Real-Time Systems

Solutions to final exam March 12, 2018 (version 20180312)

PROBLEM 1

a) False: Priority inversion occurs when a higher-priority task cannot execute (because another task holds a resource that the higher-priority task needs) and a lower-priority task is able to execute instead (thereby invalidating the priority mechanism).

b) False: A systematic time skew in the execution of a periodic task is caused by incorrect methods (e.g., by using relative `delay` statements) to calculate the arrival time of the next task instance. Systematic time skew is eliminated by using baselines and calculating new arrival times in relation to the current baseline (e.g., using `AFTER` in TinyTimber or `delay until` in Ada95).

c) True: Only so-called *number problems* may possibly have pseudo-polynomial time complexity. The special property of a number problem is that the largest number in the problem cannot be bounded by the input length (size) of the problem.

d) False: The GCC cross compiler that is used in the laboratory assignment generates code for the ARM Cortex-M4 processor core.

e) False: The utilization guarantee bound for RM-US converges towards 33.3% as the number of processors become very large.

f) False: If we know that the task set is not schedulable then a necessary test can either result in either the outcome 'True' or the outcome 'False'. This is because a necessary test can result in the outcome 'True' even though the task set is not schedulable.

PROBLEM 2

a) The message transmission delay consists of:

Frame delay: $t_{frame} = N_{frame}/R$, where N_{frame} is the message length (in bits) and R is the data rate (in bits/s).

Propagation delay: $t_{prop} = L/v$, where L is the communication distance (in m) and v is the signal propagation velocity (in m/s).

b) Message queuing delay for Ethernet can in general not be bounded because a re-transmission will happen every time a collision on the communication medium occurs.

c) The binary countdown protocol:

1. Each node with a pending message waits until bus is idle.
2. The node begins transmitting the highest-priority message pending on the node. Identifier is transmitted first, in the order of most-significant bit to least-significant bit.
3. If a node transmits a recessive bit (1) but sees a dominant bit (0) on the bus, then it stops transmitting since it is not transmitting the highest-priority message in the system.
4. The node that transmits the last bit of its identifier without detecting a bus inconsistency has the highest priority and can start transmitting the body of the message.

PROBLEM 3

a) In order to find the WCET of `main`, we need to find the WCET for the `calculateB` and `calculateA` functions.

The WCET of `calculateB(x)` is as follows:

$$
\begin{aligned}
&WCET(calculateB(x)) \\
&= \{declare, temp\} + \{sub, x-1\} + \{call, calculateA(x-1)\} + WCET(calculateA(x-1)) + \\
&\{call, calculateA(x)\} + WCET(calculateA(x)) + \{multiply, x * calculateA(x)\} + \\
&\{add, (x * calculateA(x)) + calculateA(x-1))\} + \{assign, temp\} + \{return, temp\} \\
&= 1+3+2+2+5+3+1+2+WCET(calculateA(x)) + WCET(calculateA(x-1)) \\
&= 19 + WCET(calculateA(x)) + WCET(calculateA(x-1))
\end{aligned}
$$

The WCET of `calculateA(x)` is derived based on three cases of the value of parameter x.

Case 1: $x = 0$:

$$
\begin{aligned}
&WCET(calculateA(x=0)) \\
&= \{declare, temp\} + \{compare, x=0\} + \{assign, temp\} + \{return, temp\} \\
&= 1+2+1+2 = 6
\end{aligned}
$$

Case 2: $x = 1$:

$$
\begin{aligned}
&WCET(calculateA(x=1)) \\
&= \{declare, temp\} + \{compare, x=0\} + \{compare, x=1\} + \{assign, temp\} + \{return, temp\} \\
&= 1+2+2+1+2 = 8
\end{aligned}
$$

Case 3: $x > 1$:

$$
\begin{aligned}
&WCET(calculateA(x>1)) \\
&= \{declare, temp\} + \{compare, x=0\} + \{compare, x=1\} + \\
&\{sub, x-1\} + \{call, calculateA(x-1)\} + WCET(calculateA(x-1)) + \{multiply, x * calculateA(x-1)\} + \\
&\{assign, temp\} + \{return, temp\} \\
&= 1+2+2+3+2+5+1+2+WCET(calculateA(x-1)) \\
&= 18 + WCET(calculateA(x-1))
\end{aligned}
$$

The WCET of the `main` function is as follows:

$$
\begin{aligned}
&WCET(main) \\
&= \{declare, N\} + \{declare, ans\} + \{assign, N\} + \\
&\{call, calculateB(N)\} + WCET(calculateB(N)) + \{assign, ans\} \\
&= 1+1+1+2+1+WCET(calculateB(N) \\
&= 6 + WCET(calculateB(N))
\end{aligned}
$$

Therefore, for $N = 2$, the WCET of **main** is

$$
\begin{aligned}
&WCET(main) \\
&= 6 + WCET(calculateB(2)) \\
&= 6 + (19 + WCET(calculateA(2)) + WCET(calculateA(1))) \\
&= 25 + WCET(calculateA(2)) + WCET(calculateA(1)) \\
&= 25 + (18 + WCET(calculateA(1))) + WCET(calculateA(1)) \\
&= 43 + 2 \times WCET(calculateA(1)) \\
&= 43 + 2 \times 8 \\
&= 43 + 16 = 59\mu s
\end{aligned}
$$

Since the deadline of `main` is 65 μs, the deadline is met.

b) The difference between the deadline and completion time of `main`, called slack time, is 65-59=6 μs. There are in total 2 multiply operations: 1 multiply to calculate `calculateA(2)` and 1 multiply to calculate `x*calculateA(2)` in the `calculateB(2)` function.

The slack time (6 μs) can be used to increase the cost of the multiply operation. The cost of each multiply operation can be increased by (slack time/number of multiplies)=6/2=3 μs and the main program can still meet the deadline. Therefore, the maximum cost of the multiply operation is (5+3)=8 μs.

c) WCET estimates must be pessimistic to make sure assumptions made in the schedulability analysis of hard real-time tasks also apply at run time. WCET estimates must be tight to avoid unnecessary waste of resources during scheduling of hard real-time tasks.

PROBLEM 4

A compact (table-based) solution could look similar to this:

```
#include "TinyTimber.h"

typedef struct {
    Object super;
    int Nhyper;   // number of executions within hyper period
    int table[7]; // baseline offset for next scheduled execution
                  // index 0: offset for very first execution (relative to time 0)
                  // index k: offset for execution k+1  (modulo Nhyper)
} PeriodicTask;

Object app = initObject();

PeriodicTask ptask1 = { initObject(), 6, {0,10,10,10,10,10,10} };
PeriodicTask ptask2 = { initObject(), 5, {8,10,10,18,12,10,0} };
PeriodicTask ptask3 = { initObject(), 4, {6,20,10,20,10,0,0} };


void T1(PeriodicTask *self, int u) {
    Action6ms(); // procedure doing time-critical work

    AFTER(MSEC(self->table[u]), self, T1, (u % self->Nhyper) + 1);
}

void T2(PeriodicTask *self, int u) {
    Action1ms(); // procedure doing time-critical work

    AFTER(MSEC(self->table[u]), self, T2, (u % self->Nhyper) + 1);
}

void T3(PeriodicTask *self, int u) {
    Action2ms(); // procedure doing time-critical work

    AFTER(MSEC(self->table[u]), self, T3, (u % self->Nhyper) + 1);
}

void kickoff(Object *self, int u) {
    AFTER(MSEC(ptask1.table[0]), &ptask1, T1, 1);
    AFTER(MSEC(ptask2.table[0]), &ptask2, T2, 1);
    AFTER(MSEC(ptask3.table[0]), &ptask3, T3, 1);
}

main() {
    return TINYTIMBER(&app, kickoff, 0);
}
```

PROBLEM 5

There is only one utilization-based feasibility test for global scheduling, namely the one for RM-US. Since we assume deadline-monotonic scheduling that test cannot be used. We thus need to make a more detailed schedulability analysis, beginning with response-time analysis for global scheduling.

With deadline-monotonic scheduling the static task priorities are as follows:

$prio(\tau_1) = \mathrm{H}$, $prio(\tau_2) = \mathrm{L}$, $prio(\tau_3) = \mathrm{M}$.

With the given priorities tasks τ_1 and τ_3 will run undisturbed on one processor each, and are obviously schedulable since $C_i < D_i$ for these tasks. The response time of τ_2 is then derived using the following formula:

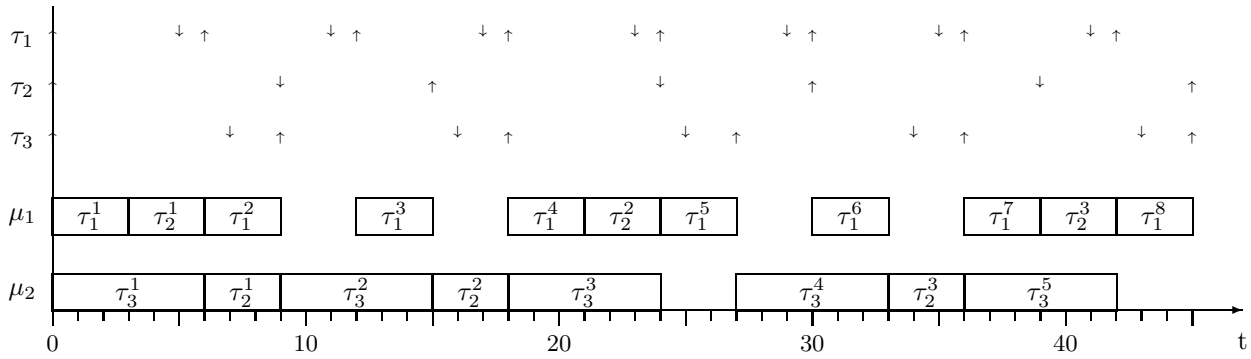
$R_2 = C_2 + \frac{1}{m}((\lceil \frac{R_2}{T_1} \rceil \cdot C_1 + C_1) + (\lceil \frac{R_2}{T_3} \rceil \cdot C_3 + C_3))$. Assume that $R_2^0 = C_2 = 6$:

$R_2^1 = 6 + \frac{1}{2}((\lceil \frac{6}{6} \rceil \cdot 3 + 3) + (\lceil \frac{6}{9} \rceil \cdot 6 + 6)) = 6 + \frac{1}{2}((1 \cdot 3 + 3) + (1 \cdot 6 + 6)) = 6 + \frac{1}{2}(6 + 12) = 6 + 9 = 15$

The derived response-time of τ_2 exceeds the deadline D_2 (= 9), so the test fails. Since the response-time test for global scheduling is only sufficient, we cannot yet determine the schedulability of the task set.

Our final option is then hyper period analysis: $\mathrm{LCM}\{T_1, T_2, T_3\} = \mathrm{LCM}\{6, 15, 9\} = 90$.

We generate a multiprocessor schedule with tasks τ_1 and τ_3 (having the highest priorities) running on one processor each. Task τ_2 is scheduled in the remaining time slots according to the following diagram (covering the first half of the hyper period, that is $t = 0$ to $t = 45$):



We observe that there is a critical instant at $t = 30$ in the diagram, maximizing the response time of τ_2 ($R_3 = 12$). Since $R_2 > D_2$ the schedule is not feasible.

Note that $t = 0$ is not a critical instant (which it would have been for a single-processor system)!

PROBLEM 6

Since deadline-monotonic scheduling is used, the static task priorities are as follows:

$prio(\tau_1) = \mathrm{H}$, $prio(\tau_2) = \mathrm{M_H}$, $prio(\tau_3) = \mathrm{M_L}$, $prio(\tau_4) = \mathrm{L}$.

We can now determine the ceiling priority for each semaphore:

$\mathrm{ceil}\{S_a\} = \max\{\mathrm{H}, \mathrm{M_H}\} = \mathrm{H}$ (since τ_1 och τ_2 may lock the semaphore)

$\mathrm{ceil}\{S_b\} = \max\{\mathrm{M_H}, \mathrm{M_L}\} = \mathrm{M_H}$ (since τ_2 och τ_3 may lock the semaphore)

$\mathrm{ceil}\{S_c\} = \max\{\mathrm{H}, \mathrm{L}\} = \mathrm{H}$ (since τ_1 och τ_4 may lock the semaphore)

We then identify, for each task τ_i, what tasks with lower priority may block τ_i and thereby cause the corresponding blocking factor B_i:

$B_1 = \max\{H_{2,a}, H_{4,c}\} = \max\{1, H_{4,c}\}$ (since τ_1 may be blocked by τ_2 and τ_4 who lock semaphores whose ceiling priorities are higher than or equal to the priority of τ_1)

$B_2 = \max\{H_{3,b}, H_{4,c}\} = \max\{3, H_{4,c}\}$ (since τ_2 may be blocked by τ_3 and τ_4 who lock semaphores whose ceiling priorities are higher than or equal to the priority of τ_2)

$B_3 = \max\{H_{4,c}\} = H_{4,c}$ (since τ_3 may be blocked by τ_4 who locks a semaphore whose ceiling priority is higher than or equal to the priority of τ_3)

$B_4 = 0$ (since τ_4 has lowest priority of all tasks, and thereby per definition cannot be subject to blocking)

We can now observe that the time $H_{1,c}$ does not affect the scheduling of the other tasks except through the execution time of τ_1. Since the time for using a resource is included in the normal execution time, we have $H_{1,c} \leq C_1 = 2$ (or, to be even more accurate, we have $H_{1,c} \leq C_1 - H_{1,a} = 2 - 1 = 1$).

On the other hand, the time $H_{4,c}$ does affect the schedulability of τ_1, τ_2 and τ_3 since that time is included in the blocking factors of these tasks. We therefore calculate the task response times and check whether they are less than or equal to the corresponding deadline:

$R_1 = C_1 + B_1 = 2 + H_{4,c} \leq D_1 = 6.$

This means that $H_{4,c} \leq D_1 - C_1 = 4$.

$R_2 = C_2 + B_2 + \lceil \frac{R_2}{T_1} \rceil \cdot C_1$. Assume that $B_2 = \max\{3, H_{4,c}\} = 4$ and $R_2^0 = C_2 = 3$:

$R_2^1 = 3 + 4 + \lceil \frac{3}{6} \rceil \cdot 2 = 3 + 4 + 1 \cdot 2 = 9$

$R_2^2 = 3 + 4 + \lceil \frac{9}{6} \rceil \cdot 2 = 3 + 4 + 2 \cdot 2 = 11$

$R_2^3 = 3 + 4 + \lceil \frac{11}{6} \rceil \cdot 2 = 3 + 4 + 2 \cdot 2 = 11 < D_2 = 12.$

This means that it still applies that $H_{4,c} \leq 4$.

$R_3 = C_3 + B_3 + \lceil \frac{R_3}{T_1} \rceil \cdot C_1 + \lceil \frac{R_3}{T_2} \rceil \cdot C_2$. Assume that $B_3 = H_{4,c} = 4$ and $R_3^0 = C_3 = 3$:

$R_3^1 = 3 + 4 + \lceil \frac{3}{6} \rceil \cdot 2 + \lceil \frac{3}{16} \rceil \cdot 3 = 3 + 4 + 1 \cdot 2 + 1 \cdot 3 = 12$

$R_3^2 = 3 + 4 + \lceil \frac{12}{6} \rceil \cdot 2 + \lceil \frac{12}{16} \rceil \cdot 3 = 3 + 4 + 2 \cdot 2 + 1 \cdot 3 = 14$

$R_3^3 = 3 + 4 + \lceil \frac{14}{6} \rceil \cdot 2 + \lceil \frac{14}{16} \rceil \cdot 3 = 3 + 4 + 3 \cdot 2 + 1 \cdot 3 = 16 > D_3 = 15$

This means that our assumption that $H_{4,c} = 4$ does not hold. Instead, we try $H_{4,c} = 3$:

$R_3^1 = 3 + 3 + \lceil \frac{3}{6} \rceil \cdot 2 + \lceil \frac{3}{16} \rceil \cdot 3 = 3 + 3 + 1 \cdot 2 + 1 \cdot 3 = 11$

$R_3^2 = 3 + 3 + \lceil \frac{11}{6} \rceil \cdot 2 + \lceil \frac{11}{16} \rceil \cdot 3 = 3 + 3 + 2 \cdot 2 + 1 \cdot 3 = 13$

$R_3^3 = 3 + 3 + \lceil \frac{13}{6} \rceil \cdot 2 + \lceil \frac{13}{16} \rceil \cdot 3 = 3 + 3 + 3 \cdot 2 + 1 \cdot 3 = 15$

$R_3^4 = 3 + 3 + \lceil \frac{15}{6} \rceil \cdot 2 + \lceil \frac{15}{16} \rceil \cdot 3 = 3 + 3 + 3 \cdot 2 + 1 \cdot 3 = 15 \leq D_3 = 15$

This means that it must apply that $H_{4,c} \leq 3$.

$R_4 = C_4 + B_4 + \lceil \frac{R_4}{T_1} \rceil \cdot C_1 + \lceil \frac{R_4}{T_2} \rceil \cdot C_2 + \lceil \frac{R_4}{T_3} \rceil \cdot C_3$. Assume that $R_4^0 = C_4 = 4$:

$R_4^1 = 4 + \lceil \frac{4}{6} \rceil \cdot 2 + \lceil \frac{4}{16} \rceil \cdot 3 + \lceil \frac{4}{20} \rceil \cdot 3 = 4 + 1 \cdot 2 + 1 \cdot 3 + 1 \cdot 3 = 12$

$R_4^2 = 4 + \lceil \frac{12}{6} \rceil \cdot 2 + \lceil \frac{12}{16} \rceil \cdot 3 + \lceil \frac{12}{20} \rceil \cdot 3 = 4 + 2 \cdot 2 + 1 \cdot 3 + 1 \cdot 3 = 14$

$R_4^3 = 4 + \lceil \frac{14}{6} \rceil \cdot 2 + \lceil \frac{14}{16} \rceil \cdot 3 + \lceil \frac{14}{20} \rceil \cdot 3 = 4 + 3 \cdot 2 + 1 \cdot 3 + 1 \cdot 3 = 16$

$R_4^4 = 4 + \lceil \frac{16}{6} \rceil \cdot 2 + \lceil \frac{16}{16} \rceil \cdot 3 + \lceil \frac{16}{20} \rceil \cdot 3 = 4 + 3 \cdot 2 + 1 \cdot 3 + 1 \cdot 3 = 16 < D_4 = 28$

The system consequently meets all of its deadlines if $H_{1,c} \leq 1$ and $H_{4,c} \leq 3$.

PROBLEM 7

We begin by calculating the utilization U_i for each task:

	C_i	T_i	U_i
τ_1	10	25	0.4
τ_2	18	40	0.45
τ_3	2	10	0.2
τ_4	$T_4/3$	T_4	1/3
τ_5	1	8	0.125

a) The Oh & Baker utilization guarantee bound for partitioned scheduling is $\mathrm{U_{RMFF}} = m(2^{1/2} - 1)$, where m is the number of processors.

For the given system, with $m = 2$, we have: $\mathrm{U_{RMFF}} = m \cdot (2^{1/2} - 1) = 2 \cdot (2^{1/2} - 1) \approx 0.828$

The total utilization of the task set is:

$\mathrm{U_{Total}} = 0.4 + 0.45 + 0.2 + 1/3 + 0.125 \approx 1.51$

Clearly, $\mathrm{U_{Total}} > \mathrm{U_{RMFF}}$ which means that the Oh & Baker test fails. However, since the test is only sufficient we cannot determine the schedulability of the task set.

b) Start by numbering the two processors μ_1 and μ_2.

According to the RMFF partitioning algorithm the tasks (temporarily excluding task τ_4, whose period is still unknown) should be assigned to the processors in the following order: $\tau_5, \tau_3, \tau_1, \tau_2$.

Based on this assignment order, we can see that three tasks can be assigned to processor μ_1, regardless of whether τ_4 is among those three or not. The Liu & Layland utilization guarantee bound for three tasks is $\mathrm{U_{RM(3)}} = n \cdot (2^{1/n} - 1) = 3 \cdot (2^{1/3} - 1) \approx 0.780$.

We have two cases, based on the relation between the periods of τ_4 and τ_1:

Case 1 ($T_4 < T_1$): Task τ_4 is assigned to μ_1. The utilization of the three assigned tasks is then $\mathrm{U} = U_5 + U_3 + U_4 = 0.125 + 0.2 + 1/3 \approx 0.658$, which is less than $\mathrm{U_{RM(3)}}$.

Case 2 ($T_4 > T_1$): Task τ_4 is not assigned to μ_1. The utilization of the three assigned tasks is then $\mathrm{U} = U_5 + U_3 + U_1 = 0.125 + 0.2 + 0.4 = 0.725$, which is also less than $\mathrm{U_{RM(3)}}$.

It is not possible to add a fourth task to processor μ_1 as the utilization of the assigned tasks would then exceed the corresponding Liu & Layland utilization guarantee bound. The remaining two tasks must therefore be assigned to processor μ_2. The Liu & Layland utilization guarantee bound for two tasks is $\mathrm{U_{RM(2)}} = n \cdot (2^{1/n} - 1) = 2 \cdot (2^{1/2} - 1) \approx 0.828$.

Again, looking at the two cases:

Case 1: The remaining two tasks to be assigned to μ_2 are tasks τ_1 and τ_2. The utilization of these tasks is $\mathrm{U} = U_1 + U_2 = 0.4 + 0.45 = 0.85$, which exceeds $\mathrm{U_{RM(2)}}$. This is consequently an infeasible assignment.

Case 2: The remaining two tasks to be assigned to μ_2 are tasks τ_4 and τ_2. The utilization of these tasks is $\mathrm{U} = U_4 + U_2 = 1/3 + 0.45 \approx 0.783$, which is less than $\mathrm{U_{RM(2)}}$. This is a feasible assignment.

Based on the reasoning above we saw that the task set can only be successfully scheduled on two processors if the period of task τ_4 is larger than the period of task τ_1, that is $T_4 > T_1 = 25$. The smallest integer value of $T_4 > 25$, that also satisfies the additional constraint that $C_4 = T_4/3$ must be an integer, is $T_4 = 27$. Then $C_4 = 9$.

The resulting assignment of tasks to processors is:

Processor μ_1: tasks τ_5, τ_3, and τ_1

Processor μ_2: tasks τ_4 and τ_2