

Beispiel: Bounded Buffer

TUHH
Technische Universität Hamburg

```
BoundedBuffer bb(20);
Semaphore lock(1);
Semaphore empty(20);
Semaphore elems(0);
```

```
produce(c) {
    empty.P();
    lock.P();
    bb.put(c);
    lock.V();
    elems.V();
}
```

```
consume() {
    elems.P();
    lock.P();
    char x = bb.get()
    lock.V();
    empty.V();
    return x;
}
```

- Ziel: Implementierung mit Semaphoren schützen

1. Buffer schützen: lock für kritischen Abschnitt — binär
2. Overflow verhindern: begrenzte Ressource verwalten — zählend
3. Nur consume wenn Element im Buffer ist: Signalisierung — zählend

Beispiel: Bounded Buffer

TUHH
Technische Universität Hamburg

```
BoundedBuffer bb(20);
Semaphore lock(1);
Semaphore empty(20);
Semaphore elems(0);
```

```
produce(c) {
    empty.P();
    lock.P();
    bb.put(c);
    lock.V();
    elems.V();
}
```

```
consume() {
    elems.P();
    lock.P();
    char x = bb.get()
    lock.V();
    empty.V();
    return x;
}
```

- Alle Verwendungsmuster von Semaphoren:

1. Buffer schützen: lock für kritischen Abschnitt — binär
2. Overflow verhindern: begrenzte Ressource verwalten — zählend
3. Nur consume wenn Element im Buffer ist: Signalisierung — zählend

Warten auf Zeit

Bellringer

TUHH
Technische Universität Hamburg

- Thread legt sich schlafen
- Kann sich dann nicht mehr selbst aufwecken

- Anderes Objekt prüft das: `Bellringer`
- Verwaltet Liste von Threads + Wartezeiten
- Thread(s) aufwecken? `check()` im Epilog von `Watch`

Bellringer

TUHH
Technische Universität Hamburg

- Mehrere Threads mit selber Wartezeit möglich
- `Bell`-Objekt, erbt von `Waitingroom`, enthält Thread(s)
- Wenn `Bell`-Alarm losgeht: Alle zugehörigen Threads aufwecken

- Effizienz wichtig: `Bellringer::check()` bei *jedem* Timer-tick

Für OS-Objekte

So bauen, dass es auch mit vielen Instanzen noch schnell ist.

Optimierung

TUHH
Technische Universität Hamburg

Wir müssen für eine Liste von Alarmen:

1. Neue Elemente einhängen (`Bellringer::job()`)
2. Schnell herausfinden ob ein Alarm dran ist (`Bellringer::check()`)

`check()` geht in $\sim$O(1)!

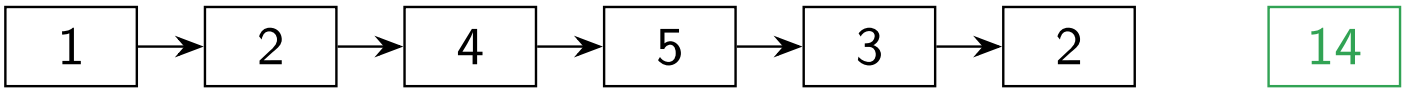
Optimierung

Wir müssen für eine Liste von Alarmen:

1. Neue Elemente einhängen (`Bellringer::job()`)
2. Schnell herausfinden ob ein Alarm dran ist (`Bellringer::check()`)

`check()` geht in $\sim$O(1)!

- Zeiten umrechnen zu: Alarm löst in *n* Watch-ticks aus
- `Bell`-Liste sortieren
- Relative Bell-Zeiten: Runterzählen der ersten Bell implizit für alle



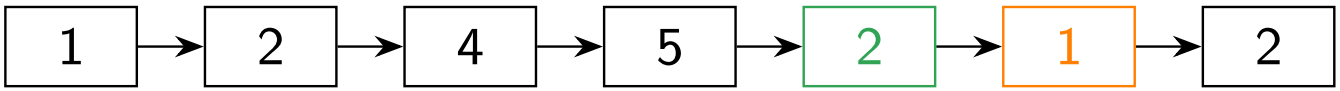
Optimierung

Wir müssen für eine Liste von Alarmen:

1. Neue Elemente einhängen (`Bellringer::job()`)
2. Schnell herausfinden ob ein Alarm dran ist (`Bellringer::check()`)

`check()` geht in $\sim$O(1)!

- Zeiten umrechnen zu: Alarm löst in n Watch-ticks aus
- `Bell`-Liste sortieren
- Relative Bell-Zeiten: Runterzählen der ersten Bell implizit für alle



CPU schlafen legen

Idle Threads

TUHH
Technische Universität Hamburg

- Alle Applications schlafen, was dann?
- CPUs brauchen Arbeit
- Ein `IdleThread : public Thread` pro CPU
- Kommt nur dran wenn ready-Liste leer ist
- Anwendungen haben Priorität

Was tun in `IdleThread::action()`?

TUHH
Technische Universität Hamburg

- Naive Lösung: `while (1) {}`
- Stromverschwendung!

- CPUs haben Instruktion die sie schlafen legt: `hlt`
- Wie lange? Bis ein IRQ kommt
- Das bedeutet auch: `cli; hlt` == Schlaf für immer

Was tun in `IdleThread::action()`?

TUHH
Technische Universität Hamburg

- Also: `while (1) {hlt;}`

Was tun in `IdleThread::action()`?

TUHH
Technische Universität Hamburg

- Also: `while (1) {hlt;}` ?

Was tun in `IdleThread::action()`?

TUHH
Technische Universität Hamburg

- Also: `while (1) {hlt;}` ?
- Nein: Würde ein Interrupt kommen -> Wakeup der CPU -> `hlt`
- CPU schläft weiter bis Timer-Interrupt `resume()` aufruft
- Also: `while (1) {hlt; scheduler.resume();}`

Was tun in `IdleThread::action()`?

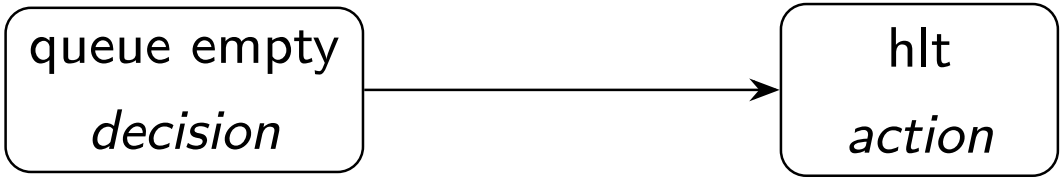
TUHH
Technische Universität Hamburg

- Also: `while (1) {hlt;}` ?
- Nein: Würde ein Interrupt kommen -> Wakeup der CPU -> `hlt`
- CPU schläft weiter bis Timer-Interrupt `resume()` aufruft
- Also: `while (1) {hlt; scheduler.resume();}` ?

Was tun in `IdleThread::action()`?

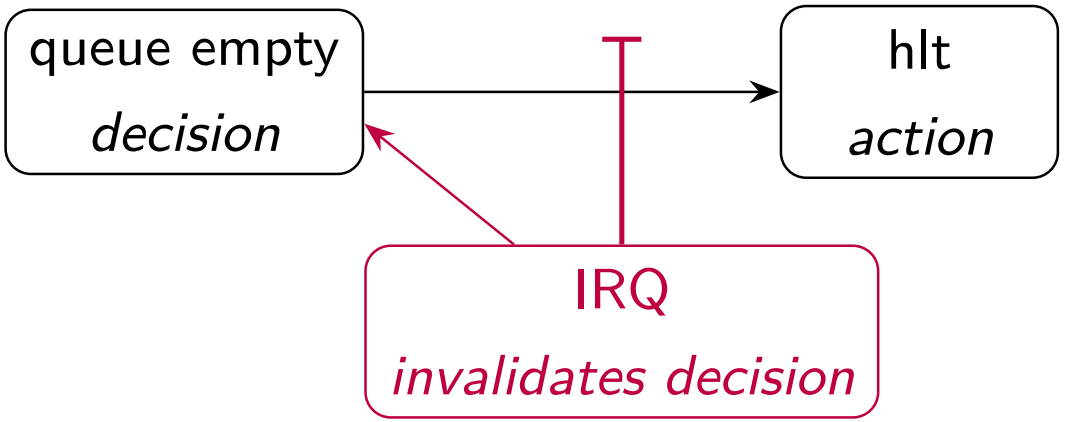
TUHH
Technische Universität Hamburg

- Also: `while (1) {hlt;}` ?
- Nein: Würde ein Interrupt kommen -> Wakeup der CPU -> `hlt`
- CPU schläft weiter bis Timer-Interrupt `resume()` aufruft
- Also: `while (1) {hlt; scheduler.resume();}` ?
- Lost-Wakeup: Interrupt kann zwischen `resume()` und `hlt` kommen.



Was tun in `IdleThread::action()`?

- Also: `while (1) {hlt;}` ?
- Nein: Würde ein Interrupt kommen -> Wakeup der CPU -> `hlt`
- CPU schläft weiter bis Timer-Interrupt `resume()` aufruft
- Also: `while (1) {hlt; scheduler.resume();}` ?
- Lost-Wakeup: Interrupt kann zwischen `resume()` und `hlt` kommen.



Unlösbares Problem?

TUHH
Technische Universität Hamburg

Interrupt unmittelbar vor `hlt`:

- `Guard` schützt nicht vor Interrupts
- Interrupts ausschalten? CPU schläft unendlich
- Es muss ein `sti` vor `hlt` kommen
- Prüfen ob ready-Liste noch leer ist: `cli; isEmpty(); sti; hlt;`

Unlösbares Problem?

TUHH
Technische Universität Hamburg

Interrupt unmittelbar vor `hlt`:

- `Guard` schützt nicht vor Interrupts
- Interrupts ausschalten? CPU schläft unendlich
- Es muss ein `sti` vor `hlt` kommen
- Prüfen ob ready-Liste noch leer ist: `cli; isEmpty(); sti; hlt;`
- **Glück**: CPU-Designer haben das Problem bedacht
- Instruktion nach `sti` ist immer atomar damit zusammen
- Es kann also kein Interrupt dazwischen kommen

Noch besser Strom sparen

TUHH
Technische Universität Hamburg

- Regelmäßiger Wakeup von Idle Thread
- Problem: Auch wenn keine Arbeit da ist
- Kein tieferer CPU sleep: Stromverschwendung

- Watch am LAPIC maskieren, nach `hlt` wieder einschalten
- Unterschied mit Stromzähler messbar

Achtung!

Bellringer kann die Watch noch brauchen!

Aufgabe 6

Aufgabe 6

TUHH
Technische Universität Hamburg

- `Waitingroom` und `Semaphore`
- Testen: Keyboard-Ausgabe in `kappl` statt Epilog
  Mit Semaphore auf Tasten warten
- `Bellringer`, queue richtig verwalten
- Testen: Threads schlafen legen (nicht zu viele)
- `IdleThread` pro CPU erstellen
- Testen: Weniger Applications als CPUs laufen lassen
- Watch in `IdleThread` blockieren