Chalmers Data- och informationsteknik
Peter Lundin

2013-08-23
Tentamen realtidssystem, gk
LET 624 0209 (6 hp)

TENTAMEN

KURSNAMN	
PROGRAM: namn **åk / läsperiod**	**REALTIDSSYSTEM, GK** **DAI2 samt EI3**
KURSBETECKNING	**LET 624**
EXAMINATOR	**Peter Lundin**
TID FÖR TENTAMEN	**Fredagen den 23/8 2013 kl. 08.30 – 12.30**
HJÄLPMEDEL	**Godkänd räknedosa**
ANSV LÄRARE: namn **telnr** **besöker tentamen kl**	**Peter Lundin** **772 5726 / 070 753 7526** **Ca 09.45 samt 11.15**
DATUM FÖR ANSLAG av resultat samt av tid och plats för granskning	**Granskning Torsdagen den 5/9 fm 10:00 - 12.15** **Peter Lundins arbetsrum plan 4 Jupiter**
ÖVRIG INFORM.	**Betygsgränser: Max 24 poäng** **3:**10 – 15**, 4:** 15,5 – 19,5 **, 5:** 20 -

Chalmers Data- och informationsteknik
Peter Lundin

2013-08-23
Tentamen realtidssystem, gk
LET 624 0209 (6 hp)

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.

Exakt analys där d ≤ p

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

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

Där b är blockeringsfaktorn , interferens från lägre prioriterade processer.

Svarstidsanalys vid processuppsättningar med fix prioritet och godtycklig deadline

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:

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

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 Data- och informationsteknik
Peter Lundin

2013-08-23
Tentamen realtidssystem, gk
LET 624 0209 (6 hp)

Uppgift 1 (2p)

I en datorsystem utan stöd för ”ömsesidig uteslutning” har Dekker visat att man kan implementera **ömsesidig uteslutning** mellan **kritiska regioner** i två processer som exekveras **pseudoparallelt**, den sk Dekkers algoritm.

Dekker beskriver sin algoritm genom att först beskriva ett antal enklare algoritmer (Dekker 1 – 4) som visar sig ha begränsningar i form av ***deadlock*** alternativt ***svält***.

a) Beskriv kort vad man menar med begreppen:

Deadlock
Svält
Kritisk respektive odelbar region. Poängtera noga skillnaden mellan de två begreppen.

b)

I en av Dekkers algoritmer används två flaggor i ett försök att skapa ömsesidig uteslutning mellan två kritiska regioner enligt nedan exempel.
Metoden kommer att i ett visst scenario av processbyten ge upphov till ett problem.
Vilket är problemet som kommer att uppstå och beskriv det tänkta scenariot som leder till problemet på lämpligt sätt.

```
int        i1=FALSE,i2=FALSE;
PROCESS P1()
{
 while(1){
  /* Icke-kritisk region */
  ----
  ----
  i1 = TRUE;
  while(i2==TRUE) ; /*vänta*/
  /* kritisk region */
  ----
  ----
  i1 = FALSE;
  /* Icke kritisk region */
  ----
  ----
 }
}
```

```
PROCESS P2()
{
while(1){
  /* Icke-kritisk region */
  ----
  ----
  i2 = TRUE;
  while(i1==TRUE) ; /*vänta*/
  /* kritisk region */
  ----
  ----
  i2 = FALSE;
  /* Icke kritisk region */
  ----
  ----
 }
}
```

Uppgift 2 (3p)

I ett tidsdelat system bestående av flera processer som exekveras ”pseudoparallellt” via tidsdelningen har man oftast ett realtidsoperativsystem som sköter tidsdelning genom processbyten som normalt genereras genom klockavbrott. Varje process har ett eget stackutrymme. För att kunna skapa ömsesidig uteslutning av kritiska regioner har realtidssystemet stöd för semaforer.

a)

En process i ett sådant system kan befinna sig i ett av tre olika tillstånd, vilka är dessa.
Beskriv även vilken händelse som initierar samtliga möjliga byten mellan de olika tillstånden.

b)

För att garantera ömsesidig uteslutning finns olika metoder för process-synkronisering. Man kan bland annat använda sig av ”Semaforer”. Beskriv vad en Semafor är och vilka operationer man kan utföra på Semaforen.

c) Exemplifiera hur man använder en Semafor för att garantera ömsesidig uteslutning samt hur man kan synkronisera händelser/processer.

Uppgift 3 (2p).

Antag att vi har ett systen som läser av en tryckt tangent på systemets tangentbord med en process ReadTang() enligt nedan samt med en process PutDisplay() skriver tecknet på en display. Systemet använder en global variabel *'tang_kod'* för att överföra tecknet mellan processerna.

För att synkronisera processerna med läsning respektive skrivning av läst tecken på displayen används två semaforer. Synkroniseringen skall vara sådan att först läses ett tecken in via ReadTang varefter tecknet skrivs på displayen utan upprepning. Processen ReadTang använder semaforerna enligt programkoden nedan.
Efter det att tecknet skrivits på displayen skall systemet vänta på ett nytt tecken från tangentbordet.

I processen **ReadTang** används funktionen
int tangNere(void)
för att indikera om en tangent är nedtryckt samt
unsigned char readTang(void)
för att läsa av aktuell nedtryckt tangent.

Skriv processen PutDisplay (void) så att den med hjälp av semaforerna säkert skriver varje inmatat tecken på displayen endast en gång.
Skrivning till display görs med befintlig funktion

void printDisplay(unsigned char tecken) ;

Antag att semaforerna har initierats enligt :
initsem(S1,1)
initsem(S2,0)

```
unsigned char tang_kod;

PROCESS ReadTang ( void){
while(1){
        wait(s1);
        while(1){
                if(tangNere()){
                        tang_kod=readTang();
                        while(tangNere());
                        signal(s2);
                        break;
                }
        }
}
```

Uppgift 4 (3p)

I en process finns nedanstående rutin som läser in ett 8-bitars tal från en inport i systemet. Man har för avsikt att teoretiskt uppskatta exekveringstiden för programdelen.
a) Beskriv vad som avgör hur många iterationer while-loopen utför i programmet.
b) Rita en programflödesgraf för raderna 1 – 8
c) Beräkna exekveringstiden [min,max] för programdelen om följande uppskattningar gäller:

BB1= [5 ,5]
BB2= [10,15]
BB3= [5,5]
BB4= [10 ,15]
BB5= [5,10]
BB6= [5,5]
BB7= [5,8]
BB8= [5 ,8]

```
    #define ReadInport *((unsigned char*) 0x9C00)
    int antal, bitpos;
    unsigned char inkod, temp;

1   antal=0;
2   inkod= ReadInport;
3   temp=inkod;
4   while( !(temp & 0x80 ) && (antal < 4 ) ){
5       inkod=inkod<<1;
6       temp = inkod;
7       antal++;
    }
8   bitpos=7-antal;
```

Uppgift 5 (3p)

Antag att vi har en uppsättning av processer med data enligt tabellen nedan.
Processerna skall schemaläggas med ett **statiskt schema**. Processerna får avbrytas.

a) Beräkna LCM-talet för processerna med de givna periodtiderna. Justera därefter periodtiderna så att ett för schemaläggningen lämpligt litet LCM-tal för periodtiderna erhålls. Beräkna därefter LCM-talet efter denna justering.

b) Rita ett statiskt schema för processerna med de justerade periodtiderna.

Process	p	c	d
P1	6	1	4
P2	9	2	6
P3	12	4	12

Uppgift 6 (2p**) ok**

Avgör om följande processer är schemaläggningsbara enligt ”Deadline monotonic”

Process	**p(ms)**	**c(ms)**	**d(ms)**
P1	16	8	16
P2	20	6	8
P3	32	4	30

Uppgift 7 (3 p)

Som bilaga till denna tentates finns en programlista ” Exempel dubbel länkad lista”.
Du skall i denna uppgift skriva till en funktion till det aktuella programmet som uppfyller följande krav:

Funktionsprototyp:

```
int delete_post( Queiterator *quetop,  int s_data);
```

Funktion:

Funktionen ska ta bort den post i listan där variabeln data har värdet s_data. Om post med data=s_data finns och tas bort returnerar funktionen 1 i annat fall returneras 0. Minnesutrymmet för den borttagna posten måste frigöras.

Uppgift 8 (3 p) ok

Tre st standard CAN-noder (A,B och C) är anslutna till en gemensam CAN-buss. Vid exakt samma tidpunkt börjar samtliga tre noder att sända varsit meddelande. Nod A sänder ett meddelande med identifieringskod enligt nedan där NOD B är konfigurerad som mottagare för detta meddelande. NOD B respektive C sänder i sin tur med identifierare enligt nedan till andra noder.CAN protokollet är av typen CSMA/CR vilket innebär att den uppkomna krocken upplöses och en av noderna får sända sitt meddelande.

Nod A : Hexadecimalt 20F
Nod B: Hexadecimalt 2F8
Nod C : Hexadecimalt 18A

Kommer meddelandet från Nod A fram till Nod B ? Om så är fallet hur många omsändningar krävs från Nod A innan meddelandet når Nod B ?
Beskriv detaljerat hur arbitreringsen utföres (avgörandet av vilken nod som får tillgång till bussen vid en kollision) .

Uppgift 9 (3p)

Man skall utveckla en programvara till ett litet realtidssystem baserat på processorn Motorola MC68340. Systemet skall styra en industriprocess och består av två inportar från vilka systemet kan läsa analoga invärden via en A/D-omvandlare. (Port_A1 resp Port_A2) samt en utport till vilken systemet kan mata ut ett digitalt värde till en D/A omvandlare (Port_D1) . Samtliga omvandlare är 8 bitars.
Programvaran till systemet består av en realtidskärna som stöder semaforhantering på samma sätt som vårt labsystem samt en dynamisk schemaläggare med prioritetsarv implementerat. Systemet skall grovt sett fungera enligt följande.

En **process P1** läser av Port_A1 med periodiciteten 30 ms . Inläst värde lagras i en gemensam cirkulär buffert. En **process P2** läser av Port_A2 med en periodicitet av 42 ms och lägger inläst värde i samma cirkulära buffert.
Ytterligare en **process P3** beräknar medelvärdet av befintliga tal i bufferten och matar ut medelvärdet till Port_D1 med periodiciteten 70 ms.
För att klara hanteringen av den gemensamma cirkulära bufferten ska man använda sig av en semafor **S1.**
Processerna P1 och P2 bör för att säkerställa funktionen hos A/D-omvandlarna ha en deadline på 20 respektive 25 ms. Deadline för P3 uppskattas till ca 40 ms för säker funktion.

Analysen av de aktuella processerna och ger följande data:

	p (ms)	c (ms)	Deadline	Anv. semafor S1 (ms)
P1	30	6	20	2
P2	42	7	25	2
P3	70	18	40	8

a)

Antag att processerna prioriteras enligt tabellen ovan. Avgör om processerna är dynamiskt schemaläggningsbara. Notera att beräkningen ska inkludera semaforanvändningen.

b) Antag att processerna P1 och P2 erhåller ett startidsjitter på ± 1ms.
Beräkna svarstiden för process P3 med hänsyn tagen till jittret.

Lycka till

Bilaga (2 sidor) till tentamen i Realtidssystem, gk 2013-08-23

Programlista:

```
/*******************************************
    Exempel dubbel länkad  lista                **
    2010-01-14  / Peter Lundin                   **
*******************************************/
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>

//#### Konstanter #####
#define MAX 20

// ##### Typdeklarationer ####
typedef struct element{
      int data;
      struct element *prev; // previus struct
      struct element *next; // next struct
}Queelement ;

typedef struct  {
      Queelement *head;
      Queelement *current;
}Queiterator;

// ##### Funktionsprototyper #####
Queiterator* newque();
void add_new_element_last( Queiterator *Quetop );
int add_post_list(Queiterator *quetop, int post_nr, int data);

//###### Huvudprogram #######
int main(int argc, char *argv[])
{
   Queiterator *que1;
   int nr=0;
   que1=newque();
   printf("\n Data ar %d", que1->head->data );
   // ------------------------------------ Lägg till 10 element till listan.
   for ( nr=0; nr<10; nr++)
      add_new_element_last( que1 );
   //------ Se till att curentpekaren pekar på element 1
   que1->current=que1->head;
    // -----skriv ut listan.
    while(!(que1->current->next==NULL)){
         printf("\n Data ar %d", que1->current->data );
         que1->current=que1->current->next;
    }
 // Skriv ut sista postens data
    printf("\n Data ar %d", que1->current->data );
  // --- fortsättningen av programmet ej med här


  //------------------

  return 0;
}
// ################################ Slut på Huvudprogrammet ##########################
```

```
// ------------------------------------------------ Befintliga funktioner   ----------------------------------
```

```
Queiterator* newque() {
   int i=0;
   Queiterator *quetemp;
   quetemp=(Queiterator*) malloc(sizeof(Queiterator));

   quetemp->head=(Queelement*) malloc(sizeof( Queelement));
   quetemp->current=quetemp->head;
   quetemp->head->next=NULL;
   quetemp->head->prev=NULL;
   quetemp->head->data=1;

   return(quetemp);
}
// ---------------- Slut Queiterator -----------------------------
```

```
void add_new_element_last( Queiterator *quetop ){
    static int nr=2;
    Queelement *newelement;
    // Se till att currentpekaren pekar på element 1
    quetop->current=quetop->head;
    // Sök upp slutet av listan.
    while( !(quetop->current->next==NULL)){
          quetop->current=quetop->current->next;
    }
    // skapa nytt
    newelement=(Queelement*) malloc(sizeof( Queelement));
    quetop->current->next=newelement;
    newelement->next=NULL;
    newelement->prev=quetop->current;
    newelement->data=nr;
    nr++;
}
// ----------------- Slut add_new_element_last ---------------
```