Übung zu Betriebssysteme

Zeitgesteuertes Warten

Wintersemester 2020/21

Bernhard Heinloth & Christian Eichler

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

Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Schlafen legen

```
App::action(){
    foo();
    sleep(13);
    bar();
}
```

- ähnlich der Funktion `sleep(3)`
- jedoch mit Wartezeit in Millisekunden (statt Sekunden)
- analog zu Wartezimmer den Thread aus Ready-Liste des Schedulers nehmen

- mit jedem Tick die Wartezeit dekrementieren
- nach Ablauf der Wartezeit Thread wieder im Scheduler einreihen

Naive Datenstruktur

Beispiel: Drei Anwendungen legen sich nacheinander schlafen

1. Thread **foo** für `666ms`
2. Thread **bar** für `23ms`
3. Thread **baz** für `42ms`

Verwaltung mittels verketteter Liste hat den Nachteil, dass bei jedem Tick die gesamte Liste durchlaufen werden muss

($\mathcal{O}(n)$, und das $1000\times$ pro Sekunde in der Epilogebene)

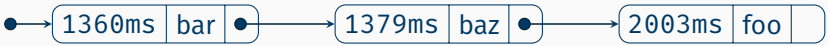
Das muss besser gehen!

Alternative Variante

- Einführung einer absoluten Zeit
  - wird mit jedem Tick inkrementiert
- Berechnung der Endzeit beim Einfügen neuer Threads
- Einordnen in einer Vorrangwarteschlange ($\mathcal{O}(n)$)
- Wenn die aktuelle Zeit T dem ersten Element entspricht:
  Thread wieder dem Scheduler übergeben ($\mathcal{O}(1)$)

Beispiel: absolute Zeit `T = 1337ms`

1. Thread **foo**: `666ms + 1337ms`
2. Thread **bar**: `23ms + 1337ms`
3. Thread **baz**: `42ms + 1337ms`



Alternative Variante

- Einführung einer absoluten Zeit
  - wird mit jedem Tick inkrementiert
- Berechnung der Endzeit beim Einfügen neuer Threads
- Einordnen in einer Vorrangwarteschlange ($\mathcal{O}(n)$)
- Wenn die aktuelle Zeit T dem ersten Element entspricht:
  Thread wieder dem Scheduler übergeben ($\mathcal{O}(1)$)

Nachteile

- Absolute Zeit ist ein neuer Zustand
- Bei 32bit Überlauf möglich (nach 49.7 Tagen)

Geht das nicht effizienter (ohne solche Probleme)?

Effiziente Variante

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit $t_0 = \texttt{0ms}$

Beispiel:

1. Thread **foo**: `666ms`

Effiziente Variante

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit $t_0 = \texttt{0ms}$
- Neue Threads nach Schlafdauer einordnen ($\mathcal{O}(n)$)
  - Nachfolgendes Element muss angepasst werden ($\mathcal{O}(\texttt{1})$)

Beispiel:

1. Thread **foo**: `643ms`
2. Thread **bar**: `23ms`

Effiziente Variante

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit $t_0 = \texttt{0ms}$
- Neue Threads nach Schlafdauer einordnen ($\mathcal{O}(n)$)
  - Nachfolgendes Element muss angepasst werden ($\mathcal{O}(1)$)

  $\Rightarrow$ keine Vorrangwarteschlange!

Beispiel:

1. Thread **foo**: 624ms
2. Thread **bar**: 23ms
3. Thread **baz**: 19ms

Effiziente Variante

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit $t_0 = \texttt{0ms}$
- Neue Threads nach Schlafdauer einordnen ($\mathcal{O}(n)$)
  - Nachfolgendes Element muss angepasst werden ($\mathcal{O}(1)$)

  $\Rightarrow$ keine Vorrangwarteschlange!
- Erstes Element wird mit jedem Tick dekrementiert und bei 0 dem Scheduler übergeben ($\mathcal{O}(1)$)

  Beispiel:

  1. Thread **foo**: 624ms
  2. Thread **bar**: 23ms
  3. Thread **baz**: 19ms