Inledande concurrent programmering

med introduktionen till async/await

Annorlunda tänkesätt

Skolbarnexemplet:

Hur hittar man det längsta barnet?

Concurrent programmering

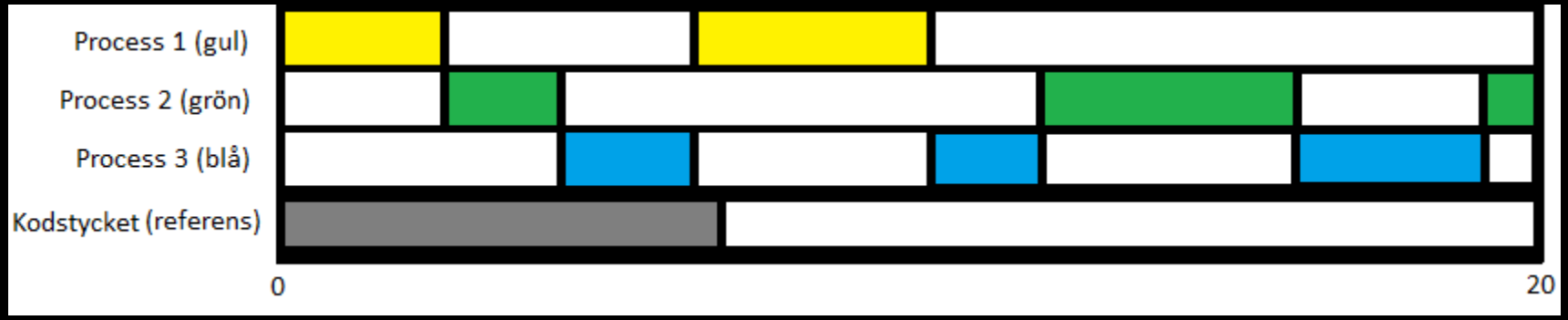
Betyder ‘samtidig’: flera processer (instanser av program - kan vara flera trådar) exekveras under samma tidsperiod

Görs med tidsfördelning mellan processer (paus och fortsättning), prioritet, mm (kan även vara parallell men det behövs inte)

Parallellprogrammering?

För att det ska vara parallellprogrammering ‘måste’ flera processer exekveras samtidigt (oftast med olika kärnor)

Annars kunde man säga att den ofta görs som en implementation av en mer abstrakt concurrent modell


Process 1 (gul)
Process 2 (grön)
Process 3 (blå)
Kodstycket (referens)
0
20


Process 1 (gul)
Process 2 (grön)
Process 3 (blå)
Kodstycket (referens)
0
20

Samtidiga (concurrent) modeller

Varför?

Flerprocess program kan bli väldigt komplicerad om man inte har en bra uttänkt struktur på hur det måste fungera.

Man kan få deadlock (lås), starvation (svält) och oväntat resultat.

Samtidiga (concurrent) begrepp

state (tillstånd) - värden vid ett specifikt tidspunkt

interleaving (vävning) - processer/tråder väver in och ut ur koden (med paus och fortsättning)

Samtidiga begrepp

atomic statments (odelbara uttryck) - bara en process få åtkomst åt gången och med samma förutsättningar ger uttrycket samma svar

liveness - eventuellt kommer ett tillstånd att vara sant (t.ex while(stop != true))

Samtidiga begrepp

critical section (kritiskt sektion) - en koddel som måste exekveras atomt (kallas även mutual exclusion på engelska)

deadlock (dödläge) - ingen process kan komma in (eller ut ur) koddelen (det fryser upp)

Samtidiga begrepp

starvation (svält) - en process exekveras aldrig (andra exekveras och ger aldrig svältprocessen möjligheten)

Finns flera koncept, men vi har nämnt de viktiga för att förstå ‘samtidiga’ problem och lösningar.

Att skicka meddelande

- En metod för att implementera concurrent programmering är att skicka ut meddelande

- För er som var med i Agil projektutveckling, är Scratch en variant av det

Meddelande - begrepp

channels (kanaler) - kommunikationskanaler som man kan lyssna på och/eller skicka över

broadcast - att man skickar till alla kanaler samtidigt

Synkron vs asynkron

synkron - både sidor är synkroniserad med varandra, så en ger och den andra tar emot direkt

asynkron - där finns en fördröjning - en sida väntar för den andra att utföra någonting innan viss kod exekveras

Synkron vs asynkron

timeout - något en asynkron process borde gör om det tar för länge

Hund exemplet!

semaphore

En semaphore är ett sätt att lösa asynkrona problem - som en person som säger till när klar

wait(S) - vänta för att få tillgång resursen

signal(S) - signalera att den är färdigt och överge resursen

C# monitor

En Monitor är mer som en dörr som bara öppnar när den är ledig - det går att vänta på eller signalera ett tillstånd

I c# använder man till och med Enter och Exit

C# monitor

```
bool entered = false;

Monitor.Enter(x, ref entered);
try { x++; }
finally { if(entered) Monitor.Exit(x) }
```

C# lock

(s.405, 474)

lock är betydligt enklare

lock(x) { x++; }

Exempel med Dekker’s algoritm

Dekker’s algoritm är en algoritm för att lösa en ‘critical section’.

Den finns i psuedokod i olika exempel, men här är en c# version.

https://github.com/robinos/DekkersAlgorithm

C# async/await

Del 5 - Kap 15, från s.461

En STOR framgång. Varför? Ganska enkel syntax för något så komplicerad och tillåter andra beräkningar och processer under tiden. (Det behöver inte alltid vara ett hårt lås)

C# async

async - nyckelordet som definerar en metod som asynkron

ie. async void ThisMethod() { }

async Task<string> ThisOtherMethod() {}

c# await

await - nyckelordet för att vänta på ett svar från en async metod

ie. await AsyncConnectToServer();
string text = await AsyncGetText();

c# Task och Task<T>

Task - en speciell klass (som en ‘wrapper’ med extra funktionalitet) som används ihop med asynkronmetoder.

ie.

```
Task<int> lengthTask = GetLength(sak);
Console.WriteLine(lengthTask.Result);
```

c# Task och Task<T>

```
Task<string> fetchTextTask = client.
GetStringAsync(url);
int length = (await fetchTextTask).Length;


static async Task<int> GetPageLengthAsync
(string url) {...}
```

Te affär exemplet med async/await

En exempel av att skicka meddelande med async/await för att simulera en teaffär.

https://github.com/robinos/TeaShop

Extra material

En bra sida för vidare läsning:

http://www.albahari.com/threading/part2.aspx#_Mutex