
1

a) Se läroboken sid 36-37

b)

Systemets program kan delas i mindre fristående körbara program 'Processer' som kan exekveras 'samtidigt'. Processerna kommer på detta vis att bli små och enklare att skriva. Genom den samtidiga exekveringen kan man erhålla bättre realtidsegenskaper.

c) Se pseudoparallell exekvering , läroboken sid 40

d) Se läroboken sid 70

Uppgift 2

Process	**Flag[0]**	**turn**	**Flag[1]**
P0	0	0	0
P0	1	1	0
Processbyte			
P1	1	1	1
	1	0	1

P1 Testar villkor i while(flag[0] && turn==0); sant vilket ger återupprepad test ej i kritisk reg P1
Processbyte till P0
P0
P0 Testar villkor while(flag[1] && turn==1); ej sant , går in i kritisk reg.
P0
Processbyte till P1
P1 Testar villkor i while(flag[0] && turn==0); sant vilket ger återupprepad test ej i kritisk reg P1
Processbyte till P0

P0	0	0	1

P0
Processbyte till P1
P1 Testar vilkor , nu ej sant in i kritisk region.
Processbyte till P0

P0	0	1	1

P0 testar P0 Testar villkor while(flag[1] && turn==1); sant dvs återupprepad test.
P0
Processbyte till P1
P1 Kör färdigt kritisk region….

Fungerar som den ska.

2b)

Metoden växer i komplexitet med antal processer
Processerna förbrukar stor CPU-tid i vänteslingan.
(Kan med viss processbytestakt ge svält för någon process)

2c)

Se läroboken sid 44 – 46

Uppgift 3

3a) Se läroboken sid 55

3b) Ex 1 Ömsesidig uteslutning:

```
Process P1{
        ….
        ….
        Waitsem(S1);
        ….
        Kritisk region.
        ….
        Signalsem(S1);
        …
}
```

```
Process P2{
        ….
        ….
        Waitsem(S1);
        ….
        Kritisk region
        ….
        Signalsem(S1);
        …
}
```

Ex 2 Synkronisering av tvåprocesser:

Antag att semaforen S1 är initierad som ledig medan S2 är upptagen.
I exemplet nedan kommer de kritiska regionerna att exekveras växelvis.

```
Process P1{
        ….
        ….
        Waitsem(S1);
        ….
        Kritisk region.
        ….
        Signalsem(S2);
        …
}
```

```
Process P2{
        ….
        ….
        Waitsem(S2);
        ….
        Kritisk region.
        ….
        Signalsem(S1);
        …
}
```

3c)

P1 anropar signalsem(S1);
If(_signal()): anropet kontrollerar om det finns annan process i WaitQ. I detta fall finns P2.
Genomför If-delen → P2 först i ReadyQ, P1 sist i ReadyQ, returnera 0 dvs If villkoret ej sant. →
Kör suspend(Running) dvs P1:s mjuka omgivning spares på stack och stackpekare i PCB. Därefter kommer dispatch() att starta första processen i ReadyQ (= P2).

Således P1 avbryts → sist i ReadyQ, P2 startar att exekveras.

Uppgift 4

a) Läroboken sid 40
b) Läroboken sid 125
c) Läroboken sid 140
d) Läroboken sid 110/115

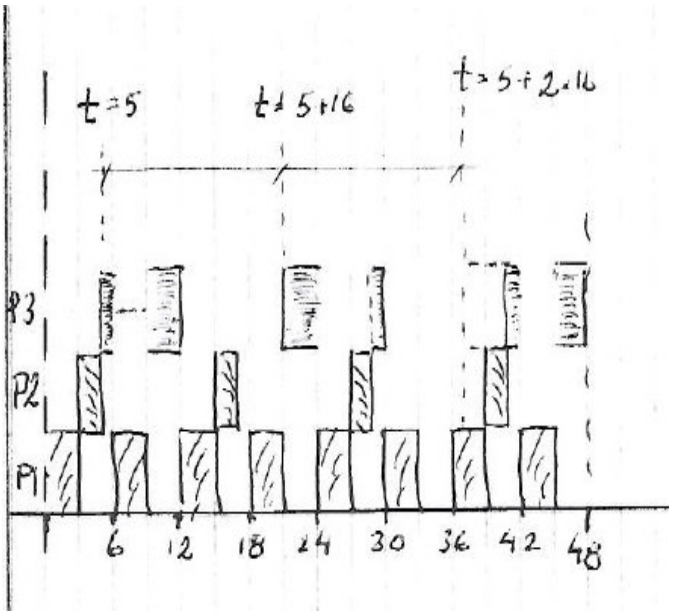
Uppgift 5

a)

LCM(6,12,16) = LCM(LCM(6,12),16)=LCM(12,16)=48 , (4x12=3x16=48)

b)

Ett statiskt schema kan erhållas enligt vidstående figur.



c)

I schemat kan man utläsa att svarstiden (räknat från den tid då processen vill starta dv st=0 :
R1 = 3 , R2 = 5 samt R3=12

d)

Beräkning enligt RMSA exakt analys ger följande för P3:

$$R_3^1 = 4 + \left\lceil \frac{4}{12} \right\rceil \bullet 2 + \left\lceil \frac{4}{6} \right\rceil \bullet 3 = 9$$

$$R_3^2 = 4 + \left\lceil \frac{9}{12} \right\rceil \bullet 2 + \left\lceil \frac{9}{6} \right\rceil \bullet 3 = 12$$

$$R_3^3 = 4 + \left\lceil \frac{12}{12} \right\rceil \bullet 2 + \left\lceil \frac{12}{6} \right\rceil \bullet 3 = 12$$

Konvergering dvs $R_3 = 12$.

Beräkning enligt samma princip ger $R_2 = 5$ samt $R_1 = 3$

Svarstiderna blir identiska med de man kan utläsa av det statiska schemats första iterationer av processerna.

e) Ej schemaläggningsbart ty R_3 > deadline för P3.

Uppgift 6

Beräkning av svarstid för P3 enligt metoden för godtycklig deadline ger följande:
q=0: dvs en start av processen:

$$w^1(0) = 20 + \left\lceil \frac{20}{100} \right\rceil \bullet 30 + \left\lceil \frac{20}{120} \right\rceil \bullet 40 = 90$$

$$w^1(0) = 20 + \left\lceil \frac{90}{100} \right\rceil \bullet 30 + \left\lceil \frac{90}{120} \right\rceil \bullet 40 = 90$$

Konvergens dvs $w_3(0)=90$, Test ger :

$w_3(0)=90 > (q+1)*p_3 = 80$ dvs fortsatt undersökning av ytterligare en start.

$$w^1(1) = 40 + \left\lceil \frac{40}{100} \right\rceil \bullet 30 + \left\lceil \frac{40}{120} \right\rceil \bullet 40 = 110$$

$$w^1(1) = 40 + \left\lceil \frac{110}{100} \right\rceil \bullet 30 + \left\lceil \frac{110}{120} \right\rceil \bullet 40 = 140$$

$$w^1(1) = 40 + \left\lceil \frac{140}{100} \right\rceil \bullet 30 + \left\lceil \frac{140}{120} \right\rceil \bullet 40 = 180$$

$$w^1(1) = 40 + \left\lceil \frac{180}{100} \right\rceil \bullet 30 + \left\lceil \frac{180}{120} \right\rceil \bullet 40 = 180$$

Konvergens kontroll om ok.

$w_3(1)=180 < (q+1)*p_3 = 160$ dvs ej ok

Ytterligare en start ger

$w_3(2)=200 < (q+1)*p_3 = 240$ dvs ok , Stoppa här , Sök max svarstid

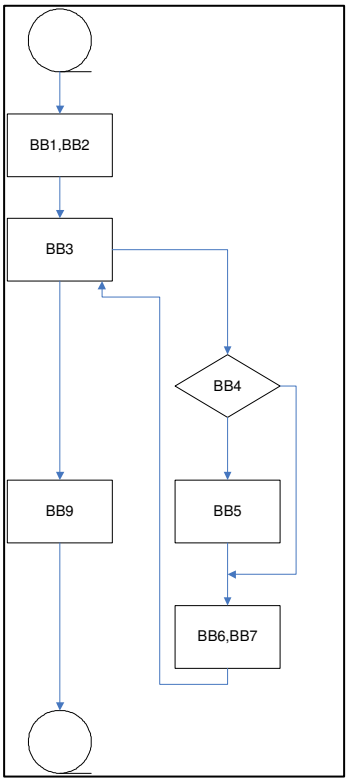
$R_3 = \max (w_3(q) – q*p_3) = 100$ ms enligt nedan sammanställning.

$R_3(0) = 90 – 0 +1 = 90$ ms
$R_3(1) = 180 – 80 = 100$ ms
$R_3(2) = 200 – 160 = 40$ ms

Svar $R_{3max} = 100$ ms

Uppgift 7

a) Se figur.
Angående programflödesgrafen: för alla BBx skall det vara BBx+1 samt att det i ruta ett ska stå BB1,BB2,BB3



b)

Min: Erhålles om ingen data finns indata[].
Min= min[BB1+BB2+BB3 BB4+BB10] = 60

Max: Erhålles om det finns MAX antal tal i indata[] och om alla talen är lika med sökt tal vilket gör att loopen och if-strukturen genomföres MAX antal gånger.

Max = max [BB1+BB2+BB3+MAX*(BB4-8) +BB4+BB4] = 3065

Uppgift 8

```
#include    <stdio.h>
#include    " can_drivers_me_05.h "  // Funktionsprototyper till befintliga funktionerna.

char  InData[9];
char  UtData[9]

void   main(){
       int  n=0;
       strcpy(InData,"000000");
       strcpy(UtData,"1223334");
       CANInit();
       CANSetupRec(0xC6,1);
       while(1){
              Delay(100);
              if(CANRec(1,InData)){
                     Buffer[n]=InData[3];
                     n++
                     if( n==20) n=0;
              }
              CANSend(0xD6,2,UtData);
       }
}
```

Uppgift 9

a)

Grundprincipen är att noderna sänder ut en bit i taget på den gemensamma bussen.Varje bittid för en nod är uppdelad i en skrivfas och en läsfas där noden kollar tillståndet på bussen. Bitvärdena påverkar bussen olika. 0:a är dominant genom att bussen, som normalt är hög via pulupmotstånd, ej påverkas av en utlagd etta medan en utlagd nolla alltid ger bussvärdet noll (dominant) .

Principen är att noderna lägger ut en bit av arbitreringskoden och läser sedan av bussen och jämför med den utlagda biten. Om tillstånd lika med egen utlagd bit så har ingen annan lagt ut någon bit alternativt lagt ut en likadan bit.
Om lika så fortsätter noden att sända nästa bit om olika så har noden sänt en etta och läst en nolla som någon annan nod måste ha lagt ut. Då slutar noden som noterat skillnad att sända och fortsätter därefter att läsa resterande meddelandet på bussen. Den nod som slutat sända försöker sända igen när det andra meddelandet är färdigsänt.

b)

Sändningen inleds med att arbitreringskoden sänds under vilken noderna skriver, läser busstatus för att avgöra om fortsatt sändning ok.
I exemplet nedan sänder noderna bitarna 0 .. 0…1….och sedan 0 för Nod A och C men ett för Nod B varför Nod B slutar sända på grund av att noden sänder bit 1 men kommer att läsa en nolla. Nod A och C Fortsätter tills läget då Non A sänder nolla och Nod C en etta. Efter detta är det enbart Nod A som sänder. Således vinner nod A arbitreringen av bussen.

```
257   =   0 0 1 0 0 1 0 1 0 1 1 1      Nod A
360   =   0 0 1 1 0 1 1 0 0 0 0 0      Nod B
25F   =   0 0 1 0 0 1 0 1 1 1 1 1      Nod C
                  ↑       ↑
```

c)

För ett datameddelande (Dataframe) kan bara en nod tillåtas sända med en viss arbitreringskod. Om de är identiska kommer ingen att vinna busstillträdet och det skulle vara omöjligt för noderna att sända eventuell data.

För en meddelande av typen Remote frame kan två noder sända med samma arbitreringskod. Detta på grund av att meddelandet ej har en datadel samt att det är en begäran till annan nod att sända. Denna begäran kan komma från flera noder samtidigt.