Solutions to the exam in Real-Time Systems 131217

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

1. Alternative A matches the alternative semaphore implementation. In `signal` the counter is always updated and the first process in the waiting queue is released. In `wait` a process that is woken up again is forced to recheck the counter value through the infinite loop construct.

Alternative B matches the ordinary semaphore implementation. In `signal` the first process in the waiting queue is released. The counter is only updated if there is no other waiting task. In `wait` a process that is woken up again can proceed directly and enter the critical section.

2.

a. To find $y(3)$ we iterate the system equation.

$$x(1) = \Phi x(0) + \Gamma u(0) = \begin{bmatrix} 0.75 & 0.1 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} 5 = \begin{bmatrix} 5 \\ 10 \end{bmatrix}$$
$$x(2) = \Phi x(1) + \Gamma u(1) = \begin{bmatrix} 0.75 & 0.1 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 5 \\ 10 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} 5 = \begin{bmatrix} 9.75 \\ 15 \end{bmatrix}$$
$$x(3) = \Phi x(2) + \Gamma u(2) = \begin{bmatrix} 0.75 & 0.1 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 9.75 \\ 15 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} 5 = \begin{bmatrix} 13.8125 \\ 17.5 \end{bmatrix}$$

Then we calculate the output of the system using

$$y(3) = Cx(3) = \begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} 13.8125 \\ 17.5 \end{bmatrix} = 48.8125$$

b. To find out if the system will converge we must check that it is stable. A discrete linear time-invariant system is stable if the poles of the pulse transfer function (or equivalently the eigenvalues of the dynamics matrix) all are inside the unit circle. The pulse transfer function of the system is

$$H(z) = C(zI - \Phi)^{-1}\Gamma = \frac{5z - 3.3}{z2 - 1.25z + 0.375}$$

and the system has its two poles in 0.5 and 0.75 and is, hence, stable. The system will eventually converge to the value $\lim_{t\to\infty} y(t) = H(1) * 5 = \frac{1.7}{0.125} * 5 = 68$.

3.

a. The observer gain matrix K is given by the equation $det(zI - (\Phi - KC)) = det(\begin{bmatrix} z-2+k_1 & -1 \\ k_2 & z-1 \end{bmatrix}) = (z-2+k_1)(z-1)+k_2 = z^2+(k_1-3)z+2-k_1+k_2 = (z-p_1)(z-p_2) = z^2$. This gives us $K = \begin{bmatrix} 3 & 1 \end{bmatrix}^T$.

b. The state feedback matrix L is given by the equation $det(zI - (\Phi - BL)) = z^2$. This gives us $L = \begin{bmatrix} 8 & 6 \end{bmatrix}$.

Inserting the control law in the state space system gives us

$$x(k+1) = \Phi x(k) + \Gamma(-Lx(k) + l_r r(k)) = (\Phi - \Gamma L)x(l) + \Gamma l_r r(k).$$

The pulse transfer function of this system is

$$H(z) = C(zI - (\Phi - \Gamma L))^{-1}\Gamma l_r = \frac{l_r}{2z^2}.$$

$H(1) = 1$ gives us $l_r = 2$.

c.
```
private double x1Hat = 0, x2Hat = 0, u = 0, y = 0, r = 0, e = 0;
private double uHelp = 0;

private final double k1 = 3, k2 = 1;
private final double l1 = 8, l2 = 6, lr = 2;
private final double phi11 = 2, phi12 = 1, phi 22 = 1;
private final double gam2 = 0.5;
private final double c1 = 1;

while (1) {
  y = getY();
  r = getReference();

  u = uHelp + lr*r;

  outputU(u);

  e = y - c1*x1Hat;
  x1HatNew = phi11*x1Hat + phi12*x2Hat + k1*e;
  x2HatNew = phi22*x2Hat + gam2*u + k2*e;

  x1Hat = x1HatNew;
  x2Hat = x2HatNew;

  uHelp = -l1*x1Hat - l2*x2Hat;

  sleep();
}
```

4.

a. All of them! Since the controllers are rate monotonic and have higher priority than the logger, the choice of T_{logger} will not affect the controllers. The two controller tasks will both meet their deadlines which can be seen from the approximate analysis (may be used since $T_i = D_i$), i.e.,

$$\frac{2}{8} + \frac{4}{12} = 0.583 \leq 2(2^{1/2} - 1) = 0.828$$

b. The first schedulability test gives that if $T_{logger} \geq \frac{5}{3(2^{1/3}-1)-\frac{2}{8}-\frac{4}{12}} \approx 25.45$, the logger is guaranteed to meet its dealines. Otherwise, the logger may not meet its deadlines. Since $T_{logger} > T_{Controller2}$ it is OK to use the approximate rate monotonic analysis.

The second schedulability test gives that

$$(\frac{2}{8} + 1)(\frac{4}{12} + 1)(\frac{5}{T_{logger}} + 1) \leq 2$$

From this one can show that $T_{logger} \geq 25$. This is also the least restrictive condition. Hence, the answer should be $T_{logger} \geq 25$.

c. Since the logger has the lowest priority it is sufficient if we do the response time analysis for this task. We then get

$$
\begin{aligned}
R^0 &= 0 \\
R^1 &= C = 5 \\
R^2 &= 5 + \left\lceil \frac{5}{8} \right\rceil 2 + \left\lceil \frac{5}{12} \right\rceil 4 = 11 \\
R^3 &= 5 + \left\lceil \frac{11}{8} \right\rceil 2 + \left\lceil \frac{11}{12} \right\rceil 4 = 13 \\
R^4 &= 5 + \left\lceil \frac{13}{8} \right\rceil 2 + \left\lceil \frac{13}{12} \right\rceil 4 = 17 \\
R^5 &= 5 + \left\lceil \frac{17}{8} \right\rceil 2 + \left\lceil \frac{17}{12} \right\rceil 4 = 19 \\
R^6 &= 5 + \left\lceil \frac{19}{8} \right\rceil 2 + \left\lceil \frac{19}{12} \right\rceil 4 = 19
\end{aligned}
$$

Hence, if we set the deadline and the period of the logger task larger or equal to 19 then all the tasks will meet their deadline.

5.

a. With the forward difference we approximate s with $s' = \frac{z-1}{h}$. This gives us the filter $\frac{2\pi f_c h}{z+2\pi f_c h-1}$. For $f_c = 20$Hz we get a discrete pole on the positive real axis (in 0.9987) , which approximates a low pass filter well. For $f_c > \frac{1}{2\pi h} \approx 15$ kHz we get a discrete pole on the negative real axis which will cause the filter output to switch sign around the filters steady state value. The pole for $f_c = 20$kHz lies at -0.309. This is a bad approximation of this first order low-pass filter which will never get this ringing behaviour.

b. With Tustin's approximation we instead approximate s with $s' = \frac{2(z-1)}{h(z+1)}$. This gives us the filter $\frac{(\pi f_c h)z+\pi f_c h}{(\pi f_c h+1)z+\pi f_c h-1}$ with the pole in $-\frac{\pi f_c h-1}{\pi f_c h+1}$. For $f_c = 20$Hz the pole will be at 0.9987 and for $f_c = 20$kHz the pole will be at 0.2088, i.e. the pole of the filter will always be on the positive real axis.

c. With $\gamma = \pi f_c h$, we get the difference equation $((\gamma+1)z + \gamma - 1)y(k) = (\gamma z + \gamma)u(k)$ or $y(k+1) = \frac{\gamma u(k+1)+\gamma u(k)-(\gamma-1)y(k)}{\gamma+1}$. The implementation is then

```
public static final double h = 1.0 / 96000;
private double uOld = 0, yOld = 0;
public double tustinLP(double u, double fc) {
   double y;
   double gamma = Math.PI*h*fc;
   y = (gamma*(u + uOld - yOld) + yOld)/(gamma+1);
   yOld = y;
   uOld = u;
```

```
    return y;
}
```

6.

a. The transfer function from u_c to y is

$$H_{yu_c} = \frac{P(H_{ff} + CH_m)}{1 + PC}$$

Choose

$$H_{ff} = \frac{H_m}{P}$$

Then

$$H_{yu_c} = \frac{P(\frac{H_m}{P} + CH_m)}{1 + PC} = H_m$$

b. In order for H_{ff} to be implementable it must be causal and stable. It is causal if the order of the denominator is larger than or equal to the order of the numerator. It is (asymptotically) stable if the poles are inside the unit circle. Since

$$H_{ff}(z) = \frac{H_m}{P} = \frac{\frac{B_m(z)}{A_m(z)}}{\frac{B(z)}{A(z)}} = \frac{B_m(z)A(z)}{A_m(z)B(z)}$$

The constraint on causality gives that the $H_m(z)$ must have at least the same pole excess as P. The constraint on stability gives that any unstable zeros of $P(z)$, i.e., the unstable parts of $B(z)$, must also be included in $H_m(z)$, i.e., be part of $B_m(z)$.

c. The reason for the ringing is zero in -0.9672, i.e., on the negative real axis. The transfer function, H_{uu_c} from u_c to u is given by

$$H_{uu_c} = \frac{P(\frac{H_m}{P} + CH_m)}{P(1 + PC)} = \frac{H_m}{P}$$

Hence, the poorly damped zero of P is the cause of the ringing.

The problem is avoided by including not only the unstable parts of $B(z)$ in H_m, but also poorly damped zeros and zeros on the negative real axis. This can be done by modifying H_m to

$$H_m(z) = \frac{0.034147(z + 0.9672)}{(z - 0.7408)^2}$$

7. The solution below does not handle spurious wakeups.

```
public class CyclicBarrier {

  private int parties;
  private int barrierLimit;

  public CyclicBarrier(int parties) {
    this.parties = parties;
```

```
    barrierLimit = parties;
  }

  public synchronized int await() {
    int threadNum:
    parties = parties - 1;
    threadNum = parties;
    if (parties == 0) {
      parties = barrierLimit;
      notifyAll();
      return threadNum;
    } else {
        wait(); // Here the exception handling around wait() has been omitted
      }
      return threadNum;
    }
  }
}
```