Realtidssystem

- Schemaläggning -

EDAF85 - Realtidssystem (Helsingborg)
Roger Henriksson

Föreläsning 6

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)

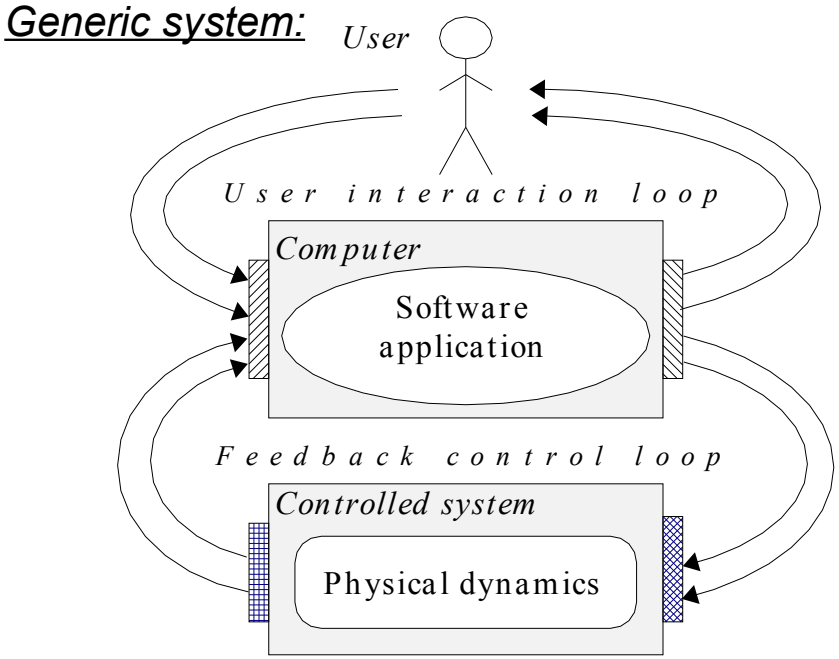
Innehåll

- Schemaläggning
  - Realtid - krav, garanti, exekveringstider
  - Schemaläggningsprinciper
  - Prioritetstilldelning
  - Prioritetsinvertering

Från parallellitet till realtid - Realtidskrav

Ett realtidssystem måste

- utföra alla beräkningar *logiskt korrekt*
- reagera på inmatningar *jämlöpande*
- alltid ha *konsistent* data att arbeta med
- producera alla (del-)resultat *i tid*



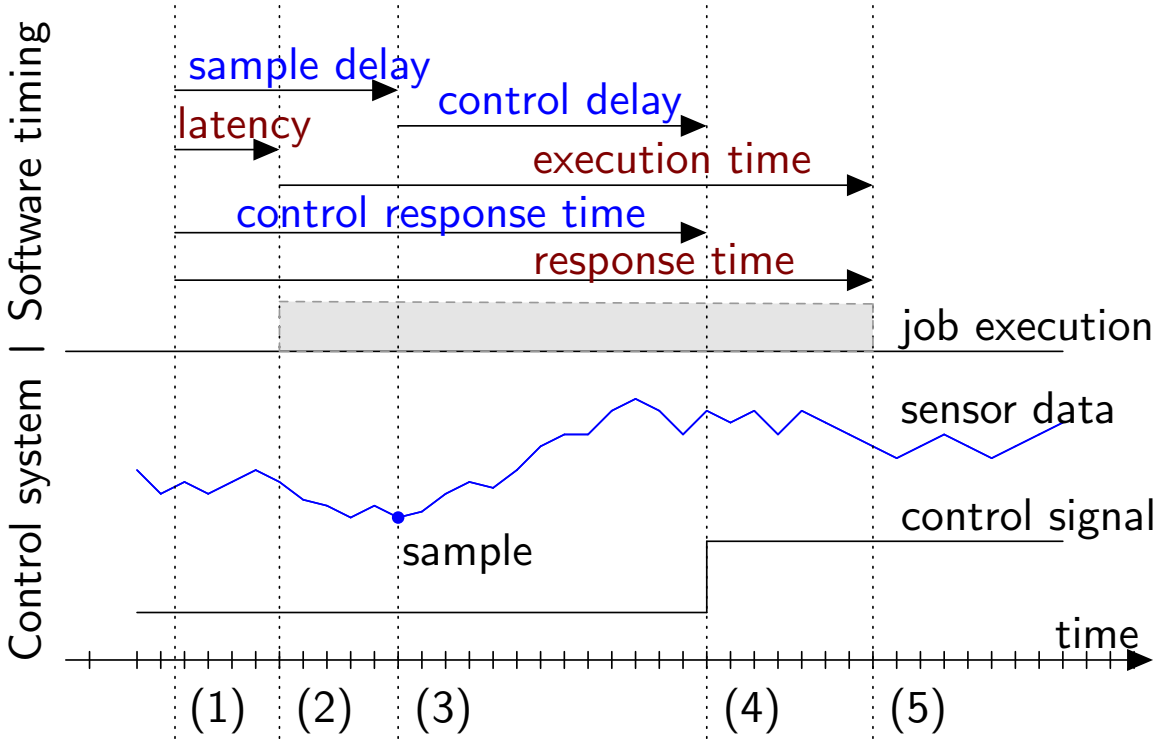
För att uppnå ***"realtids-korrekthet" (real-time correctness)*** måste mjukvaran säkerställa, att det "jämlöpande-korrekta" ***resultatet produceras pålitligt i tid***.

Ofta handlar det om att hantera både periodiskt upprepade aktiviteter parallellt med händelsestyrda aktiviteter

Tidsfördröjningar och väntetider i återkopplingsreglering

Tidpunkterna i grafiken:

1. Starttid för regleringen, önskad början av perioden
2. Faktiska starten av regleringsberäkning, efter trådbyte (release time)
3. Givarens mätdata är tillgänglig
4. De beräknade regleringsvärden blir utgivna
5. Reglering och exekvering är avslutade, inkluderande förberedelse för nästa mätomgång



Control delay och *control response time* är något som folk i reglerteknik tittar på, medans vi behandlar *execution time* (exekveringstid) och *response time* (svarstid) från mjukvaruperspektivet.

Tiden som en aktivitet (tråd) förbrukar

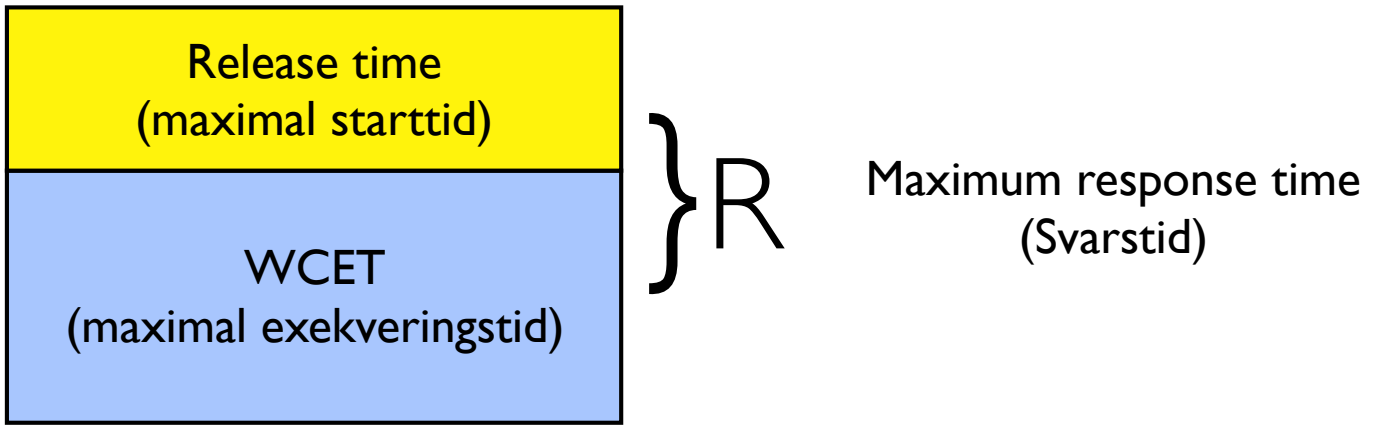
Att hantera något i realtid betyder att garantera ett korrekt (användbart) svar inom en given tid

Vi måste då titta på två olika typer av tider som ingår i en tråds sammanlagda "arbetstid", alltså den *maximala svarstiden* (maximum response time, R) som vi vill garantera:

1. *Exekveringstiden* i värsta fall (Worst Case Execution Time, WCET), plus väntetider, ifall det finns delade resurser och preemption.

2. *Starttid* (som hänger direkt ihop med andra trådars exekveringstid och trådbytestider).

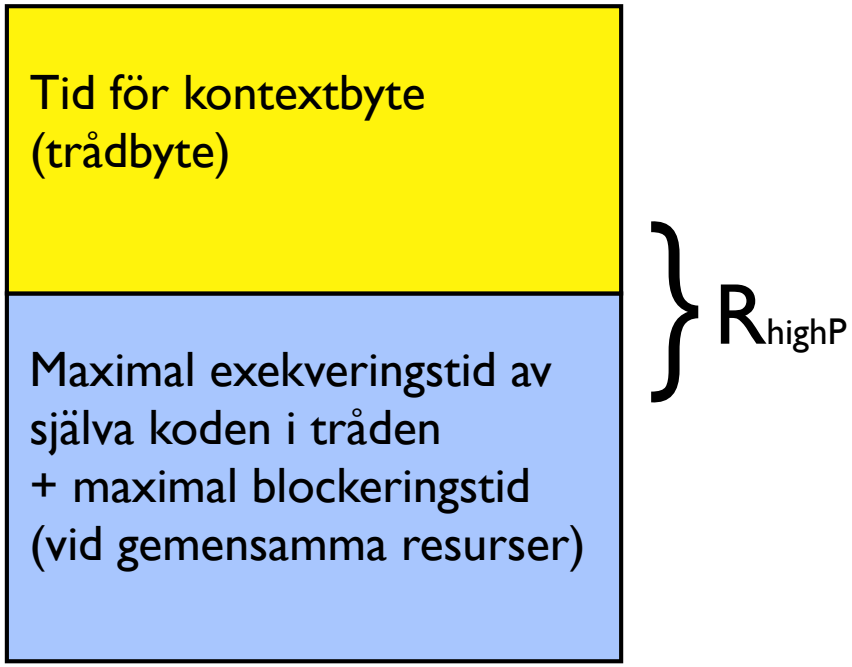
Eftersom vi vill kunna ge garantier, kommer vi alltid utgå från det värsta möjliga läget,



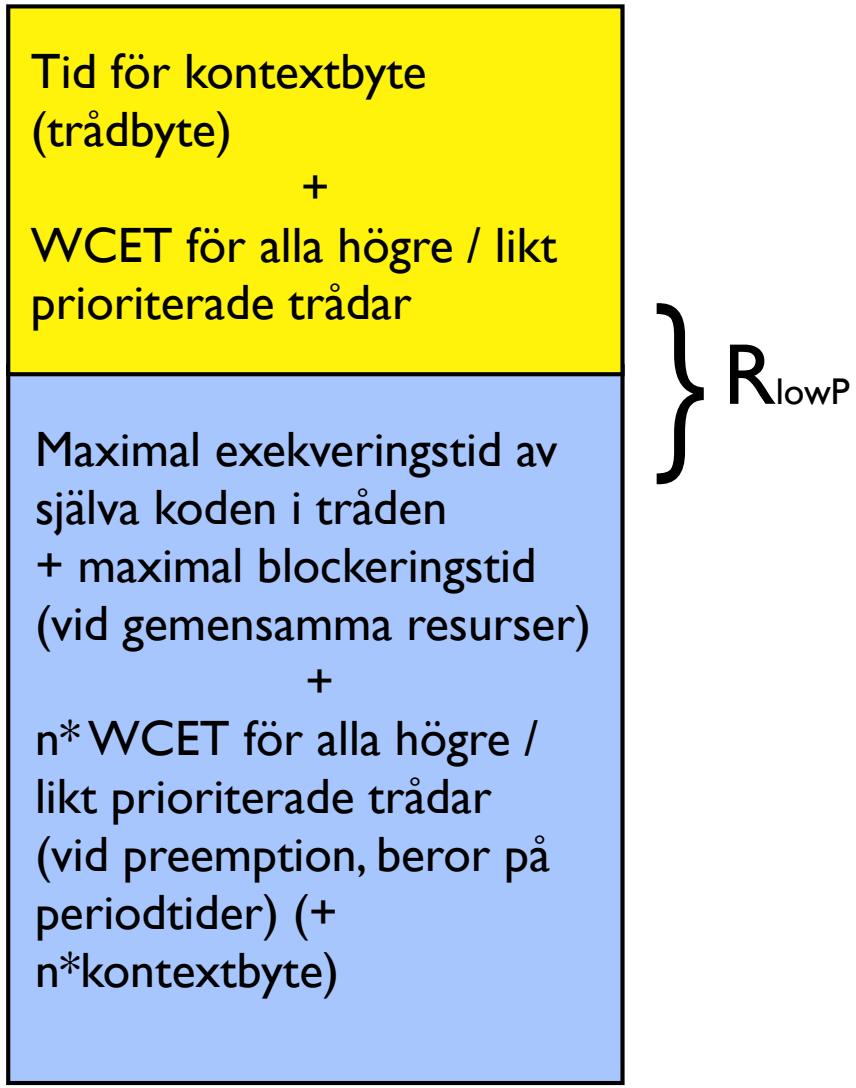
Svarstider och prioritet

Svarstiderna, dvs väntetider och fördröjningar beror också på trådens prioritet

Högst prioriterade tråden



Andra trådar

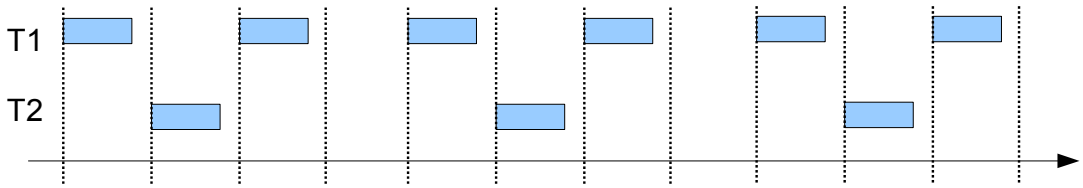


Beräkningen av den maximala svarstiden (Worst case response time) hänger alltså ihop med den valda schemaläggningen och trådarnas periodiska aktivitet

Statisk schemaläggning

Används för extremt kritiska trådar och i enkla styrsystem

- Tiden blir delad i små intervall (slots)
- Alla aktiviteter (trådars exekveringstid) måste vara tillräckligt små så att de passar in i ett intervall
- Aktiviteterna är schemalagda i passande tidsintervaller innan systemet börjar köras
- Cyklisk upprepade exekvering av schemat



T1: 50 gånger / sek, 8ms WCET,
T2: 25 gånger / sek, 8ms WCET,
10ms tidsintervall

Fördelar

- Garanterad schema, alla får köra, allt utförs inom angivna tidsramar
- Varje aktivitet kan alltid utföras färdigt, inga avbrott, inga kritiska sekvenser
- Lätt att beräkna den maximala svarstiden

Nackdelar

- Fragmentering, inget bra nyttjande av CPU-tiden
- Aktiviteter måste eventuellt delas upp i onaturliga avsnitt för att få dem in i en enstaka tidsintervall
- Schemat kan bli ganska komplext, och måste eventuellt göras om när programmet ändras

Dynamisk schemaläggning

Används i de flesta, inte helt så kritiska system, ofta också där resurserna måste utnyttjas till högsta möjliga grad

- Beroende på den valda schemaläggningsprincip hanteras trådarna dynamiskt allt eftersom de startas up
  - Strict priority (Prioritetsbaserad) - trådarna sorteras i en väntekö enligt deras prioritet
  - Round robin - trådarna sorteras i en FIFO-kö
- Vi måste anta dynamisk, prioritetsbaserad schemaläggning för att kunna bygga realtidssystem, annars måste man känna till alla trådar i systemet (så att man kan räkna in dem i schemaläggningsanalysen)

Prioritetsbaserad schemaläggning

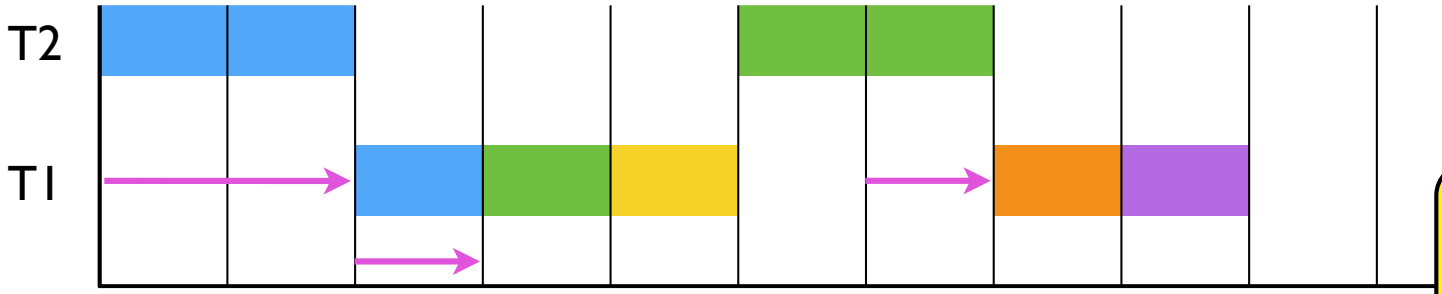
Strict priority order - strikt prioritetsbaserad schemaläggning

- Dynamisk schemaläggning bestämmer online vilken tråd (av de som är flaggad "ready") ska köra
- De flesta system-schemaläggare är baserad på prioriteter (som syns i Java-klasserna)
- Om prioriteterna är fasta, alltså inte ändras efter att tråden har skapats, pratar vi om *fixed priority scheduling*
- Hur sätter man då en tråds prioritet?
  - Trådens prioritet påverkar schemaläggningen, dvs väljer man "fel" kan det bli omöjligt att hitta ett fungerande schema
  - Både för att bestämma prioritet och för att sen kunna garantera en maximal svarstid, måste man analysera värsta fallet - schemaläggningsanalys med WCET.

Prioritetstilldelning

Exempel: T1 exekverar 1ms varje 2ms, T2 exekverar 2ms varje 5ms.
Trådarna måste exekveras färdigt en gång per period
Tidsintervallet vi måste titta på är minsta gemensamma multipeln av perioderna, dvs 2x5 = 10ms

A: T2 får högre prioritet än T1:



T1 får inte köra i sin första period (FEL)

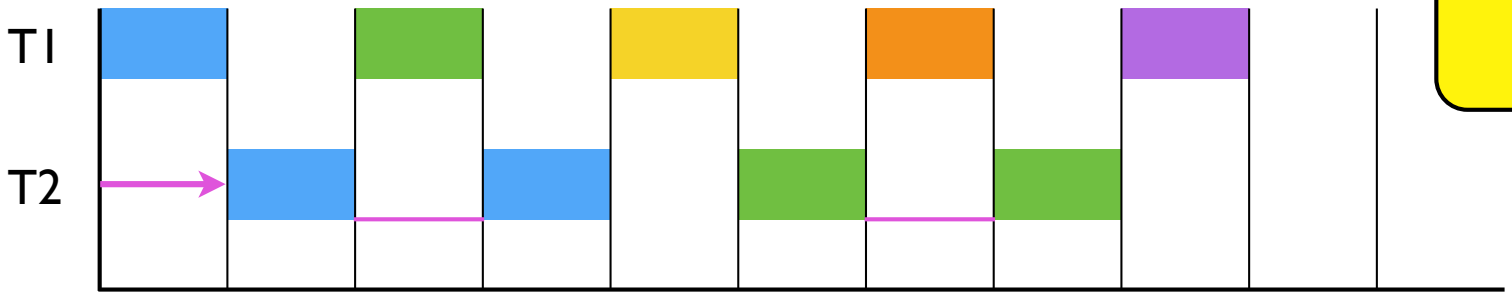
Det fungerar alltså om

högst frekvens

får

högst prioritet

B: T1 får högre prioritet än T2:



T2 blir avbruten av T1 vid 2ms och 6ms, men får ändå köra inom den tänkte 5ms-perioden varje gång

RMS - Rate monotonic scheduling

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

Fråga: Hur bra är denna tumregel? 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

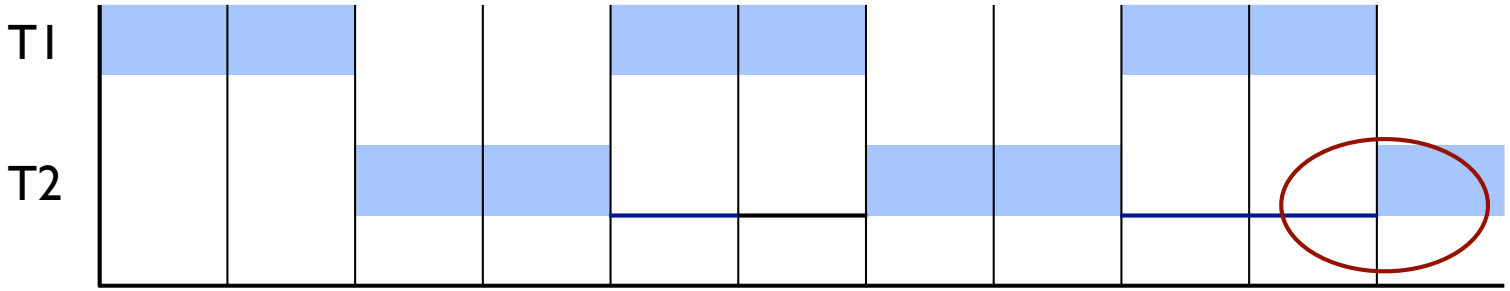
Notation

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

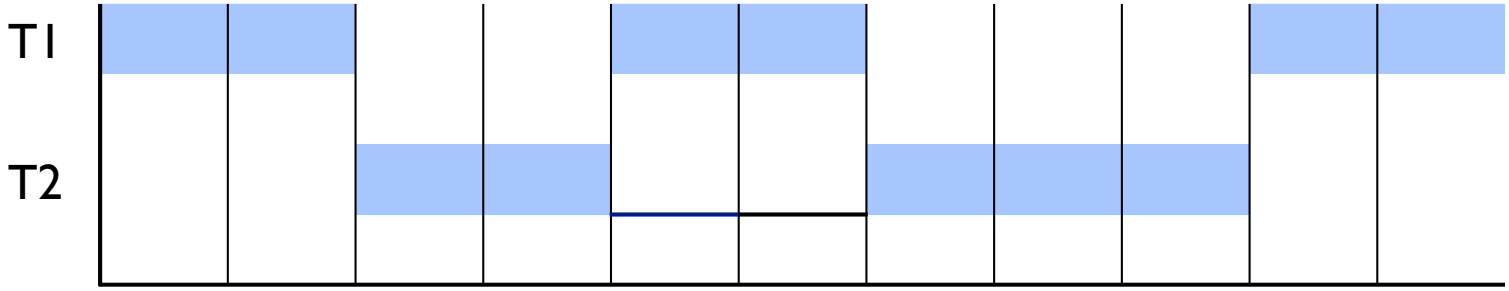
Värsta fallet som fortfarande går att schemalägga för två trådar utan blockering

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

EDF- earliest deadline first

En annat schemaläggningsprincip, som alltid ger CPUn till den tråden som är närmast sin deadline

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



Här kan man få 100% CPU användning,

men: dyrt att implementera samt ger riktigt dåliga resultat när det blir fel någonstans, då kommer alla trådar missa sina deadlines

Svarstider under tillfällig blockering

Trådar i ett realtidssystem bör exekveras med absolut prioritetsbaserad ordning, dvs en tråd med hög prioritet ska inte behöva vänta på en tråd med lägre prioritet under en obestämt tidsperiod

Därför (i ett realtidssystem):

- CPUn tilldelas alltid den högst prioriterade tråden som är "redo" (ready)
- Semaforer och monitorer har prioritetskö

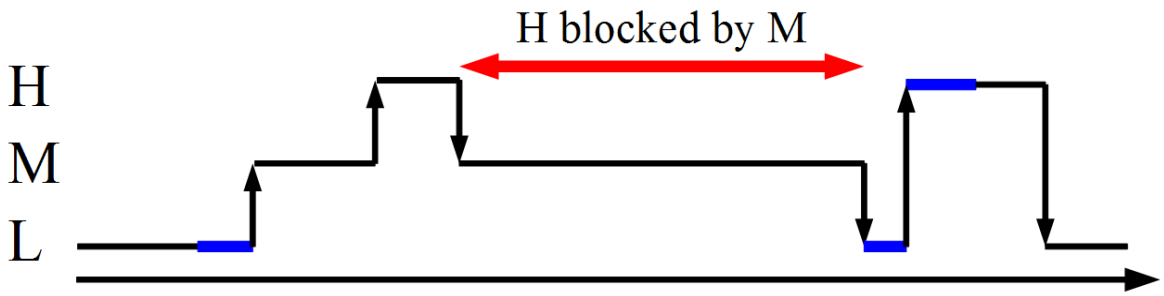
Kan vi då garantera rätt exekveringstid för högt prioriterade trådar utan att känna till exekveringstiden av mindre prioriterade trådar? Vad händer när det blir blockering på delade resurser?

Prioritetsinvertering

Ett problematiskt scenario:

Tre trådar med H(ög), M(edel) och L(åg) prioritet, där H och L delar en resurs.

- L exekveras och kommer in i en kritisk region (kommer in i resursen)
- M avbryter (preemption) och börjar exekvera
- H avbryter och försöker komma in i den delade resursen, som L håller. H är blockerad.
- M fortsätter exekvera under en obestämd tidsperiod, och blockerar både L och H.



Prioritetsarv

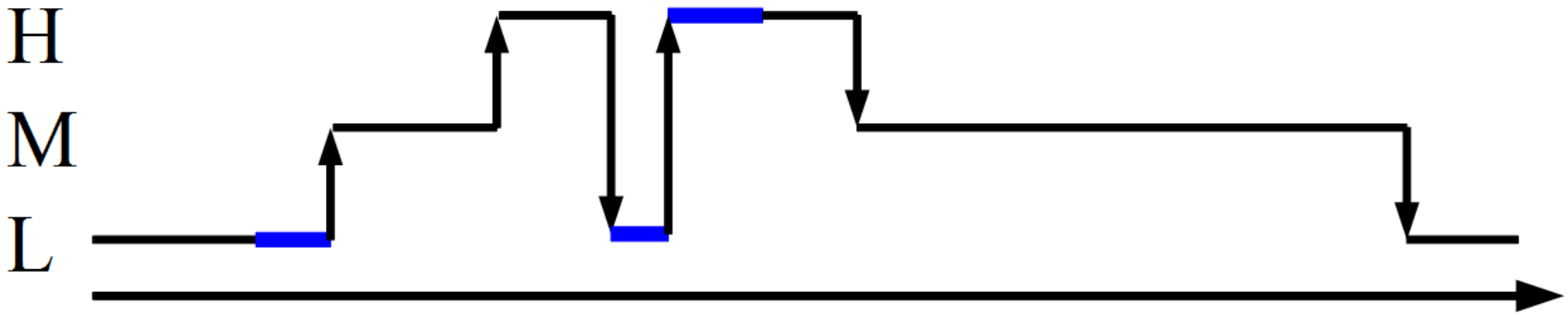
Ett "botemedel" mot prioritetsinvertering är prioritetsarv

Idé: Låt en tråd med låg prioritet som håller i resurser som behövs av högre prioriterade trådar "ärva" den högre prioriteten, så att de blir klara så snart som möjligt och inte blockerar längre.

Det finns olika protokoll som implementerar prioritetsarv:

- Dynamiskt prioritetsarv (basic inheritance protocol)
- Prioritetstak (priority ceiling protocol)
- Direkt prioritetsarv (immediate inheritance protocol)

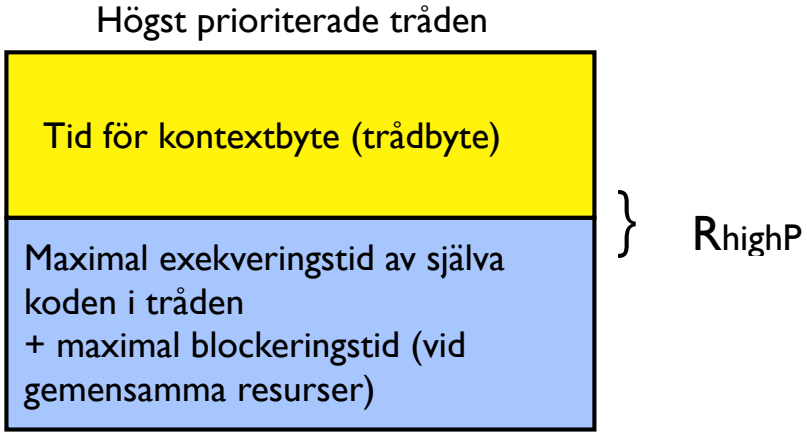
Dynamiskt prioritetsarv

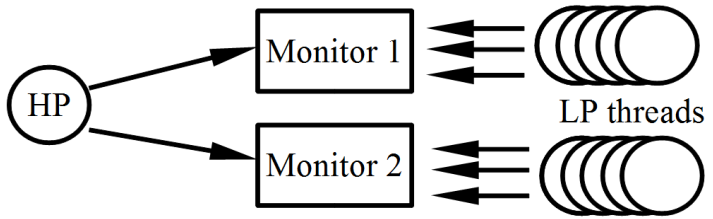


Det "problematiska scenariot" räddat med prioritetsarv (?)

- L exekveras och kommer in i en kritisk region (kommer in i resursen)
- M avbryter (preemption) och börjar exekvera
- H avbryter
- och försöker komma in i den delade resursen, som L håller. H är blockerad. L ärver Hs prioritet och avslutar jobbet i den kritiska sekvensen. L får sen tillbaka "sin egen" prioritet och blir avbruten av H.
- H avslutar och M fortsätter exekveringen
- Både H och M väntar enligt sina tidsperioder. L får avsluta.

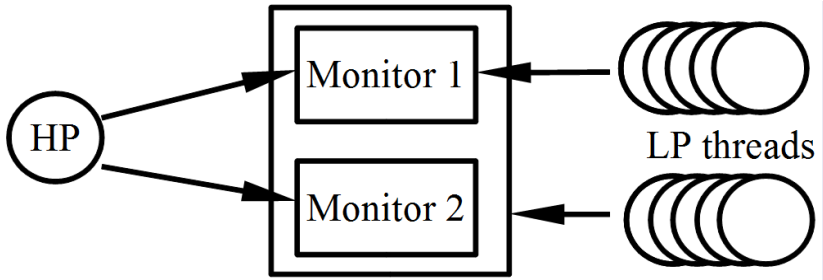
Dynamiskt prioritetsarv - problem





Det kan förekommer multippla blockeringar på olika resurser av flera lägre prioriterade (LP) trådar, som tillfälligt får hög prioritet enligt prioritetsarv. Detta kan man komma förbi med:

- Prioritetstak (priority ceiling protocol): Enbart EN LP-tråd får komma åt någon av resurserna som HP-tråden behöver vid ett tillfälle.
- Direkt prioritetsarv (immediate inheritance protocol): LP-tråden som innehar en kritisk region (sekvens) får alltid den högsta prioriteten som är förknippad med den aktuella resursen.
- Egenskaper: Det blir ett "staket" kring en grupp av resurser, som då också garanterar att det hela är dödlägesfritt. Nackdelen är något högre hanteringskostnader, eftersom man måste hålla reda på trådarnas resursbehov.



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 (kan vara prioritetsbaserad eller "Round-robin")
  - 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!

Sammanfattning

- Introducerat begreppen exekveringstid, svarstid, blockeringstid
- Introducerat schemaläggningsbegreppet och -principer (RMS, EDF)
- Introducerat begreppen prioritetsinvertering och prioritetsarv

- Ska kunna förklara de introducerade begreppen och principerna.

- Lästips:
  - e-bok: del 2, kap 11 + 12, sida 263 - 294