Realtidssystem

- Schemaläggningsanalys -

EDAF85 - Realtidssystem (Helsingborg)
Roger Henriksson

Föreläsning 7

Kursens innehåll motsvarar tidigare omgångar under beteckning EDA698
Föreläsningsbilder av Elin A. Topp
Stora delar baserad på: Föreläsningsmaterial EDA040 (Klas Nilsson, Mathias Haage) samt EDA698 (Mats Lilja)

Innehåll

- Schemaläggningsanalys
  - Grundläggande analys, övre gräns
  - Grundläggande analys, exakt
    - Med schemaläggningsdiagram
    - Formellt (beräknad)
  - Allmän schemaläggningsanalys

Schemaläggning - begrepp

- Schedulability (möjligt att schemaläggas):
  - Ett system är schemaläggningsbart (schedulable) när alla trådar håller sina deadlines
- Schedulability test (test för möjlig schemaläggning)
  - Givet ett antal trådar och kunskap om deras prioriteter (RMS - perioder räcker!), besvarar ett sådant test frågan "Är detta system schemaläggningsbart?"
  - Svaret blir "ja" eller "nej".
- Scheduling analysis (schemaläggningsanalys)
  - Arbetet att bedöma ett systems egenskaper när det gäller schemaläggningen, utfört under systemets uppbyggande
- Scheduling (schemaläggning)
  - Specialfall: statisk schemaläggning
  - Normalfall: dynamiskt schemaläggning
  - OBS: Prioriteter och deadlines är till för korrekt realtidshantering (realtime correctness), medan den korrekta hanteringen av jämlöpande exekvering (concurrency correctness) INTE får vara kopplad till prioriteter!

Grundläggande schemaläggningsanalys

- Förenklingar:
  - Periodiska trådar
  - **Ingen blockering**
  - Deadline = period
  - Fixed-priority, e.g., RMS

- Beteckningar
  - T = Period
  - C = Exekveringstid (värsta fall)
  - U = C/T = CPU-användning
  - R= Response time (svarstid)
  - D = deadline (= T)

RMS - Rate monotonic scheduling

RMS regel: välj prioritet i relation till perioden; Kort period motsvarar hög prioritet.

Fråga: Hur bra är denna tumregel? Kan vi granskar ett system mera noggrant? Hur exakt kan vi bestämma, om ett system kommer att fungera?

Fråga: Hur mycket CPU-tid kan vi använda? Går det att utnyttja CPUn till 100% om flera trådar ska samsas om den, som var för sig har en viss grad av nyttjande av CPU-tid?

Scheduling analysis enligt RMS: Hur mycket CPU-användning kan vi ha och fortfarande garantera att det finns ett schema som fungerar?

Förenklingar

Till en början antar vi:
- Periodiska trådar
- Ingen blockering på resurs
- Deadline = period

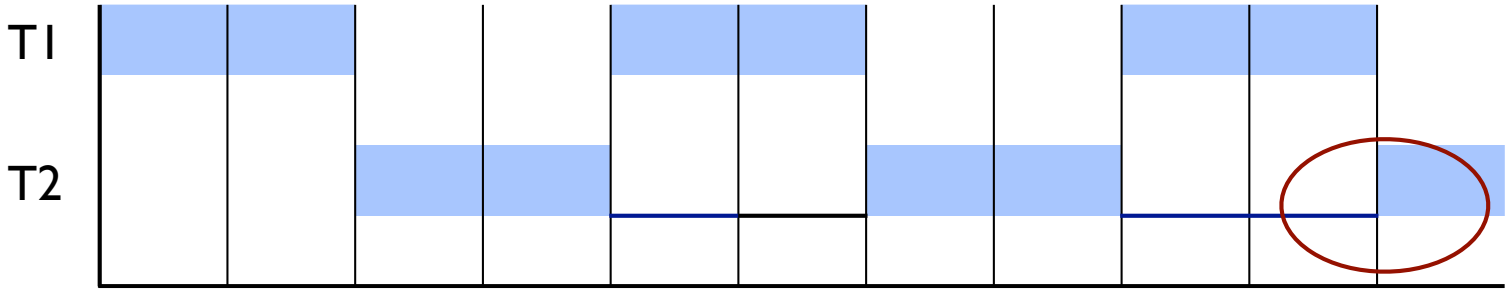
Notation

För varje tråd vet vi:
- T = Period
- C = Exekveringstid
- U = C/T = CPU-användning

Exempel (RMS)

Vi försöker med 100% CPU utnyttjande

T1: C=2ms, T=4ms, C/T=0.5 T2: C=5ms, T=10ms, C/T=0.5



Men med T2: C=2ms, T=4ms, C/T=0.5 hade det fungerat...
problemet ligger alltså i relationen mellan perioderna

Värsta fallet som fortfarande går att schemalägga för två trådar utan blockering

Med T1: C=1ms, T=2ms (C/T=0.5) får man
T2: C=1ms, T=3ms som minsta U = C/T som fortfarande går att schemalägga.
Totalt blir U = C/T = 1/2 + 1/3 ≈0.83

RMS - analys av övre gränsen

(Liu & Layland, 1973)

Generellt är det möjligt att garantera att det finns ett schema (schedulability) om

$$\sum \frac{C_i}{T_i} < n(2^{1/n} - 1)$$

med n = antal trådar.

n=1, U=1;

n=2, U≈0.83;

n=3, U≈0.78;

n=∞, U≈0.69;

OBS: Ett system KAN fortfarande vara "schemaläggningsbart", även om U ligger högre än 69%. Då måste man göra en ***exakt*** analys av det aktuella systemet.

RMS - exakt analys

(Joseph & Pandya)

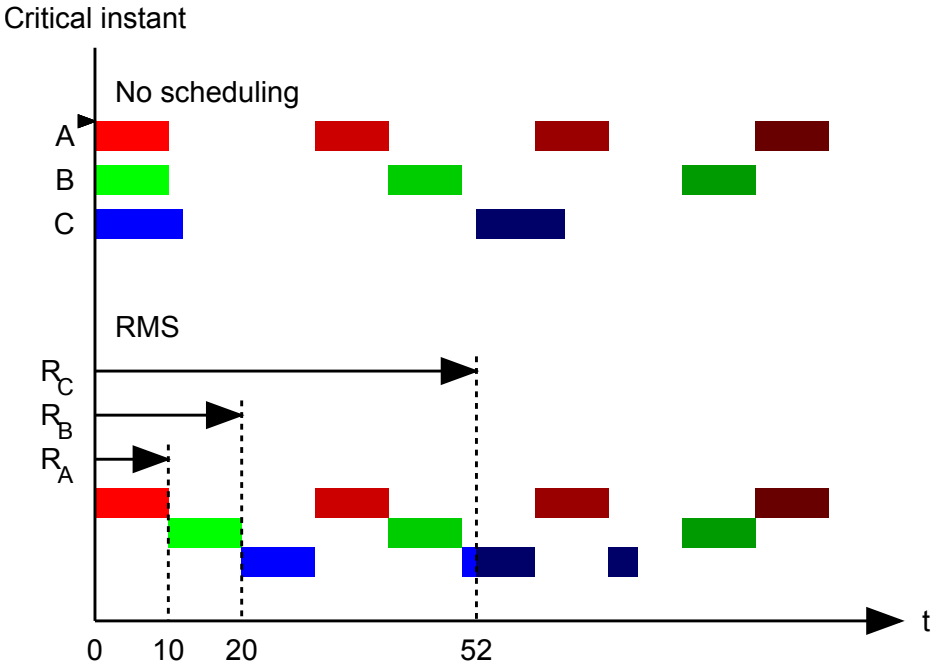
Analysen av den övre gränsen för RMS (Liu and Layland, Upper Bound analysis of RMS) lämnar ganska stora marginaler; Om det ska vara mera exakt, måste man titta på R (worst case response time), inte den summerade belastningen av CPUn.

Vi tittar på vad som händer vid den mest kritiska tidpunkten (critical instant) - när alla trådar vill dra igång samtidigt.

Teorem: Om alla trådar håller sin första deadline efter en sådan kritisk tidpunkt, kommer de att hålla alla följande deadlines, eftersom de blir mindre kritiska och "enklare" att hantera.

RMS: exakt analys med schemaläggningsdiagram

Thread	C	T
A	10	30
B	10	40
C	12	52



$$U_A = \frac{C_A}{T_A} = \frac{10}{30} = 0.33$$
$$U_B = \frac{C_B}{T_B} = \frac{10}{40} = 0.25$$
$$U_C = \frac{C_C}{T_C} = \frac{12}{52} = 0.23$$

$$U = \sum U_i = 0.81 > 0.78$$

Worst-case response times:

$$R_a = C_a = 10 \qquad R_b = C_b + C_a = 20 \qquad R_c = C_c + C_a + C_b + C_a + C_b = 52$$

RMS: exakt analys av svarstiderna

Worst case response time (svarstiden i värsta fall) är den minsta tiden R_i som uppfyller följande formel:

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

$hp(i)$ mängd aktiviteter (trådar) med högre prioritet än *i*

$\left\lceil \frac{R_i}{T_j} \right\rceil$ antal gånger aktivitet *i* kan bli avbruten av den högre prioriterade aktiviteten *j*

$\lceil \rceil$ tak (ceil), avrundning uppåt, e.g., $\lceil 1.6 \rceil = 2$

RMS: exakt analys, iterativt exempel

Vi använder samma trådsystem som för den grafiska analysen

Thread	C	T
A	10	30
B	10	40
C	12	52

$$R_i^{k+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^k}{T_j} \right\rceil C_j$$

Iterativ beräkning av R_i för i = A, B, C:

$R_a^0 = 0$

$R_a^1 = 10$ (*)

$R_a^2 = 10$ (stable)

$R_b^0 = 0$

$R_b^1 = 10 + 0 \cdot 10 = 10$

$R_b^2 = 10 + 1 \cdot 10 = 20$

$R_b^3 = 10 + 1 \cdot 10 = 20$ (stable)

$R_c^0 = 0$

$R_c^1 = 12 + 0 \cdot 10 + 0 \cdot 10 = 12$

$R_c^2 = 12 + 1 \cdot 10 + 1 \cdot 10 = 32$

$R_c^3 = 12 + 2 \cdot 10 + 1 \cdot 10 = 42$

$R_c^4 = 12 + 2 \cdot 10 + 2 \cdot 10 = 52$

$R_c^5 = 12 + 2 \cdot 10 + 2 \cdot 10 = 52$ (stable)

* no higher priority threads

Allmän schemaläggningsanalys vid RMS

- **Förenklingarna** vi hade, som faktiskt nästan aldrig uppfylls:
  - Enbart periodiska trådar
  - **Ingen blockering**
  - Deadline (D) = period (T)
  - **Omedelbar kontextbyte** utan hanteringstid (release time)
- Med den **allmänna** analysen (Generalized Rate Monotonic Analysis) utvidgar man metoden:
  1. Kortare deadlines (D < T)
  2. Tillåter **blockering** (på delade resurser, tidsbegränsad med prioritetsarv)
  3. Icke-periodiska trådar tillåtna (undvik därmed extremt pessimistiska resultat, annars hade man behövt tilldela perioder för säkerhets skull)
  4. Icke-omedelbar kontextbyte (“release jitter”) etc.

Punkterna 1 och 2 bör man känna till och kunna använda i analysen.
Punkterna 3 och 4 bör man känna till.

Allmän schemaläggningsanalys, D < T

- Situationen med Deadline = Period förekommer sällan (oftast i "toy examples" i kursen ;-), oftast har vi egentligen D < T
  - Händelser som förekommer sällan, men måste då hanteras kvickt.
- Test för möjlig schemaläggning ger svaret "ja" eller "nej" på frågan "Är det här systemet av trådar schemaläggningsbart?"
  - Beräkna svarstiden i värsta fall, R_i, för alla trådar i
  - Om alla R_i <= D_i är systemet schemaläggningsbart
- D < T - Deadline Monotonic Scheduling (variant av RMS):
  - Ge prioritet enligt D istället för T
  - Beräkna R som tidigare
  - Kolla att R < D
  - Analysen enligt Joseph & Pandya håller fortfarande

Allmän schemaläggningsanalys med blockering

- Svarstiden i värsta fallet när man räknar in tiden en tråd *i* kan vara blockerad på en resurs blir:

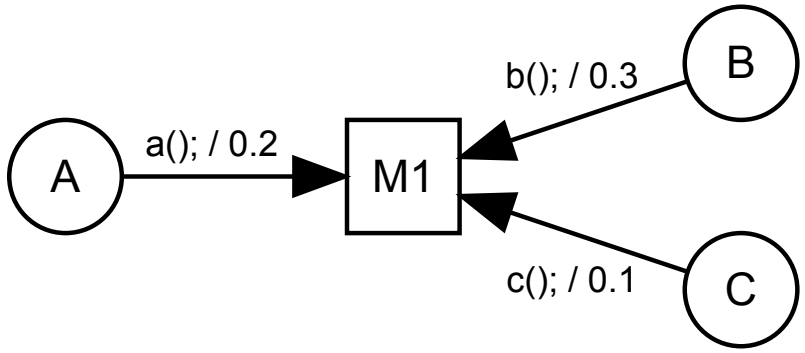
$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

- B_i är blockeringsfaktorn (eller blockeringstiden), dvs den maximala tiden tråd *i* kan vara blockerad av lägre (!) prioriterade trådar. Blockeringstiden är summan av:
  - Direkt blockering (normal blocking) - Tråden är blockerad när den väntar på en resurs som någon annan tråd håller.
  - Indirekt blockering (push-through blocking) - Tråden är blockerad av en tråd med lägre prioritet som ärver högre prioritet när den annars blockerar en högt prioriterad tråd (enbart när prioritetsarv tillämpas)

Blockering, exempel 1

Går det att schemalägga följande system?

Thread	C	D	T
A	1	2	10
B	2	3	15
C	4	10	20



Iterativ beräkning av R_i för i = A, B, C:

$$R_A^0 = 0$$
$$R_A^1 = 1 + max(0.3, 0.1) + 0 = 1.3$$
$$R_A^2 = 1 + max(0.3, 0.1) + 0 = 1.3$$

$$R_B^0 = 0$$
$$R_B^1 = 2 + 0.1 + 0 = 2.1$$
$$R_B^2 = 2 + 0.1 + 1 \cdot 1 = 3.1$$
$$R_B^3 = 2 + 0.1 + 1 \cdot 1 = 3.1 \quad (*)$$

$$R_C^0 = 0$$
$$R_C^1 = 4 + 0 + 0 = 4$$
$$R_C^2 = 4 + 0 + (1 \cdot 1 + 1 \cdot 2) = 7$$
$$R_C^3 = 4 + 0 + (1 \cdot 1 + 1 \cdot 2) = 7$$

* $R_B > D_B$, *not* ok

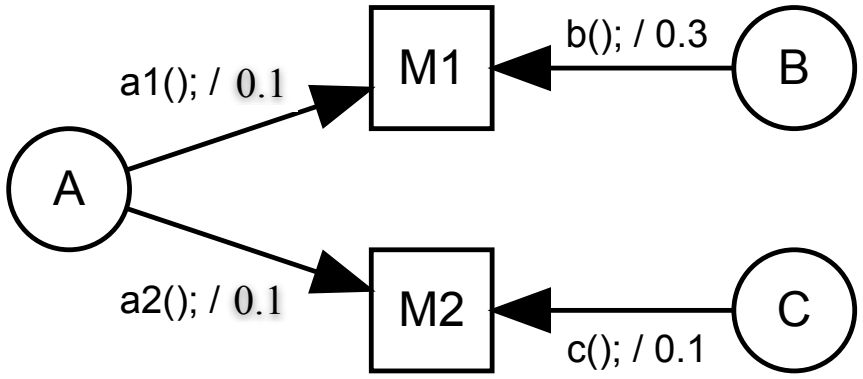
Resultat: Tråd B kan inte hålla sin deadline, eftersom den blockeras av tråd C.

Ej schemaläggningsbart!

Blockering, exempel 2

Går det att schemalägga det första systemet om man delar M1 i M1 och M2?

Thread	C	D	T
A	1	2	10
B	2	3	15
C	4	10	20



Iterativ beräkning av R_i för i = A, B, C:

$R_A^0 = 0$
$R_A^1 = 1 + (0.3 + 0.1) + 0 = 1.4$
$R_A^2 = 1 + (0.3 + 0.1) + 0 = 1.4$

$R_B^0 = 0$
$R_B^1 = 2 + 0.1^{(**)} + 0 = 2.1$
$R_B^2 = 2 + 0.1 + 1 \cdot 1 = 3.1$
$R_B^3 = 2 + 0.1 + 1 \cdot 1 = 3.1$ (*)

$R_C^0 = 0$
$R_C^1 = 4 + 0 + 0 = 4$
$R_C^2 = 4 + 0 + (1 \cdot 1 + 1 \cdot 2) = 7$
$R_C^3 = 4 + 0 + (1 \cdot 1 + 1 \cdot 2) = 7$

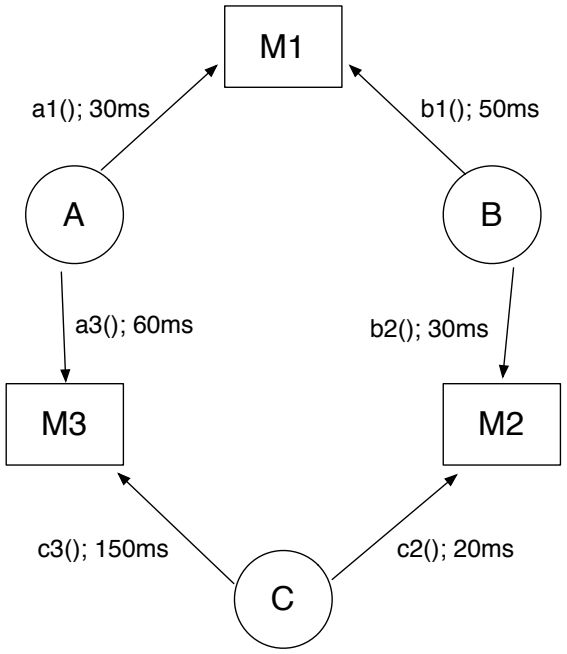
* $R_B > D_B$, *not* ok

** push-through blocking

Resultat: Tråd B kan inte hålla sin deadline, eftersom den kan indirekt blockeras av tråd C, som kör med As prioritet på M2. Ej schemaläggningsbart!

Blockeringsgraf

Blockeringsgraf för systemet:



Trådarnas perioder och exekveringstider:

	T	C
A	300	105
B	500	125
C	800	205

$B_C = 0$ (den lägst prioriterade tråden kan inte blockeras på resurs)
$B_B = max(c2(), c3()) = max(20ms, 150ms) = 150ms$ (indirekt blockering av C via M3)
$B_A = b1() + c3() = 50ms + 150ms = 200ms$ (direkt blockering av C och B)

Allmän schemaläggningsanalys
- sporadiska trådar -

En sporadisk tråd sätts igång vid oförutsägbara tidpunkter.

- Pessimistisk analys (det som vi gjorde hittills med RMS baserad analys):
  - Modellera tråden som en periodisk tråd med en periodtid som motsvarar den minsta tidsperioden mellan händelser som drar igång tråden
  - Sådana minimi-perioder existerar oftast i verkligheten med tanke på fysiska begränsningar av omgivningen
  - Använd tidigare föreslaget test för schemaläggning
- Mindre pessimistisk är analysen med användning av en *sporadisk server*
  - Idé: Reservera en viss del av CPUn för hantering av sporadiskt förekommande händelser
  - Konstruera en periodisk tråd, den *sporadiska servern* som hanterar alla sporadiska uppgifter (jobs). Låt tråden köra som mest $C_{sporadic}$ tidsenheter under varje period, $T_{sporadic}$
  - Använd föreslaget test för schemaläggningen på den *sporadiska server*-tråden
  - Behöver inte en minimi-tid mellan externa händelser
  - Kan dock inte garantera att alla deadlines hålls för alla sporadiskt förekommande händelser

Allmän schemaläggningsanalys
- inkludera starttider-

Release jitter (starttidsvariation)

- Variationer i tiden det tar för en tråd *i* att verkligen komma igång efter den har satts igång av en händelse (J_i)

Kontextbyte / trådbyte

- Det tar tid att byta till en ny tråd (C_{sw})

Klockade avbrott (clock interrupts)

- Själva avbrotten (interrupts) driver trådavbrott (preemption) och kontextbyten med en viss "arbetstid" ($C_{tick}, T_{tick}, C_{queue}$)

$$\omega_i = C_i + 2C_{sw} + B_i + \sum_{j \in hp(i)} \left\lceil \frac{\omega_i + J_i + T_{tick}}{T_j} \right\rceil (C_j + 2C_{sw})$$

$$+ \sum_{j \in alltasks} \left\lceil \frac{\omega_i + J_i + T_{tick}}{T_j} \right\rceil C_{queue} + \left\lceil \frac{\omega_i}{T_{tick}} \right\rceil C_{tick}$$

$$R_i = J_i + T_{tick} + \omega_i$$

Själva exekveringstider då?

Hur kan man bestämma *C*, *B*, etc, som vi behöver för analysen? Fortfarande ett öppet problem!

- Mäta
  - Köra applikationen (aktiviteten) många gånger med olika inmatningar
  - Mäta exekveringstiden med lämpliga analysverktyg, spara undan den längsta tiden
  - Använd den längsta påträffade tiden som C (eller B, när det är mycket inom en resurs) (optional: lägg på 10-50%)
  - Det kan aldrig bli någon garanti på att man har sett det värsta fallet!
  - CPU-egenskaper som cacheing och pipelining kan göra saken mera komplicerad
- Formell kodanalys
  - Analysera koden rad för rad; summera exekveringstiderna (värsta fall) för alla kommandon
  - Hur hantera loops? For, while, if-förgreningar?
  - Man hamnar då (igen) i ganska pessimistiska lägen
  - Cacheing och pipelining är fortfarande ett problem för uppskattningen
  - Automatiska verktyg? Gärna, men få att utnyttja!

Sammanfattning

- Recap: prioritetstilldelning, schemaläggningsprinciper, schemaläggningsanalys, prioritetsinvertering och prioritetsarv
- Schemaläggningsanalys (uppskattad, exakt, med / utan blockering)

- Nu borde ni kunna bedöma om ett (litet förenklat) system går att schemalägga

- Lästips:
  - e-bok: Kapitel 13 + 14 + 15, s 295-360