Eerste Deeltoets Concurrency

2 oktober 2012, 11.00–13.00, Educ-Gamma.

Motiveer je antwoorden *kort*! Zet je mobiel uit. Stel geen vragen over deze toets; als je een vraag niet duidelijk vindt, schrijf dan op hoe je de vraag interpreteert en beantwoord de vraag zoals je hem begrijpt.
Cijfer: Vragen 1 t/4 zijn elk 2pt en vragen 5 t/m 7 elk 4pt, samen 20, totaal delen door 2.

1. **Beantwoord met Amdahls Regel:** Een programma bestaat voor 80% uit parallelliseerbare code en wordt gedraaid op een quad-core.

   (a) Welke speedup wordt gehaald?
   (b) Om de executie verder te versnellen kunnen we kiezen tussen (1) het programma verbeteren zodat 90% parallelliseerbaar is; (2) meer cores bijschakelen. Hoeveel cores heb je bij optie (2) nodig om dezelfde speedup te halen als bij optie (1)?

   Oplossing: (a) Volgens Amdahl wordt de executietijd (bij α parallelliseerbaarheid en p cores): $(1-\alpha)+\frac{\alpha}{p}$, dus hier: $0,20+\frac{0,80}{4}=0,4$. De speedup is dan een factor $1/0,4=2,5$.
   (b) Met $\alpha=0,90$ wordt de executietijd op quad-core: $0,1+\frac{0,9}{4}=0,325$. Om dat te bereiken met $\alpha=0,8$ moet gelden $0,2+\frac{0,8}{p}=0,325$, wat geeft $p=\frac{0,8}{0,125}=6,4$.

   Beoordeling: Voor elk deelantwoord een punt. Beoordelingscodes:

   A = Verkeerde vraag beAntwoord (bv: hoeveel cores heb je nodig om de speedup uit (a) te halen als $\alpha=0,9$, met antwoord 3). Geen punt.
   G = Vervang 6,4 door Geheel getal 6 of 7, prima, ook 1pt.
   F = Geen Amdahl gebruikt, onduidelijk wat berekend is.
   M = Goede Methode of formule maar verkeerd antwoord, halve punt.
   R = Benoem speedup als 0,4 (de Reciproke) ipv 2,5, dit is ook goed en levert 1pt.
   Z = Door bij (b) de speedup af te ronden op 3 krijg je Zes cores, dit is ook goed, 1pt.

2. **Safety of Liveness:** Zeg van elk van deze uitspraken of ze een *Safety* of een *Liveness* eigenschap beschrijven en waarom (in 1 zin).

   (a) De relschoppers worden opgespoord, berecht en gestraft!
   (b) Asielaanvragen worden binnen zes maanden afgehandeld.
   (c) Als een of meer threads aan de operatie beginnen, zal tenminste een de operatie in eindige tijd kunnen voltooien.
   (d) Als een thread t_2 de ticket-keuze begint nadat thread t_1 de ticket-keuze heeft voltooid, ontvangt t_2 een hoger ticket dan t_1.
   (e) De scheduler voor threads $t1$, $t2$ en $t3$ is fair.

   Oplossing: Een eigenschap is Safety als je een schending kunt constateren in eindige tijd.

   (a) Liveness, want zolang de relschoppers nog niet berecht zijn, kan het altijd later nog gebeuren.
   (b) Safety, want je kunt schending constateren wanneer een aanvraag *niet* binnen zes maanden is behandeld.
   (c) Liveness (vanwege de onbepaaldheid van "eindige tijd"): zolang alle threads nog bezig zijn, kan er altijd nog eentje afronden.
   (d) Safety (als je de uitspraak opvat als een bewering over de *waarde* van t_2's ticket en niet het *verkrijgen ervan*): te schenden in eindige tijd, namelijk als t_2 toch een kleiner ticket krijgt.
   (e) Liveness, want hoe lang een thread ook stil staat, hij kan altijd later nog weer een instructie gaan uitvoeren.

   Beoordeling: Een halve punt per goed antwoord vanaf het tweede; dus 1 goed is 0pt, 2 goed is 1/2pt, etc. Beoordelingscodes:

D = Safety bij (a) omdat je schending constateert bij Dood van relschopper; origineel en slim, reken goed.
L = Safety bij (c) omdat je schending constateert bij Livelock; Fout omdat livelock niet de enige vorm van schending van de eigenschap is.
M = Geen Motivatie, -1/2.

3. **Schoppen:** Klaverjassen wordt gespeeld met 32 kaarten, waarvan 8 van type Schoppen. Je begint met een "hand" van acht kaarten. Wat is de kans dat je begint met precies zes Schoppen-kaarten?

   Oplossing: Gebruik de formule $Pr(E) = \frac{\#E}{\#S}$. De mogelijke handen zijn alle combinaties van 8-uit-32, dat zijn er $\left(\begin{array}{c} 32 \\ 8 \end{array} \right) = \frac{32!}{8!.24!} = 10518300$. Een hand met zes Schoppen krijg je met een greep van 6-uit-8 Schoppen (kan op $\left(\begin{array}{c} 8 \\ 6 \end{array} \right) = 28$ manieren) aangevuld met een greep van 2-uit-24 non-Schoppen (kan op $\left(\begin{array}{c} 24 \\ 2 \end{array} \right) = 276$ manieren). De beschreven gebeurtenis bestaat dus uit $\left(\begin{array}{c} 8 \\ 6 \end{array} \right) \cdot \left(\begin{array}{c} 24 \\ 2 \end{array} \right) = 7728$ elementaire uitkomsten. De kans is $\frac{7728}{10518300} = 0,000735$.

   Beoordeling: Twee punten. Beoordelingscodes:
   A = Vergeet de C(24,2) manieren om de 6 schoppen Aan te vullen tot acht kaarten, antwoord is dan 2,66e-6; 1pt.
   A+ = Neemt som ipv product van twee binomiaalcoefficienten in teller; 1pt.
   U = Geen Uitleg en antwoord fout; 0pt.
   V = Berekent de kans op *een* van de volgorden, antwoord 2,62e-5; 1pt.

4. **Drie waarden gooien:** Je gooit herhaald met een dobbelsteen (zes-kantig) en gaat hiermee door totdat je *drie verschillende* uitkomsten hebt gezien. Wat is het verwachte aantal keren dat je moet gooien?

   Oplossing: Als bij de Coupon Collector kun je uitrekenen hoe lang je op elke "nieuwe" waarde moet wachten. Noem x_i het verwachte aantal worpen voor de i-de waarde. Je eerste waarde zie je zeker bij je eerste worp dus $x_1 = 1$. Bij het wachten op je tweede waarde zijn vijf van de zes uitkomsten gunstig, je succeskans is dus $\frac{5}{6}$, dus je verwachte aantal worpen (Bernoulli) $\frac{6}{5}$. Voor je derde waarde moet je verwacht $\frac{6}{4}$ keer gooien. Vanwege lineariteit van verwachting mag je de verwachtingen optellen. Dus verwacht aantal worpen is $x_1+x_2+x_3 = 1+\frac{6}{5}+\frac{6}{4} = 3,7$.

   Beoordeling: Twee punten. Beoordelingscodes:
   B = Herkent er een opvolging van Bernoulli trails in met verschillende kansen; 1pt.
   G = Denkt dat de verwachting de waarde is met Grootste kans (hier 3); 0pt.
   K = Tel Kansen op ipv verwachtingen (1 + 5/6 + 4/6); 0pt.
   U = Geen Uitleg, fout antwoord; 0pt.
   V = Lost Verkeerd probleem op (bv: kans op drie waarden na drie trekkingen); 0pt.

5. **LockTwo:** Hier staan de lock en unlock van de LockTwo klasse (voor thread i).

```
public void lock()
{ victim = i;
  while (victim == i) {}  }

public void unlock()
 { }
```

   (a) Aan welke drie eisen moet een lock implementatie voldoen?
   (b) Welke van deze eisen is/zijn voor LockTwo niet voldaan? Waarom?
   (c) Als je de opdracht `victim = i` verplaats naar de `unlock`, is dan het probleem opgelost?

   Oplossing: (a) Een lock moet voldoen aan (1) **Mutual Exclusion**: hoogstens 1 thread heeft

het lock (2) **No Deadlock**: Als threads het lock willen krijgen, en het is vrij, moet eentje het krijgen (3) **No Starvation**: elke thread die het lock vraagt, zal het ooit krijgen.
(b) LockTwo dwingt af dat de twee threads het lock om de beurt hebben. Het voldoet niet aan **No deadlock**: als een thread het lock wil en de andere is er niet in geinteresseerd, blijft de thread wachten, terwijl het lock vrij is. Omdat het strikt om de beurt gaat, is er wel **No starvation** mits beide threads steeds weer het lock vragen.
(c) Nee, omdat threads willekeurig traag of snel kunnen werken, maakt het niet uit of je de eerste opdracht van lock verplaats naar het eind van unlock.

Beoordeling: Voor a en c elk 1pt, voor b 2pt. Beoordelingscodes:

D = Zegt bij (c) dat 't nu misgaat bij Drie of meer threads. Dat is echter bij de gegeven LockTwo ook al, dus niet goed; 0pt.
I = Zegt bij (c) dat er iets mis gaat omdat `victim` niet is geïnitialiseerd. Reken ik niet goed omdat bij de wijziging natuurlijk ook de initialisatie aangepast kan worden.
M = Bij (a) worden de namen genoemd maar geen oMschrijving van de eisen; 1/2pt.
S = Zegt NoStarvation bij (b). Omdat NoStarvation *sterker* is dan NoDeadlock, is schending van NoStarvation zwakker dan schending van NoDeadlock, daarom is dit een minder sterk antwoord; 1,5pt.
W = Bij (b) wordt wel de geschonden eis genoemd, maar niet Wanneer die geschonden wordt (scenario); 1pt.

6. **Assignments:** In een situatie waar `x` is 0 en `v` is true, worden deze twee threads opgestart:

```
Thread 1:        Thread 2:
  x = 1;           while (v) {};
  v = false;       y = x;
```

Is het zeker dat de waarde van `y` na afloop 1 is wanneer
(a) read/write atomicity geldt? Waarom?
(b) dit wordt uitgevoerd in Java of C#? Waarom?
(c) `v` als `volatile` wordt gedeclareerd?

Oplossing: (a) Ja. De waarde van `x` is al 1 voordat de toekenning aan `v` begint. Thread 2 ziet false in `v` voordat hij met lezen van `x` begint. Dus thread 2 begint met lezen nadat het schrijven is voltooid en leest zeker 1.
(b) Nee. Het is door de interactie via `v` wel zeker dat thread 2 pas `x` gaat lezen nadat thread 1 met het schrijven ervan klaar is. Maar het is niet zeker dat die wijziging ook al zichtbaar is voor thread 2.
(c) Ja. Declareren als `volatile` beperkt niet alleen herordening van operaties op `v` zelf, maar ook herordening van `v` ten opzichte van `x`. De volatile write op `v` komt gegarandeerd pas in het geheugen *na elke* operatie die ervoor zit[1]. Het werken met `volatile` is overigens zo ingewikkeld dat het doorgaans wordt afgeraden[2].

Beoordeling: Voor a 1pt, b 2pt, c 1pt. Beoordelingscodes:

A = Zegt ten onrechte dat Read/Write in C# Atomair is (bij b); 0pt.
M = Zegt bij (c) ja zonder motivatie; hoogstens 1/2, en helemaal 0 als bij (b) ook ja werd geantwoord.
V = Geeft bij (c) het sceario dat de read op `x` van thread 2 nog de oude waarde geeft nadat `v` false gezien was. Slim, maar de Volatile op `x` voorkomt dit; 1/2pt.

7. **Multivalued register:** Je kunt een m-waardig *regular* register maken uit een array van m regular bits. De Writer schrijft waarde x door bit x op 1 te zetten en de lagere bits op 0, de

[1] http://msdn.microsoft.com/en-us/library/aa645755%28v=vs.71%29.aspx
[2] http://stackoverflow.com/questions/6526432/volatile-with-release-acquire-semantics

reader zoekt vanaf positie 0 naar de eerste 1:

```
Write(x):                    Read:
  r[x].write(1)                for(i=0; i<m; i++)
  for (i=x-1; i>=0; i--)         if (r[i].read == 1)
    r[i].write(0)                  return i
```

(a) Laat zien dat de Reader kan falen wanneer de Writer ook de hogere bits op 0 zet (met een loopje `for (i=x+1; i<m; i++) r[i].write(0)`).
(b) Is het gebouwde register ook atomic? Leg uit!

Oplossing: (a) Stel dat de Writer eerst Write(4) doet en dan Write(2). Na de Write(4) zijn alle bits voor positie 4 op 0, neem aan dat de Reader begint en de 0len leest tot positie 3. De Writer voert zijn complete Write(2) uit, daarna staan alle bits voorbij positie 2 op 0. De Reader gaat verder en leest alleen maar 0len.
(b) Het is niet atomic, een inversie is mogelijk, omdat de gebruikte bits zelf alleen maar regular zijn. Scenario hiervoor: stel de waarde is 2 maar de Writer doet een Write(3). De Writer begint met r[3] op 1 te zetten, en gaat dan r[2] op 0 zetten met `r[2].write(0)`. Tijdens deze write kan de Reader een volledige Read afwerken, waarbij de read op r[2], die met de r[2].write(0) overlapt, de nieuwe waarde al geeft zodat de Read resultaat 3 geeft. Daarna kan de Reader weer een volledige Read afwerken, waarbij de r[2].read, die weer met de r[2].write(0) overlapt, de oude waarde nog teruggeeft zodat deze Read 2 oplevert.
We *vermoeden* dat de constructie *wel* een atomair register zou opleveren als de gebruikte bits atomair zijn. Als dat klopt, kun je een scenario alleen maar geven door niet-atomair gedrag van bits erin te bouwen.

Beoordeling: Voor a en b elk 2pt. Beoordelingscodes:
A = Motiveert een nee bij (b) met een Atomair scenario; 0pt. *Bijvoorbeeld:* een "inversie" bij twee overlappende reads, dit kan niet want als twee Reads overlappen kan nooit sprake zijn van inversie.
B = Beredeneert bij (b) dat het atomair is, maar vanuit Atomaire bits. Slim, maar de gegeven bits zijn slechts regular; 1pt.
D = Noemt alleen Definitie van atomic maar bewijst er niets over, 0pt.
L = Beweert dat een Latere writewaarde kan worden geretourneerd. Kan niet, zou ook niet regular zijn; 0pt.
M = Beredeneert bij (b) incorrectheid met een MultiWriter scenario, terwijl deze constructie alleen bedoeld is voor SingleWriter; 0pt.
R = Beredeneert Regular ipv atomair, 0pt.
S = Zegt bij (b) nee en noemt inversie, maar zonder Scenario te geven; 1pt.
T = Beweert bij (b) dat het niet atomair is omdat Timestamps nodig zijn; 1pt.
W = Voert bij (a) een scenario op waarin het fout gaat met twee Writers; zie M, 1pt.