Ätande filosofer

Mattias Cederlund

09-11-2012

1 Uppgiften

Fem filosofer sitter runt ett runt middagsbord med en skål nudlar framför sig och en ätpinne mellan varje filosof. Filosoferna sitter för det mesta och drömmer men till slut blir de hungriga och bestämmer sig för att äta. Då tar de båda pinnarna som ligger närmast, äter ett tag och lämnar sedan tillbaka pinnarna. Det som är trickigt är när flera filosofer får för sig att de ska äta samtidigt vilket kan resultera i en dead-lock där alla filosofer sitter med en pinne vardera och väntar på att den andra pinnen ska bli ledig. Uppgiften är att modellera filosofernas beteende och se vad som gick fel. En lösning ska hittas där vi undviker dead-locks och därmed ser till att alla filosofer får något att äta.

2 Ansatts

Första deluppgiften gick ut på att skapa en process som modellerar en ätpinne, chopstick. Den ska ha olika tillstånd beroende på om den är ledig, eller om någon filosof har tagit den för att äta. När processen startas är ätpinnen i ett läge där den är tillgänglig. Då kan filosoferna plocka upp den genom att skicka ett meddelande innehållande atomen request och sitt processid. Ätpinnens process väntar på meddelanden i en receive-sats och om ett sådant meddelande kommer skickar den tillbaka ett meddelande granted till filosofen och lägger sig i ett läge där ätpinnen inte är tillgänglig i en ny receive-sats. När filosofen har ätit klart skickar den ett meddelande, returned och ätpinnen lägger sig i ett tillstånd då den åter igen är tillgänglig.

I andra deluppgiften ska filosofernas process utformas. Filosofen startar i ett läge där den drömmer tills den bestämmer sig för att äta. Tiden filosofen drömmer utgörs av en sleep i 0,5-1 sekund. Därefter begär den att få pinnarna som ligger till höger och vänster genom att skicka ett meddelande till ätpinnarnas processer. Den skickar en tuple innehållande atomen request och self() - filosofens processid. Därefter sätts filosofprocessen i ett tillstånd där den väntar på att pinnarna ska svara med ett meddelande vardera innehållande atomen granted. Är så fallet att filosofprocessen mottager två

sådana meddelanden har den tillgång till båda pinnarna och kan börja äta. Filosofen sätts då i ett tillstånd där den äter ett tag, modellerat med en sleep. Därefter skickas atomen returned till båda ätpinnarnas processer och filosofens hunger minskar. Om filosofen ätit nog många gånger och är mätt skickas ett meddelande till controller-processen, annars återgår filosofen till dröm-tillståndet igen. När alla filosofer är mätta kommer controllerprocessen skicka meddelandet quit till alla chopstick-processer och middagen är över, vilket ordnas av given control-process. Controllerprocessens uppgift är att skapa övriga processer och stänga ner chopstick-processerna när filosoferna ätit klart.

I deluppgift fyra experimenterar vi lite med hur filosoferna beter sig vid middagen. Med de ursprungliga processerna kunde filosoferna äta sig mätta, då det vid varje testkörning var en filosof som fick två pinnar i början. Därefter följde självklart att en filosof alltid hade två pinnar i och med att den som nyss lämnat ifrån sig sina pinnar drömde ett tag innan den försökte ta upp pinnarna igen. Detta hände även om tiden filosoferna drömde minskades eller till och med sattes till noll, vilket verkar underligt. Jag antar att det är något om hur processerna ges tillgång till processorn som ligger bakom att det blir så.

För att försäkra sig om att en dead-lock-situation kan uppstå sätter vi en delay mellan att första och andra ätpinnen begärs. Alla filosofer kommer nu begära och få vänster pinne och vänta ett tag innan de begär höger pinne och därmed hamnar i en dead-lock då höger pinne redan är upptagen för alla. Detta avhjälper vi genom att när en lång tid har gått och filosofen endast fått en pinne ge upp, lämna tillbaka pinnen och återgå till att drömma. Detta ser vi till med en timeout i receive-satsen, after (1000+ random:uniform(2000)).

```
after (1000+ random:uniform(2000)) ->
      io:format("~s giving up~n", [Name]),
     if First == Right ->
           Left ! abort;
     true ->
          Right ! abort
     end,
     First ! returned,
     dreaming(Hungry, Right, Left, Name, Ctrl)
```

När tiden har gått och vi kommer in i after-satsen skickar filosofen även ett meddelande till den chopstick som vi inte har lyckats få tag på, innehållande abort. För att göra detta möjligt behövde vi ändra på granted-meddelandet som ätpinnen skickade tillbaka så att det även innehöll information om vilken ätpinne vi fick tillgång till. Vi tar alltså reda på vilken pinne vi har fått, och jämför med högerpinnen. Är det samma ätpinne skickar vi meddelandet till vänsterpinnen, annars till högerpinnen. När chopstick-processen får meddelandet abort går den igenom de meddelanden den har

och ta bort den request som filosofen tidigare skickat. Detta får vi genom att när chopstick-processen mottager ett meddelande med abort anropar en funktion abort() som kollar om det finns några request-meddelanden. Är så fallet anropas abort igen och meddelandet har annullerats. Receive-satsen avslutas med en after 0. Alltså när det inte hittats något meddelande så återgår den till tillståndet gone där ätpinnen inte är tillgänglig.

```
abort() ->
     receive
          {request, _} ->
               abort()
          after 0 ->
               gone()
     end.
```

För att försäkra oss om att granted- och abort-meddelandet inte gått förbi varandra på grund av att meddelanden mellan processer är asynkrona så har vi i början av dream-tillståndet använt oss av en receive-sats. Om ett granted-meddelande kommer in medans filosofen drömmer lämnas ätpinnen direkt tillbaka och efter en slumpmässig tid går man ut receive-satsen och begär pinnarna igen genom att använda en after. After-satsen ser till att funktionen har samma egenskaper som tidigare när sleep användes i början av dröm-tillståndet.

Ett annat sätt för att undvika dead-locks är att använda sig av en waiter-process som kontrollerar hur många filosofer som äter samtidigt. Filosoferna skickar ett meddelande till waitern och frågar om den får äta. Filosofen lägger sig då i en receive-sats och väntar på svaret ja innan den begär sina två pinnar, äter och lämnar sedan tillbaka pinnarna. Sedan meddelar filosofen att den har ätit klart med ett meddelande till waitern som då kommer ge ett godkännande till någon annan filosof att äta, osv. Modellen med waitern är kort och gott en semafor. Antalet filosofer som kan äta samtidigt kontrolleras med en mängd states som waitern har. Max fyra filosofer kan begära ätpinnar samtidigt utan att vi får en dead-lock. Om vi tillåter 4 filosofer att äta så skulle det innebära att det alltid är endast en filosof som kan ha två pinnar och därför är det effektivare om waitern endast tillåter att två filosofer äter samtidigt. Sitter dessa med minst en filosofs avstånd kan de äta samtidigt vilket är omöjligt om man tillåter att fler filosofer begär pinnar samtidigt.

3 Utvärdering

Vid testkörning av lösningen som innebar att filosoferna gav upp när de inte hade fått sin andra ätpinne inom en viss tid är resultatet att alla filosofer lyckas äta sig mätta. Antalet gånger filosoferna ger upp och lämnar tillbaka sin ätpinne är varierande men det brukar landa på 6-8 gånger. Det kan vara

värt att tänka på att vi i början ser till att det kan ske och kommer ske en dead-lock genom att låta alla filosofer begära sin vänsterpinne först och sedan vänta på att begära högerpinnen. I detta tillfälle i början kommer dead-lock då ske och vi får att flera filosofer ger upp och lämnar tillbaka sin pinne efter slumpmässig men lång tid. Det resulterar till att en minst en filosof inte behöver lämna tillbaka sin ätpinne och kan börja äta. Med dessa resultat anser jag att systemet har anpassat sig bra och antalet misslyckade försök att äta är inte för många. Nedan följer utskrift från början när deadlock sker, någon filosof ger upp och någon får då sin andra pinne.

```
22> Confucios received first chopstick
22> Avicenna received first chopstick
22> Plato received first chopstick
22> Kant received first chopstick
22> Descartes received first chopstick
22> Kant giving up
22> Descartes received second chopstick
22> Descartes started eating
22> Plato giving up
```

4 Sammanfattning

Det största och kanske mest frustrerande problemet jag hade var att slumptalsgeneratorn alltid började på ett bestämt seed, vilket gjorde att alla filosofer bestämde sig för att äta samtidigt. Därefter följde i försöken att utveckla mekanismer där de ger upp efter ett tag att alla också gav upp samtidigt, och på nytt bad om ätpinnar samtidigt. Dead-locken kunde på så sätt inte undvikas. För att avhjälpa detta genererar jag i början, när filosofprocessen startas i tillståndet init/5 en seed till slumpfunktionen utav en tidsstämpel, innan tillståndet övergår till dreaming. Därefter flöt allt på bra som det skulle.

Ett annat problem som dök upp var att filosoferna kunde ha utestående requests till ätpinnarna även efter att de hade ätit klart, i och med att de har möjligheten att ge upp. Tilldelades de då pinnar så lämnade de aldrig tillbaka dem, eftersom de ansågs klara. På grund av detta ordnades en funktion vid utlämnandet av pinnar som jag kallade send(To) vilken kallades på när ett request-meddelande hade mottagits där To är filosofens processid. Denna funktion gick igenom inkomna meddelanden och plockade bort multipla requests från den filosof som skulle tilldelas ätpinnen genom att när ett sådant meddelande påträffats anropa send(To) igen. Därefter, med hjälp av en after 0 skickades granted-meddelande till filosofen. Problemet med att filosoferna inte kunde äta klart på grund av att det saknades chopsticks löste sig därefter.