Chalmers Campus Lindholmen
Peter Lundin/ Data- och informationsteknik

TENTAMEN

KURSNAMN	
PROGRAM: namn **åk / läsperiod**	**REALTIDSSYSTEM , GK** **DAI2 samt EI3**
KURSBETECKNING	**LET 623 0101 (5p)**
EXAMINATOR	**Peter Lundin**
TID FÖR TENTAMEN	**Tisdagen den 23/10 2007 kl. 14.00 – 18.00**
HJÄLPMEDEL	**Godkänd räknedosa**
ANSV LÄRARE: namn **telnr** **besöker tentamen kl**	**Peter Lundin** **772 5726 / 070 753 7526** **Ca 15.00 samt 16.30**
DATUM FÖR ANSLAG av resultat samt av tid och plats för granskning	**Granskning Torsdag 8/11 12:15 sal J121 (OBS)**
ÖVRIG INFORM.	**Betygsgränser : Max 23 poäng** **3:**10 – 14 **4:** 14,5 – 18,5**, 5:** 19 -

OBS : Lösning anslås på kursens hemsida Onsdagen den 24/10

Chalmers Campus Lindholmen
Peter Lundin/ Data- och informationsteknik

2007-10-17
Tentamen realtidssystem
LET 623 0101(4p)

FORMELSAMLING

RMSA

***FÖRUTSÄTTNINGAR*:**

- Varje process i uppsättningen är periodisk
- Deadline och periodtid är samma
- Konstant exekveringstid för processer
- Ingen processkommunikation förekommer
- Alla processer är avbrytbara

n	$n(2^{1/n}-1)$
1	1,000
2	0,828
3	0,780
4	0,757
5	0,743

En uppsättning processer ($P_1,P_2,...P_n$) är schemaläggningsbar om:

$$\sum_{i=1}^{n}\frac{c_i}{p_i} \le n\,(2^{1/n}-1)$$

där c_i är är exekveringstiden för P_i och p_i är periodtiden för P_i.

Exact analys

Beräkning av svarstiden för process ***i*** under interferens från högre prioriterade processer ***j*** ges av sambandet

$$R_i^{n+1}(q) = c_i + b_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^n}{p_j} \right\rceil c_j$$

Där b är blockeringsfaktorn , interferens från lägre prioriterade processer.

Svarstidsanalys vid processuppsättningar med fix prioritet

För en uppsättning processer gäller att maximal svarstid *R*, inom ett fönster *w*, kan beräknas. För blockeringsfaktorn *b* gäller att prioritetstaksprotokoll (max en blockering) är implementerat.

$$w_i^{n+1}(q) = (q+1)c_i + b_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i^n(q) + J_j}{p_j} \right\rceil c_j$$

Iterationen utförs tills : $w_{i,q} <= (q+1)\,p_i$

Svarstiden R ges av:

$$R_i = \left(\max_{q=0,1,2,\ldots} \left(w_i(q) - qp_i \right) \right) + J_i$$

och där:

- *w* är det ”fönster” inom vilket vi betraktar svarstider
- *q* är den start, numrerad 0,1,2..., inom fönstret vi betraktar
- *c* är processens exekveringstid
- *b* är blockeringsfaktor för processen
- *p* är processens periodtid
- *J* är maximalt jitter för processen

Chalmers Campus Lindholmen
Peter Lundin/ Data- och informationsteknik

2007-10-17
Tentamen realtidssystem
LET 623 0101(4p)

Uppgift 1 (4p)

a)
Beskriv kort vad man i samband med realtidssystem menar med begreppen **kritisk** respektive **odelbar region** samt **ömsesidig uteslutning**. (1,5p)

b) Beskriv skillnaden mellan statisk och dynamisk schemaläggning? (0,5p)

c) Vad menas med en semafor? Hur kan den användas och varför? (1p)

d) Vad innebär prioritetsarv vid användning av semaforer och varför behövar man införa detta vid semaforanvändning? (1p)

Uppgift 2 (2p)

I en realtidskärna utan stöd för ”ömsesidig uteslutning” på maskinnära nivå försöker man att implementera ömsesidig uteslutning mellan två processers kritiska regioner med flaggor enligt nedan. Visa med ett scenario av processbyten eller beskriv på annat sätt att metoden ej kommer att fungera.

```
-------------------------------------------------------------
int i1=FALSE, i2=FALSE;

PROCESS P1()
{
   while(1){
     /* icke-kritisk region */

     i1 = TRUE;
     while(i2==TRUE) ;
    /* kritisk region 1 */
     ……..
      ……
     i1 = FALSE;
    /* slut kritisk region */
  }
}


PROCESS P2()
{
   while(1){
     /* icke-kritisk region */

     i2 = TRUE;
     while(i1==TRUE) ;
    /* kritisk region 1 */
     ……..
      ……
     i2 = FALSE;
    /* slut kritisk region */

  }
}
```

Uppgift 3 (2p)

Vidstående tabell visar data för tre processer i antal ms för ett system. Rita ett möjligt statiskt schema för processerna om man tillåts göra mindre justeringar av periodtiden i syfte att erhålla ett rimligt litet LCM-tal. Periodtiden får endast justeras så att utnyttjandegraden U uppfyller villkoret U < 0,85 efter justering.

Proc	Pri	p (ms)	c(ms)
P1	0	6	2
P2	1	13	3
P3	2	25	5

Uppgift 4 (4p)

Ett realtidssytem där fyra processer med data enligt tabellen nedan exekveras med hjälp av en realtidskärna som stödjer prioritetsarv i samband med semaforhantering.

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

a)
Antag att processerna tilldelas prioritet enligt tabellordning, där P1 har högsta prioritet, och att processerna inte använder sig av semaforer.
Visa enligt metoden för RMSA att systemet inte är schmaläggningsbart vid en dynamisk schemaläggning.

b)
Om man prioriterar processerna enligt någon annan ordning kommer processernas svarstider att bli annorlunda. Genom val av lämplig prioritetsordning kan systemet eventuellt bli schemaläggningsbart. Avgör om det finns någon prioritetsordning som gör systemet ovan schemaläggningsbart samt visa att det är schemaläggningsbart.

c)
Hur påverkas systemets schemaläggningsbarheten om processerna P1 och P4 delar en semafor S1 med exekveringstiderna för den kritiska regionen enligt tabellen. Prioritetsordning enligt svar i deluppgift b).

Uppgift 5 (3p)

I ett system använder man sig av ett antal olika processer med väl avgränsade uppgifter enligt följande:

Process A (PA) :
Läser ett värde från en givare , omvandlar värdet till en intern representation. Läsningen skall ske med en periodicitet om 100 Hz. Exekveringstiden för processen PA är 3 ms.

Process B (PB) :
Utför en beräkning baserad på inläst värde från PA. Periodiciteten för processen kan sättas till 100 Hz. Exekveringstiden för processen PB är 3 ms.

Process C (PC) :
Omvandlar det beräknade värdet från process PB till ett styrvärde som matas ut till ett ställdon. Ställdonet kräver en uppdatering av styrvärde var 5:e ms (200 Hz) varför processens periodtid blir 5 ms. Exekveringstiden för PC är 1,5 ms.

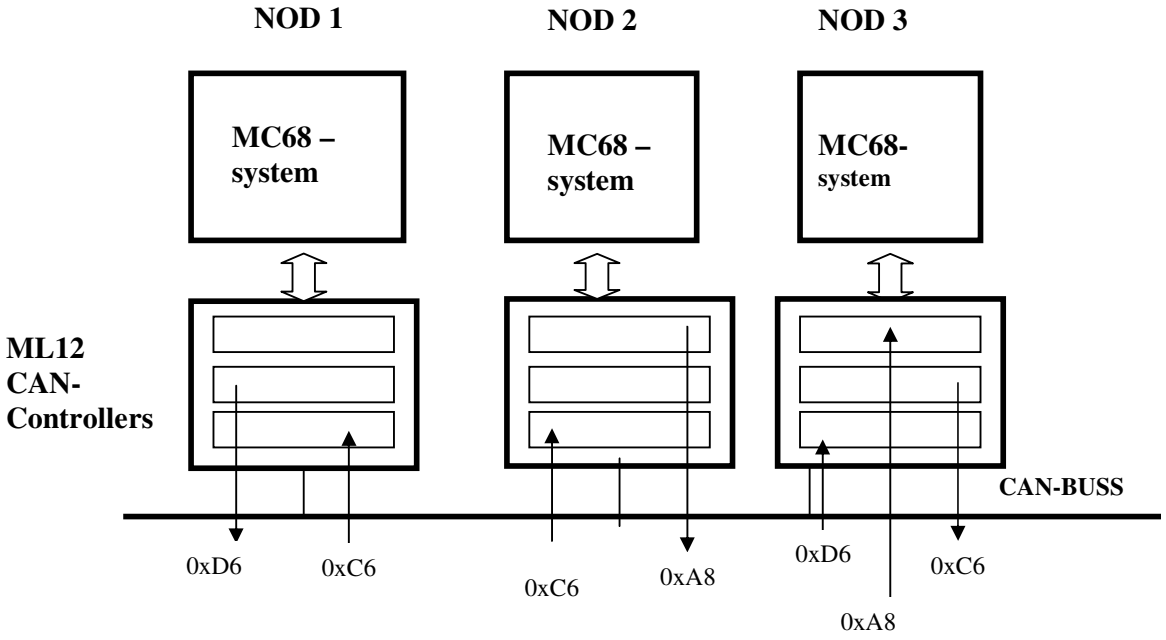
Processen PA har ett starttidsjitter avseende exekveringstiden på motsvarande +/- 0,6 ms.

Antag att vi bestämmer att processerna skall prioriteras i följande ordning : PA, PB samt PC , där PC har lägst prioritet.

Beräkna den maximala svarstiden för **PC** enligt metoden för godtycklig deadline och fix prioritet. Det finns ingen semaforhantering att ta hänsyn till.

Uppgift 6 (4 p)

Antag att vi har ett system bestående av tre datornoder anslutna till varandra via en CAN-buss enligt nedanstående figur.



Till programutvecklingsmiljön finns följande funktioner avsedda för CAN-noderna:

CANInit() :
Initierar noden till viss kommunikationsstandard vilket bland annat innefattar att alla arbitreringsbitar måste vara lika för att ett meddelandet skall läsas in.

CANSetupRec(int *arbitrering*, int *msgobject*) :
Aktiverar nodens meddelandeobjekt nr *msgobjekt* för mottagning av meddelande med arbitreringsinnehåll enligt *arbitrering*.

int CANRec(int *msgobject*, char **buffer*) :
Läser eventuellt inkommit meddelande object . Parametern *msgobjekt* anger aktuellt objekt nr i noden medan pekaren **buffer* överför en pekare till mottagande sträng.
Rutinen returnerar ’1’ om nytt objekt finna att läsa samt ’0’ i annat fall.

CANSend(int *arbitrering*, int *msgobject*, char **data*):
Läser in data (8 tecken) till angivet meddelandeobjekt i noden som sedan sändes med angiven *arbitrering* då bussen ledig.

Programstrukturen för varje enskild nod ges av följande:

Programskal

```
// Programskal NOD nr 1 - 3
#include      <stdio.h>
#include      " can_drivers_me_05.h "  // Funktionsprototyper till befintliga funktionerna.

char   InData[9];
char   UtData[9]

void    main(){

strcpy(InData,"000000");
strcpy(UtData,”111111”);     // ”222222” för NOD 2 samt ”333333” för NOD 3
                .......
                .......
                .......
        While(1){
                ……
                ……

        }
}
```

a) (2p)

Skriv programkod i C för noder N1 och N3 om programmen ska ha följande funktion:

Generellt för alla noder är att de skall ha ett program som återupprepat i en oändlig loop kontrollerar om det kommit något meddelande till just den noden. Eventuell indata läses in till strängen InData som sedan skrivs ut till serieporten med hjälp av funktionen puts(char *) .

Var 50:e loop sänder noden ut meddelandet ’UtData’ till andra noder i de fall noden ska skicka något vilket framgår av nedan beskrivning.

Kommunikationen mellan de tre noderna skall vara följande (se även bilden ovan):

NOD 1 sänder ut Utdata med arbitreringskoden 0xD6 samt tar emot data med arbitreringskoden 0xC6
NOD 2 sänder Utdata med arbitreringskoden 0xA8 samt tar emot data med arbitreringskoden 0xC6
NOD 3 sänder ut Utdata med arbitreringskoden 0xC6 samt tar emot data med arbitreringskoderna 0xA8 respektive 0xD6

b) (1p)
I en CAN-buss kan det inträffa att två noder börjar sända samtidigt. Antag att NOD 1 och 3 i systemet ovan börjar sända samtidigt.

Vad inträffar ?
Kommer båda meddelandena att slutligen bli sända ? (Motivering)
Vilket meddelande i systemet ovan kommer att ha kortaste garanterade överföringstid ?

c) (1p)
Beskriv i grova drag hur CAN-protokollet i grunden fungerar när det gäller arbitreringen till bussen (bussaccessen).

Uppgift 7 (2p)

I ett realtidssystem skall man kontinuerligt med viss periodicitet läsa in en analog ingång via en AD-omvandlare med 8 bitars upplösning och lagra värdet i en buffert *databuf[]*.

AD-omvandlaren har följande funktionen :
- eventuell kontroll att omvandlaren redo för ny inläsning görs genom att kontrollera att bit nr 2 i 'AD_status_reg' är nollställd.
- en inläsning och omvandling startas genom att ettställa bit nr 0 i tillhörande register 'AD_control_reg'.
- efter start av omvandling inväntas att omvandlingen blir klar vilket anges genom att bit nr 2 i registret 'AD_status_reg ' blir ettställd. Biten nollställs vid avläsningen av dataregistret.
- inläst värde avläses från 'AD_data_reg'

Programmet skall utformas så att det är en huvudprocess som med avsedd periodicitet anropar en funktion **unsigned char Las_in AD (void)** som genomför start av omvandling och avläsning. Funktionen returnerar inläst värde som sedan lagras i bufferten databuf[] av anropande huvudprocess. Funktionen skall vara utformad så att den vi ett anrop alltid utför uppgiften och att den ska kunna anropas av flera godtyckliga processer.
För att lösa det senare ska man använda sig av en semafor S1 som ser till att bara en process åt gången kan använda AD-omvandlaren dvs starta en ny omvandling.

Ni skall skriva den aktuella funktionen Las_in AD () som ska göra följande:
Starta en omvandling om omvandlaren ej upptagen av annan process.
Kontrollera om omvandling klar . Om ej överlåt CPU-tiden till annan process genom anrop av operativsystemets funktion yield(). Om omvandling klar returneras inläst värde.

Antag att semaforerna har initierats enligt :
initsem(S1,1) respektive initsem(S2,1) samt att de operationer som får utföras på semaforen är : waitsem() samt signalsem().

Funktionen anropas från aktuell huvudprocess enligt följande:

```
……..
data= Las_in AD ();
Waitsem(S2);
Write_nr++;
Databuf[write_nr]=data;
If(write_nr==100) write_nr=0;
Signalsem(S2);
……
……
```

```
//  ------------------ realtidssystem tenta_okt_07 --------------------

// -------     Systemadresser  ------
#define AD_control_reg      0x0800
#define AD_status_reg       0x0800
#define AD_data_reg         0x0801

#define readstatus     *( ( unsigned char *)  AD_status_reg  )
#define readdata   *( ( unsigned char *)  AD_data_reg  )
#define write_control(x)  *(( unsigned char *)  AD_status_reg  )=x

// --- Semaforbenämningar -----
#define S1   1
#define S2   2

// ---- Globala variabler ----------
uinsigned char databuf[100];
int  write_nr = 0;
int read_nr=0;
```

// ---- Skriv endast denna funktion --------

```
Unsigned char Las_in AD(void){
   unsigned char indata;




}
```

Uppgift 8 (2p)

I en realtidskärna som stödjer semaforer har man definierat processorkontrollblocket (PCB) för processerna samt för semaforerna enligt nedan kod:

```
typedef struct pcb {
struct pcb      *next;
struct pcb      *prev;
char*   stkptr;
int     stat;
int     exitcode;
char    *name;
char    stack[PROC_STACK_SIZE];
}PCB;


typedef struct sem{
        PCB    *proc;
        int     cnt;
}SEM;
```

Semaforerna används via funktionsanropen waitsem() respektive signalsem(). Funktionen waitsem() är implementerad enligt nedan kod.

Förklara genom hänvisning till programkoden hur en process påverkas när den anropar witsem(S) både om semaforen är ledig repektive om semaforen är upptagen. Beskriv även påverkan på andra processer i systemet

```
static int      __wait(int semaphoreID)
{
        disable();      // Förhindrar avbrott
        if (WaitQ[(short)(semaphoreID-1)].cnt > 0) {
                (WaitQ[(short)(semaphoreID-1)].cnt) --;
                Enable(); // Tillåter avbrott.
                return 1;
        }


        Running->stat = PROC_WAITING;
        insert_last( Running, &(WaitQ[(short)(semaphoreID-1)].proc) );
        return 0;
}

void    waitsem(int semaphoreID)
{
        disable();
        if( __wait(semaphoreID) ){
                enable();
                return;
        }
        suspend(Running); /*low level routine does housekeeping... */
        dispatch();
}
```

Not:
- disable() : stänger av avbrottsmöjligheten
- enable() : gör att avbrott tillåts igen.

Lycka till önskar Peter och Sven.