

## **Tentamensskrivning - Realtidsprogrammering** **2005-03-30, kl 8.00-13.00**

### **Anvisningar**

Tillåtna hjälpmedel: inga utöver Java snabbreferens.

DAT040: En uppdelning av betyg 4 för G resp VG kommer att ske, f.ö. se EDA040.

EDA040: *För godkänt betyg* krävs att större delen av de 7 första uppgifterna (teori) *samt uppgiften 8 (den första av de båda konstruktionsuppgifterna)* behandlas *nöjaktigt*. För högre betyg krävs dessutom en acceptabel lösning till uppgift 9 (design), och för högsta betyg ett tillräckligt (c:a 30p) totalt poängantal.

Senast 2005-04-05 anslås på institutionens anslagstavla vilka som deltagit i tentamen men som, enligt institutionens noteringar, ännu inte redovisat övningarna, laborationerna eller projektet. Deras skrivningsresultat anslås eller registreras inte förrän rättelse skett eller dispens erhållits från ansvarig lärare. Rättelse skall göras senast 2005-04-13.

På anslagstavlan anslagen restlista samt resultat av tentamen kommer även att publiceras via kursens hemsida ([www.cs.lth.se/EDA040](http://www.cs.lth.se/EDA040)), i tillämpliga fall lösenordsskyddat (id=eda040, pwd=ht04).

### **Uppgifter**

- 1.** Vad innebär, och vad händer vid, ett kontextbyte (eng: context switch)? (1p)
  
- 2.** I realtidssystem används prioriteter för att begränsa svarstiderna för de viktiga trådarna.
  - a)** Varför skall man inte använda prioriteter för implementering av kritiska regioner (tex genom att tillfälligt under exekvering i den kritiska regionen låta tråden ha en högre prioritet än andra involverade trådar)? (1p)
  - b)** För att bygga stora programvarusystem lär man sin redan i de grundläggande programmeringskurserna att i möjligaste mån hålla data/information lokalt (i funktioner, klasser, paket, etc.). Hur stämmer nyttjandet av prioriteter som grund för fördelning av processortid med denna grundläggande kapslingsprincip? (1p)
  
- 3.** I kursen har begreppet *statisk schemaläggning (static cyclic scheduling)* förekommit.
  - a)** Hur hanterar man exekvering av kod som inte ryms inom en statisk tidslucka och som måste avbrytas av exekvering av andra (snabbare) statistiskt schemalagda aktiviteter? (1p)
  - b)** Under vissa omständigheter kan ett statistiskt schemalagt system ge dåligt processorutnyttjande. Nämn en anledning och hur man kan undvika problemet? (1p)
  
- 4.** I moderna processorer används olika tekniker såsom 'pipelining' och 'caching' (för data respektive instruktioner) för att öka prestanda, t.ex. för skrivbords- och server-tillämpningar. Med prestanda avses då vad vi brukar kall medelprestanda. Vad är problemet med denna typ av högpresterande datorer i realtidssystem med hårda svarstidskrav? (1p)

5. Betrakta ett realtidssystem som är implementerat med hjälp av fyra stycken trådar, A, B, C och D, med karaktäristika enligt nedanstående tabell ( $C$  = maximal exekveringstid/period och  $T$  = periodtid):

Tråd	$C$ (ms)	$T$ (ms)
A	4	20
B	5	32
C	3	12
D	2	8

För samtliga trådar gäller även att deadline är lika med periodtiden. Vi antar vidare att trådarna schemaläggs enligt principen RMS (Rate Monotonic Scheduling) och att trådarna är helt oberoende av varandra, dvs de kommunicerar inte med varandra och kan således inte blockera varandra (de kan dock naturligtvis avbryta varandra genom s.k. preemption - påtvungad tidsdelning). Vi bortser också från kostnaden för trådbyten et cetera. Systemet exekverar på en styrdator med en enda CPU.

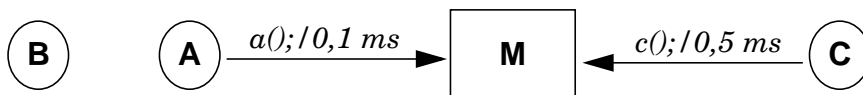
a) Vad blir värstafallssvarstiden för var och en av de fyra trådarna? (3p)

b) Antag att vi byter ut styrdatorn i uppgiften mot en dator med två processorer (CPU:er), och fördelar trådarna mellan processorerna så att CPU nummer 1 enbart ägnar sig åt att exekvera tråd D ovan medan de övriga trådarna (A, B och C) får dela på CPU nummer 2. De två processorerna anses exekvera helt parallellt utan att dela på några hårdvaruresurser. Vi får alltså ur analysynpunkt två helt oberoende realtidssystem. Visa hur man på ett enkelt sätt kan övertyga sig om att alla de totalt fyra trådarna på de två processorerna kommer att klara sina deadlines utan att beräkna/känna till några värstafallssvarstider för de individuella trådarna. (Ledning:  $5/32 < 0,157$ ) (2p)

6. Ett realtidssystem är implementerat med hjälp av tre periodiska trådar kallade A, B och C. Tråd A har högst prioritet, B näst högst prioritet och C har lägst prioritet. Trådarnas värstafallsexekveringstid ( $C$ ), period ( $T$ ) och deadline ( $D$ ) ges av följande tabell:

Tråd	$C$ (ms)	$T$ (ms)	$D$ (ms)
A	2	40	40
B	3	50	50
C	4	60	60

Tråd A kommunicerar med tråd C via en monitor kallad M. Varje gång A respektive C körs anropar de en monitoroperation (en och endast en gång per period) enligt nedanstående figur. Tråd B kommunicerar inte med någon annan tråd. Den maximala exekveringstiden för de olika monitoroperationerna är angiven i figuren.



a) Antag att systemet schemalägger trådarna enligt strikt prioriterad schemaläggning med dynamiskt prioritetsarv (basic priority inheritance protocol). Vad blir värstafallssvarstiden för var och en av de tre trådarna? (2p)

b) Antag att systemet ovan schemalägger trådarna enligt strikt prioritetsbaserad schemaläggning men *utan någon form av prioritetsarv*. Hur lång tid kan i så fall var och en av de tre trådarna bli fördröjda av lägre prioriterade trådar i värsta fall? (2p)

7. I ett Javaprogram hittar vi tre typer av trådar, T1, T2 och T3, som i sina respektive `run()`-metoder exekverar följande linjära sekvenser av semaforoperationer på de fem mutex-semaforerna A, B, C, D och E (mellanliggande kod som är beroende av semaforerna representeras av funktionsanropen på formen `useXY()`; där `XY` avser att semaforerna X och Y måste vara tagna när koden utförs):

<b>T1</b>	<b>T2</b>	<b>T3</b>
<code>A.take();</code>	<code>D.take();</code>	<code>C.take();</code>
<code>E.take();</code>	<code>E.take();</code>	<code>D.take();</code>
<code>useAE();</code>	<code>useDE();</code>	<code>B.take();</code>
<code>E.give();</code>	<code>E.give();</code>	<code>useBCD();</code>
<code>A.give();</code>	<code>D.give();</code>	<code>B.give();</code>
<code>...</code>	<code>...</code>	<code>D.give();</code>
<code>E.take();</code>	<code>B.take();</code>	<code>C.give();</code>
<code>A.take();</code>	<code>A.take();</code>	
<code>useAE();</code>	<code>useAB();</code>	
<code>A.give();</code>	<code>A.give();</code>	
<code>E.give();</code>	<code>B.give();</code>	

a) Rita en resursallokeringsgraf för systemet. (1,5p)

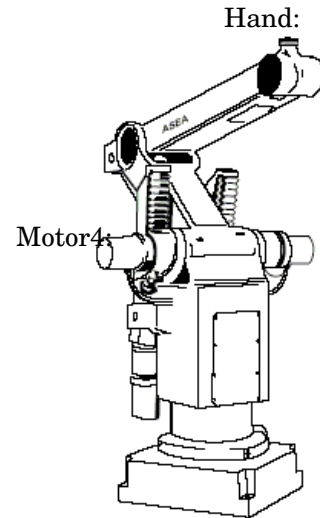
b) Programmet ovan riskerar att råka ut för dödläge. Hur kan man se det i resursallokeringsgraf? (1p)

c) Föreslå en ändring i programmet ovan som gör att risken för dödläge elimineras utan att programmets synkroniseringsbehov riskeras. Dvs, när en funktion på formen `useXY()`; anropas måste motsvarande semaforer vara tagna. (0,5p)

## 8. Börvärden för motorstyrning

I en robotarm finns ett antal leder, där för varje led rörelsen sker genom att ett en motorvinkel regleras till önskad vinkel/position. Normalt finns 5 eller 6 sådana leder för en robotarm, exempelvis styrs handledsvinkeln via inre mekanik av motorn Motor4 enligt figur.

För att robothanden (som monteras på handfästet vid Hand enligt figur) skall utföra önskad rörelse, exempelvis en rät linje i rummet, så beräknas i den färdiga delen av styrsystemet uppnåeliga börvärden till respektive motor. Dessa börvärden representerar steg längs den önskade banan, med en steglängd motsvarande 20ms i tid. Dessa steg utgör målvärden (kallade target nedan) och kan inte beräknas tätare i tiden eftersom varje värden kräver omfattande dynamiska modeller och beräkningar. Dock behöver regleringen av respektive motor ske med en kortare periodtid, som vi i det följande antar är 4ms. För att då inte introducera ryck och vibrationer i robotarmen behöver



1. Varje 20ms steg interpoleras till mindre steg motsvarande 4ms
2. Önskat hastighetsbörvärde skall beräknas, motsvarande lägesändring per tidsenhet.

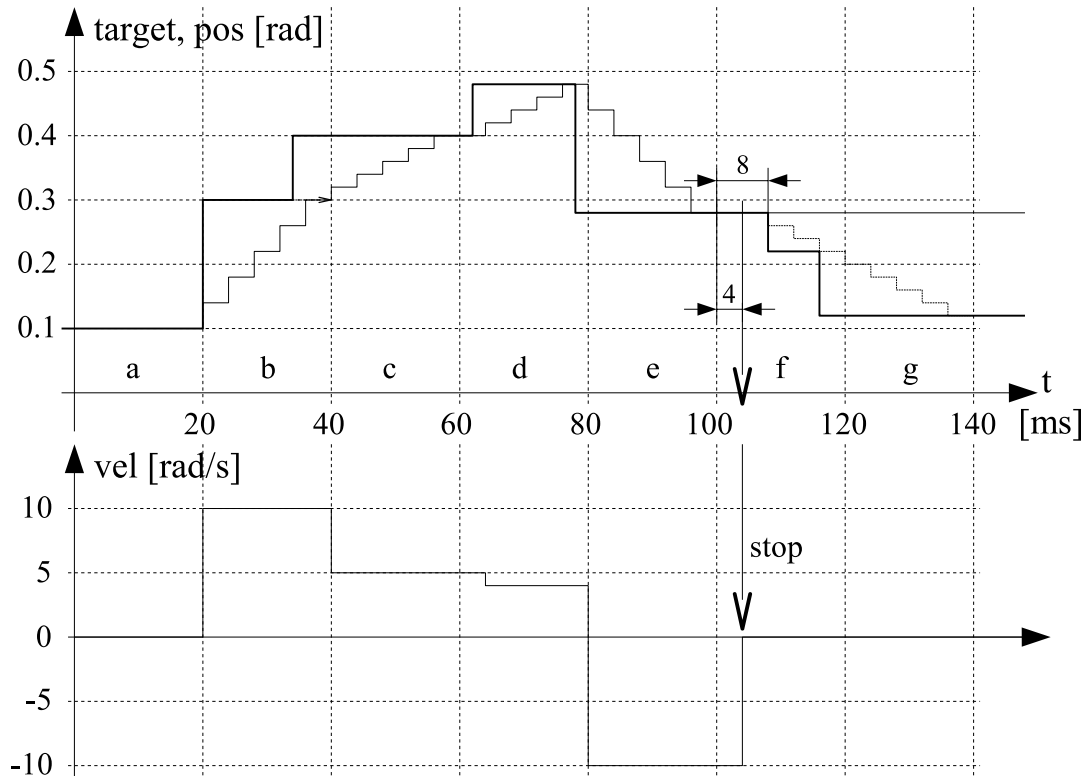
Vi begränsar oss nu till att betrakta en motor, och vi studerar nu endast den mycket avgränsade delen av systemet som åstadkommer punkterna 1 och 2 ovan. Eftersom övriga delar av systemet är färdiga så vet vi ganska väl hur vår kombinerade interpolering och derivering skall ske. Vi vet också att tillhandahållandet av target-värden med perioden 20ms sker asynkront med regleringen som behöver börvärden med periodtiden 4ms. Vi skall således åstadkomma kopplingen mellan två aktiviteter/trådar, vilket innebär att ett trådsäkert passivt objekt skall implementeras; vår design är klar så när som på att vi kan bestämma eller ändra vilka argument våra metoder skall ha.

Du skall utforma och implementerar en klass som vi kan kalla SetpointMonitor. Om vi antar att vi har en instans ipol av denna så skall vi genom att mata objectet med target-värden



erhålla interpolerade positionsvärden (pos), deriverade hastighetsvärden (vel), samt en stop-signal om någon deadline överskrids. Följande figur illustrerar med ett exempel hur signalerna beräknas, där vi vid de olika perioderna a till f har följande förlopp:

- Under period a, före 20ms, har ingen rörelse beordrats. Motorn står still med positionen/vinkeln 0.1 rad.
- Efter 20ms sätts ett nytt target-värde på 0.3 rad, tillsammans med ett tidsvärde på 40ms, vilket anger att target-värdet skall nås efter 40ms. När är target-värde sätts finns inga antaganden eller kunskap om vilken periodtid som interpoleringen kommer att ske för, men vi vet att utperioden (för pos) alltid är kortare än inperioden (för target).
- Under period b, medan interpolerade positioner erhålles av regulatortråden med en period av 4ms, så sättes ett nytt target-värde på 0.4 rad vid tiden 34 men med angivelse att värdet skall gälla från tiden 40.
- Interpoleringen fortsätter under period c. Nästa target-värde, som skall gälla från tiden 60, skulle egentligen satts under denna period men tillhandahålles efter den aktuella tiden (en mindre och tolererad avvikelse från tidskraven).



- Intervall d startar utan att ett nytt målvärde har satts (att nås vid tiden 80). Vi tolererar dock en så mindre fördröjning (max 4ms) så befintliga värden för pos och vel behålles vid periodens början. Egentligen skulle då hastigheten sättas till noll eftersom positionen är konstant, men även hastigheten fryses för att undvika onödiga transienter för det fall att ett nytt target-värde strax inkommer. Detta sker vid tiden 62 och interpoleringen fortsätter mot den nya målpositionen 0.48, med uppdaterat hastighetsvärde från tiden 64.
- Vid tiden 78 sätts ett nytt target-värde som skall uppnås vid tiden 100. Detta nya värde är mindre så nu blir tidsderivatan (hastigheten) från tiden 80 mindre än noll.
- Vid tiden 108 tillhandahålles nästa target-värde, som skulle verka redan från tiden 100 för att nås vid tiden 120. Detta värde kommer alltför sent eftersom den maximala tillåtna fördröjningen är 4ms. Vid tiden 104, i samband med att regulatortråden erhåller sin positionsreferens, detekteras denna överskridna maximala svarstid (deadline), och objektet ipol övergår till tillståndet (nöd)stopp. Detta innebär att positionsbörvärdet hålles kvar konstant på aktuellt värde och den önskade hastigheten sätts till noll.

Om tidskravet vid tiden 104 inte hade missats så skulle interpoleringen fortsatt längs den prickade linjen. Din uppgift är att utveckla den saknade programkoden i (eller kring) följande klass.

```
public class SetpointMonitor {

    /**
     * Maximum permitted delay of new target position. If setTarget
     * is called more than maxDelay after the time when the target
     * should be reached, this is an emergency and the motion should
     * be stopped.
     */
    public static final long maxDelay = 4;
```

```

// Your private attributes go here.....

/**
 * Provide a new target position (motor angle) and the time when that
 * position should be reached (in terms of the setpoint to the motion
 * control as returned by getPos). After the setting of one target (by
 * calling this method) and the target time, the next target (with a
 * new time) has to be set (again, by calling this method) within the
 * current target time plus the maxDelay value (in ms). Failure to
 * meet this deadline (such as the 8ms being larger than the 4ms in
 * the figure/graph of the problem description) will result in a stop
 * of the motion (equivalent to calling emergency with a true argument)
 * and false is returned from this call.
 *
 * If a motion is known in advance, the thread computing the target
 * positions (along the desired path) can compute the next target and
 * call this method in advance. However, only one extra/pending target
 * can be provided; additional calls of this method will block until
 * any pending target position has been taken care of (by the feedback
 * control calling getPos up to the previous target time).
 *
 * @param youDecide what arguments you want to have.....
 * @return the success of the setting of the new target. That is, targets
 *         provided too late or during stop/emergency results in false.
 */

public /* you choice */ setTarget(What youDecide) {
    // This is for you to implement.
}

/**
 * Obtain the commanded position to be applied at time t. The calling
 * thread (which typically is the one with the highest priority in
 * the system) is responsible for calling this method frequently
 * enough and before time t is passed. That is, the timing of the
 * thread performing feedback control is checked elsewhere, and this
 * method simply computes the desired position by interpolating
 * between the target positions. Furthermore, if the feedback control
 * would advance slower in time than expected (supplied argument t for
 * each call increases less than the real time), the calls of setTarget
 * would block longer and delay the setting of new targets, but the
 * obtained position would still be valid for the time supplied.
 *
 * @param t the time for which the computed position should apply.
 * @return the computed position to be used as the control setpoint.
 */

public /* your choice */ double getPos(long t) {
    // You also write this method
}

/**
 * Get desired velocity, determined as change of desired position
 * per time unit, which in turn is the change of target position
 * per time unit. If the target position is updated late, but before
 * stop is commanded, the obtained velocity remains unchanged (see
 * time 60-62 in the figure/graph of the problem description).
 *
 * @return the desired speed in motor-radians per second.
 */

public /* your choice */ double getVel() {
    // Another method to develop for you...
}

```

```

/**
 * Emergency stop of motion, or reset of emergency state to continue
 * after a previous stop. This method is available to all parts of
 * the system (e.g. to the operator interface) but may also be
 * called internally if deadlines are missed.
 *
 * @param on If true, stop motion by setting target position to
 *           current position, and notify any caller of awaitStop.
 *           If false, block until the velocity of the commanded
 *           motion is zero, and then permit new motions.
 */
public /* your choice */ void emergency(boolean on) {
    // Another method to develop for you...
}

/**
 * Block caller until there is an emergency stop, due to a call of
 * the emergency method and/or a missed deadline. To avoid excessive
 * context switches, considering that unblocking should be unusual,
 * this method is not synchronized. Instead, it is made thread safe
 * by synchronizing on a an internal object specifically maintained
 * for stopping only.
 */
public void awaitStop() {
    // The last one to develop...
}

// Private stuff, classes and methods, whatever you decide...
}

```

Således, skriv klart klassen SetpointMonitor så att egenskaperna enligt befintliga kommentarer implementeras. Om ytterligare klasser behövs/önskas så kan dessa läggas till som inre klasser alternativt som yttre extra klasser om så önskas.

(9p)

## 9. Tunnelövervakning

En principiell konstruktionsutformning (design) av ett övervakningssystem för en biltunnel behöver utföras för projektering och ev beslut om att gå vidare med implementering och installation. En motsvarande produkt, som kanske kan kompletteras med det SDK som kan köpas till, finns exempelvis på <http://www.autoscope.com> där informationen till höger har hämtats. Klarlägg med figurer och text vilka trådar, resurser, kommunikationskanaler, etc. som behövs för ett system som uppfyller följande specifikationer:

1. En tunnel med dubbelriktad biltrafik skall övervakas i båda ändarna av tunneln.
2. Vid vardera tunnelmynning finns en kameraservert som har till uppgift att:
  - a) Med 1Hz skicka bilder till central operatörsdator.
  - b) Med 5Hz detektera om det förekommer någon rörelse i bilden.
  - c) Vid rörelse fånga 25Hz bildsekvenser med längden 1 sekund.
  - d) För varje fångad bildsekvens urskilja förekommande bilnummer och för detekterat fordon beräkna riktning och hastighet.
  - e) För varje fordon, som kan ha detekterats ur flera sekvenser, skall dess hastighet och riktning skickas till central dator.
  - f) Om fordon eller rörelse detekteras men inget reg.nr., så skickas bildsekvensen till central operatör (vid central dator).
3. Kommunikation mellan olika datorer stöder skickandet av hela objekt. All läsning av objekt från nätverket är blockerande.
4. Operatörsdatorn finns på central plats; inte vid någon av tunnelmynningarna.
5. Ett operatörsgränssnitt visar uppdaterade stillbilder med frekvensen 1Hz (idle), inkomna bildsekvenser, samt detekterade problemfordon enligt följande punkt.
6. Alla inkomna reg.nr. (med plats och riktning) skickas till en separat tråd i operatörsdatorn. Denna tråd eliminerar bilar som passerat genom tunneln på normalt sätt. Resterade fordon klassas med ett eller flera av attributen: Fortkörare, U-svängare, Bortkommen, Nyskapad. Resultatet loggas i fil och visas i användargränssnittet.

Utför design av systemet, men ingen implementering. Algoritmer antas finnas i färdiga klasser (som du själv kan namnge). (9p)

### **Autoscope Video Detection** (product example)

In recent years, a number of aboveground technologies have emerged to complement or replace inground inductive loops, which are expensive to reconfigure, have limited capabilities, and fail frequently. These new technologies include video detection, radar, ultrasonic, infrared and laser. Of these, video detection has been the most successful, providing unsurpassed richness of data as well as video images, wider coverage areas and greater versatility for demanding applications. To determine the best technology for your applications, ask yourself these questions:

- Do you want wide area detection? Accuracy in measuring vehicle counts and speed?
- Do you want to detect stopped vehicles? Congestion? Vehicles going the wrong way?
- Do you want to be able to reconfigure the detectors easily during road construction or to reflect changes in road geometry?
- Do you want "snapshots" of traffic?

Video detection is the only technology that can provide all these functions and benefits. Video detection is now more cost-effective, accurate and reliable than ever, outperforming inground inductive loops and other aboveground detection technologies under all weather and light conditions. The Autoscope system is the leader in video vehicle detection.

### **Tunnels**

Tunnels are safer than ever, thanks to video-based incident detection and traffic management. Autoscope systems can quickly signal an alarm when incidents are detected, enabling fast response by an operator. Autoscope systems are proving their unique value to this challenging environment by detecting: Stopped vehicles, Slow traffic, slow vehicles, Wrong way vehicles, etc.

Autoscope is being utilized with SCADA (Supervisory Control And Data Acquisition) for incident management in the Clyde Tunnel. In 1992, local authorities in Glasgow decided to adopt the new European tunnel guidelines for a highly driven tunnel and included Autoscope in their refurbishing program.

[[www.autoscope.com](http://www.autoscope.com)]