Deadlocks

detektera och undvik

Enkla exempel

- Smal bro med en fil
- En fyrvägskorsning
- Fyra vägkorsningar
- Två lås

```
P: Lock A, Lock B .. Rel. A, Rel. B
Q: Lock B, Lock A .. Rel. B, Rel. A
```

Vad motsvarar Resurser? Vad motsvarar Trådar?

Dining philosophers

- Ett runt bort med obegränsad mängd spagetti.
- Fem tallrikar.
- Fem gafflar, en mellan varje tallrik.
- Fem filosofer sitter runt bordet. De antingen äter eller tänker. När de tänker blir de hungriga, när de ätit behöver de tänka.
- Varje filosof behöver både gaffeln till höger och gaffeln till vänster om sin tallrik.

När en filosof blir hungrig

- Vänta tills höger gaffel är ledig
- Ta upp höger gaffel
- Vänta tills vänster gaffel är ledig
- Ta upp vänster gaffel
- Ät tills du är mätt
- Lägg ned vänster gaffel
- Lägg ned höger gaffel
- Tänk (filosofera) tills du blir hungrig igen

Implementation

```
for ever ()
{
  acquire(right_fork)
  acquire(left_fork)
  eat()
  release(left_fork)
  release(right_fork)
  think()
}
```

Steg 1: Vad händer? (ibland)

- Filosof 1 tar gaffel 5, trådbyte...
- Filosof 2 tar gaffel 1, trådbyte...
- Filosof 3 tar gaffel 2, trådbyte...
- Filosof 4 tar gaffel 3, trådbyte...
- Filosof 5 tar gaffel 4, trådbyte...

Rita resursallokeringsgraf!

Resursallokeringsgrafer

- Trådar ritas som cirklar
- Resurser ritas som en fyrkant med en prick för varje resurs av den sorten.
- En resurs som är reserverad av en tråd ritas med en pil från pricken till trådens cirkel.
- En tråd som väntar på en resurs ritas med en pil från trådens cirkel till resursens fyrkant.
- En tråd som vi vet kan komma resevera en resurs ritas som en väntande tråd men med streckad pil.

Steg 2: Vad händer? (ibland)

- Filosof 1 väntar på gaffel 1, trådbyte...
- Filosof 2 väntar på gaffel 2, trådbyte...
- Filosof 3 väntar på gaffel 3, trådbyte...
- Filosof 4 väntar på gaffel 4, trådbyte...
- Filosof 5 väntar på gaffel 5, trådbyte...

Rita resursallokeringsgraf!
Varför ritas inte en låda med 5 prickar för gafflarna?
Deadlock! Cirkulärt beroende i grafen!

En resursallokeringsgraf

- Det finns tre processer, P1, P2 och P3
- Det finns en R1, två R2, en R3 och tre R4
- P1 håller en R2, väntar på en R1
- P2 håller en R2, R1 och väntar på en R3
- P3 håller en R3
- P3 väntar på en R2, deadlock uppstår

- Antag en extra R2 som är allokerad av P4.
- Är det deadlock?

Hantering av deadlock

- Prevention
  - Förhindra möjligheten att deadlock *kan* uppstå. Deadlock *kan inte* uppstå.
- Avoidance
  - Deadlock *kan* uppstå i teorin, men vi undviker att det uppstår i praktiken.
- Detection
  - Deadlock *kan* uppstå och *kommer att* uppstå. Men vi upptäcker när det uppstår och reder ut situationen.
- Do nothing
  - Operativsystemet gör inte ett smack åt saken. Applikationsprogrammerare är smarta nog att själva klara skivan på bästa sätt. En vanlig lösning! (Lösning??)

Vad är *tillräckligt* för deadlock?

- Circular wait
  - Det finns en kedja av resursallokeringar sådan att:
  - Tråd A har reserverat resurs 1 och väntar på resurs 2.
  - Tråd B har reserverat resurs 2 och väntar på resurs 3.
  - Tråd C har reserverat resurs 3 och väntar på resurs 4.
  - Och så vidare tills...
  - Tråd X har reserverat resurs N och väntar på resurs 1.
- Detta är tillräckligt för deadlock och innebär att de tre nödvändiga villkoren är uppfyllda.

Vad är *nödvändigt* för circular wait?

- Mutual exclusion
  - Det finns resurser som kräver att endast en tråd i taget kan använda dem.
  - Systemet har minst en kritisk sektion som kräver mutual exclusion.
- Hold and wait
  - Det finns trådar som har en resurs reserverad och under tiden behöver (vänta på) en annan resurs.
  - En tråd håller en resurs och väntar på en annan.
- No preemption *of resources*
  - En *resurs* kan *endast* ges tillbaka frivilligt av den tråd som använder den. Ingen kan tvinga en tråd att ge upp en resurs, eller stjäla en resurs från någon annan.

Prevention

Några exempel på metoder:

- Resurser är numrerade och reserveras alltid i ordning med lägst nummer först. (ingen circular wait kan uppstå)
- Trådar får bara hålla en resurs i taget. (ingen hold and wait kan uppstå)
- Högre prioriterade trådar kan ta över resurser från lägre prioriterade (som får backa och allokera om resursen). (preemption of resources)

Avoidance

- En algoritm kontrollerar om en resursallokering kan leda till deadlock. Kan deadlock uppstå stoppas resursallokeringen tills den kan göras på ett säkert sätt.

- Bankers algorithm. Utvecklades ursprungligen för att banker alltid skulle ha tillräckligt med medel att (på sikt) tillfredsställa alla kunders behov.

Detection

- En algoritm kör regelbundet och kontrollerar om nuvarande resursallokeringar och väntande trådar innebär att deadlock finns.
- Variant av Bankers algorithm kan användas.
- Eller hitta cykler i resursallokeringsgraf.

Bankers algorithm

- Räkna alla processer som får finnas i systemet.
- Räkna alla resurser som får finnas i systemet.
- Lista vilka resurser varje process behöver som mest.

1. Håll reda på vilka resurser som är allokerade och av vilken process i en matris.
2. Finns det *någon* process där systemets lediga resurser räcker för att processen skall få alla ytterligare resurser den behöver?
3. Antag att den processen kör klart och därmed frigör alla sina resurser. Börja om på 2.
4. Kunde alla processer köra klart? Systemet är då i ett ***säkert läge*** där deadlock *inte har uppstått*.
5. Annars är systemet i ett läge där deadlock *kanske har* eller *kommer* uppstå.

Bankers algorithm: Max behov

RESOURCES TO CLAIM			
PROCESSES	A	B	C
P	2	0	3
Q	3	2	5
R	1	5	1
TOTAL	4	5	5

Bankers algorithm: Allokeringar

PROCESS	RESOURCES ALLOCATED		
	A	B	C
P	2	0	1
Q	2	2	0
R	0	1	0
FREE	0	2	4

Bankers algorithm: Kvar behov

```
RESOURCES NEEDED      FREE AFTER
P        A   B   C      FINISHED
R ---+---+---+---+
O  P | 0 | 0 | 2 | #1 => 2 2 5
C ---+---+---+---+
E  Q | 1 | 0 | 5 | #2 => 4 4 5
S ---+---+---+---+
S  R | 1 | 4 | 1 | #3 => 4 5 5
E ---+---+---+---+
S FREE 0 | 2 | 4 |
```

Bankers algorithm: Exempel

- Ställ upp och räkna!
  - Är systemet i ett säkert läge?
  - P försöker allokera en resurs av typ A. Skall det tillåtas?
  - R behöver allokera en resurs av typ C. Skall det tillåtas?

Bankers algorithm: nackdelar

- Förutsätter ett givet antal resurser
  - Ta bort en USB-disk och förutsättningen är bruten
- Utgår från att alla resurser en tråd behöver är kända på förhand
  - Kan bero på t.ex. indata från användare...
- Komplexitet O(res_count*process_count^2)

Bankers för detektering

- Istället för informationen om en process återstående resursbehov används information om vilka resurser processen begär just nu.

- I övrigt samma beräkningar. Går det inte ut finns ett deadlock.

Rita resursallokeringsgraf

Vi vet inte vad som hänt tidigare men får veta att följande är det enda som händer från nu:

- Många okända händelser…
- P1 reserverar resurs A (ok)
- P2 försöker reservera resurs A (väntar)
- P1 reserverar resurs B (ok)
- P1 försöker reservera resurs C (väntar)
  - Vilka villkor för deadlock är uppfyllda?
  - Kan deadlock ha uppstått?
  - Kan resurs B ingå i ett eventuellt deadlock?

Lösning

- Rita resursallokeringsgrafen.
- Om P2 sedan tidigare reserverat resurs C så gör givna händelserna att deadlock uppstår förutsatt att endast en resurs finns av A och C.
- B kan inte ingå i deadlock. Eftersom B var ledig så kan ingen ha väntat på den tidigare. Alltså uppfylls inte hold and wait för resurs B och den kan då inte ingå i circular wait.