Nebenläufige Programmierung

Perspektiven der Informatik
27. Januar 2003
Gert Smolka

Telefon-Szenario

- Eine Telefonzelle
- Mehrere Personen wollen telefonieren
- Immer nur eine Person kann telefonieren

- Ressource
- Prozesse
- Gegenseitiger Ausschluß

Brotstand-Szenario

- Brotstand mit einer Verkäuferin
- Mehrere Personen wollen Brot kaufen
- Immer nur eine Person wird bedient

- Fairness
- Verhungern
- Warteschlange

Konto-Szenario

- Konto bei Bank
- Geldautomaten, Geschäfte und Banken wollen auf Konto zugreifen
- Immer nur einer kann auf Konto zugreifen

- Mit Computern realisiert
- Indeterminismus

Schi-Szenario

- Wochende im Schnee
- Auto mieten (1. Ressource)
- Schier mieten (2. Ressource)

- Autovermietung hat nur noch ein Auto
- Sportgeschäft hat nur noch ein Paar Schier
- Karl und Berta, die sich nicht kennen, wollen beide in den Schnee

4 Möglichkeiten

- Karl bekommt Auto und Schier
- Berta bekommt Auto und Schier
- Karl bekommt Auto und Berta die Schier
- Berta bekommt das Auto und Karl die Schier

- Verklemmung (Deadlock)
- Verklemmungsgefahr besteht, wenn ein Prozess mehrere Ressourcen gleichzeitig benötigt

Dining Philosophers

- Drei Philosophen sitzen am Tisch
- Es gibt nur drei Essstäbchen
- Man braucht zwei Stäbchen zum Essen

- [Dijkstra, etwa 1960]
- Probleme wurden entdeckt bei der Konstruktion von Betriebssystemen

Wichtige Begriffe

- Ressourcen und Prozesse
- Gegenseitiger Ausschluss
- Konkurrenzsituation
- Fairness / Verhungern
- Verklemmungsgefahr
- Indeterminismus

Betriebssystem

- Verwaltungsprogramm für Computer
- Kann mehrere Programme ausführen
- Prozess = laufendes Programm
- Ein Computer, viele Prozesse
- BS bietet Kommunikationsmöglichkeiten für Prozesse
- Vernetzung => Prozesse auf verschiedenen Computern können miteinander kommunizieren

Parallelität

- Gleichzeitige Ausführung von Operationen
- Computer mit mehreren Prozessoren
- Prozesse ≠ Prozessoren
- Nebenläufigkeit ≠ Parallelität
- Parallelisierung von Programmen
- Parallele Algorithmen
- Prozessoren parallelisieren intern

Primäre HW/SW-Schnittstelle ist (noch) sequentiell

Nebenläufige Software

↓

Sequentieller Befehlssatz

↓

Paralleler Prozessor

Threads

- Thread = sequentielle Ausführung innerhalb eines Prozesses
- Mehrere Threads pro Prozess
- Prozesse haben getrennte Speicherbereiche
- Threads eines Prozesses sehen denselben Speicherbereich

Rest des Vortrags

- Nebenläufige Programmierung mit der Programmiersprache Alice
- Erweiterung von Standard ML
- Plattform zum Experimentieren
- Entwicklung am Lehrstuhl Programmiersysteme
- www.ps.uni-sb.de/alice/

Threads und Futures

`spawn` *Ausdruck*

- Erzeugt neuen Thread, der Ausdruck auswertet
- Liefert Future als Stellvertreter für den Wert des Ausdrucks

⇒ Demo

Datenfluss-Synchronisation

```
(spawn f x) + (spawn g y)
```

- Operationen blockieren automatisch, wenn sie Wert benötigen
- Threads können parallelisiert werden

- [Multilisp, etwa 1985 am MIT]

Portale und Ströme

- Nebenläufige Datenstruktur, ermöglicht many-to-one Kommunikation zwischen Threads

```
port : unit → (α → unit) * α list
```

- Prozedur legt Botschaft in einen Strom
- Strom ist angefangene Liste mit Future

$x_1::x_2::x_3::x_4::x_5::\mathtt{Future}$

⇒ Demo

Bedarfsgesteuerte Berechnung

`lazy` *Ausdruck*

- Liefert Future als Stellvertreter für den Wert des Ausdrucks
- Erst wenn der Wert benötigt wird, wird der Ausdruck auswertet
- Programmieren mit unendlichen Listen

⇒ Demo

Bedarfsgesteuerte Berechnung

- Bedarfsgesteuertes Laden von Programmkomponenten (z. B. in Java)
- Bedarfsgesteuertes Übersetzen von Programmen (JIT Compiler)
- Haskell ist eine funktionale Sprache, in der bedarfsgesteuerte Berechnug voll ausgereitzt wird; lazy functional programming

Monitore

- Ressource wird durch Operationen zugegriffen (z. B. Konto)
- Monitor sequentialisiert Operationsanwendungen automatisch
- [Hoare, Per Brinch Hansen, etwa 1975]

```
monitor  : unit → monitor
register : monitor → (α→β) → (α→β)
```


Remote Procedure Calls

- Zwei Prozesse A und B
- In B gibt es eine Prozedur p
- A will eine Prozedur p in B aufrufen
  - A braucht eine Referenz auf p in B
  - Argument muss von A nach B gebracht werden
  - Ergebnis muss von B nach A gebracht werden
- Proxy ist Stellvertreter für Prozedur p
  - Erledigt Transfer von Argument und Ergebnis
  - Kann zwischen Prozessen hin und her gereicht werden

Proxies

- Erzeugen ist einfach

```
proxy : (α→β) → (α→β)
```

- Aber wie kommt ein Proxy in einen anderen Prozess?

Offer und Take

```
offer : α → ticket
take  : ticket → α
```

- Take muss prüfen, ob der übergebene Wert den erwarteten Typ hat
- Offerieren von Strukturen wäre besser
- Strukturen sind aber keine Werte

Pakete

```
pack   : structure * signature → package
unpack : package * signature → structure
```

- Ein Paket enthält eine Struktur und eine Signatur gemäß der die Struktur benutzt werden kann
- Unpack erledigt dynamische Typprüfung
- Offer und take tauschen Pakete aus

Compute Server

- Proxy für

```
fun appy(f,x) = f x
```

Abschließende Bemerkungen

- Nebenläufigkeit in Betriebssysteme
- Nebenläufigkeit in Programmiersprachen
- Middleware (z. B. Corba)

Locks

```
lock   : unit → lock
obtain : lock → unit
return : lock → unit
```

- Lock ist primitive Ressource mit eingebautem wechselseitigen Ausschluss
- Fairness mit Warteschlange
- Dijkstra‘s Semaphore, etwa 1960

Locks mit Wait

```
lock   : unit → lock
obtain : lock → unit
return : lock → unit
wait   : lock → unit
wakeup : lock → unit
```

Java hat Monitore mit Wait

Kanäle

```
channel: unit → α channel
put    : α channel → α → unit
get    : α channel → α
```

- Get blockiert, wenn kein Wert im Kanal ist
- Zweifache Fairness Bedingung