Datorteknik och Realtidssystem Fö1

Introduktion

Kent Palmkvist

LINKÖPINGS UNIVERSITET

Datorteknik och realtidssystem : Agenda

- Varför är det bra att lära sig om realtidssystem?
- Vad är ett realtidssystem?
- Var finns och används realtidssystem?
- Hur kan man uppnå realtidsfunktionalitet?
- Vilka är de centrala delarna i ett realtidsoperativsystem?
- Formalia
  -Föreläsningar
  -Laborationer
  -Kursmaterial
  -Examination

Varför är det bra att lära sig om realtidssystem?

Exempel ur verkligheten ...

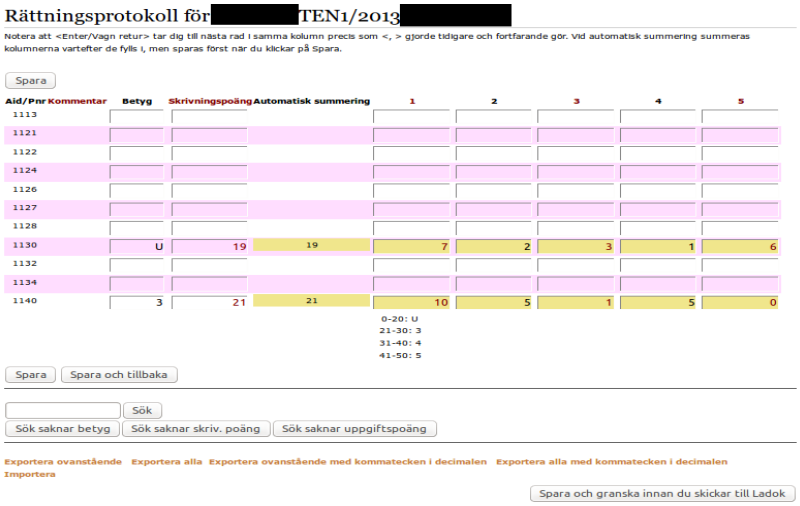
LINKÖPINGS UNIVERSITET

Gridlock

Kallas i datorsammanhang för ***deadlock*** (ett cykliskt beroende mellan processer)

LINKÖPINGS UNIVERSITET

Ladok



Rättningsprotokoll för [redacted] TEN1/2013 [redacted]

Notera att <Enter/Vagn retur> tar dig till nästa rad i samma kolumn precis som <, > gjorde tidigare och fortfarande gör. Vid automatisk summering summeras kolumnerna vartefter de fylls i, men sparas först när du klickar på Spara.

Spara

Aid/Pnr	Kommentar	Betyg	Skrivningspoäng	Automatisk summering	1	2	3	4	5
1113									
1121									
1122									
1124									
1126									
1127									
1128									
1130		U	19	19	7	2	3	1	6
1132									
1134									
1140		3	21	21	10	5	1	5	0

0-20: U
21-30: 3
31-40: 4
41-50: 5

Spara | Spara och tillbaka

Sök

Sök saknar betyg | Sök saknar skriv. poäng | Sök saknar uppgiftspoäng

Exportera ovanstående | Exportera alla | Exportera ovanstående med kommatecken i decimalen | Exportera alla med kommatecken i decimalen
Importera

Spara och granska innan du skickar till Ladok

Ladok

Hypotetisk källkod med parallella aktiviteter

```
// Aktivitet 1: Autosummera regelbundet
// nuvarande rad
while (true) {
  int sum=0;
  for (i=0; i=last_column; i++) {
    sum = sum + get_fields(current_line, i);
  }
  update_sum_field(current_line, sum);
  wait_250_milliseconds();
}
```

```
// Aktivitet 2: Körs var gång man
// trycker på knappen TAB
if (current_column == last_column) {
  current_line++;
  current_column = 0;
} else {
  current_column++;
}
```

Vad är problemet?
Vad händer om man i sista fältet skriver in en siffra och sedan (snabbt) trycker på TAB?
Denna typ av fel kallas för ***race condition*** (kapplöpning)

Therac-25 : Strålbehandlingsmaskin (1980-tal)

```
PATIENT NAME   : JOHN DOE
TREATMENT MODE : FIX     BEAM TYPE: X     ENERGY (MeV): 25

                           ACTUAL     PRESCRIBED
    UNIT RATE/MINUTE          0            200
    MONITOR UNITS           50  50         200
    TIME (MIN)              0.27          1.00

GANTRY ROTATION (DEG)       0.0             0     VERIFIED
COLLIMATOR ROTATION (DEG) 359.2           359     VERIFIED
COLLIMATOR X (CM)          14.2          14.3     VERIFIED
COLLIMATOR Y (CM)          27.2          27.3     VERIFIED
WEDGE NUMBER                  1             1     VERIFIED
ACCESSORY NUMBER              0             0     VERIFIED

DATE   : 84-OCT-26   SYSTEM : BEAM READY   OP.MODE: TREAT AUTO
TIME   : 12:55. 8    TREAT  : TREAT PAUSE           X-RAY 173777
OPR ID : T25VO2-RO3  REASON : OPERATOR     COMMAND:
```


Therac-25 : Strålbehandlingsmaskin (1980-tal)

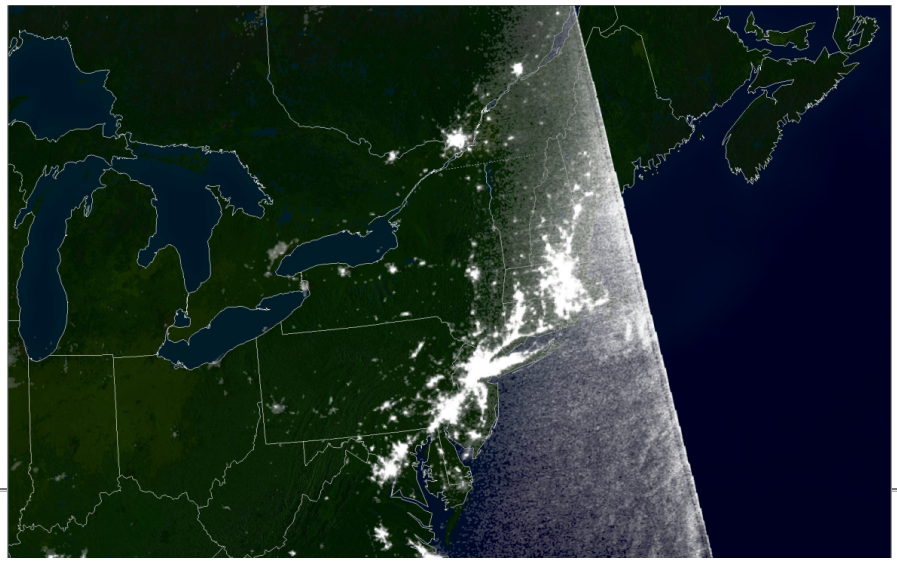
- Tre moder: X-ray treatment, electron beam treatment, field light
- Funktion: I X-ray-mode tar det ung. 8 sekunder att initiera maskinens magneter, lite beroende på maskinens ålder.
- Någon fick en idé: Hårdvara degraderar med tiden, mjukvara gör inte det, alltså: Låt mjukvaran sköta alla säkerhetskontroller.
- Problem: Det uppstod ett race condition mellan användarinterfacet och andra kritiska delar av mjukvaran.
- Om operatören bytte från X-ray till electron beam under dessa 8 sekunder upptäcktes inte det av alla delar i systemet.

Therac-25 : Strålbehandlingsmaskin (1980-tal)

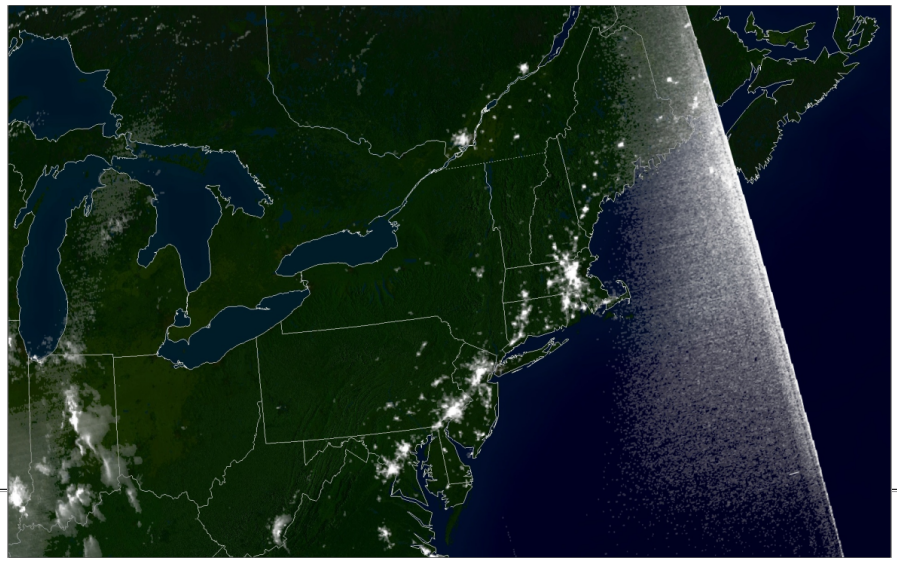
- Resultat: Patienten fick en stråldos på ca 20000 rad istf 200 rad
- Sjukhuset kunde initialt inte diagnosticera skadorna då man aldrig tidigare sett så allvarliga strålskador, dvs det tog ett tag att upptäcka problemet
- Risken att dö är ca 50% efter en helkropps-stråldos av 500 rad.
- Orsak till problemet: Felaktig implementering av parallella aktiviteter vilket orsakade ett ***race condition***

Källa : https://en.wikipedia.org/wiki/Therac-25

Nordamerika 2003, 13 augusti



Nordamerika 2003, 14 augusti



Nordamerika 2003, massivt strömavbrott

- En bug i övervakningssystemet till strömförsörjningsnätet orsakade ett ***race condition***, då två processer försökte uppdatera en gemensam datastruktur samtidigt
- Övervakningssystemet gick in i en oändlig loop
- Personalen förstod inte att övervakningssystemet slutat fungera
- 50 miljoner människor blev utan ström (tur att det inte var vinter)
- Uppskattad kostnad: $4.5-$10 miljarder

Källa : https://en.wikipedia.org/wiki/Northeast_blackout_of_2003

Patriot missile defense system

- Under Gulf-kriget (1990-1991) falerade systemet att upptäcka inkommande SCUD-missiler
- Tiden i Patriot-systemet representerades med 24-bitars fixpunktstal `000000001001.101100000000 = 8+1+1/2+1/8+1/16 = 9.6875`
- Tiden räknades upp med ett inkrement av 0.1 sekunder,vilket inte kan representeras exakt med fixpunktstal
- Efter 100 timmars upptid har tiden driftat ca 0.3 sekunder
- En SCUD-missil hinner ganska långt, ca 500 m, på 0.3 sekunder …
- 28 amerikaner dog, ca 100 skadades
- 2018 beslutade Sverige att köpa Patriot-systemet🤔

Källa : http://www-users.math.umn.edu/~arnold/disasters/patriot.html

Vad är ett realtidssystem?

En kort definition

Vad är ett realtidssystem?

- Ett realtidssystem är ett datorsystem med särskilda krav beträffande responstider
  *Dvs systemet *ska* ge respons inom en viss given tidsram och det räcker inte med att ge respons ”så snabbt det går”*
- Ett realtidssystem måste inte nödvändigtvis vara snabbt, och ett snabbt datorsystem är inte nödvändigtvis ett realtidssystem
  Kravet kan t ex vara att sampla data vid givna tidsintervall med god precision, inte att sampla så snabbt som möjligt
- De viktigaste egenskaperna hos ett realtidssystem är determinism, och att de ingående processerna möter sina deadlines
  Om systemet är förutsägbart/deterministiskt så går det att avgöra om det är möjligt att schemalägga aktiviteter inom givna tidsramar

Vad är ett realtidssystem?

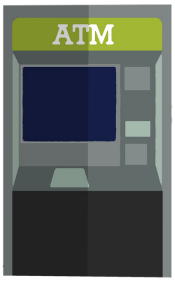
- För ett **hårt** realtidssystem måste kraven på responstid uppfyllas för att systemet ska anses fungera
  T ex en krockkudde får inte aktiveras för sent eller för tidigt, annars ”fungerar” den inte
- För ett **mjukt** realtidssystem är kraven att uppfylla tider inte lika strikta, utan systemet kan anses fungera i alla fall
  T ex en videouppspelare kan missa en bildruta ibland utan att det märks
- Det finns de som menar att alla realtidssystem måste vara hårda för att räknas som realtidssystem

Var finns och var används realtidssystem?

Några exempel

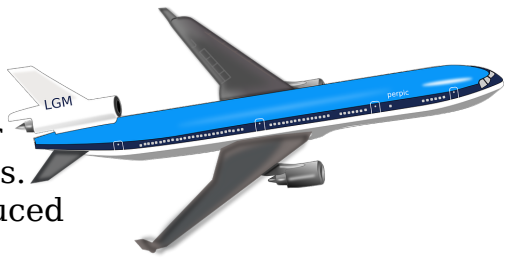
LINKÖPINGS UNIVERSITET

Var finns och var används realtidssystem?



Bankomat: Kort, pengar och kvitto ska levereras till användaren i ”lagom” takt efter det att koden matats in. Tidskraven på när dessa steg sker är kanske inte så hårda, ”bara vi får pengarna” inom rimligt tid.

Flygplan: Här finns massor med exempel på svarstider. T ex tiden från det att piloten rör styrspaken till dess att roderytorna aktiveras. Tar det för lång tid ökar risken för Pilot-induced Oscillations (JAS Gripen, Linköping 1989)



Källa : https://en.wikipedia.org/wiki/Accidents_and_incidents_involving_the_JAS_39_Gripen

Var finns och var används realtidssystem?

Kamera: En bild ska kunna tas, kanske med en viss tidsfördröjning, och med korrekt slutartid. Den tagna bilden måste hinna läsas från sensorn och sparas på minneskortet innan nästa bild tas.

Bil: Innehåller många olika styrsystem med krav på svarstider. T ex krockkudden som måste utlösas vid rätt tidpunkt.

Var finns och var används realtidssystem?

Mobiltelefon: Bör ju reagera ”hyggligt” snabbt när man använder touchskärmen. Måste också reagera inom en mikrosekund på inkommande datapaket för att kalibrera radiomottagarens förstärkare.

Klocka: Ska ju förstås visa rätt tid utan att dra sig. Strängt taget, finns det egentligen någon klocka som håller tiden exakt, över tid?

Var finns och används realtidssystem?

- Kortfattat kan vi säga att realtidssystem finns nästan överallt. Det är exempelvis mycket få inbyggda system som ej kräver en viss mängd realtidsprogrammering.
- Däremot så varierar kraven på svarstider samt variationer i svarstiden markant mellan olika applikationer.
- Ska man jobba med inbyggda system (som ju finns överallt) behöver man känna till realtidsprogrammering. Dessutom, då alla ”vanliga” datorer har flera kärnor behöver man känna till parallellprogrammeringsaspekten för att få ut maximal prestanda.

Var är **inte** ett realtidssystem?

- Webben, kanske?

  Internet innehåller massor med teknik som var för sig har realtidsfunktionalitet, men sammantaget är det svårt att ställa realtidskrav på hela webben.

Hur kan man uppnå realtidsfunktionalitet?

Några exempel

Hur kan man uppnå realtidsfunktionalitet?

- Som ett ”vanligt” program i ett ”vanligt” operativsystem?
  Fungerar troligen inte så bra, särskilt om realtidsfunktionen ska uppnås via tajmade vänteloopar

```
for (i=0; i<60; i++)
  {
    data[i] = sample();
    wait_ms(1000);
  }
```

Väntetiden kan förskjutas om andra processer i operativsystemet avbryter loopen, dessutom tar övrig programkod såsom for-loop och sampling viss tid

Hur kan man uppnå realtidsfunktionalitet?

- Utan ett operativsystem, s k Bare Metal?

-Utan avbrott : Endast för extremt enkla applikationer

```
for (i=0; i<60; i++)
  {
    data[i] = sample();
    wait_ms(986);
  }
```

3-4 ms
10 ms
986 ms

Hur kan man uppnå realtidsfunktionalitet?

- Utan ett operativsystem, s k Bare Metal?

-Med avbrott : t ex foreground/background

Foreground:

```
enable_timer_ms(1000);
while (i<60)
  {
  }
disable_timer();
```

Background: 1 gång / 1000 ms

```
timer_interrupt()
  {
    data[i] = sample();
    i++;
  }
```

Hur kan man uppnå realtidsfunktionalitet?

- Utan ett operativsystem, s k Bare Metal?

-Foreground/background scheduling:

- Använd en huvudloop (*foreground*) som gör sådant arbete som inte är tidskritiskt.
- Lägg till avbrott (*background*) som kan initiera aktiviteter med jämna mellanrum, vid specifika tidpunkter eller reagera direkt på vissa händelser.
- Ett samverkande (*concurrent*) system uppnås, där processorkraften delas mellan huvudloop och olika avbrottshanterare.
- Huvudloopen och avbrottshanterare kan betraktas som parallella aktiviteter

Hur kan man uppnå realtidsfunktionalitet?

- Med ett realtidsoperativsystem, s k RTOS!

-De flesta operativsystem har något slags stöd för realtidsprogrammering t ex Linux

```
for (i=0; i<60; i++)
  {
    data[i] = sample();
    wait_until(1000);
  }
```

```
wait_until(int d)
  {
    t = t + d;
    clock_sleep(ABS, t);
  }
```

Hur kan man uppnå realtidsfunktionalitet?

- Med ett realtidsoperativsystem, s k RTOS!

Det finns speciella realtidsoperativsystem, vanligen med mindre funktionalitet men med just den funktionalitet som krävs av en realtidsprogrammerare. T ex OSE, VxWorks, uC/OS-III (kommersiella) eller FreeRTOS, Simple-OS (open source).

I kursen:
-Linux i praktiken (laborationer)
-Simple-OS (och Linux) i teorin

Vilka är de centrala delarna i ett realtidsoperativsystem?

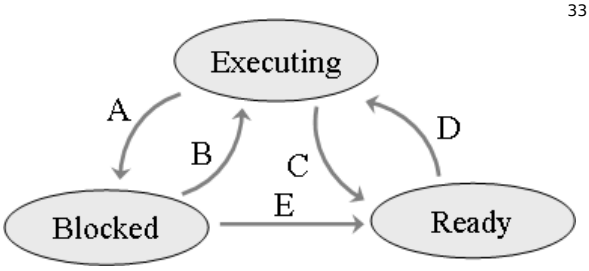
Centrala delar i ett RTOS

- Avbrott – Det normala programflödet avbryts, en avbrottsrutin körs, varefter det normala programflödet återupptas där det avbröts.
- Processorregister – Innehåller status (kontext) för den programkod som för tillfället exekverar. Dessa register behöver sparas (typiskt på en stack) då en avbrottsrutin ska köra. De register som håller programräknaren och stackpekaren är av särskilt intresse.
- Stack – En stack är en LIFO-lista (Last In First Out) i minnet och adresseras med ett stackpekarregister.

Realtidsoperativsystem, RTOS

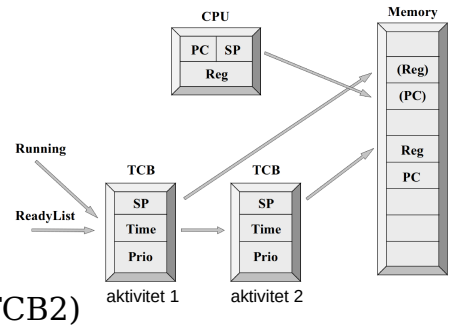
- Ett realtidsoperativsystem är ett operativsystem designat för realtidskrav.
- Ett realtidsoperativsystem kan hantera parallella aktiviteter, ofta kallade *tasks*.
- *Tasks* kan även kallas för *trådar* eller *processer*.
- Trådar har oftare närmare kommunikation med varandra än processer, då trådar delar minnesutrymme medan processer (med separata minnesutrymmen) får kommunicera på annat sätt (t ex filer, sockets, pipes)
- Processer skulle kunna vara vanliga program, och trådar kan löpa inom ett program.

RTOS, Task states



- A – En process vill använda en gemensam resurs, och den resursen används redan av en annan process
- B – En process väntar på en gemensam resurs, den blir ledig och den väntande processen får exekvera
- C – En process med högre prioritet än nuvarande körande process blir redo för att köra
- D – En process blir körande, för att den har högre prioritet eller för att en annan process blir blockerad
- E – En gemensam resurs, som en process väntar på, blir ledig men den väntande processen får inte lov att köra, t ex pga lägre prioritet

RTOS, Task switch (processbyte)



1. Spara alla register för aktivitet 1 på dess stack
2. Spara stackpekare och programräknare för aktivitet 1 på lämpligt ställe som hör till aktivitet 1 (TCB1 – Task Control Block)
3. Läs in stackpekare som hör till aktivitet 2 (från TCB2)
4. Läs in alla register för aktivitet 2 från dess stack
5. Läs in programräknare (dvs återhoppsadress) för aktivitet 2 (från TCB2)

Processbyte kan ske frivilligt (en process begär själv att processbyte ska ske) eller ofrivilligt (RTOS tvingar ut den för en annan process med högre prioritet)

Realtidsoperativsystem, RTOS (fler frågor)

- Kan alla processer köra samtidigt?
- Om inte, hur bestäms det vilka processer som får köra?
- Kan processer köra i en viss ordning?
- Vad händer om flera processer använder samma resurs?
- Hur skyddar man gemensamma resurser?
- Var avgörs det vilken process som får en resurs?
- I programmet? I operativsystemet? I processorn?
- Hur vet man att alla processer hinner med sitt arbete?
- Vad händer om dom inte gör det?

Formalia

Kursmoment

- Föreläsningar
- Laborationer
- (Inlämningsuppgift)
- Tentamen

Föreläsningar

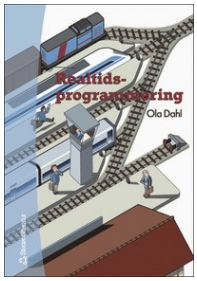
1. Introduction
2. Shared Resources (L1)
3. Task synchronization (L2)
4. Monitors, message passing (L3, L4)
5. Real-time kernel + OS (L5)
6. Scheduling + Real-time tasks in Linux (L6)
7. (Spare)

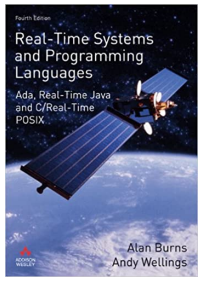
Laborationer

1. Shared Resources
2. Alarm Clock
3. Lift with Monitors
4. Lift with Message Passing
5. Lift, performance analysis (inlämningsuppgift)
6. Real-time Linux

OBSERVERA: Merparten av labarbetet måste göras utanför labtid.
Labtiden är främst till för redovisning och hjälp.
Laborationerna görs i C under Linux. Kan göras på egen Linuxdator, MEN måste redovisas på datorerna i labbet.

Kursmaterial





- Kursbok (finns inte längre)
  Realtidsprogrammering
  Ola Dahl

  Enstaka ex finns att låna på LiU's bibliotek

- Alternativ kursbok
  Real-Time Systems and Programming Languages
  Alan Burns and Andy Wellings

- Kursens web-sida
  Fö-underlag + powerpoints, labbmaterial, tidigare tentor

Tentamen

Bygger framför allt på det teoretiska stoffet från kursbok och föreläsningsunderlag, men erfarenhetsmässiga kunskaper från laborationerna kan också ingå.

Uppgifterna kan tänkas använda både kursbokens Simple OS och laborationernas Linux för olika realtidsprimitiver och begrepp.

Programmeringsuppgifter använder sig av C, men det är inte en tenta i C-programmering. Dvs, syntax och funktionsnamn är underordnat, men det behöver entydigt framgå i ett svar vad som menas.

Resultat från tidigare år, samt förändringar till i år

- 30 studenter läste kursen 2022
- 7 av dessa svarade på kursutvärderingen (Evaliuate), dvs 23%
- Helhetsbetyg 4.57

www.liu.se

LINKÖPINGS UNIVERSITET