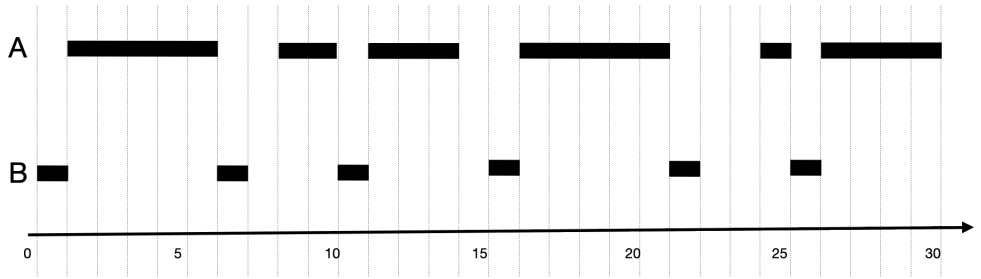
Lösningar, Tentamen
EDAF85 – Realtidssystem

2023-10–28, 08.00–13.00

1. a) I varje ögonblick låter schemaläggaren den tråd köra som har närmast till sin nästa deadline. Det finns alltså inga fasta prioriteter utan det kan mycket väl vara så att en tråd med lång periodtid får köra före en tråd med kort periodtid när den närmar sig sin deadline.

b)



2. a) Rita en resursallokeringsgraf! Då ser man att det krävs minst två T3-trådar för att det ska bli dödläge. Alltså: n=2. Det finns en annan cykel i grafen som består av två T1-noder och en T2-nod (dvs n=0), men eftersom det bara finns en T1-tråd så är detta en falsk cykel.

b) T1: rad 15, T2: rad 28, första T3:rad 42, andra T3: rad 45

3. $R_A = 5,3ms$, $R_B = 16ms$, $R_C = 2,2ms$

4. $B_A = max(3 + max(2;4);6) = 7ms$ – enbart direkt blockering
$B_B = max(6;3+4) = 7ms$ – direkt eller kombination av direkt och indirekt blockering
$B_C = 4ms$ – direkt eller indirekt blockering; båda fallen inträffar när D tagit M2
$B_D = 0ms$ – lägst prioritet

5. Prioritetsinversion är ett fenomen som kan inträffa när vi schemalägger trådar enligt principen fixed priority scheduling och inte har något prioritetsärvningsprotokoll. Då kan följande scenario uppträda:

1. En lågprioriterad tråd (L) kör och tar ett lås (t.ex. genom att anropa en monitormetod).
2. En tråd med medelhög prioritet (M) avbryter L och börjar köra.
3. En högprioriterad tråd (H) avbryter M och börjar i sin tur köra.
4. H försöker ta låset som L håller och blockeras.
5. Nu får M köra eftersom M är den körbara tråd som har högst prioritet. Tiden som M kan behöva på sig för att exekvera färdigt kan vara väldigt lång om vi har otur.
6. Först när M har kört färdigt får L köra och kan släppa låset som H vill ha.
7. Till slut får H tillgång till låset och kan köra vidare.

Problemet med detta är att värstafallssvarstiden för H kan bli väldigt lång och väldigt svår att beräkna vilket är oacceptabelt i ett system med strikta tidskrav.

6. Busy-wait innebär att en tråd ligger i en loop och gör inget annat än testar ett villkor om och om igen i väntan på att det ska bli uppfyllt. Detta kan leda till att andra trådar, särskilt de med lägre prioritet, utsätts för svält och får inte köra. Dessa trådar kan då inte uppfylla villkoret som den första tråden väntar på. Dessutom går mängder av CPU-kraft åt i onödan vilket slöar ner andra processer och ökar strömförbrukningen på datorn.

7.

```
public class RWLock {
    private int nbrReadersActive = 0;
    private int nbrWritersWaiting = 0;
    private boolean writerActive = false;

    public synchronized void lockR() throws InterruptedException {
        while (nbrWritersWaiting > 0 || writerActive) {
            wait();
        }
        nbrReadersActive++;
    }

    public synchronized void unlockR() {
        nbrReadersActive--;
        notifyAll();
    }

    public synchronized void lockW() throws InterruptedException {
        try {
            nbrWritersWaiting++;
            while (nbrReadersActive > 0 || writerActive) {
                wait();
            }
            writerActive = true;
        } finally {
            nbrWritersWaiting--;
            notifyAll();
        }
    }

    public synchronized void unlockW() {
        writerActive = false;
        notifyAll();
    }
}
```

8.

```
while(true) {
    try {
        int m = receive();
        if (m==queue.CUSTOMER_ID) {
            queue.customerHandler.send(++lastCustomerArrived);
        } else {
            if (!clerkFree[m-1]) {
                clerkFree[m-1] = true;;
                nrOfFreeClerks++;
            }
        }
        if (lastCustomerArrived!=lastCustomerServed && nrOfFreeClerks>0) {
            do {
                lastClerkAssigned = (lastClerkAssigned+1) % queue.NO_OF_CLERKS;
            } while (!clerkFree[lastClerkAssigned]);
            nrOfFreeClerks--;
            clerkFree[lastClerkAssigned] = false;
            DispData d = new DispData();
            d.ticket = ++lastCustomerServed;
            d.counter = lastClerkAssigned+1;
            queue.displayHandler.send(d);
        }
    } catch(InterruptedException e) {
        throw new Error("Unexpected interrupt");
    }
}
```