# Solutions to the exam in Real-Time Systems 170603

These solutions are available on WWW:
*http://www.control.lth.se/course/FRTN01/*

**1.** The ceiling of a semaphore is the priority of the highest priority task that can lock that semaphore. Semaphore s1 is locked by Task A and Task B, therefore $ceil(s1) = \max\{10, 5\} = 10$. Semaphore s2 is locked by Task B and Task C, therefore its ceiling is $ceil(s2) = \max\{5, 8\} = 8$. Semaphore s3 is locked by Task A and Task C, its ceiling being $ceil(s3) = \max\{10, 8\} = 10$. Semaphore s4 is locked by all the tasks, therefore its ceiling is $ceil(s4) = \max\{10, 5, 8\} = 10$.

**2.** One example is tracking-based anti-windup in controllers like PID. In this case, the process that is controlled is the integral part of the PID controller. The employed controller is a Proportional controller, and the control law (written in the discretized form) is the following.

```
v := P + I + D;
u := sat(v,umax,umin);
I := I + (K*h/Ti)*e + (h/Tr)*(u - v);
```

Another example is the reference model generator in state feedback control. Here the desired reference model dynamics is obtained through state feedback applied as an open loop model of the process. The process that is controlled is the open loop process model. The controller type is state feedback and the control law is given by

$$\begin{aligned} x_m(k+1) &= \Phi x_m(k) + \Gamma u_{ff}(k) \\ u_{ff}(k) &= L_c u_c(k) - L_m x_m(k) \end{aligned}$$

A third example is the corrector term in an observer. This term can be interpreted as a proportional feedback term, with the error between the measured output and the estimated output as the input of the control system. The output of the control system is the correction term that is added to the state estimate.

**3.**

**a.** For task T1 and T3 the deadline and the periods are different, therefore one should use Deadline Monotonic for priority assignement. Priority 1 is assigned to task T5, which has the smallest deadline. Priority 2 is assigned to task T1, 3 to task T4 or T3, while the non chosen one has priority 4. Priority 5 is finally assigned to task T2, which has the longest relative deadline.

**b.** The taskset is not schedulable with the approximate analysis. In fact:

$$U = \sum_{i=1}^{5} \frac{C_i}{D_i} = \frac{1}{5} + \frac{10}{50} + \frac{2}{20} + \frac{2}{20} + \frac{1}{4} = 0.85 \not\leq 5 \cdot (2^{1/5} + 1) = 0.7435.$$

Reducing $C_2$ can make the taskset schedulable according to the approximate analysis. If $C_2 = 4$, in fact

$$U = \sum_{i=1}^{5} \frac{C_i}{D_i} = \frac{1}{5} + \frac{4}{50} + \frac{2}{20} + \frac{2}{20} + \frac{1}{4} = 0.73 \leq 5 \cdot (2^{1/5} + 1) = 0.7435.$$

**4.**

**a.** The transfer function $G(s)$ is given by

$$G(s) = C(sI - A)^{-1}B = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} s & 1 \\ -1 & s \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{s^2 + 1}$$

The poles are thus located at $s = \pm i$

**b.** Approximating $H(z)$ using forward difference is done with $H(z) = G(s')$ where $s' = \frac{z-1}{h}$. The pulse transfer function is thus given by

$$H(z) = G(s') = \frac{1}{(s')^2 + 1} = \frac{1}{(\frac{z-1}{h})^2 + 1} = \frac{h^2}{z^2 - 2z + 1 + h^2}$$

The poles are thus given by

$$z = 1 \pm \sqrt{h^2} = 1 \pm ih.$$

The poles can therefore never be located inside (or on) the unit circle for any $h > 0$. The poles are thus never located within the unit-circle.

**c.** Approximating $H(z)$ using backward difference one instead uses $H(z) = G(s')$ with $s' = \frac{z-1}{zh}$ leading to the following transfer function

$$H(z) = G(s') = \frac{1}{(\frac{z-1}{zh})^2 + 1} = \frac{z^2 h^2}{z^2 - 2z + 1 + h^2 z^2} = \frac{z^2 \frac{h^2}{1+h^2}}{z^2 - \frac{2}{1+h^2} z + \frac{1}{1+h^2}}$$

The backward difference maps the left complex half-plane into a circle that lies completely inside the unit circle. Hence, the poles are always located inside the unit circle.

**5.**

**a.** In a $Q\,m.n$ representation, $m$ bits are dedicated to the integer part, and $n$ bits are devoted to the representation of the decimal part. Additionally, 1 bit is used for the sign.

You need to use at least 6 bits for the integer part. In fact, $2^6 = 64$, while $2^5 = 32$. If the resolution should be at least 0.01, you need 7 bits for the decimal part. In fact, $2^{-7} = 0.078$, while $2^{-6} = 0.0156$, therefore the first number of bits that guarantees a resolution equal or lower than 0.01 is 7. The smallest number of bits needed for the representation of the number is therefore $1 + 6 + 7 = 14$. The number is represented using a $Q\,6.7$ notation.

The constant $k_p = 4.5$ can be converted to its corresponding fixed point representation using $round(4.5 \cdot 2^7) = 576$. Its binary representation is 00001001000000.

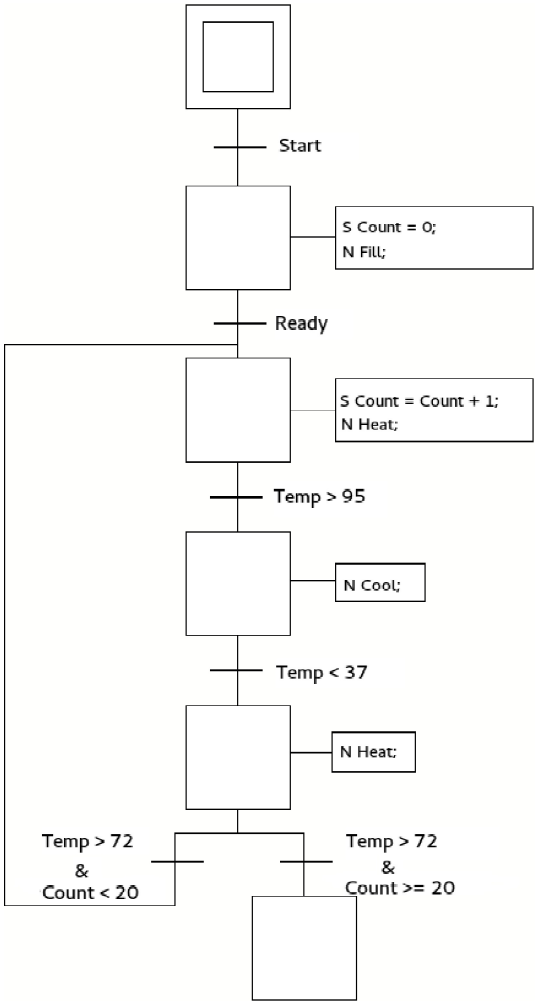



**Figure 1** Solution to Problem **??**.

**b.** Given the need of at least 14 bits, a processor with words of 16 bits is enough for the given problem. The type `short` – or `int16_t` – can be used to represent the given data.

**6.** A possible solution is shown in Figure 1.

**7.**

**a.** The intention behind this question was to ask when purely oscillatory behavior occurs. The answer to this is that in discrete-time purely oscillatory behavior occurs for systems with eigenvalues on the unit circle.

However, as the question was posed the answer is that oscillatory behaviour occurs in all cases except when the eigenvalues are on the positive real axis.

**b.** Yes it is possible. In discrete-time purely oscillatory behavior occurs for first-order systems with the eigenvalue in $-1$.

**8.**

**a.** The problem is that the limitation that the WriteOutput function provides when the argument is too large or too small is not known to the observer. Hence, the problem arises whenever the control signal is limited by the actuator, e.g., for large reference changes or large disturbances.

**b.** The correct pseudo-code implementation is

```
while (true) {
 y = ReadInput();
 ref = getReference;
 u = lr*ref - L*x_hat;
 u = limit(u,u_min,u_max);
 WriteOutput(u);
 x_hat = (Phi - K*C)*x_hat + Gamma*u + K*y;
 //sleep
}
```

**9.**

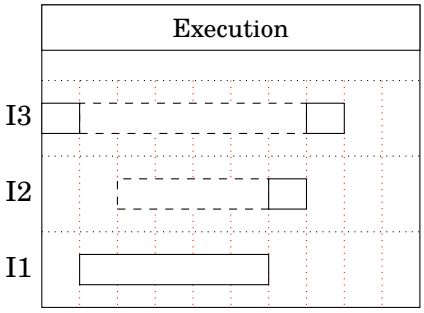
```
public class Semaphore {
    private boolean s;

    public synchronized void give() {
        s = false;
        notify();
    }

    public synchronized void take() {
        while (s) {
            try {
                wait();
            }
            catch(Exception e) {}
        }
        s = true;
    }

    public synchronized boolean tryTake() {
        if (s) {
            return false;
        }
        else {
            s = true;
            return true;
        }
    }
}
```

**10.**

**a.** The periods of the tasks are multiple of each other. The most critical situation happens when all the tasks are triggered at the same time (every 100 $ms$). In this case, the interrupt levels determine which task is executed before the others. The schedule is therefore I1, I2 and then I3. I1 will not miss

its deadline, terminating with an execution time of $5\,ms$. I2 has a period of $100\,ms$ but in order for it to not interfere with I3, it should terminate giving to I3 enough time to complete before the next activation ($5\,ms$ after the termination of I1). Given that the duration of I3 is $2\,ms$, the maximum duration of I2 to guarantee all the deadlines is $3\,ms$.

**b.** At time $0\,ms$ interrupt I3 is triggered. Its handler executes for $1\,ms$ and then there is a trigger of interrupt I1, which has higher priority and is therefore executed first. For the next $5\,ms$ (from time $1\,ms$ to time $6\,ms$) I1 is executed and terminated. Meanwhile, at time $2\,ms$ I2 was triggered, but its handler did not start its execution because a higher priority handler was executing. At time $6\,ms$ the handler for I2 starts executing and terminates within $1\,ms$. The duration the execution of I2 comprehends the time from the triggering time ($2\,ms$) to the termination of the handler ($7\,ms$), therefore being $5\,ms$. Finally, at time $7\,ms$ the execution of the handler of I3 is restored, and the handler completes when its remaining duration ($1\,ms$) has passed. The total execution time for the handler of I3 is therefore from time $0\,ms$ to time $8\,ms$ with a duration of $8\,ms$.




Since the total execution has been less than the period ($10\,ms$), the same exact conditions are repeated every $10\,ms$. The mentioned response times are therefore identical for all the executions. The following drawing indicates the schedule, where dashed part denotes that the interrupt is ready to be executed but not executed and the solid lines denote execution.

**11.** A backward approximation of the I-part gives the result

```
I(k) = I(k-1) + (Kh/Ti)e(k)
```

A forward approximation of the D-part gives the result

$$\frac{T_d}{N}\frac{D(t_{k+1}) - D(t_k)}{h} + D(t_k) = -KT_d\frac{y(t_{k+1}) - y(t_k)}{h}$$

$$D(t_{k+1}) = \frac{T_d - Nh}{T_d}D(t_k) - NK(y(t_{k+1}) - y(t_k))$$

The code below shows the newimplementation.

```
// Code executed in the Regul thread

y := AIIn(ychan); // Sample y
e := yref - y;
```

```
I := I + ki*e;
D := D - kd*y;
v := K*(beta*yref - y) + I + D;
u := sat(v,umax,umin)
DaOut(u,uchan); // Output u
D := ((Td-N*h)/Td)*D + N*K*y;
I := I + (h/Tr)*(u - v);


// Code that is executed only when a controller parameter is updated
ki := K*h/Ti;
kd := N*K;
```