Lösningar, EDAF85 Realtidssystem

2025–08–28, 08.00–13.00

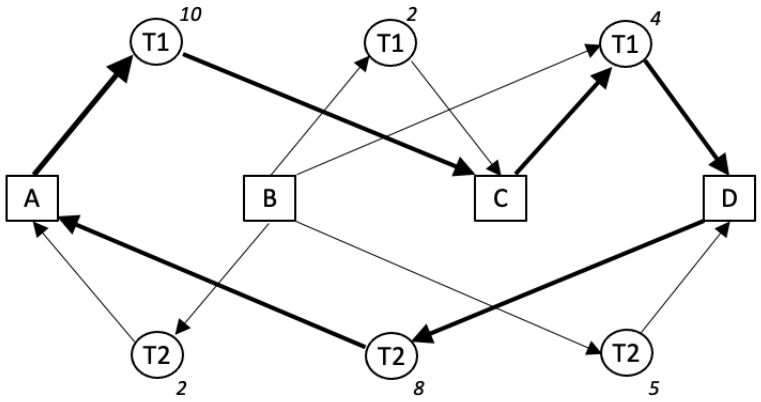
1. Prioritetsinversion är ett fenomen som inträffar när en högprioriterad tråd blir blockerad av en lägre prioriterad tråd utan att denna direkt blockerar en resurs som den första tråden försöker låsa och som leder till långa fördröjningar som kan vara svåra eller till och med omöjliga att beräkna i förväg.

   Exempel: Tre trådar, A, B, och C, där A har högst prioritet och C lägst. A och C kommunicerar med varandra genom en monitor M. C kör och går in i M (låser monitorn). B börjar köra och avbryter C i kraft av sin högre prioritet. A börjar köra och avbryter i sin tur B på grund av sin ännu högre prioritet. A försöker gå in i M, men blockerar eftersom den är låst av C. B fortsätter köra eftersom den nu är den tråd som har högst prioritet av de körbara trådarna (B och C). Först när B kört klart får C köra och kan lämna M så att A kan fortsätta sin körning. Resultatet är att A har blockerats inte bara av att C behövde lämna C, men också av att B skulle köra färdigt trots sin lägre prioritet.

   Prioritetsinversion kan undvikas genom att införa prioritetsarv. Det kan t.ex. göras genom att låta C tillfälligt få As prioritet så länge som A väntar på att C ska lämna en resurs som A behöver.

2. 1. Falskt. Det är bara synkroniserade anrop i samma *objekt* som blockerar varandra. Inte anrop i olika objekt.
   2. Sant. Enligt föregående.
   3. Falskt. Eftersom `getAbs` inte är synkroniserad kommer den heller aldrig att blockera när den anropas. Låset på C1 påverkar inte anropet av metoden.
   4. Falskt. Vi kan få felaktiga värden som returneras från metoden ifall `re` och/eller `im` skulle uppdateras under tiden beräkningen av formeln utförs.

3. a) Resursallokeringsgraf med cykel markerad samt radnummer för varje hold-wait-situation.



   b) Vi måste ha två instanser av T1, som exekverar rad 4 respektive 10, samt en instans av T2 som exekverar rad 8 för att det ska bli dödläge.

   c) Byt ordning på rad 9 och 10 i T1 så att C låses före A. Då vänder vi på T1-beroendet så att går från C till A i stället för tvärtom och cykeln bryts.

4. DMS ger prioritetsordningen B, C, A för trådarna.

   a) Blockeringstiden för A, B_A, är 0 *ms* eftersom A har lägst prioritet och det därmed inte finns några trådar med lägre prioritet som kan blockera den.

   För B påverkas blockeringstiden B_B endast av direktblockeringar då B har högst prioritet. Det finns två fall då A kan läsa antingen M1 eller M2, men inte båda samtidigt. Om A låser M2 kan C inte låsa denna. Vi får då: $B_B = max((0,1+0,4),(0,2)) = 0,5\ ms$.

   I fallet med C kan potentiellt både direkt och indirekt (push-through) blockering förekomma. Vi får två fall beroende på om A låser monitorn M1 eller M2 (indirekt respektive direkt blockering) vilket ger: $B_C = max((0,4),(0,2)) = 0,4\ ms$.

   b) Vi använder iterationsmetoden för att beräkna värstafallssvarstiderna:

   $R_B = 2 + 0,5 = 2,5$ Eftersom vi inte har något R_B i högerledet finns det ingen anledning att fortsätta iterationen.

$$
\begin{aligned}
R_C &= 4 + 0,4 = 4,4 \\
R_C &= 4 + 0,4 + \left\lceil \tfrac{4,4}{5} \right\rceil \cdot 2 = 6,4 \\
R_C &= 4 + 0,4 + \left\lceil \tfrac{6,4}{5} \right\rceil \cdot 2 = 8,4 \\
R_C &= 4 + 0,4 + \left\lceil \tfrac{8,4}{5} \right\rceil \cdot 2 = 8,4
\end{aligned}
$$

$$
\begin{aligned}
R_A &= 3 + 0 = 3 \\
R_A &= 3 + 0 + \left\lceil \tfrac{3}{5} \right\rceil \cdot 2 + \left\lceil \tfrac{3}{20} \right\rceil \cdot 4 = 9 \\
R_A &= 3 + 0 + \left\lceil \tfrac{9}{5} \right\rceil \cdot 2 + \left\lceil \tfrac{9}{20} \right\rceil \cdot 4 = 11 \\
R_A &= 3 + 0 + \left\lceil \tfrac{11}{5} \right\rceil \cdot 2 + \left\lceil \tfrac{11}{20} \right\rceil \cdot 4 = 13 \\
R_A &= 3 + 0 + \left\lceil \tfrac{16}{5} \right\rceil \cdot 2 + \left\lceil \tfrac{10}{20} \right\rceil \cdot 4 = 13
\end{aligned}
$$

   Svarstiderna blir alltså: $R_B = 2,5\ ms$, $R_C = 8,4\ ms$ och $R_A = 13\ ms$.

   c) Nej, det kan vi inte eftersom den metoden bara är tillämplig om trådarna är oberoende av varandra (ingen blockering) och om deadline är lika med periodtid. Inget av detta gäller i vårt fall.

5.

```
class VideoProcessing {

    // reading thread
    ActorThread<DataBuffer> t1 = new ActorThread<>() {
        @Override
        public void run() {
            try {
                while (true) {
                    DataBuffer b = receive();
                    b.read();
                    t2.send(b);
                }
            } catch (InterruptedException unexpected) {
                throw new Error(unexpected);
            }
        }
    };

    // processing thread
    ActorThread<DataBuffer> t2 = new ActorThread<>() {
        @Override
        public void run() {
            try {
                while (true) {
                    DataBuffer b = receive();
                    b.process();
                    t1.send(b);
                }
            } catch (InterruptedException unexpected) {
                throw new Error(unexpected);
            }
        }
    };

    void startThreads() {
        t1.start();
        t2.start();

        t1.send(new DataBuffer());
        t1.send(new DataBuffer());  // added
    }

    public static void main(String[] args) {
        new VideoProcessing().startThreads();
    }
}
```

6.

```
class DistanceSensing {
  Sample val[] = new Sample[3];

  /** Constructor */
  public DistanceSensing() { }

  /** Stores a new sensor value in the monitor.
      Old sensor value from the same sensor is
      overwritten. This method never blocks. */
  public synchronized void put(Sample s) {
    val[s.getId()] = s;
    notifyAll();
  }

  /** Blocks until sensor values arrived later than time
      (as measured by System.currentTimeMillis())
      are available from all three sensors and then returns
      the sample with smallest value. If any sensor does
      not have, or receives, a sufficiently recent sensor
      value (having arrived later than time) within
      timeout milliseconds after getMinimum() being called,
      null is returned.
  */
  public synchronized Sample getMinimum(long time, long timeout) {
    long deadline = System.currentTimeMillis()+timeout;
    Sample ret = minimum(time);
    while (ret==null) {
      try {
        long delay = deadline-System.currentTimeMillis();
        if (delay>0) {
          wait(delay);
        } else {
          break;
        }
      } catch(InterruptedException e) {
        break;
      }
      ret = minimum(time);
    }
    return ret;
  }

  private Sample minimum(long time) {
    Sample ret = null;
    int valid = 0;
    for(int i=0;i<3;i++) {
      if (val[i]!=null) {
        if (val[i].getTimestamp()>time) {
          valid++;
          if (ret==null || val[i].getDistance()<ret.getDistance()) {
            ret = val[i];
          }
        }
      }
    }
    if (valid<3) {
      ret = null;
    }
    return ret;
  }
}
```