Lösningar, EDAF85 Realtidssystem

2025–10–24, 08.00–13.00

1. Earliest Deadline First, EDF, låter hela tiden den tråd som har närmast till sin nästa deadline köra. Trådarna har alltså inte någon fix prioritet. Deadline Monotonic Scheduling, DMS, är däremot ett fixprioritetsprotokoll där prioriteten bestäms av hur korta deadlines trådarna har. Ju kortare deadline desto högre prioritet.

2. $B_A = 0{,}3ms$ – direkt blockering i M1.
   $B_B = max(0{,}3; 0{,}1) = 0,3ms$ – värsta fallet är om D befinner sig i M1 vilket ger indirekt blockering
   $B_C = 0{,}3ms$ – indirekt blockering när D ärver prioriteten från A.
   $B_D = 0ms$ – inga lägre prioriterade trådar

3. Eftersom RMS används blir prioritetsordningen D–C–A–B. Vi använder iterationsmetoden för att beräkna värstafallssvarstiderna:

   $R_D = 2 + 0,2 = 2,2$

   $R_C = 2 + 0{,}5 = 2{,}5$
   $R_C = 2 + 0{,}5 + \left\lceil \frac{2{,}5}{8} \right\rceil \cdot 2 = 4{,}5$
   $R_C = 2 + 0{,}5 + \left\lceil \frac{4{,}5}{8} \right\rceil \cdot 2 = 4{,}5$

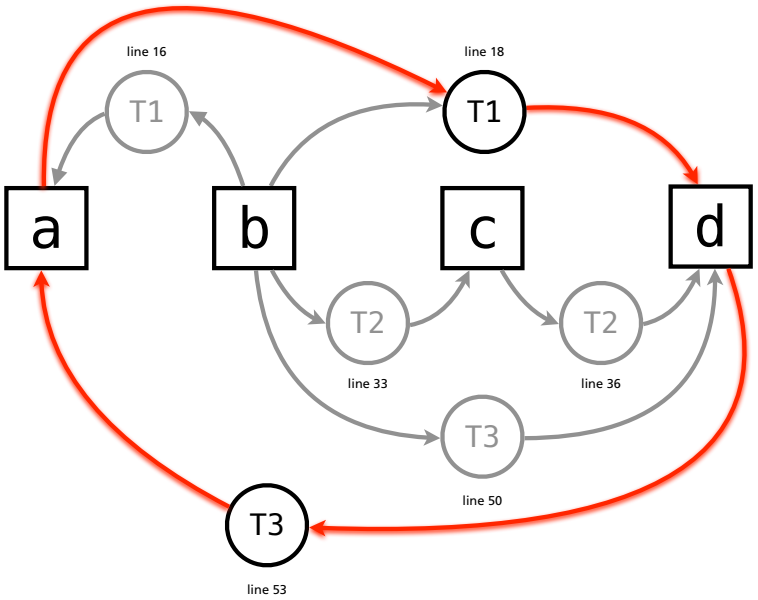
   $R_A = 4 + 0{,}3 = 4{,}3$
   $R_A = 4 + 0{,}3 + \left\lceil \frac{4{,}3}{8} \right\rceil \cdot 2 + \left\lceil \frac{4{,}3}{10} \right\rceil \cdot 2 = 8{,}3$
   $R_A = 4 + 0{,}3 + \left\lceil \frac{8{,}3}{8} \right\rceil \cdot 2 + \left\lceil \frac{8{,}3}{10} \right\rceil \cdot 2 = 10{,}3$
   $R_A = 4 + 0{,}3 + \left\lceil \frac{10{,}3}{8} \right\rceil \cdot 2 + \left\lceil \frac{10{,}3}{10} \right\rceil \cdot 2 = 12{,}3$
   $R_A = 4 + 0{,}3 + \left\lceil \frac{12{,}3}{8} \right\rceil \cdot 2 + \left\lceil \frac{12{,}3}{10} \right\rceil \cdot 2 = 12{,}3$

   $R_B = 5 + 0 = 5$
   $R_B = 5 + 0 + \lceil \frac{5}{8} \rceil \cdot 2 + \lceil \frac{5}{10} \rceil \cdot 2 + \lceil \frac{5}{15} \rceil \cdot 4 = 13$
   $R_B = 5 + 0 + \left\lceil \frac{13}{8} \right\rceil \cdot 2 + \left\lceil \frac{13}{10} \right\rceil \cdot 2 + \left\lceil \frac{13}{15} \right\rceil \cdot 4 = 17$
   $R_B = 5 + 0 + \left\lceil \frac{17}{8} \right\rceil \cdot 2 + \left\lceil \frac{17}{10} \right\rceil \cdot 2 + \left\lceil \frac{17}{15} \right\rceil \cdot 4 = 23$
   $R_B = 5 + 0 + \lceil \frac{23}{8} \rceil \cdot 2 + \lceil \frac{23}{10} \rceil \cdot 2 + \lceil \frac{23}{15} \rceil \cdot 4 = 25$
   $R_B = 5 + 0 + \lceil \frac{25}{8} \rceil \cdot 2 + \lceil \frac{25}{10} \rceil \cdot 2 + \lceil \frac{25}{15} \rceil \cdot 4 = 27$
   $R_B = 5 + 0 + \lceil \frac{27}{8} \rceil \cdot 2 + \lceil \frac{27}{10} \rceil \cdot 2 + \lceil \frac{27}{15} \rceil \cdot 4 = 27$

   Svar: $R_A = 12{,}3ms$, $R_B = 27ms$, $R_C = 4{,}5ms$ och $R_D = 2{,}2ms$

4.

a)



b) Det är tråd T1 och tråd T3 som orsakar dödläget. Tråd T1 försöker låsa **d** på rad 18 och T3 försöker låsa **a** på rad 53.

c) Om T1 och T3 hamnat i dödläge beror det på att de har tagit, och håller, låsen **a**, **b** och **d**. Förr eller senare kommer tråd T2 att försöka ta någon av dessa lås, vilket kommer att misslyckas. Då fastnar även tråd T2. Detta kan ske på rad 31 (försöker ta **b**) eller på rad 36 (försöker ta **d**).

d) Det är lite komplicerat att lösa detta fall, men ett (av flera) sätt att göra det är att byta ut rad 53 mot:

```
d.unlock();
a.lock();
d.lock();
```

Det förutsätter förstås att det går bra att tillfälligt släppa låset på **d**. Annars kan man t.ex. flytta upp `a.lock()` på rad 53 till före rad 50. Nackdelen med denna lösning är att vi i ”onödan” låser **a** medan koden som representeras av ”`// Use b and d`” körs.

5. Tricket är att se till att monitorerna `a1` och `a2` låses i samma ordning i de två trådarna. Se övning 3 på Canvas.

   Ett sätt att göra det är att ändra den andra tråden (rad 12–16) enligt följande:

```
new Thread(() -> {
  while (true) {
    synchronized(a1) {
      a2.m1(a1);
    }
  }
}).start();
```

   Nu kommer båda trådarna att börja med att låsa `a1` för att sedan i nästa steg låsa `a2`.

   Ett sätt att åstadkomma samma sak, fast lite krångligare, är att införa en synkroniserad metod i klassen `A` som inte gör annat än anropar `m1()` i det andra monitorobjektet och anropa denna i själva tråden:

```
new Thread(() -> {
  while (true) {
    a1.m1caller(a2,a1);
  }
}).start();

...

class A {
  synchronized m1caller(A callee, A argument) {
    callee.m1(argument);
  }

  synchronized void m1(A a) {
    System.out.println("m1-1");
    a.m2();
    System.out.println("m1-2");
  }

  synchronized void m2() {
    System.out.println("m2");
  }
}
```

6. **Filtrering av sensorvärden**

Det enda som krävdes som svar var de bitar kod som skulle skjutas in i det givna programmet, men här inkluderas även det som var givet från början för att ge en helhetsbild av lösningen. Här modelleras också meddelanden med hjälp av separata klasser för varje typ av meddelande vilket kanske är lite överarbetat. Det kan säkert göras betydligt kortare (om möjligen än mindre renlärigt ur modelleringssynpunkt), t.ex. med hjälp av records och en enum som anger meddelandetyp.

Man kan även diskutera hur man ur modelleringssynpunkt bäst hanterar de olika typer av meddelanden som anländer till trådarna. Här använder vi `instanceOf` för att avgöra vad som ska hända, men ofta brukar virtuella metoder vara ett bättre alternativ än `instanceOf`. I det här fallet skulle det i så fall dock innebära att vi implementerade trådens funktionalitet i virtuella metoder i meddelandeklasserna där det knappast kan sägas höra hemma. Jämför med principen om enkelt ansvar – Single Responsibility Principle.

```
abstract class FilterMessage {}

class SensorMessage extends FilterMessage {
  private int id;
  private double value;

  public SensorMessage(int id, double value) {
    this.id = id;
    this.value = value;
  }

  public int getId() { return id; }

  public double getValue() { return value; }
}

class RequestMessage extends FilterMessage {
  private long time;
  private ActorThread sender;

  public RequestMessage(long time, ActorThread sender) {
    this.time = time;
    this.sender = sender;
  }

  public double getTime() { return time; }

  public ActorThread getSender() { return sender; }
}

abstract class SupervisorMessage {}

class DangerMessage extends Supervisor {}

class MinimumMessage extends SupervisorMessage {
  private double value;

  public MinimumMessage(double value) {
    this.value = value;
  }

  public double getValue() { return value; }
}
```

```
class SensorThread extends Thread {
  private int id;
  private ActorThread filter;

  public SensorThread(int id, ActorThread filter) {
    this.id = id;
    this.filter = filter;
  }

  public void run() {
    while(true) {
      double sensorValue = SensorIO.getValue(id);
      filter.send(new SensorMessage(id,sensorValue));
    }
  }
}

class SupervisorThread extends ActorThread<SupervisorMessage>  {
  private ActorThread filter;

  public SupervisorThread(ActorThread filter) {
    this.filter = filter;
  }

  public void run() {
    long nextTime = System.currentTimeMillis();
    while(true) {
      filter.send(new RequestMessage());
      try {
        SupervisorMessage m = receive();
        if (m instanceof MinimumMessage) {
          MinimumMessage mm = (MinimumMessage) m;
          System.out.println(m.getValue());
        } else {
          System.out.println("Danger!");
        }
      } catch(InterruptedException e} {
        System.err.println("Unexpected interrupt:");
        System.err.println(e);
        System.exit(1);
      }
      nextTime = System.currentTimeMillis()+1000;
    }
  }
}
```

```
class FilterThread extends ActorThread<FilterMessage> {
  private static final double DANGER_DIST = 0.8;

  public void run() {
    double [] value = new double[3];
    long [] timestamp = new long[3];
    ActorThread sender = null;
    long timestamp = 0;

    mainloop:
    while (true) {
      try {
        FilterMessage m = receive();
        if (m instanceOf RequestMessage) {
          RequestMessage mm = (RequestMessage) m;
          sender = mm.getSender();
          timestamp = mm.getTime();
        } else {
          SensorMessage mm = (SensorMessage) m;
          value[m.getId()] = mm.getValue();
          timestamp[m.getId()] = System.currentTimeMillis();
        }
      } catch(InterruptedException e} {
        System.err.println("Unexpected interrupt:");
        System.err.println(e);
        System.exit(1);
      }
      double minimum = Double.MAX_VALUE;
      int valid = 0;
      for(int i=0;i<3;i++) {
        if (value[i]<DANGER_DIST && timestamp[i]>0 && sender!=null) {
          sender.send(new DangerMessage());
          sender = null;
          continue mainloop;
        }
        if (timestamp[i]>=timestamp) {
          valid++;
        }
        if (value[i]<minimum) {
          minimum = value[i];
        }
      }
      if (valid==3 && sender!=null) {
        sender.send(new MinimumMessage(minimum));
        sender = null;
      }
    }
  }
}

public class FilterTask {
  public static void main() {
    FilterThread filter = new FilterThread();
    filter.start();
    new SupervisorThread(filter).start();
    for(int i=0;i<3;i++) {
      new SensorThread(i,filter).start();
    }
  }
}
```

7. **SupervisedBuffer**
   Vi redovisar återigen hela koden för helhetens skull.

```
public class SupervisedBuffer {
  private LinkedList<Object> queue = new LinkedList<>(); // Stores buffered messages
  private long activity;

  /** Adds message to the buffer. Never blocks */
  public synchronized void add(Object message) {
    queue.add(message);
    activity = System.currentTimeMillis();
    notifyAll();
  }

  /** Returns the next available message. The method blocks if the
      queue is empty until a new message is added. */
  public synchronized Object take() throws InterruptedException {
    while (queue.isEmpty()) {
      wait();
    }
    activity = System.currentTimeMillis();
    return queue.remove(0);
  }

  /** This method blocks until timeout milliseconds have passed since
      a message was last added to or returned from the buffer. Can be
      used by a supervisor thread to detect communication stall. */
  public synchronized void awaitInactivity(long timeout) throws InterruptedException {
    // If we want to wait at least timeout milliseconds from this method
    // was called add: "activity = System.currentTimeMillis();" here. That
    // might perhaps be more useful...
    while (System.currentTimeMillis()<=activity+timeout) {
      long delay = activity+timeout-System.currentTimeMillis();
      if (delay>0) {
        wait(delay);
      }
    }
  }
}
```