LUNDS TEKNISKA HÖGSKOLA
Datavetenskap

Realtidsprogrammering : EDA040
DAT040

Tentamensskrivning - Realtidsprogrammering
2006-04-20, 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 8 första uppgifterna (teori) *samt uppgiften 9 (den första av de båda konstruktionsuppgifterna) behandlas nöjaktigt*. För högre betyg krävs dessutom en acceptabel lösning till uppgift 10 (design), och för högsta betyg ett tillräckligt (c:a 30p) totalt poängantal.

Senast 2006-04-25 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 2006-05-05.

Uppgifter

1. Vad är ömsesidig uteslutning? Ge ett exempel på en situation som kräver ömsesidig uteslutning. (1p)

2. Vilka fyra villkor brukar man säga måste vara uppfyllda för att det ska kunna uppstå dödläge (*deadlock*) mellan trådar som synkroniseras med hjälp av mutexsemaforer? (2p)

3. I samband med schemaläggningsanalys brukar man prata om det så kallade “kritiska ögonblicket” (*critical instant*). Vad menas med detta? (1p)

4. Under vissa förutsättningar brukar man säga att rate monotonic scheduling (RMS) är en “optimal” schemaläggningsmetod.

a) Vad menar man med att rate monotonic scheduling är optimal? (1p)

b) Nämn två förutsättningar med avseende på realtidssystemet för att rate monotonic scheduling ska vara optimal. (1p)

5. Ibland används prioritetsärvningsprotokollet “priority ceiling protocol” i stället för det vanligare “basic inheritance protocol”. Nämn en fördel med priority ceiling protocol jämfört med basic inheritance protocol. (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	25
B	2	5
C	4	16
D	3	30

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

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 mutexsemaforerna 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):

T1	**T2**	**T3**
`B.take();`	`C.take();`	`B.take();`
`C.take();`	`A.take();`	`A.take();`
`useBC();`	`E.take();`	`useAB();`
`C.give();`	`useACE();`	`B.give();`
`B.give();`	`E.give();`	`D.take();`
`...`	`A.give();`	`useAD();`
`E.take();`	`C.give();`	`D.give();`
`B.take();`		`A.give();`
`useBE();`		
`B.give();`		
`E.give();`		

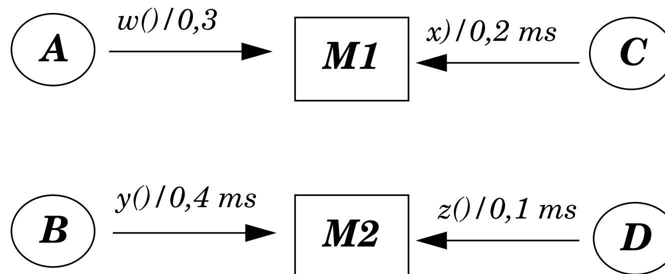
Observera att vi kan ha ett godtyckligt antal instanser av varje trådtyp.

a) Rita en resursallokeringsgraf för systemet. (2p)

b) Finns det risk för dödläge i systemet? Motivera ditt svar. (1p)

c) Föreslå ändringar i programmet ovan som gör att risken för dödläge elimineras utan att programmets synkroniseringsbehov förändras. Dvs, när en funktion på formen "useXY();", anropas måste motsvarande semaforer vara tagna. (2p)

8. Ett realtidssystem är implementerat med hjälp av fyra trådar - A, B, C och D - som kommunicerar med varandra med hjälp av två monitorer (för att åstadkomma ömsesidig uteslutning) - M1 och M2 enligt nedanstående figur.



Monitoroperationerna `w`, `x`, `y` och `z` anropas av trådarna A, B, C och D enligt figuren en gång varje respektive tråds period. För varje monitoroperation anges även den maximala tiden det kan ta att exekvera operationen. Tråden A har högst prioritet, B näst högst prioritet och D har lägst prioritet. Vi förutsätter att trådarna schemaläggs med hjälp av rate monotonic scheduling och dynamisk prioritetsärvning (*basic inhertance protocol*).

Vad blir den maximala tiden som var och en av trådarna A, B, C och D blockeras av *lägre prioriterade trådar*? (3p)

9. Redundant kommunikation

Bakgrund: Vid implementation av feltoleranta system behövs kommunikation med redundanta kanaler. För prototyputveckling och simulering av sådana system vill vi utveckla en speciell version av `RTEventBuffer`, utökad så att varje meddelande ska tas emot av två trådar. Om en av de mottagande trådarna inte hämtat ett meddelande inom en viss tid ska detta detekteras och hanteras. En möjlig lösning vore naturligtvis att faktiskt använda två stycken buffertar och att alltid låta producent-trådarna skicka meddelanden till båda.

Detta är emellertid ingen bra lösning. Den existerande applikationen innehåller kod som skickar meddelanden till en `RTEventBuffer` som fåtts utifrån. För att slippa göra ändringar i sådan kod måste den redundanta kommunikationen vara transparent för producent-trådar. Att ha två separata buffertar skulle även göra implementationen komplicerad, eftersom ett meddelande ska ligga kvar i båda buffertarna tills det hämtats av båda konsument-trådarna (eller, om för lång tid gått, ska meddelandet tas bort ur båda buffertarna).

Den bifogade klassen `BufferPlain` ingår i den aktuella versionen av våra stödklasser för Java-baserad realtid. Det är en förenkad version av dess basklass, `RTEventBuffer`, som använder de attribut som listas i början av klassen `BufferPlain`. Den har även metoder som är `synchronized`, och `wait/notify` används som i en vanlig Java-monitor. I denna uppgiften får du begränsa problemet till att bara behandla metoderna `doPost` och `doFetch`, och timeout-värdet för de redundanta anropen till `doFetch` (som förklaras nedan och inte ska förväxlas med den `timeout` som används i `try`-metoderna) kan antas vara 200ms.

Uppgift: Din uppgift är att designa och implementera en sådan buffert med två kanaler, med egenskapen att ett meddelande ligger kvar i bufferten tills båda konsument-trådarna har hämtat det, eller för lång tid har passerat (timeout). Om bara en av trådarna lyckas hämtar ett meddelande i tid ska den informeras om att den andra kanalen fallit bort. I det fall att det bara finns en konsument-tråd registrerad ska den, på samma sätt, få meddelanden och informeras om att den är den ende mottagaren.

Således, skriv en ny subklass `BufferDual` (som ärver från `BufferPlain`) och som:

1. Har en extra registreringsmetod som tillåter två trådar att registrera sig som mottagare av meddelanden. (Att leverera samma `RTEvent` till två konsumenter är inte tillåtet på grund av ägarskapskontrollen. Här får du emellertid ignorera sådana problem och/eller sätta owner till null.)
2. När den första tråden hämtar ett meddelande ska tidpunkten lagras och meddelandet lämnas kvar i bufferten.
3. När den andra tråden anropar `doFetch` för att hämta samma meddelande ska detta returneras om anropet kommer inom 200ms från att första tråden anropade `doFetch`.
4. Om den andra tråden anropar `doFetch` för sent ska den få ett exception. Meddelandet ska plockas bort, så att nästföljande anrop returnerar nästa meddelande i bufferten.
5. I fallet att den andra tråden inte hämtat det senaste meddelandet i tid ska meddelandet plockas bort ur bufferten, den första tråden få ett exception, och den andra tråden ska få ett exception när/om den senare anropar `doFetch`.
6. Begreppen första och andra tråden beror endast på vilken tråd som försöker hämta ett visst meddelande först; rollerna tilldelas alltså inte vid registreringen.

Skriv de nya metoderna och förklara din design och dina beslut.

Ledning: Om du vill returnera ett event även i fallet att ett exception ska genereras, kan en `Throwable`-instans innehålla en referens till objektet som är orsak till detta exception. Attributet `cause` sätts med någon av konstruktorerna

```
Throwable(String message, Throwable cause)
```

eller

```
Throwable(Throwable cause)
```

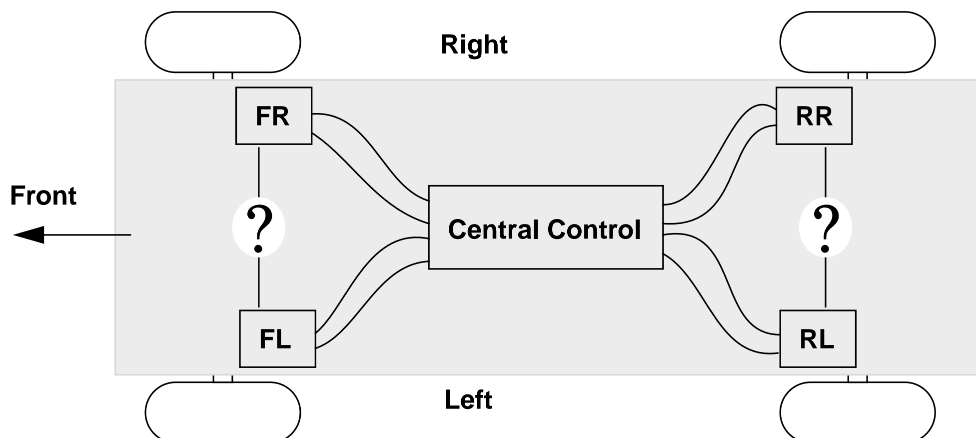
och metoden

```
Throwable getCause()
```

returnerar värdet av attributet `cause`. (10 p)

10 Brake-by-wire

Bakgrund: För att reducera kostnaden försöker man inför s.k. "X-by-wire" i moderna fordon, där X kan betyda "steer", "fly", "brake", etc. I denna uppgift studeras bromsning (brake) av fyrhjuliga fordon, med en central styrdator och en distribuerad mikroprocesson per hjul. Den tillgängliga programvaran för den centrala processorn använder `RTEventBuffer` för att skicka styrorder till hjulen och för att ta emot svar från dessa. Eftersom det är av avgörande betydelse att fordonet förblir säkert i fallet att en nätverkskabel bryts, så kan vi förslagsvis använda redundans och klassen `BufferDual` som utvecklades i uppgift 9. Antag att det finns rendundanta nätverkskopplingar i form av dubblerade kablage mellan centraldatorn och var och en av bromsnoderna (FR, FL, RR, och RL) såsom visas i följande figur:



Linjerna med frågetecken mellan bromsnoderna för axelgemensamma hjul markerar att du om så behövs (enligt specifikationerna nedan) kan introducera ytterligare förbindelser.

Uppgift: Utforma det distribuerade bromssystemet så att:

- Den central styrprogramvaran (som körs av main-tråden i den datorn) skickar bromskommandon eller **börvärden till var och en av bromsregulatorerna**. Du bestämmer när och hur detta sker (exempelvis periodiskt).
- **Varje bromsstyrning utför en periodisk regleralgoritm**. Då kan även ev maskinvarufel upptäckas, vilket skall resultera i felmeddelande som skickas till centraldatorn.
- Om **avbrott** inträffar i en enda nätverkskabel mellan centralen och bromsningen så skall fordonet **fungera såsom om inget fel funnits**, förutom att en varning (centralt) skall visas för föraren genom anrop av `Panel.brakeWarning();`
- Varje **bromsregulator skall inom 40ms detektera ifall en eller flera av dess förbindelser med centralen är bruten**. Du kan anta att all kommunikation är predikterbar och att data/objekt alltid överförs inom 2ms. Antag ytterligare att alla nätverksmetoder är potentiellt sett blockerande (vid läsning då inga data finns).
- Cetraldatorn skall **övervaka att inget kablage brutits**. Om meddelanden kan tas emot från en förbindelse så kan man antaga att alla ledningar är OK. Läsning av data kan dock forsatt vara blockerande (utan exceptions) även om förbindelsen bryts.
- En bromsenhet **utan förbindelse med centraldatorn skall bromsa lika mycket som den andra enheten på samma axel**. Om det finns något kommunikationsproblem med den andra bromsen på samma axel så skall detta hanteras likadant som om en av förbindelserna med centrladatorn är bruten.
- En helt bortkopplad bromsenhet skall broma med halv maxeffect efter 500ms. En återupptagen förbindelse inom denna tid är OK men skall resultera i en `brakeWarning`.

Utforma systemet i termer av trådar och förbindelserna mellan dessa. Illustrativa figurer krävs men programkod behöver inte skrivas. (8 p)