Uppgift 1.

a) En process är ett körbart program med tillhörande minnesutrymme, stack samt ett sk Processorkontrollblock.

b) Processorkontrollblocket är en skyddad del av processens minnesdel och innehåller viktiga variabler för att kunna stoppa samt återstarta processen samt hantera processen vid samtidig hantering av flera processer. (Exempelvis köhantering) . Viktiga data i PCB är :

Pekare till nästa process i eventuell kö
Plats för att lagra processens stackpekare.
Processens status (Running, wiating mm)
Processens stack

Uppgift 2.

a Kritisk region: En del av ett programmet i vilken man hanterar med andra processer delade resurser (data mm). För att ett system skall fungera måste man garantera ömsesidig uteslutning mellan kritiska områden. En kritisk region får avbrytas av RTOS och andra processer kan exekveras dock ej de processer som delar de kritiska områdena.

Odelbar region: En programdel som absolut ej får avbrytas utan måste exekveras i en oavbruten följd, exempelvis ett processbyte.

b). En semafor är en heltalsvariabel > = 0 som hanteras av ett realtidsoperativsystem (RTOS) . Värdet på variabeln kan endast påverkas genom anrop av operationerna waitsem(int SemId) och signalsem(int SemId).
waitsem(S) : Räknar ner semaforen S om S>0, i annat fall kommer anropande process att suspenderas.
signalsem(S) : Om S= 0 och process väntar på semaforen så startas denna väntande processen annars räknas S upp.

c) Semaforer kan användas för att garantera ömsesidig uteslutning av mellan processer delade kritiska områden eller resurser samt till att synkronisera processer.

Ex ömsesidig uteslutning en eller flera processer med struktur enligt nedan.

```
Process Px(){
 DO_FOREVER{
        ….
        wait(S1);
        // Kritisk region
        …
        signal(S1);
        …
}
```

Ex synkronisering:

```
Process P1(){
DO_FOREVER{
        …
        wait(S1){
        ….
        ….
        signal(S2);
        …
        ..
}
```

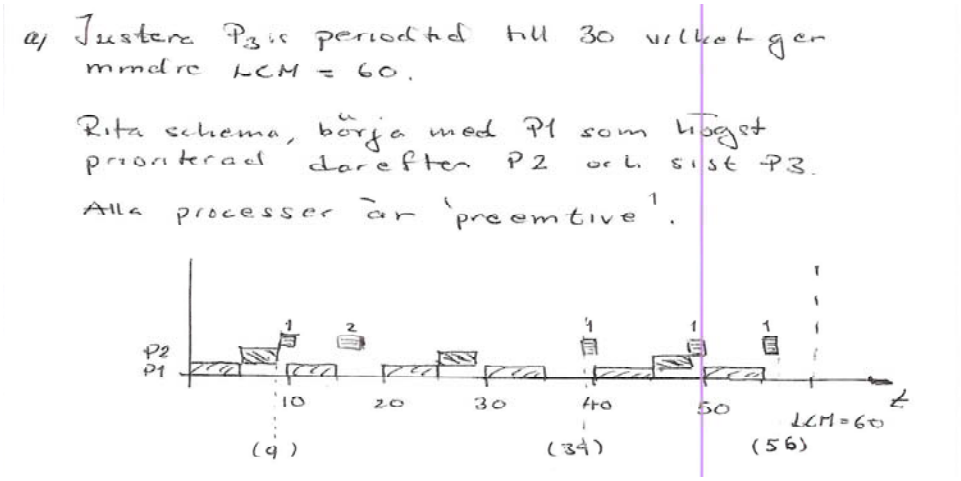
```
Process P1(){
DO_FOREVER{
        …
        wait(S2){
        ….
        ….
        signal(S1);
        …
        ..
}
```

Området mellan wait och signal i de båda processerna kommer utföras i turordning efter varandra om semaforen S1 är ledig och S2 upptagen vid start.

d) En process kan vara i ett av de tre tillstånden:

Running	Exekveras , till Waiting om försöker ta upptagen semafor till Ready om CPU tiden (Timeslice) är ute och processbyte ska ske
Waiting	Väntar på att få ta en semafor. Om först i Waiting-kön till Ready-kö när semaforen ledig , signal () (Vanligen till Running direkt) .
Ready	Står i Ready-kön och väntar på körtid. Om först i Ready-kön till Running vid processbyte.

Uppgift 3



b)

Om man studerar det statiska schemat enligt deluppgift a) ser man att P1 och P2 har svarstider som motsvarar exekveringstiderna och som är mindre än respektive deadline. P3 har för sina två olika starter en svarstid som är större än den andra och enligt figuren lika med 17 ms som i sin tur är större än det angivna deadline som är 15 ms. Således är systemet ej schemaläggningsbart om man skall uppfylla kraven.

Uppgift 4 a

Proc	Pri	p	d	c	s_i	b
P1	0	6	6	2	2	3
P2	1	13	8	3		3
P3	2	25	15	5	3	-

[ms]

a/ Villkor för enkel RMSA ej uppfyllt då $p \neq d$ ⇒ Exakt RMSA krävs.

Beräkna svarstider, enl. $R_i^{n+1} = c_i + b_i + \sum_{\forall j<i} \left\lceil \frac{R_i^n}{p_j} \right\rceil \cdot c_j$

P3: Ansätt $R_3^0 = 5$

$R_3^2 = 5 + \left\lceil \frac{5}{13} \right\rceil \cdot 3 + \left\lceil \frac{5}{6} \right\rceil \cdot 2 = 10$

$R_3^3 = 5 + \left\lceil \frac{10}{13} \right\rceil \cdot 3 + \left\lceil \frac{10}{6} \right\rceil \cdot 2 = 12$; $R_3^4 = 5 + \left\lceil \frac{12}{13} \right\rceil \cdot 3 + \left\lceil \frac{12}{6} \right\rceil \cdot 2 = 12$

Konvergens ⇒ $R_3 = 12 < d_3 = 15$.

P2: Ansätt $R_2^1 = 3$; $R_2^2 = 3 + \left\lceil \frac{3}{6} \right\rceil \cdot 2 = 5$

$R_2^3 = 3 + \left\lceil \frac{5}{6} \right\rceil \cdot 2 = 5$ Konvergens ⇒ $R_2 = 5 < d_2 = 8$

P1: $R_1 = c_1 = 2$

∴ Schemaläggningsbar för $\forall R_i < d_i$

Uppgift 4 b

Bestämning av blockeringsfaktorer

P1 & P3 delar semaforen S. Under 3 ms kommer P3 att erhålla samma prioritet som P1 (Priority inheritance) P1 & P2 kommer då att blockeras maximalt 3 ms under en periodtid dvs blockeringsfaktorn blir 3 ms för P1 & P2

Svarstiderna behöver räknas om. P3 får samma då $b_3 = 0$.

P2: $R_2^0 = c_2 + b_2 = 6$

$R_2^1 = 6 + \left\lceil \frac{6}{6} \right\rceil \cdot 2 = 8$

$R_2^2 = 6 + \left\lceil \frac{8}{6} \right\rceil \cdot 2 = 10$; $R_2^3 = 6 + \left\lceil \frac{10}{6} \right\rceil \cdot 2 = 10$

Konv. $R_2 = 10 > d_2 = 8$

⇒ Processerna ej schemaläggningsbara enligt RMSA.

Uppgift 5

Följande tabell kan skapas avseende processerna:

Process	Prioritet	Periodtid p	Exekveringstid c	Jitter J
PA	0	10	3	1
PB	1	10	3	0
PC	2	5	1	1

Maximala svarstiden beräknas enligt metoden för fix prioritet samt godtycklig deadline.

$$w_c^{n+1}(q) = c_c(1+q) + \sum_{\forall j} \left\lceil \frac{w_c(q) + J_j}{p_j} \right\rceil c_j$$

EN START $q=0$: Ansätt $w^0(0) = 1 + \left\lceil \frac{1+0}{10} \right\rceil \cdot 3 + \left\lceil \frac{1+1}{10} \right\rceil \cdot 3 =$

$= 7$

$w(0) = 1 + \left\lceil \frac{7+0}{10} \right\rceil \cdot 3 + \left\lceil \frac{7+1}{10} \right\rceil \cdot 3 = 7 \Rightarrow$ KONVERGENS

TESTA OM $w(0) + J_c \leq p_c(q+1) = 5$ SVAR NEJ

NY TEST $q=1$, Ansätt $w^0(1) = (q+1) \cdot 1 = 2$

$w^1(0) = 2 + \left\lceil \frac{2+0}{10} \right\rceil \cdot 3 + \left\lceil \frac{2+1}{10} \right\rceil 3 = 8$

$w^2(1) = 2 + \cdots\cdots = 8$ KONVERGENS

TESTA $w(1) + J_c \leq p_c(q+1) = 10 \Rightarrow$ SVAR JA.

SÖK SVARSTIDER FÖR STARTERNA.

$q=0 \quad R_c(0) = w_c(0) - q \cdot P_c + J_c = 8$ - NOT 1

$q=1 \quad R_c(1) = w_c(1) - q \cdot P_c + J_c = 4$

SVAR $R_{MAX} = 8$

NOT 1: Hantering av Jitter har ej tydligt i lärobok samt inget övningsex. på övning. $\Rightarrow$ Ej poängavdrag för detta SVAR $R_{MAX} = 7$ ok

Uppgift 6

a)

Antag att vi endast sänder tre hexadecimala siffrorna i arbitreringsfältet och som sändes efter den
inledande startbiten:

243 = 0 0 1 0 0 1 0 0 0 0 1 1 Nod A
340 = 0 0 1 1 0 1 0 0 0 0 0 0 Nod B

Starta sänd arbitreringsfältet med MSB – biten. Antag att noderna börjar lägga ut samtidigt på bussen.
Noderna sänder ut en bit i taget på den gemensamma bussen. På bussen som normalt är hög via pullup- motstånd kommer en hög bit ej att påverka bussens tillstånd medan om någon nod lägger ut en nolla kommer bussen att bli noll oavsett vad andra noder lägger ut.

Principen är att noderna lägger ut en bit av arbitreringskoden och läser sedan av bussen innan nästa bit läggs ut. Det avlästa värdet jämförs med den egna 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 en skillnad mellan sänd bit och buss-status att sända och fortsätter därefter att läsa resterande del av meddelandet på bussen. Den nod som slutat sända försöker återigen sända när det andra meddelandet är färdigsänt.
I exemplet ovan sänder noderna bitarna 0 .. 0…1….och sedan 0 för Nod A och ett för Nod B varvid Nod B slutar sända på grund av att noden sänder bit 1 men kommer att läsa en nolla.

b)

För att arbitreringen enlig delfråga a skall fungera måste varje nod hinna läsa annan nods utsända bit innan noden kan lägga ut en ny bit. En spänningsändring fortplantas längs ledaren med en begränsad hastighet vilket gör at t det tar en i förhållande till bit-tiden märkbar tid för spänningsändringen att fortplanta sig ett antal meter i en ledare, dvs mellan två noder. Med en viss bittid i sändningen blir det ett visst maximalt avstånd mellan de två yttersta noderna i ett system för att det säkert skall fungera enligt CAN-protokollet.

Uppgift 7

a)

```
void Las_AD_port( int proc_nr){
        waitsen(S1);
        ADPort_control=0x04;  // Ettställer bit nr 2 (vikt 4 )
        While(!(ADPort_status & 0x02));  // Väntar tills omvandling klar .
        Ad_Box[proc_nr]=ADPort_data;  // lägger omvandlat värde i global buffert
        Signalsem(S1);
}
```

b)

```
void Las_AD_port( int proc_nr){
        waitsen(S1);
        ADPort_control=0x04;  // Ettställer bit nr 2 (vikt 4 )
        While(!(ADPort_status & 0x02)){
                yield();
        }
        Ad_Box[proc_nr]=ADPort_data;  // lägger omvandlat värde i global buffert
        Signalsem(S1);
}
```

Uppgift 8

a)

MAX = 1
Ger en lista bestående av två post av typen REGTYP
Med värden i enligt fig.

0
'next post'
NULL

'nr'
NULL
'prev post'

MAX=5 ger en lista med 6 poster där
post ett ser likadan ut som tidigare. Post 2 får ändrat
innehåll så att next-pekaren pekar på nästa post .
Följande 3 poster har samma struktur på innehållet.
Den sista posten har innehållet motsvarande post två
tidigare dvs nextpekaren = NULL.

b)

```
void add_last(REGTYP* temp, int data){
// Funktion som lägger till ett element sist i den länkade listan samt
// lägger in talet data i elementets fält tal.
  REGTYP *new_post;
  new_post=(REGTYP*) malloc(sizeof(REGTYP));
  // Sök sista posten
  while(temp->next!=NULL){
      temp=temp->next;
  }
  temp->next=new_post;
  new_post->tal=data;
  new_post->prev=temp;
  new_post->next=NULL;

}
```

Uppgift 8 old:

a) och b9 Se läroboken sid 57-58