Uppgift 1.

a)
CR : Carrier Sense : Innan nod börjar sända läser noden bussens tillstånd och avgör om någon annan nod sänder. Om ingen sänder kan noden börja sända. Medans noden sänder läser noden busstillstånd och avgöra om någon annan nod börjat sända, dvs om det uppstått kollision. Om detta inträffat finns olika sätt att hantera detta.

MA : Multiple Access : Alla noder kan börja sända utan annan kontroll än ovan

CR : Collision Resolution : Finns en metod (protokoll) som gör att en eventuell kollision upplöses .

b) Bussarbitrering**:** Metod för att avgöra vilken nod som skall få sända om flera vill börja samtidigt.

Uppgift 2.

a) 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.

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) Ex ömsesidig uteslutning en annan eller flera processer med motsvarande kritisk region enligt nedan princip.

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

Ex på synkronisering enligt vidstående programexempel.

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.

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

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

d)
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 vilket innebär att man endast kan exekvera en av de kritiska regionerna samtidigt. En kritisk region får avbrytas av RTOS och andra processer kan exekveras dock ej de processer som delar de kritiska områdena.

Uppgift 3

t1: P1 exekverar, tar semaforen READ. Ställs i WaitQ för semaforen WRITE_OK.

t2: P2 exekverar, ställs i WaitQ för semaforen READ.

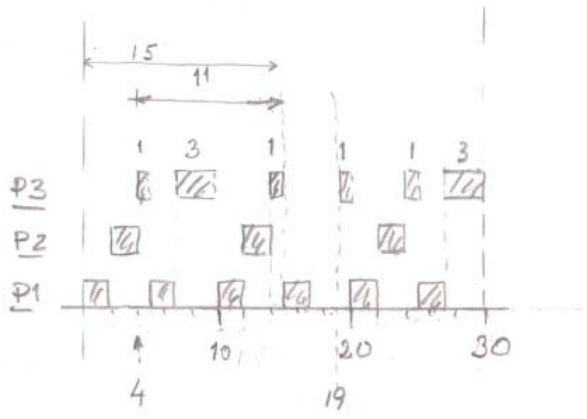
t3: P3 exekverar, tar semaforen WRITE, tilldelar data=3, frigör WRITE_OK ,
Ger att P1 exekverar tar WRITE_OK, tilldelar tal=3, frigör READ_OK, Frigör READ,
Ger att P2 exekverar , ställs i WaitQ för WRITE_OK. (får nu vänta tills nytt värde skrivs av channelWrite.)

Uppgift 4

a) LCM(6,10,16) = 2^4 * 3*5= 240
b) Periodjustera till p (5,10,15) som ger LCM = 30.
c) Se figur

d) Beräkning enligt RMSA ger : R3= 15 , R2=4 samt R1=2.

I det statiska schemat kan man se att om man betraktar t=0 som redo tiden för alla tre processerna så ser man att de beräknade tiderna är tiden fram till dess att processen är klar första gången.



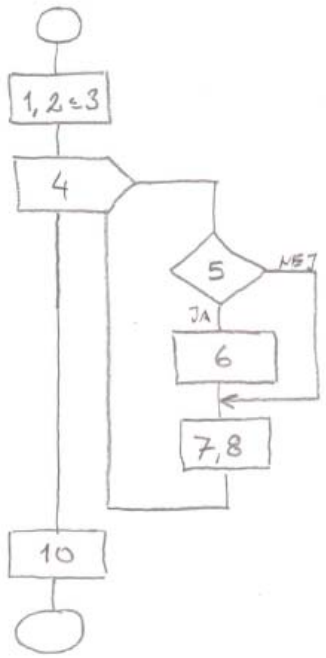
e) Ej enligt dynamisk schemaläggning ty R3 > än d3. Däremot gäller att det statiska schemat är ok då man tolkar starttid som verklig startid i schemat eftersom starterna upprepas periodiskt efter detta. Exempelvis är R3 då 11 enligt det ritade schemat.

Uppgift 5:

Semaforhanteringen ger blockeringsfaktorerna : b1=3 ms samt b2 = 3 ms.

R3 beräknas till 12 ms vilket är mindre än d3=15 ms
R2 beräknas till 10 ms vilket är större än d2=8 ms .

Systemet är således ej schemaläggningsbart enligt RMSA.



Uppgift 6:

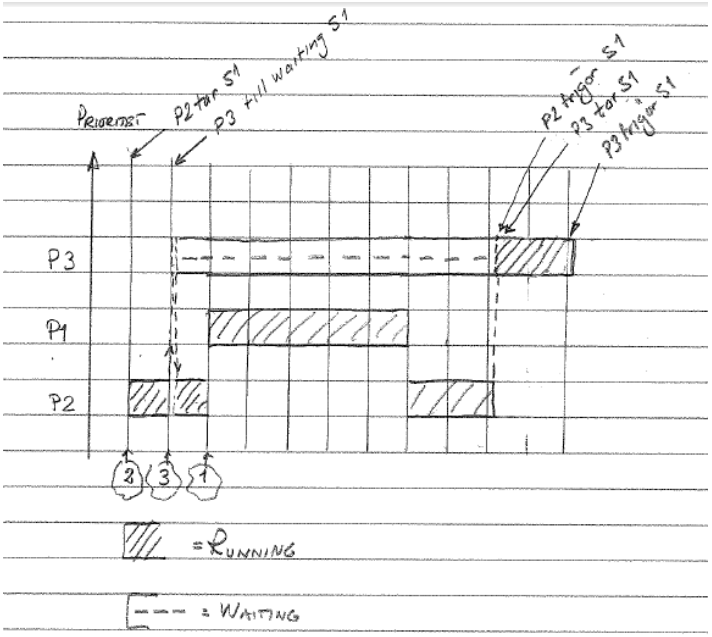
a) Programflödesgraf:

T_{min} ges av programslingan 1,2,3,4 samt 10 .
T_{min} = 55 ms

Tmax ges av att while-loopen samt if satsen utföres för alla tal vilket innebär att tabellen innehåller MAX antal tal av det sökta talet .
T_{max} ges då av slingan 1,2,3, MAX*(4,5,6,7,8) samt 4 och 10
T_{max} = 610 ms

Uppgift 7

a) Händelserna ger följande tidsdiagram med hänsyn tagen till prioritet och semaforhantering.



b) P3, som har högsta prioritet, får i och med att P2 har tagit semaforen S1vänta tills P2 frigör semaforen. P1, som har högre prioritet än P2 kommer att få exekvera sin händelsefunktion. Som figuren visar kommer P3 , trots sin höga prioritet , att få invänta både P1 och P2. P1 och P2 verkar i detta läge som om de hade en högre prioritet än P3. Effekten kallas för Prioritetsinversion.

c) Genom att då P2 tar semaforen låta P2 erhålla den högsta prioriteten hos de processer som hanterar semaforen kommer P2 att få exekvera den kritiska regionen (händelsefunktionen) obrutet (i detta fall) vilket minimerar fördröjningen av P3. Metoden kallas för Prioritetsarv

Uppgift 8
Programmet för NOD 2 kommer att ha samma utseende som NOD 1 :s program så när som på att man måste ändra enligt nedan:

ptrCAN0->CAN_AR_MR_ID[4]=0x0AAAAA**AC**; // ID sänd
ptrCAN4->CAN_AR_MR_ID[0]=0x0AAAAA**BA**; // ID acc reg
Om texten "ID ok skall skrivas ut ändra till:
If(identifier==0x0AAAAABA) puts("ID ok \n\r");

Uppgift 9
Om den sista posten ej innehåller aktiv data (namn) vilket i stort sett alla inlämnade lösningar utgår från och som ger full poäng.

```
REGTYP* find_person(  char *name,  REGTYP *top){
  int not_ok=1;
  while(top!=NULL){
    if(strcmp(name,top->enamn)==0){
       not_ok=0;
       break;
    }
    top=top->next;
  }
  if(not_ok)  return (NULL);
  else return top;
}
```

Om sista posten även har ett aktivt namn justeras sista två raderna till följande:

```
  if(strcmp(name,top->enamn)==0) not_ok=0;
  if(not_ok)  return (NULL);
  else return top;
}
```