Parallell programmering

- Bakgrund
- Grundläggande begrepp
- Semaforer
- Monitorer
- Meddelanden
- Java threads

Varför parallellitet?

- Speciella problem i operativsystem
- Lösa problem med inneboende parallellitet
  - Simulering av fysiska system
- Flexiblare program
- Utnyttja datorarkitekturer med flera processorer

2

Nivåer

- Instruktionsnivå
- Satsnivå
- Underprogram
- Program

3

Grundläggande begrepp

- Process
- Synkronisering
  - samarbetande
  - Tävlande
- Scheduler

4

Problem med parallellitet

- Icke-determinism
- Hastighetsberoende
- Deadlock
- Svältning

5

Oberoende processer

- Om C ; K är ekvivalent till C // K
- Inträffar om det inte spelar någon roll om alla komponenter i C exekveras före eller efter alla komponenter i K
- Tillräckligt villkor är att varken C eller K uppdaterar en variabel som den andra inspekterar eller uppdaterar

6

Tävlande processer

- C och K vill var och en ha ensam tillgång till en resurs r
- Exekveras då antingen som C ; K eller som K ; C
- Vilken ordningen blir kan vanligtvis inte förutsägas

7

Kommunicerande processer

- Om C_2 i C måste föregå K_2 i K, C_2 producerar något som K_2 konsumerar säger man att det är kommunikation från C till K
- Kedja av processer - pipeline
- I UNIX: C_1 | C_2 | ...

8

Semaforer

- En enkel mekanism för att stödja synkronisering mellan processer
- En datastruktur som består av ett heltal och en kö där uppgiftsbeskrivningar lagras
- "Vakter" begränsar tillgången till delad resurs och sköter accessen
- Två operationer: *wait* och *release*

9

"Wait"

```
Wait(aSemaphore)
if aSemaphore's räknare > 0 then
  minska aSemaphore's räknare
else
  Lägg anroparen i aSemaphore's kö
  Försök överföra kontrollen till
   en process som är redo
end
```

10

"Release"

```
Release(aSemaphore)
If aSemaphore's kö är tom {ingen
  process väntar}then
  Öka aSemaphore's räknare
Else
  Lägg anropande process i "redo"-
   kön
  Överför kontrollen till en
   process i aSemaphore's kö
end
```

11

Semaforer - problem

- Statisk kontroll av korrekt användning ej möjlig
- Deadlock
- Programmeraren ansvarar för att de används korrekt

12

Monitorer

- Per Brinch Hansen beskrev begreppet 1973
- Tony Hoare myntade termen monitor 1974
- Inkapsling
- Synkroniseringen (tävlings-) sköts från run-timesystemet
- Concurrent Pascal var först

13

Concurrent Pascal

```
type monitor_namn = monitor (formella
  parametrar)
---deklarationer av delade variabler---
---definitioner av lokala procedurer---
---definitioner av exporterade procedurer---
---initieringskod---
end
```

14

Monitorer - problem

- För att hantera samarbetande synkronisering har vi samma problem som för semaforer
- Ej naturligt för distribuerade system

15

Meddelanden

- Brinch Hansen & Hoare 1978
- Monitorer bra metod för tävlingssynkronisering med delat minne, för att hantera distribuerade system passar inte monitorer
- Tillåter en "task" att specificera till andra "tasks" när den kan ta emot meddelanden
- När en överföring kan ske kallas det *rendezvous*, inträffar när både sändare och mottagare är redo

16

The History of the Development of Parallel Computing

1955-1993

Http://ei.cs.vt.edu/~history/Parallel.html

17