1

a)

En kritisk region i ett program är en följd av instruktioner som behandlar data eller resurser som andra processer också kan behandla. För att säkerställa att inte flera processer samtidigt påverkar den gemensamma datan eller resursen ser man till att endast en process åt gången kan vara inne i den kritiska regionen. Processbyte till process som ej hanterar den gemensamma resursen kan förekomma.

Odelbar region i ett program är en följd av instruktioner som av olika anledningar ej får avbrytas utan måste exekveras i en obruten följd. Exempelvis rutiner för processbyte.

Ömsesidig uteslutning av kritiska regioner innebär att man garanterar att systemet ej är inne i mer än en kritisk region åt gången. Systemet får göra processbyte inom det kritiska området men då endast exekvera programdelar som ej är kritiska I förhållande till den aktuella regionen.

b) Se lärobok sid 110 respektive 115.

c) Se läroboken sid 55

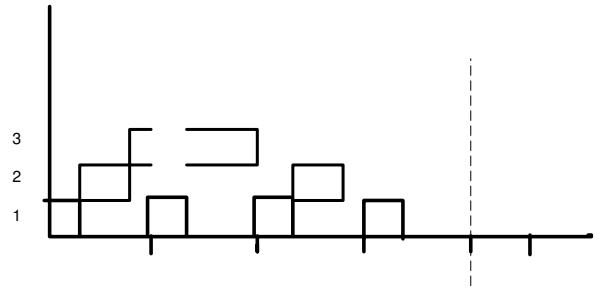
d) Se läroboken sid 120

2 Se beskrivning Dekkers algoritm nr 3

3

Justera periodtiden till värden enligt tabellen nedan:

	P	c
P1	6	2
P2	12	3
P3	24	5



LCM{6,12,24}=24

U=2/6+3/12+5/24=0,79 : Not: Justering till P=5,10,20 ger U>0,85 vilket ej var tillåtet.
Ett statiskt schema kan erhållas enligt ovan figur.

Uppgift 4

a) Antingen beräknar man svarstiderna enligt RMSA exakt analys för processerna alla processerna och jämför svaret med respektive process deadline eller så ser man att P2 inte kan få en svarstid mindre än eller lika med 5 eftersom den egna exekveringstiden är 3 ms och den högre prioriterade processen P1 har exekveringstiden 3 ms.

Kan även visas genom att direkt räkna ut P2:s svarstid:
Beräkning enligt RMSA exakt analys ger följande för P2:

$$R_2^1 = 3 + \left\lceil \frac{3}{15} \right\rceil \bullet 3 = 6$$

$$R_2^1 = 3 + \left\lceil \frac{6}{15} \right\rceil \bullet 3 = 6$$

Konvergens ger R2 = 6 ms vilket är större än deadline för P2 dvs systemet är ej schemaläggningsbart.

b)
Man vet att den optimala prioritetsordningen är enligt deadline monotonic (DM) dvs prioritering efter deadline. I vårat fall låter vi således P2 och P1 byta prioritet och erhåller då följande prioritetsordning:

Process	**p**	**c**	**d**	**S1**	**b**
P2	20 ms	3 ms	5 ms		-
P1	15 ms	3 ms	7 ms	1 ms	2ms
P3	10 ms	4 ms	10 ms		2ms
P4	20 ms	3 ms	20 ms	2 ms	-

Nu behöver vi visa att alla processers svarstider är mindre än respektive daeline:

P4:

$$R_4^1 = 3 + \left\lceil \frac{3}{10} \right\rceil \bullet 4 + \left\lceil \frac{3}{15} \right\rceil \bullet 3 + \left\lceil \frac{3}{20} \right\rceil \bullet 3 = 13$$

$$R_4^1 = 3 + \left\lceil \frac{13}{10} \right\rceil \bullet 4 + \left\lceil \frac{13}{15} \right\rceil \bullet 3 + \left\lceil \frac{13}{20} \right\rceil \bullet 3 = 17$$

$$R_4^1 = 3 + \left\lceil \frac{17}{10} \right\rceil \bullet 4 + \left\lceil \frac{17}{15} \right\rceil \bullet 3 + \left\lceil \frac{17}{20} \right\rceil \bullet 3 = 20$$

$$R_4^1 = 3 + \left\lceil \frac{20}{10} \right\rceil \bullet 4 + \left\lceil \frac{20}{15} \right\rceil \bullet 3 + \left\lceil \frac{20}{20} \right\rceil \bullet 3 = 20$$

Konvergering dvs $R_4 = 20$ som är >= deadline för P4.

Beräkning enligt samma princip ger $R_3 = 10$, $R_1 = 6$ samt $R_2 = 3$ dvs samtliga mindre än eller lika med motsvarande deadline vilket gör systemet schemaläggningabart.

c)
Införandet av semaforanvändning för P1 och P4 ger blockeringsfaktorer för de olika processerna enligt tabellen ovan. P1 och P3 kommer att påverkas så att svarstiderna förlängs med minst motsvarande respektive blockeringsfaktor varför vi utan beräkningar kan konstatera att både P3 får en svarstid som överstiger deadline vilket gör att systemet ej är schemaläggningsbart.

Uppgift 5

Vi har följande processdata:

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

$$w^1(0) = 1{,}5 + \left\lceil \frac{1{,}5}{10} \right\rceil \bullet 3 + \left\lceil \frac{1{,}5+1{,}5}{10} \right\rceil \bullet 3 = 7{,}5$$

$$w^1(0) = 1{,}5 + \left\lceil \frac{7{,}5}{10} \right\rceil \bullet 3 + \left\lceil \frac{7{,}5+1{,}5}{10} \right\rceil \bullet 3 = 7{,}5$$ w(0) > (q+1)*p_3 varför vi testar med ytterligare en start av samma process dvs q=1.

$$w^1(1) = 3 + \left\lceil \frac{3}{10} \right\rceil \bullet 3 + \left\lceil \frac{3+1{,}5}{10} \right\rceil \bullet 3 = 9$$

$$w^2(1) = 3 + \left\lceil \frac{9}{10} \right\rceil \bullet 3 + \left\lceil \frac{9+1{,}5}{10} \right\rceil \bullet 3 = 12$$

$$w^3(1) = 3 + \left\lceil \frac{12}{10} \right\rceil \bullet 3 + \left\lceil \frac{12+1{,}5}{10} \right\rceil \bullet 3 = 15$$

$$w^4(1) = 3 + \left\lceil \frac{15}{10} \right\rceil \bullet 3 + \left\lceil \frac{15+1{,}5}{10} \right\rceil \bullet 3 = 15$$

Konvergens dvs $w_3(1)=15$
$w_3(1)=15 > (q+1)*p_3 = 10$ dvs fortsatt undersökning av ytterligare en start dvs q=2.

$$w^1(2) = 4{,}5 + \left\lceil \frac{4{,}5}{10} \right\rceil \bullet 3 + \left\lceil \frac{4{,}5+1{,}5}{10} \right\rceil \bullet 3 = 10{,}5$$

Osv till konvergens för w(2) = 16,5
$w_3(2)=16{,}5 > (q+1)*p_3 = 15$ dvs fortsatt undersökning av ytterligare en start dvs q=3.

$w_3(3)$ kommer på samma sätt som ovan att konvergera för $w_3(3)=18 < (q+1)*p_3 = 20$ **dvs ok**

$R_3 = \max (w_3(q) - q*p_3) = 35$ ms enligt nedan sammanställning.

$R_3(0) = 7{,}5 - 0 = 7{,}5$ ms
$R_3(1) = 15 - 5 = 10$ ms
$R_3(2) = 16{,}5 - 10 = 6{,}5$ ms
$R_3(3) = 18 - 15 = 3$ ms

Svar $R_{3max} = 10$ ms

Uppgift 6

Program nod 1:

```
// --- Mainprog nod 1 ----
void main(){
        int n =0;
        strcpy(InData,"000000");
        strcpy(UtData,"111111");

        CANInit();          /* Init hardware in this node */
        CANSetupRec(0xC6, 2); /* setup msg obj 2 as receiver with arbkod = 0xC6 */
        while(1){
                /* Spin for a message block */
                if(CANRec(2,InData)){
                        InData[8]='\0';
                        puts(InData);
                }
                n++;
                if ( n= =50 ) {
                        n=0;
                        CANSend(0xD6,1,UtData);
                        /* Send UtData via msg obj 1 med arb. kod 0xD6 */
                }
        }
}
- - - - - - - - - - - -- -- - - - -- - -

- - - - - - - - - - - - - - - - - - -
```

Program nod 3:

```
// -- Main nod nr 3
void main(){
        int n =0;
        strcpy(InData,"000000");
        strcpy(UtData,"333333");
        CANInit();                      /* Init hardware in this node */
        CANSetupRec(0xD6, 2);           /* setup msg obj 2 as receiver with arbkod = 0xD6 */
        CANSetupRec(0xA8, 3);           /* setup msg obj 3 as receiver with arbkod = 0xA8 */

        while(1){
                /* Spin for a message block */
                If(CANRec(2,InData){
                        InData[8]='\0';
                        puts(InData);
                }

                if(CANRec(3,InData)){
                        InData[8]=\0';
                        puts(InData);
                }

                n++;
                if ( n= =50 ) {
                        n=0;
                        CANSend(0xC6,1,UtData);
                        /* Send UtData via msg obj 1 med arb. kod 0xC6 */
                }
        }
}
- - - - - - - - - - -
```

b) Om nod 1 och nod 3 börjar sända samtidigt så kommer de båda noderna att börja skicka sina respektive arbitreringskod först bit för bit.
Under det att en enskild bit ligger ute på bussen läser båda noderna av busstillståndet .
Bussen är så konstruerad att en nolla är dominant, dvs om någon lägger ut en nolla på bussen så kommer bussens tillstånd att bli noll oavsett om någon annan nod lägger ut en etta. (Man säger att ettan är recessiv och nollan dominant).
Om busstillstånd är lika med utsänd bit för en nod fortsätter noden att sända. Om ej så avslutar noden sändningen och övergår till att läsa. Efter det att arbitreringsdelen avslutats ska det bara återstå en sändande nod som då slutför sändningen av data-delen mm.
Nod som tvingas avsluta under arbitreringsdelen försöker att sända på nytt efter föregående meddelande slutförts. Arbitreringsmetoden gör att nod med lägsta arb. kod har högst prioritet.
I vårt fall blir det nod 3 som får sända till slut om nod 1 och nod 3 börjar samtidigt. Nod 1 kommer att sänder sitt meddelande efter det att nod 3 slutfört sitt meddelande under förutsättning att inte nod 2 börjar sända.

c) Se beskrivningen i inledningen av b).

Uppgift 7

```
Unsigned char Las_in AD(void){
   unsigned char indata;
        waitsem(S1);
        write_control(0x01);
        while(1){
                if((readstatus & 0x04)= = 0x04 ){
                        indata=readdata;
                        signalsem(S1);
                        return(readdata);
                }else {
                        yield;
                }
        }
}
```

Uppgift 8

Följande olika tillstånd kan förekomma som påverkar utförandet av funktionen waitsem():
1) Ingen process finns i väntekön till aktuell semafor.
2 Det finns minst en process i väntekön till den aktuella semaforen.

Fallet 1: Ingen process i väntekön varför WaitQ[semaphoreID-1].cnt är > 0 och funktionen __wait() räknar ned WaitQ[semaphoreID-1].cnt med ett och returnerar 1 till den anropande funktionen waitsem() som då direkt ger kommandot return dvs återhopp till anropande process som fortsätter med raden efter anropet waitsem(SemID).

Fallet 2:
Minst en process i väntekön,
If() –vilkoret i funktionen __wait() är inte uppfyllt varför funktionens nedre del utföres vilket innebär följande:

- stausvariabeln i anropande process PCB skrivs till PROC_WAITING.
- Anropande process placeras sist i kön till aktuell semafor. (insert_last(Running…….)
- 0 returneras till funktionen waitsem(). Som utför de sista instruktionerna suspend(Running) som lagar aktuell process register på stack och stackpekare till PCB varefter funktionen dispatch() startar nästa process i RaedyQ.