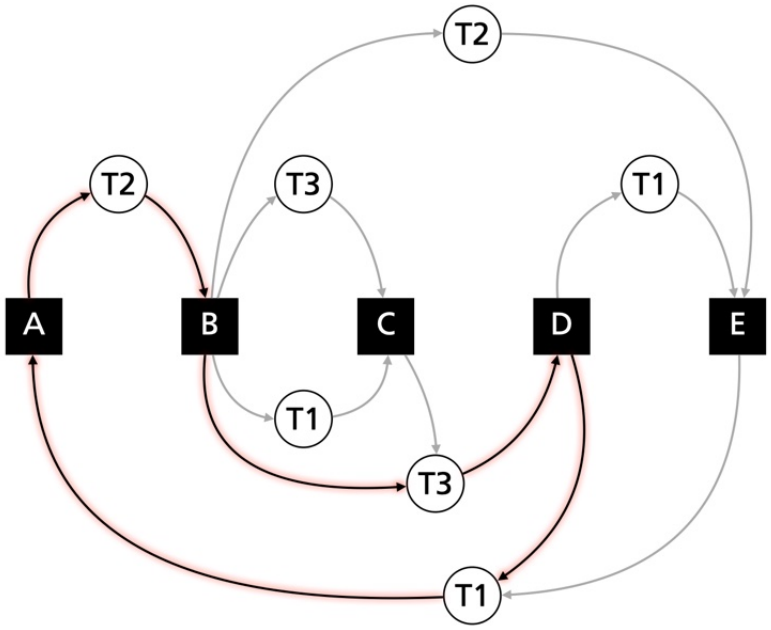
Lösningsförslag till Tentamen, EDAF85 Realtidssystem

2022–08–25, 14.00–19.00

1. a) Kalla resurserna A, B, C, D och E (motsvarande lockA, lockB, lockC, lockD och lockE). Dessutom har vi tre trådar. Kalla dessa T1 (den första tråden), T2 (den andra tråden) och T3 (huvudprogrammet). Då får vi följande resursallokeringsgraf med cykeln A-T2-B-T3-D-T1-A:



b) Tråd T1 måste befinna sig på rad 14 (`lockA.lock()`), T2 på rad 39 (`lockB.lock()`) och T3 på rad 53 (`lockD.lock()`).

c) Byt plats på rad 38 och 39.

2. a) Under vissa förutsättningar, såsom att varje tråds deadline var lika med dess periodtid, trådarna schemalades enligt RMS och att inga blockeringar mellan trådarna förekom, skulle vi, om det totala CPU-utnyttjandet var under $n(2^{1/n} - 1)$, kunna sluta oss till att systemet var schemaläggningsbart. Men då dessa förutsättningar inte är uppfyllda i detta fall kan vi inte dra några som helst slutsatser om schemaläggningsbarheten.

b) Först måste vi avgöra prioritetsordningen bland trådarna. Eftersom DMS skulle användas ges det av kolumn D i tabellen. Vi får då (prioritet efter stigande periodtid): B, C, D, A. Vi kan därefter resonera oss fram till blockeringstiderna genom att analysera den givna blockeringsgrafen.

För B (högst prioriterad) gäller att bara direkt blockering kan vara aktuell. Blockering kan då bara ske i M1 av A. Vi får då $B_B = 0,4ms$.

För C dels direkt blockering förekomma i M2 (av D) samt indirekt blockering av A om A skulle ärva Bs prioritet $B_C = 0,3 + 0,4 = 0,7ms$.

För D gäller att enbart indirekt blockering kan komma ifråga. Även i detta fallet handlar det om att A kan ärva Bs prioritet. Vi får alltså: $B_D = 0,4ms$.

Till slut har vi den lägst prioriterade tråden kvar, A. Eftersom det inte finns några trådar med lägre prioritet kan den heller inte blockeras av lägre prioriterade trådar. Vi får alltså: $B_A = 0ms$.

c) Vi använder iterationsmetoden som beskrivet ovan:
$R_B = 1 + 0,4 = 1,4$ Eftersom vi inte har något R_B i högerledet finns det ingen anledning att fortsätta iterationen.

$$R_C = 3 + 0,7 = 3,7$$
$$R_C = 3 + 0,7 + \left\lceil \frac{3,7}{4} \right\rceil \cdot 1 = 4,7$$
$$R_C = 3 + 0,7 + \left\lceil \frac{4,7}{4} \right\rceil \cdot 1 = 5,7$$
$$R_C = 3 + 0,7 + \left\lceil \frac{5,7}{4} \right\rceil \cdot 1 = 5,7$$

$$R_D = 2 + 0,4 = 2,4$$
$$R_D = 2 + 0,4 + \left\lceil \frac{2,4}{4} \right\rceil \cdot 1 + \left\lceil \frac{2,4}{12} \right\rceil \cdot 3 = 6,4$$
$$R_D = 2 + 0,4 + \left\lceil \frac{6,4}{4} \right\rceil \cdot 1 + \left\lceil \frac{6,4}{12} \right\rceil \cdot 3 = 7,4$$
$$R_D = 2 + 0,4 + \left\lceil \frac{7,4}{4} \right\rceil \cdot 1 + \left\lceil \frac{7,4}{12} \right\rceil \cdot 3 = 7,4$$

$$R_A = 4 + 0 = 4$$
$$R_A = 4 + 0 + \left\lceil \frac{4}{4} \right\rceil \cdot 1 + \left\lceil \frac{4}{12} \right\rceil \cdot 3 + \left\lceil \frac{4}{24} \right\rceil \cdot 2 = 10$$
$$R_A = 4 + 0 + \left\lceil \frac{10}{4} \right\rceil \cdot 1 + \left\lceil \frac{10}{12} \right\rceil \cdot 3 + \left\lceil \frac{10}{24} \right\rceil \cdot 2 = 12$$
$$R_A = 4 + 0 + \left\lceil \frac{12}{4} \right\rceil \cdot 1 + \left\lceil \frac{12}{12} \right\rceil \cdot 3 + \left\lceil \frac{12}{24} \right\rceil \cdot 2 = 12$$

Svarstidena blir alltså: $R_B = 1,4ms$, $R_C = 5,7ms$, $R_D = 7,4ms$ och $R_A = 12ms$.

d) Eftersom samtliga svarstider är mindre än respektive tråds deadline så kommer systemet att vara schemaläggningsbart.

3. Med rendez-vous avses en situation där två trådar möts på ett bestämt ställe i respektive tråds kod och där en tråd väntar medan den andra tråden vidtar någon slags åtgärd. Det kan t.ex. handla om att en tråd tillhandahåller data av något slag som den andra tråden sedan utför någon slags beräkning på datan i fråga och den första tråden behöver invänta resultatet av beräkningen. I detta fall kan två två signalerande semaforer användas enligt exemplet nedan. Här antas tråd A tillhandahålla något slags data som tråd B sedan ska göra något slags beräkning på medan tråd A väntar tills beräkningen blir klar. Semaforerna antas ha startvärdet 0 från början.

```
// Tråd A
PrepareData();
semaphoreA.release();
/* Wait for thread B to
   complete rendez-vous */
semaphoreB.acquire();
...
```

```
// Tråd B
/* Wait for thread A to initiate rendez-vous
semaphoreA.acquire();
DoCalculation();
semaphoreB.release();
...
```

4. a) Detta är ett exempel på en s.k. *kapplöpning* (eng. race condition):

a) Först kontrolleras värdet på `s` i villkoret (`s != null`). Det intressanta fallet inträffar när uttrycket är sant, annars returneras värdet -1 och det kan inte uppstå något `NullPointerException`.

b) När det väl konstaterats att uttrycket är sant kommer `s.length()` att evalueras som ett andra steg.

Mellan steg 1 och 2 är det dock möjligt att det sker ett kontextbyte (eng. context switch) och då kan en annan tråd tänkas anropa `set()` och ändra värdet på `s` till `null`. När den ursprungliga tråden så småningom får fortsätta exekvera har den redan passerat null-kontrollen och anropar `s.length()` i övertygelsen om att `s` är skilt från `null` vilket resulterar i en `NullPointerException`.

b)
```
public int len() {
    String a = s;   // stack-confined copy of reference
    return (a != null) ? a.length() : -1;
}
```

5.
```
public class CountDownLatch {

    private int count;

    public CountDownLatch(int count) {
        this.count = count;
    }

    public synchronized void await() throws InterruptedException {
        while (count != 0) {
            wait();
        }
    }

    public synchronized boolean await(long timeout) throws InterruptedException {
        long now = System.currentTimeMillis();
        long finish = now + timeout;
        while (now < finish && count != 0) {
                wait(finish - now);
                now = System.currentTimeMillis();
        }
        return (count == 0);
    }

    public synchronized void countDown() {
        count--;
        if (count == 0) {
                notifyAll();
        }
    }
}
```

6.
```
class WheelController {
    private int wheelno;
    private CountDownLatch latch;

    public WheelController(int wheelno,CountDownLatch latch) {
        this.wheelno = wheelno;
        this.latch = latch;
    }

    public void run() {
        initiateConnectionToWheel(wheelno);
        latch.countDown();
        wheelControlLoop(wheelno);
    }
}

class ABSSystem {
    public boolean initiateABS() {
        CountDownLatch latch = new CountDownLatch(4);
        for (i=0;i<4;i++) {
            new WheelController(i,latch).start();
        }
        return latch.await(3000);
    }
}
```