Theoretische Aspecten van Programmatuur

HC 3 (week 5)

Bob Diertens

Theory of Computer Science (TCS)

date: Maart, 7

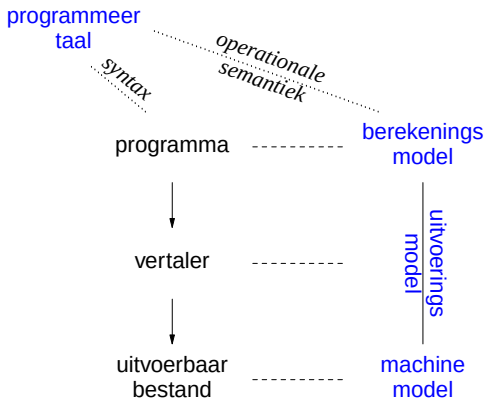
version: March 3, 2017

Wat

- Bestudering programmeer-constructies.
- Uitbreiding PGA (vnl. basis-instructies)
- Functies
- Berekeningsmodellen voor functies
- Concurrency support
- Communicatie
- Berekeningsmodel voor objecten
- Object-orientatie
- Gereedschap: Generalisatie & Abstractie

Concurrency

Uitvoering Programma



Allen gebaseerd op sequentiele uitvoering van instructions
Concurrentie?

Concurrentie vs Parallellisme

Concurrentie is het behandelen van een aantal dingen tegelijkertijd.

Parallellisme is het uitvoeren van een aantal dingen tegelijkertijd.

Ondersteuning voor concurrentie in machine modellen

Hoe deze ondersteuning te gebruiken?

Zoals andere toevoegingen

bibliotheek van functies

bv: mathematische functies bibliotheek, grafische functies bibliotheek, ...

Bibliotheek voor meervoudige draadjes / multi-threading

...?

zie
[P.A. Buhr, *Are Safe Concurrency Libraries Possible?*, 1995]
[H.J. Boehm, *Threads Cannot be Implemented as a Library*, 2005]

Thread Library

- varierend op verschillende platforms
  → gebruik standaard (Posix)
- berekenings model gebaseerd op sequentiele uitvoering van instructies
- vertaling gebaseerd op sequentieel uitvoerings model

Vertaling

- efficiente code
- correct voor sequentiele uitvoering van instructions
- mogelijk incorrect (of onverwacht) voor concurrente uitvoering van instructions
  oorzaak: gedeelde bronnen (shared resources)
  *** communicatie tussen draadjes (threads) door gedeeld geheugen (interferentie lees/schrijf acties, herschikking (reordering) van code)
- moeten problemen voorkomen zoals race conditions, deadlocks, data inconsistentie, etc

Problemen

thread1	thread2
`x ++`	`x ++`

De waarde van x?

thread1	thread2
`y1 = x`	`while(1) {`
`sleep(...)`	`  x++`
`y2 = x`	`}`
`d = y2 - y1`	

De waarde van d?

thread1	thread2
`y = x`	`z = y`

x, y, z zijn 64 bit integers op een 32 bit machine.

thread1	thread2
`k ++`	`x = k`
`y = x`	
`k ++`	

De waarde van y?

. . .

Problemen (cont'd)

- Cache
  L1, L2, ...
- Meerdere cores
- . . .

Ondersteuning voor concurrentie

Programming Languages

— extensies

— nieuw

Ada, Java, C#

Implementaties

— meestal met huidige vertalers / vertaler technologie, met gebruik van toegevoegde draadjes bibliotheek

— kan beter

- transparent
- geen idee wat er gaande is
  :-(
  gebruikers vriendelijk :-)

— Ontbreken van berekenings model

Ondersteuning voor concurrentie (cont'd)

Er worden wel oplossingen geboden (maar niet afgedwongen)

Blijft moeilijk om problemen te omzeilen.

bv met geheugen modellen
zie
[W. Pugh, *The Java Memory Model is fatally flawed*, 2000]
[J. Manson, W. Pugh, S. Adve, *The Java Memory Model*, 2005]

[A. Alexandrescu, H.J. Boehm, K. Henney, D. Lea, B. Pugh, *Memory Model for C++*, 2004]
[A. Alexandrescu, H.J. Boehm, K. Henney, B. Hutchings, D. Lea, B. Pugh, *Memory Model for C++*, 2005]

Ondersteuning voor concurrentie (cont'd)

Hoe dan wel?

Communiceer niet dmv het delen van geheugen,
maar deel geheugen dmv communicatie.

Gebruik een programmeertaal die dit ondersteunt.

bv. The Go Programming Language (golang.org)

De programmeertaal Go

- go functie(args)
- channels
- select
  wachten op meerdere communicatie operaties (+ random keuze)
- iedere core een eigen scheduler
- load balancer
- transparant
  - — # cores aanwezig
  - — # cores beschikbaar
  - — wat draait op welke core
  - — optimaal?
    echt nodig?

De programmeertaal Go (cont'd)

voorbeeld

```
func main() {
      var s string

      c1 := make(chan string)
      c2 := make(chan string)
      c3 := make(chan string)

      go workcell1(c1, c2)
      go workcell2(c2, c3)

         .
         c1 <- s
         .
         s = <- c3
         .
}
func workcell1(cin <- chan string, cout chan <- string) {
      var s string

         .
         s = <- cin
         .
         cout <- s
         .
}
```


Concurrency in PGA

Threads

Nieuwe instructie:

```
fork #k
```

Splitsing van huidige thread.

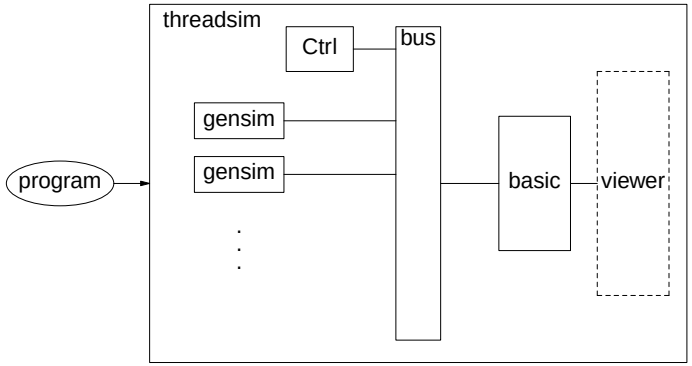
Een thread gaat verder bij de volgende instructie.
Andere thread gaat verder bij de k-de instructie vanaf huidige locatie.

Typisch gebruik:

```
fork #2;
##L1;
##L2;
```

Bij L1 en L2 staat de code voor de threads.

Thread simulator



→ basis-instructie kan als ondeelbare actie worden beschouwd

Communicatie

Communicatie in Sequentieel Model

Communicatie tussen functies:

- parameters
- teruggeef-waarde
- gedeelde bronnen
  - geheugen (globale variabelen)
  - ...

In een sequentieel model is deling van bronnen geen probleem

Communicatie in Concurrent Model

Communicatie tussen functies:

- gedeelde bronnen
- berichten uitwisseling

In een concurrent model is deling van bronnen een probleem

Communicatie Vormen

- direct
  - een functie stuurt een bericht
  - andere functie ontvangt het bericht
  - in abstractie, sturen en ontvangen van bericht kan gezien worden als ondeelbare actie
- indirect
  - een functie zet een bericht in een bepaalde lokatie
  - andere functie haalt het bericht op uit de lokatie
  - geen ondeelbare actie, wegens interferentie van delen van de communicatie
- een-richtings
- twee-richtingen
  - bestaat uit twee een-richtings communicatie

Problemen

Indirect Communicatie

- als schrijf- en lees-acties deelbaar zijn, dan onderlinge interferentie
- een nieuw bericht kan geschreven worden voordat het oude gelezen is
- als lokatie toegankelijk is voor andere functies, dan interferentie met deze functies mogelijk
- ...

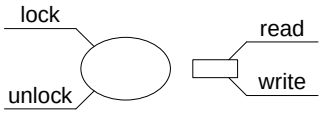
Twee-richtings Communicatie

- direct
  - beide functies willen tegelijk een bericht sturen of ontvangen (deadlock)
- indirect
  - indien zelfde lokatie voor opslag - berichten kunnen overschreven worden

Oplossingen

- Zorvuldig programmeren
  - lastig
  - niet standaard
  - slecht onderhoudbaar
- Mechanisme voor exclusieve toegang
  - standaard (abstractie)
  - robuust

Exclusieve Toegang



- lock - geeft exclusieve toegang
- unlock - geeft toegang vrij

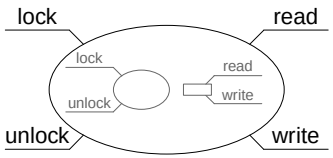
Wordt algemeen gebruikt in software.

Typische implementatie volgens Dekker's algoritme met gebruik van 2 semaforen
[E.W. Dijkstra, *Cooperating Sequential Processes*, 1965]

Probleem

- vereist zorgvuldig programmeren
- ingewikkelder voor toegang van meer dan twee
- toegang ook mogelijk zonder gebruik van mechanisme

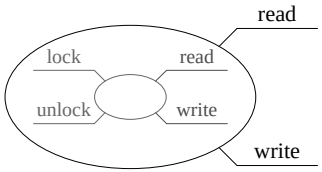
Exclusieve Toegang (cont'd)



- inkapseling van mechanisme met lokatie
- alleen combinatie van acties
  - lock - write - unlock
  - lock - read - unlock
  - maar ook lock - [read | write]* - unlock

Basis Afsluitings Mechanisme

Eenrichtings Indirecte Communicatie



Gebruiker

- lock - read - unlock ⇒ read
- lock - write - unlock ⇒ write

Mechanisme zorgt voor lock-en unlock-en

Basis Communicatie Mechanisme

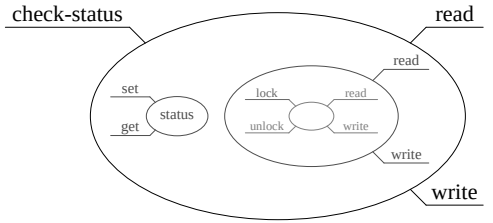
Eenrichtings Indirecte Communicatie (cont'd)

Abstractie

- mechanisme is een (concurrente) functie
- eenrichtings directe communicatie

⇒ eenrichtings indirecte communicatie bestaat uit (2x) eenrichtings directe communicatie

Eenrichtings Indirecte Communicatie (cont'd)



Bij eenrichtings indirect communicatie

- alleen read na een write
- alleen write na read vorige bericht

Mechanisme dat dit afdwingt

- status-vlag
- mogelijkheid om status-vlag te checken

Toestand Gebaseerde Communicatie

Tweerichtings Indirecte Communicatie

- 2 x eenrichtings communicatie
- gebruik zelfde lokatie
  - — extra probleem door overschrijving bericht
  - — oplossing dmv meer states

Speciale Gevallen

Laatste Bericht

- eenrichtings
  - — toestand gebaseerde communicatie (igv check voor write)
  - — basis comunicatie mechanisme
- tweerichtings
  - — toestand gebaseerde communicatie (check voor bericht)

Geen Richting

- geen communicatie met elkaar, maar met lokatie
  dus shared memory
  - — basis communicatie mechanisme
- nieuw bericht op basis van laatste bericht
  lock - read - generate - write - unlock
  - — basis afsluitings mechanisme

Conclusies

Er is alleen directe communicatie met concurrente functie.

Bij sequentiele uitvoering kan van alles weg gelaten worden ter optimalisatie.

Bij concurrente uitvoering, ipv sequentiele uitvoering + threads, alles in concurrente functies.