LUND UNIVERSITY, Faculty of Engineering Department of Computer Science

Lösningar, EDAF85 Realtidssystem

2024–04–11, 08.00–13.00

1. 1. Sant
   2. Falskt
   3. Sant
   4. Sant

2. a) A: $5 + 3 = 8\mu s$ (direkt blockering i M1 och M2)
   B: $5 + 3 = 8\mu s$ (indirekt blockering av C och D i M1 och M2)
   C: $3\mu s$ (indirekt blockering av D i M2)
   D: $0\mu s$ (lägst prioritet...)

   b) Nej, ingen tråd kommer att få en längre värstafallsblockering. De nya blockeringstiderna blir:
   A: $5\mu s$ (direkt blockering)
   B: $5\mu s$ (indirekt blockering av C)
   C: $3\mu s$ (direkt blockering av D)
   D: $0\mu s$ (lägst prioritet...)

   Möjligen kommer blockering att ske oftare, men värstafallsblockeringen blir inte längre.

   c) Av resonemanget i föregående delfråga så ser man att tråd A och B får kortare värstafallsblockeringstider.

   d) Alla trådar kör fortfarande lika länge, dvs deras respektive C är oförändrade, men kanske i en annan ordning. Alltså måste CPU-utnyttjandet vara oförändrat.

3. a) $U = 1/8 + 3/10 + 2/4 = 0,125 + 0,3 + 0,5 = 0,925$

   b) $R_C^0 = 2 + 0,5 = 2,5$
   $R_C^1 = 2 + 0,5 = 2,5$

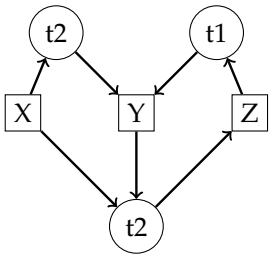
   $R_A^0 = 1 + 0,2 = 1,2$
   $R_A^1 = 1 + 0,2 + \left\lceil \frac{1,2}{4} \right\rceil \cdot 2 = 3,2$
   $R_A^2 = 1 + 0,2 + \left\lceil \frac{1,2}{4} \right\rceil \cdot 2 = 3,2$

   $R_B^0 = 3 + 0 = 3$
   $R_B^1 = 3 + 0 + \lceil \frac{3}{4} \rceil \cdot 2 + \lceil \frac{3}{8} \rceil \cdot 1 = 6$
   $R_B^1 = 3 + 0 + \lceil \frac{3}{4} \rceil \cdot 2 + \lceil \frac{3}{8} \rceil \cdot 1 = 6$
   $R_B^1 = 3 + 0 + \lceil \frac{6}{4} \rceil \cdot 2 + \lceil \frac{6}{8} \rceil \cdot 1 = 8$
   $R_B^1 = 3 + 0 + \lceil \frac{8}{4} \rceil \cdot 2 + \lceil \frac{8}{8} \rceil \cdot 1 = 8$

   Svar: $R_A = 3,2\mu s$, $R_B = 8\mu s$ och $R_C = 2,5ms$

   c) Eftersom alla värstafallssvarstider är mindre än respektive tråds deadline är systemet schemaläggningsbart.

4. a)



b) t1: F
t2: D

5.
```
public class SimpleReentrantLock {

    private Thread owner = null;
    private int lockings = 0;

    public synchronized void lock() throws InterruptedException {
        Thread curr = Thread.currentThread();

        while (owner != null && owner != curr) {
            // another thread has taken this lock
            wait();
        }
        lockings++;
        owner = curr;
    }

    public synchronized void unlock() {
        Thread curr = Thread.currentThread();

        if (owner != curr) {
            throw new IllegalMonitorStateException();
        }
        lockings--;
        if (lockings == 0) {
            owner = null;
            notifyAll();
        }
    }

    public synchronized boolean tryLock(long millis) throws InterruptedException {
        Thread curr = Thread.currentThread();
        long now = System.currentTimeMillis();
        long deadline = now + millis;

        while (owner != null && owner != curr && now < deadline) {
            // another thread has taken this lock
            wait(deadline - now);
            now = System.currentTimeMillis();
        }
        if (owner == null || owner == curr) {
            lockings++;
            owner = curr;
            return true;
        } else {
            return false;
        }
    }
}
```

6.

```
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");
    }
}
```