Introduction to Real-Time Systems

Solutions to final exam May 31, 2017 (version 20170531)

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: For a sporadic task the time interval between two, subsequent, arrivals is guaranteed to never be less than a minimum value.

c) False: The response-time test for global fixed-priority scheduling is a sufficient feasibility test since one extra instance of each higher-priority task must be (pessimistically) accounted for in the interference analysis.

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

e) True: TinyTimber's AFTER() construct allows the programmer to call a method after a delay relative to the calling method's baseline, thereby eliminating any systematic time skew.

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 four conditions for deadlock is:

- Mutual exclusion – only one task at a time can use a resource
- Hold and wait – there must be tasks that hold one resource at the same time as they request access to another resource
- No preemption – a resource can only be released by the task holding it
- Circular wait – there must exist a cyclic chain of tasks such that each task holds a resource that is requested by another task in the chain

b) The basic idea of a priority ceiling protocol is as follows: Each resource is assigned a priority ceiling equal to the priority of the highest-priority task that can lock it. Then, a task τ_i is allowed to enter a critical region only if its priority is higher than all priority ceilings of the resources currently locked by tasks other than τ_i. When the τ_i blocks one or more higher-priority tasks, it temporarily inherits the highest priority of the blocked tasks.

PROBLEM 3

a) The WCET of `Control` is dependent on the WCET of function `Calc`.

WCET of "Calc":

$WCET(Calc(x)) =$
$\{Declare, i\} + \{Declare, r\} + \{Assign, i\}) + \{Assign, r\} +$
$(3+1) \cdot \{Compare, i < 3\} + 3 \cdot (\{Multiply, r * x\} + \{Assign, r\} + \{Add, i + 1\} + \{Assign, i\}) +$
$\{Subtract, r - 1\} + \{Assign, r\} + \{Return, r\} \ =$
$1 + 1 + 1 + 1 + 4 \cdot 2 + 3 \cdot (5 + 1 + 3 + 1) + 3 + 1 + 2 = 4 + 8 + 30 + 6 = 48$

WCET of "Control":

$WCET(Control) =$
$\{Declare, c\} + \{Declare, r\} + \{Assign, c\}) +$
$\{Call, Calc(c)\} + WCET(Calc(c)) + \{Divide, Calc(c)/3\} + \{Assign, r\} + \{Compare, r <= 800\} +$
$\max(\{Shift, r\} + \{Assign, r\}, \{Multiply, 3 * r\} + \{Divide, 3 * r/289\}) + \{Add, 3 * r/289 + 2\}) +$
$\{Assign, r\}) \ =$
$1 + 1 + 1 + 2 + WCET(Calc(c)) + 8 + 1 + 2 + \max(2 + 1, 5 + 8 + 3 + 1) + 1 =$
$17 + \max(3, 17) + WCET(Calc(c))$

The function `Calc(x)` calculates the polynomial $x^4 - 1$ which, with the given input port data range $[-9, +9]$, has the largest value $9^4 - 1 = 6560$. The comparison in the `if`-statement in `Control` then becomes $6560/3 = 2187 \leq 800$, which is false. Thus, the longer path in the `if`-statement will be executed.

$WCET(Control) = 17 + \max(3, 17) + WCET(Calc(c)) = 17 + 17 + 48 = 82 > 73$

The deadline is <u>not</u> met!

b) We notice that, if the shorter path in the `if`-statement would be executed, we get:

$WCET(Control) = 17 + 3 + WCET(Calc(c)) = 17 + 3 + 48 = 68 < 73$

Thus, in order to find the largest input port data range for which `Control` will meet its deadline we must make sure that the shorter path is always taken. This happens when $Calc(c)/3 \leq 800$, that is, when $Calc(c) \leq 2400$. Since `Calc(x)` calculates the polynom $x^4 - 1$ the largest permitted data range is $[-7, +7]$, since $7^4 - 1 = 2400$.

c) The new function `Calc(x)` is functionally compatible with the old function since $(x^2-1)((x^2-1)+2) = (x^2 - 1)(x^2 + 1) = x^4 - 1$. However, the WCET of the new function is significantly smaller:

$WCET(Calc(x)) =$
$\{Declare, r\} + \{Multiply, x * x\} + \{Subtract, x * x - 1\} + \{Assign, r\} +$
$\{Add, r + 2\} + \{Multiply, r * (r + 2)\} + \{Assign, r\} \ + \{Return, r\} \ =$
$1 + 5 + 3 + 1 + 3 + 5 + 1 + 2 = 21$

With the original input port data range $[-9, +9]$ we get:

$WCET(Control) = 17 + \max(3, 17) + WCET(Calc(c)) = 17 + 17 + 21 = 55 < 73$

The deadline is met!

PROBLEM 4

a) A compact solution could look similar to this:

```
#include TinyTimber.h

typedef struct {
    Object super;
    char *id;
} PeriodicTask;

Object app = initObject();

PeriodicTask ptask1 = { initObject(), "Task 1" };
PeriodicTask ptask2 = { initObject(), "Task 2" };

void T1(PeriodicTask *self, int u) {
    Action1(); // procedure doing time-critical work

    SEND(MSEC(95), MSEC(40), self, T1, 0);
}

void T2(PeriodicTask *self, int u) {
    Action2(); // procedure doing time-critical work

    SEND(MSEC(160), MSEC(85), self, T2, 0);
}

void kickoff(PeriodicTask *self, int u) {
    SEND(MSEC(0), MSEC(40), &ptask1, T1, 0);
    SEND(MSEC(70), MSEC(85), &ptask2, T2, 0);
}

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

b) The priorities for scheduled activities are given by the deadlines in SEND() or BEFORE() calls, since the TinyTimber kernel uses earliest-deadline-first scheduling.

c) If shared data is stored within an object, TinyTimber guarantees that mutual exclusion applies for the methods defined with the object if called with SYNC or ASYNC.

d) TinyTimber uses the Deadline Inheritance Protocol, combined with deadlock detection via the return value of the SYNC call.

PROBLEM 5

a) RM uses static task priorities that are assigned according to the following rule: tasks with shorter periods (= higher rate) get higher priority.

b) RM is an optimal priority assignment for preemptive scheduling on one processor if (i) priorities are static and (ii) deadline equals period for all tasks.

c) If the exact values of the task periods are not known, we only have access to the individual task utilizations. It may then seem natural to try to apply Liu & Layland's sufficient utilization-based test. Unfortunately, the test's utilization bound for two tasks ($\approx 83\%$) is significantly lower than the actual utilization of the tasks ($= 100\%$), which means that the test does not tell us anything regarding schedulability. Neither can we apply any type of detailed analysis within a hyper-period since we do know know which task has the highest priority. And even if we make the assumption that τ_1 has the highest priority (and thereby meets all of its deadlines), we still cannot answer our question because $C_1/T_1 + C_2/T_2 = 1 \rightarrow T_2C_1 + T_1C_2 = T_1T_2 \rightarrow T_2 = T_2/T_1C_1 + C_2$. Here, we can see that, in order to decide whether τ_2 is schedulable or not, we must know how many instances of τ_1 that interferes with the execution of τ_2 within T_2. If T_2/T_1 is not an integer number there is a risk that task τ_2 misses a deadline (e.g. for $T_2 = 10$ and $T_1 = 4$).

d) With the new information that $T_2 = 2T_1$, it is now possible to decide schedulability. First, we now know with certainty that task τ_1 has highest priority according to RM and thereby meets its deadlines. In addition, we know from sub-problem (c) that $T_2 = T_2/T_1C_1 + C_2$. Therefore, we can conclude that $T_2 = 2C_1 + C_2$. That is, we now know that, within T_2, there is room for exactly two instances of task τ_1 and one instance of task τ_2. It is then clear that it is possible to do an analysis within one hyper-period ($= T_2$) since the system is not overloaded, and the behavior of the tasks will be repeated for each new hyper-period. And, since we already know that $T_2 = T_2/T_1C_1 + C_2$, we also know that task τ_2 will be able to execute C_2 time units within its period. The system is consequently schedulable.

PROBLEM 6

a) The Liu & Layland utilization-based test for EDF cannot be used since it does not apply to all tasks that $D_i = T_i$.

b) Perform processor-demand analysis:

First, determine LCM of the task periods: $\text{LCM}\{T_1, T_2, T_3\} = \text{LCM}\{4, 10, 20\} = 20$.

Then, derive the set K of control points: $K_1 = \{4, 8, 12, 16, 20\}$, $K_2 = \{4, 14\}$ and $K_3 = \{15\}$ which gives us $K = K_1 \cup K_2 \cup K_3 = \{4, 8, 12, 14, 15, 16, 20\}$.

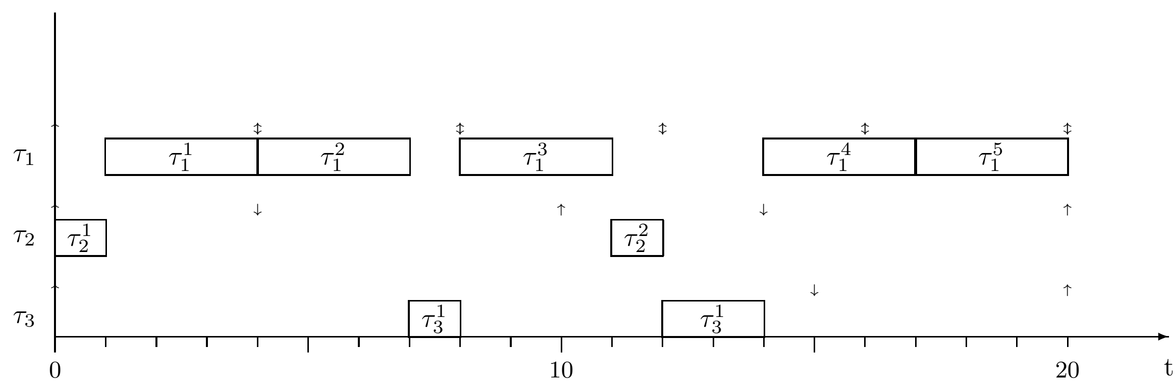
Schedulability analysis now gives us:

L	$N_1^L \cdot C_1$	$N_2^L \cdot C_2$	$N_3^L \cdot C_3$	$C_P(0,L)$	$C_P(0,L) \leq L$
4	$(\lfloor\frac{(4-4)}{4}\rfloor + 1) \cdot 3 = 3$	$(\lfloor\frac{(4-4)}{10}\rfloor + 1) \cdot 1 = 1$	$(\lfloor\frac{(4-15)}{20}\rfloor + 1) \cdot 3 = 0$	4	OK
8	$(\lfloor\frac{(8-4)}{4}\rfloor + 1) \cdot 3 = 6$	$(\lfloor\frac{(8-4)}{10}\rfloor + 1) \cdot 1 = 1$	$(\lfloor\frac{(8-15)}{20}\rfloor + 1) \cdot 3 = 0$	7	OK
12	$(\lfloor\frac{(12-4)}{4}\rfloor + 1) \cdot 3 = 9$	$(\lfloor\frac{(12-4)}{10}\rfloor + 1) \cdot 1 = 1$	$(\lfloor\frac{(12-15)}{20}\rfloor + 1) \cdot 3 = 0$	10	OK
14	$(\lfloor\frac{(14-4)}{4}\rfloor + 1) \cdot 3 = 9$	$(\lfloor\frac{(14-4)}{10}\rfloor + 1) \cdot 1 = 2$	$(\lfloor\frac{(14-15)}{20}\rfloor + 1) \cdot 3 = 0$	11	OK
15	$(\lfloor\frac{(15-4)}{4}\rfloor + 1) \cdot 3 = 9$	$(\lfloor\frac{(15-4)}{10}\rfloor + 1) \cdot 1 = 2$	$(\lfloor\frac{(15-15)}{20}\rfloor + 1) \cdot 3 = 3$	14	OK
16	$(\lfloor\frac{(16-4)}{4}\rfloor + 1) \cdot 3 = 12$	$(\lfloor\frac{(16-4)}{10}\rfloor + 1) \cdot 1 = 2$	$(\lfloor\frac{(16-15)}{20}\rfloor + 1) \cdot 3 = 3$	17	Not OK!
20	$(\lfloor\frac{(20-4)}{4}\rfloor + 1) \cdot 3 = 15$	$(\lfloor\frac{(20-4)}{10}\rfloor + 1) \cdot 1 = 2$	$(\lfloor\frac{(20-15)}{20}\rfloor + 1) \cdot 3 = 3$	20	OK

The processor demand in one of the strategic time intervals ($L = 16$) exceeds the length of that interval, so not all tasks will meet their deadlines.

c) From sub-problem b): $\text{LCM}\{4, 10, 20\} = 20$.

A simulation of the tasks using EDF scheduling in the interval $[0, LCM]$ gives the following timing diagram. Here, we see that task τ_1 misses its deadline at time $t = 16$.



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 $\text{U}_{\text{RMFF}} = m(2^{1/2} - 1)$, where m is the number of processors.

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

The total utilization of the task set is:

$\text{U}_{\text{Total}} = 0.4 + 0.45 + 0.2 + 1/3 + 0.125 \approx 1.51$

Clearly, $\text{U}_{\text{Total}} > \text{U}_{\text{RMFF}}$ 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 $\text{U}_{\text{RM(3)}} = n \cdot (2^{1/n} - 1) = 3 \cdot (2^{1/3} - 1) \approx 0.780$.

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 $\text{U} = U_5 + U_3 + U_4 = 0.125 + 0.2 + 1/3 \approx 0.658$, which is less than $\text{U}_{\text{RM(3)}}$.

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

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 $\text{U}_{\text{RM(2)}} = n \cdot (2^{1/n} - 1) = 2 \cdot (2^{1/2} - 1) \approx 0.828$.

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 $\text{U} = U_1 + U_2 = 0.4 + 0.45 = 0.85$, which exceeds $\text{U}_{\text{RM(2)}}$. 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 $\text{U} = U_4 + U_2 = 1/3 + 0.45 \approx 0.783$, which is less than $\text{U}_{\text{RM(2)}}$. 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