

Tentamen

Realtidsprogrammering

2007-04-11, 8.00-13.00

Hjälpmedel: inga utöver Java snabbreferens och miniräknare.

DAT040: En uppdelning av betyg 4 för G respektive VG kommer att ske, för övrigt se EDA040.

EDA040: För *godkänt betyg* krävs att större delen av de 7 första uppgifterna (teori) *samt uppgiften 8 (programmering) behandlas nöjaktigt*. Poäng från lösning av uppgift 9 och 10 bidrar till högre betyg.

1. Vad menas med ömsesidig uteslutning?

(1p)

2. Det finns flera varianter på definition av en semafor som skiljer sig åt genom hur de väljer vilken tråd att starta vid ett anrop av `give()`. Hur skall lämpligen en semafor avsedd för realtidssystem vara definierad?

(1p)

3. Antag att vi beträffande korrekthet för realtidsprogram med jämlöpande exekvering definierar följande nivåer

0. Felaktigt program.
1. Logiskt korrekt.
2. Jämlöpande korrekt.
3. Realtidsmässigt korrekt.

där korrekthet på en nivå förutsätter korrekthet på lägre nivå. Dvs nivå 3 kräver att nivå 1 och 2 är uppfyllda, och nivå 2 kräver att nivå 1 är uppfyllt. Om inte ens nivå 1 är uppfyllt så är programmet felaktigt även i traditionell sekventiell mening och befinner sig på nivå 0. Logiskt korrekthet gäller således således programmets sekventiella delar utan hänsyn taget till parallellitet och tidskrav.

Inför drifttagning av ett större reglersystem testas ett antal olika programvaror vilka (via konfigurationsfiler) tillåter att interna tråders prioritet ändras. Ange för vart och ett av fallen a tom c vilken grad av korrekthet systemet uppfyller.

- a) Oberoende av prioritetssättning så beräknas en numeriskt korrekt styrsignal, och med systemets ordinarie prioritetsval beräknas värdena i tid, men med vissa val av prioriteter kommer uppdateringen av styrsignalen för sent.
- b) För vissa val av prioriteter blir styrsignalen både för sent beräknad och får fel numeriskt värde. Med ordinarie prioriteter blir dock styrsignalen korrekt för alla val av indata.
- c) För en av systemets övervakningsfunktioner sker samtliga beräkningar sekventiellt inom en tråd, men för vissa sekvenser av indata erhålls numeriskt felaktiga utdata.

(3p)

4. Förklara begreppen *prioritetsinversion* respektive *prioritetsarv*. Vilka är konsekvenserna i de båda fallen (prioritetsinversion och prioritetsarv) för en högprioriterad tråds förmåga att uppfylla ställda tidskrav?

(3p)

5. Metoderna i en klass måste vara *trådsäkra*, eller s.k. *reentrant*, för att kunna anropas från flera trådar samtidigt.

a) Nämn två typer av programmeringsfel som gör att en metod inte blir *reentrant*, dvs inte trådsäker eller återanropningsbar.

(1p)

b) Varför behöver exempelvis funktionerna/metoderna `sin()` och `cos()` i standardklassen `Math` inte vara deklarerad `synchronized`, trots att de kan anropas från många trådar parallellt?

(1p)

6. 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 - påtvingad 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)

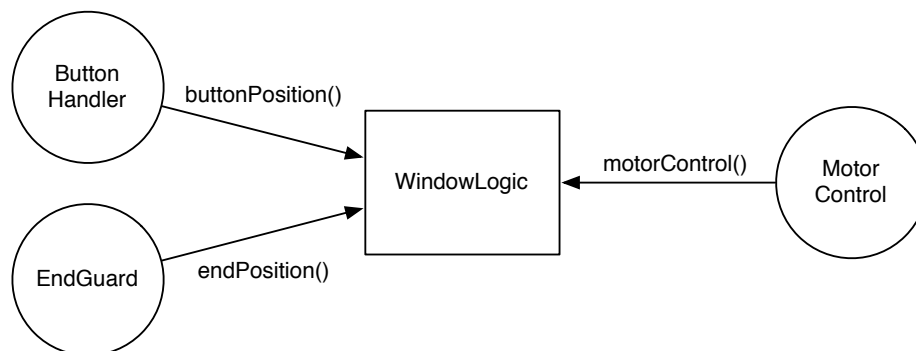
7. Vilka fyra villkor måste vara uppfyllda för att dödläge (deadlock) skall kunna uppträda i ett system? Vilket av dessa krav är det som vi vid realtidsprogrammering normalt försöker att inte uppfylla? Varför vill vi inte förbjuda att övriga villkor är uppfyllda?

(3p)

8. Elektrisk fönsterhiss

Den elektriska fönsterhissen i en bil styrs med en återfjädrande knapp med tre lägen: *upp*, *neutralt* respektive *ner*. Normalt vevas fönstret ner så länge man håller *ner* intryckt, och upp när man håller *upp* nertryckt. Om knappen hålls i läge *ner* mer än 3 sekunder inträder en automatisk funktion. Fönstret fortsätter då sin rörelse nedåt tills fönstret är helt öppet även om knappen släpps. Hastigheten på fönstret ökar dessutom för att markera att hissen gått över i automatläge. Denna automatiska rörelse kan stoppas genom att man trycker på *upp*, varvid fönstret åter vevas upp tills knappen släpps (eller fönstret nått sitt övre läge). Motorn får inte startas om fönstret är i sitt övre läge och man trycker på *upp*, och inte heller om fönstret är i sitt nedre läge och man trycker på *ner*. På samma sätt måste motorn stoppas när fönstret når något av sina ytterlägen.

För att styra motorn har man beslutat att använda sig av en monitor WindowLogic som anropas av tre stycken trådar enligt nedanstående figur.



Tråden ButtonHandler bevakar knappen och informerar WindowLogic om varje ändring i knappens läge. EndGuard känner av om fönstret når något av sina ytterlägen och informerar då WindowLogic. Baserat på informationen från ButtonHandler och EndGuard tar monitorn WindowLogic beslut om hur fönsterhissens motor ska styras och låter tråden MotorControl, som sköter själva styrningen, ta del av dessa beslut.

Monitorn WindowLogic ska ha följande gränssnitt:

```

public class WindowLogic {

    public static final int UP = 1;
    public static final int STOP = 0;
    public static final int NEUTRAL = 0;
    public static final int DOWN = -1;
    public static final int FAST_DOWN = -2;

    /**
     * Constructor
     */
    public WindowLogic();

    /**
     * To be called when the state of the control button changes.
     * @param state the new state of the button: UP, NEUTRAL, or DOWN
     */
    public synchronized void buttonPosition(int state);

    /**
     * To be called when the window reaches one of its endpoints.
     * @param endpoint the endpoint reached: UP or DOWN
     */
    public synchronized void endPosition(int endpoint);
  }
  
```

```
/**
 * The calling thread is blocked until the status of the motor needs to
 * change.
 * @return the new motor status: UP, STOP, DOWN, or FAST_DOWN
 */
public synchronized int motorControl();

}
```

Uppgift

Implementera monitorn `WindowLogic` tillsammans med eventuella extra hjälptrådar du kan behöva för att monitorn ska uppföra sig enligt beskrivningen ovan. Inför extra hjälpmetoder i monitorn efter behov. Du ska *inte* implementera trådarna `ButtonHandler`, `EndGuard` och `MotorControl`.

Ledning: Det kan vara lämpligt att tänka sig monitorn `WindowLogic` som en tillståndsmaskin som byter tillstånd när olika insignaler anländer.

(10p)

9. Synkron broadcast-buffert

I ett programsystem har man behov av att skicka samma meddelande från en producent till ett antal konsumenter. Det finns ett fixt antal konsumenter, numrerade 1..N. För att undvika kopiering av meddelanden och se till att konsumenterna hanterar meddelandena synkront vill man använda en monitor som sköter buffring av meddelanden och synkroniseringen av konsumenterna.

Monitorn ska ha följande gränssnitt:

```
public class BroadcastBuffer {

    /**
     * Constructor.
     * @param n the number of consumers
     * @param size the maximum number of messages the buffer can hold
     */
    public BroadcastBuffer(int n,int size);

    /**
     * Deposits a message in the buffer. Messages can be of arbitrary object
     * type. The method blocks the calling thread only if the buffer is full.
     * @param message the message to deposit
     */
    public synchronized void deposit(Object message);

    /**
     * Retrieves a message from the buffer. The method blocks the calling thread
     * if the buffer is empty or if not all other consumers have yet fetched the
     * last message fetched by the calling thread.
     * @param id the number of the calling consumer (1..N)
     * @return a reference to the next message
     */
    public synchronized Object retrieve(int id);

}
```

Producenttråden anropar metoden `deposit()` för att lägga in nya meddelanden i bufferten. Producenttråden blockeras endast om bufferten är full. De N stycken konsumenttrådarna anropar `retrieve()` för att hämta meddelanden. Ett och samma meddelande ska hämtas av alla konsumenter. En konsumenttråd kan blockeras i anropet av `retrieve()` av två olika orsaker. Den första orsaken är att bufferten är tom och inga meddelanden finns att hämta. Tråden ska också blockeras om den försöker hämta nästa meddelande innan samtliga andra konsumenter har hämtat det föregående meddelandet. På detta sätt kommer konsumenterna att arbeta synkront.

Uppgift

Implementera monitorn `BroadcastBuffert`.

(5p)

10. Temperaturhållning

Denna uppgift går ut på att skriva den del av ett styrprogram som sköter temperaturhållningen i en industriprocess. Temperaturen kan höjas med hjälp av en mikrovågskälla medan den kan sänkas endast genom avsvälning. Vid normalläge skall temperaturen hållas i ett givet intervall varvid gränserna inte får över- respektive underskridas (annat än under uppstart av processen). För övervakning och styrning finns följande statistiska metoder tillgängliga:

```
public class ProcessControl {  
  
    /**  
     * Returns the current temperature (degrees Celsius).  
     * @return the current temperature  
     */  
    public static double readTemp();  
  
    /**  
     * Switch the microwave source on or off.  
     * @param on true if the microwave source should be switched on, false to switch off  
     */  
    public static void heating(boolean on);  
  
    /**  
     * Returns the target temperature.  
     * @return the target temperature  
     */  
    public static double targetTemp();  
  
}
```

Uppgift

Skriv en tråd som styr temperaturen. Temperaturen får inte överskrida den som returneras av `targetTemp()`, men inte heller underskrida en temperatur som är 5°C lägre. När uppvärmningen är påslagen stiger temperaturen med 5°C per sekund och när den är avstängd sjunker den med 1°C per sekund. Tröghet i systemet och temperaturmätningen som sådan medför dessutom ett mindre fel, men vi väljer att bortse från det i uppgiften.

Lösningen måste vara effektiv i meningen att reglering inte ska ske oftare än vad som krävs för att hålla temperaturen inom intervallet. Vidare ska värmen sättas av/på så sällan som möjligt för att inte slita i onödan på det mekaniska relä som bryter/sluter strömmen till mikrovågskällan. Gör en rimlig avvägning mellan samplingstid och maximalt utnyttjande av temperaturintervallet. För att få helt godkänt på uppgiften krävs en motivering till varför din lösning uppfyller kraven ovan.

(3p)