Real-Time Systems

Solutions to final exam March 13, 2017 (version 20170313)

PROBLEM 1

a) False: It is possible to construct a task set consisting of five (or more) tasks with harmonically related periods that has an accumulated utilization equal to 100% and is schedulable with RM. For example, a task set where each task has an execution time of 0.2 and a period of 1.

b) False: For a sporadic task there is a guaranteed minimum time interval between two subsequent arrivals.

c) False: For an NP-complete problem to have pseudo-polynomial time complexity the largest number in the problem cannot be bounded by the input length (size) of the problem

d) False: The critical instant refers to a point in time when the response time of an analyzed task is maximized. In single-processor system the critical instant occurs when the task arrives at the same time as all tasks with higher priority.

e) False: A false path is a piece of code that will never be executed when the program is running. For example, due to conditional statements in the code.

f) True: If we know that the task set is schedulable then a necessary test must have resulted in the outcome 'Yes'. This is because, for necessary tests, the outcome 'No' always means that 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

The pulse widths are determined by the relation between the values that are read from the input ports. Denote by I_1 the value read from `Inport1`. Denote by I_2 the value read from `Inport2`.

In the following solutions we assume that a write to the output port takes effect as the corresponding assignment operation ends.

a) The section of the program code that affects the width of the pulse on bit b_1 on the output port is:

```
while (c > 0)
   c = c - 1;

Outport = 0x00;   // end pulses on bits 0 and 1 (negative flanks)
```

Let C_2 be the value of variable `c` when this code section begins. The logical condition of the `while` statement will then be evaluated $C_2 + 1$ times, and the body of the `while` loop will be executed C_2 times. The pulse width W_{b_1} is then:

$$W_{b_1} = (C_2+1)\cdot\{\mathit{Compare},\, c > 0\} + C_2\cdot(\{\mathit{Subtract},\, c-1\} + \{\mathit{Assign},\, c\}) + \{\mathit{Assign},\, \mathit{Outport}\} =$$
$$(C_2+1)\cdot 2 + C_2\cdot(4+1) + 1 = 7\cdot C_2 + 3$$

The maximum width $W_{b_1}^{\max}$ is achieved if C_2 has the value 77 when the code section begins. This happens when $I_1 < I_2$. The maximum pulse width is then:

$$W_{b_1}^{\max} = 7\cdot 77 + 3 = 542\ \mu\text{s}$$

The minimum width $W_{b_1}^{\min}$ is achieved if C_2 has the value 61 when the code section begins. This happens when either $I_1 = I_2$ or $I_1 > I_2$. The minimum pulse width is then:

$$W_{b_1}^{\min} = 7\cdot 61 + 3 = 430\ \mu\text{s}$$

Answer: The minimum width of the pulse on bit b_1 is $W_{b_1}^{\min} = 430\ \mu$s, which fulfills the minimum-width constraint 400 μs.

b) The width of the pulse on bit b_0 on the output port consists of two parts. We derived the latter part since the program code that affects the width of that part is the same as in sub-problem a).

The section of the program code that affects the first part, $W_{b_{0a}}$, of the width of the pulse on bit b_0 on the output port is:

```
while (c > 0)
   c = c - 1;

if (d1 < d2)
   c = 77;
else
   c = 61;

Outport = 0x03;    // start pulse on bit 1 (positive flank)
```

Let C_1 be the value of variable `c` when this code section begins. The logical condition of the `while` statement will then be evaluated $C_1 + 1$ times, and the body of the `while` loop will be executed C_1 times. The pulse width $W_{b_{0a}}$ is then:

$$W_{b_{0a}} = (C_1+1)\cdot\{\mathit{Compare},\, c > 0\} + C_1\cdot(\{\mathit{Subtract},\, c-1\} + \{\mathit{Assign},\, c\}) +$$
$$\{\mathit{Compare},\, d1 < d2\} + \max(\{\mathit{Assign},\, c\}, \{\mathit{Assign},\, c\}) + \{\mathit{Assign},\, \mathit{Outport}\} =$$
$$(C_1+1)\cdot 2 + C_1\cdot(4+1) + 2 + \max(1,1) + 1 = 7\cdot C_1 + 6$$

The maximum width of the first part, $W_{b_{0a}}^{max}$, is achieved if C_1 has the value 47 when the code section begins. This happens when $I_1 = I_2$:

$W_{b_{0a}}^{max} = 7 \cdot 47 + 6 = 335\ \mu s$

The minimum width of the first part, $W_{b_{0a}}^{min}$,is achieved if C_1 has the value 31 when the code section begins. This happens when either $I_1 < I_2$ or $I_1 > I_2$:

$W_{b_{0a}}^{min} = 7 \cdot 31 + 6 = 223\ \mu s$

It is now possible to calculate the total pulse width, W_{b_0}, of the pulse on bit b_0 on the output port. Depending on how the `if` statements affect the number of iterations of the `while` loops we need to analyze three cases, namely $I_1 < I_2$, $I_1 = I_2$ and $I_1 > I_2$:

$I_1 < I_2$: Pulse width $W_{b_0} = W_{b_{0a}}^{min} + W_{b_1}^{max} = 223 + 542 = 765\ \mu s$.

$I_1 = I_2$: Pulse width $W_{b_0} = W_{b_{0a}}^{max} + W_{b_1}^{min} = 335 + 430 = 765\ \mu s$.

$I_1 > I_2$: Pulse width $W_{b_0} = W_{b_{0a}}^{min} + W_{b_1}^{min} = 223 + 430 = 653\ \mu s$.

Answer: The maximum width of the pulse on bit b_0 is $W_{b_0}^{max} = 765\ \mu s$, which fulfills the maximum-width constraint 800 μs.

Note: The analysis above shows that the combination $W_{b_{0a}}^{max} + W_{b_1}^{max} = 335 + 542 = 877$ never occurs. The reason for this is that the conditions $I_1 = I_2$ and $I_1 < I_2$ would have to be true at the same time, which is impossible. The scenario of assignments $C_1 = 47$ and $C_2 = 77$ being valid at the same time is thus a false path in the program code.

c) With the given values, $I_1 = -121$ and $I_2 = 93$, the pulse widths for the case $I_1 < I_2$ apply:

$W_{b_1} = 542\ \mu s$

$W_{b_0} = 765\ \mu s$

PROBLEM 4

a) Critical region is called via a `SYNC` statement:

```
#include "TinyTimber.h"

typedef struct {
    Object super;
    char whoami;
    int deadline;
} Task;

Task T1 = { initObject(), '1', 650 };
Task T2 = { initObject(), '2', 950 };
Task T3 = { initObject(), '3', 800 };

void Critical(Task*, int);

void Non_Critical(Task *self, int unused) {

    SYNC(self, Critical, 0);

    if (self->whoami == '1') {
                                                              // TODO: insert TinyTimber code
        BEFORE(USEC(T2.deadline), &T2, Non_Critical, 0);
    }
    if (self->whoami == '2') {
                                                              // TODO: insert TinyTimber code
        BEFORE(USEC(T3.deadline), &T3, Non_Critical, 0);
    }
    if (self->whoami == '3') {
                                                              // TODO: insert TinyTimber code
        SEND(USEC(1000), USEC(T1.deadline), &T1, Non_Critical, 0); // restart loop
    }
}

void Critical(Task *self, int unused) {
    Action250(); // Do critical work for 250 microseconds
}

void kickoff(TaskObj *self, int u) {
    BEFORE(USEC(T1.deadline), &T1, Non_Critical, 0);
}

main() {
    return TINYTIMBER(&T1, kickoff, 0);
}
```

b) Access to the critical region is handled by means of a set of semaphores:

```
#include "TinyTimber.h"
#include "semaphore.h"

Semaphore Sem1 = { initObject(), 1, 0 }; // TODO: set initial semaphore value
Semaphore Sem2 = { initObject(), 0, 0 }; // TODO: set initial semaphore value
Semaphore Sem3 = { initObject(), 0, 0 }; // TODO: set initial semaphore value

typedef struct {
    Object super;
    char whoami;
    int deadline;
    CallBlock cb; // where call-back information is stored
} Task;

Task T1 = { initObject(), '1', 650, initCallBlock() };
Task T2 = { initObject(), '2', 950, initCallBlock() };
Task T3 = { initObject(), '3', 800, initCallBlock() };

void Critical(Task*, int);

void Non_Critical(Task *self, int unused) {
    self->cb.obj = (Object *) self; // provide call-back information
    self->cb.meth = (Method) Critical;
    if (self->whoami == '1')
        ASYNC(&Sem1, Wait, (int) &self->cb ); // TODO: select semaphore to acquire
    if (self->whoami == '2')
        ASYNC(&Sem2, Wait, (int) &self->cb ); // TODO: select semaphore to acquire
    if (self->whoami == '3')
        ASYNC(&Sem3, Wait, (int) &self->cb ); // TODO: select semaphore to acquire
}

void Critical(Task *self, int unused) {
    Action250(); // Do critical work for 250 microseconds
    if (self->whoami == '1')
        SYNC(&Sem2, Signal, 0); // TODO: select semaphore to release
    if (self->whoami == '2')
        SYNC(&Sem3, Signal, 0); // TODO: select semaphore to release
    if (self->whoami == '3')
        SYNC(&Sem1, Signal, 0); // TODO: select semaphore to release
    SEND(USEC(1000), USEC(self->deadline), self, Non_Critical, 0); // restart loop
}

void kickoff(TaskObj *self, int u) {
    BEFORE(USEC(T1.deadline), &T1, Non_Critical, 0);
    BEFORE(USEC(T2.deadline), &T2, Non_Critical, 0);
    BEFORE(USEC(T3.deadline), &T3, Non_Critical, 0);
}

main() {
    return TINYTIMBER(&T1, kickoff, 0);
}
```

PROBLEM 5

a) The utilization-based test 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: $K_1 = \{4, 8, 12, 16, 20\}$, $K_2 = \{18\}$ and $K_3 = \{3, 13\}$ which gives us $K = K_1 \cup K_2 \cup K_3 = \{3, 4, 8, 12, 13, 16, 18, 20\}$.

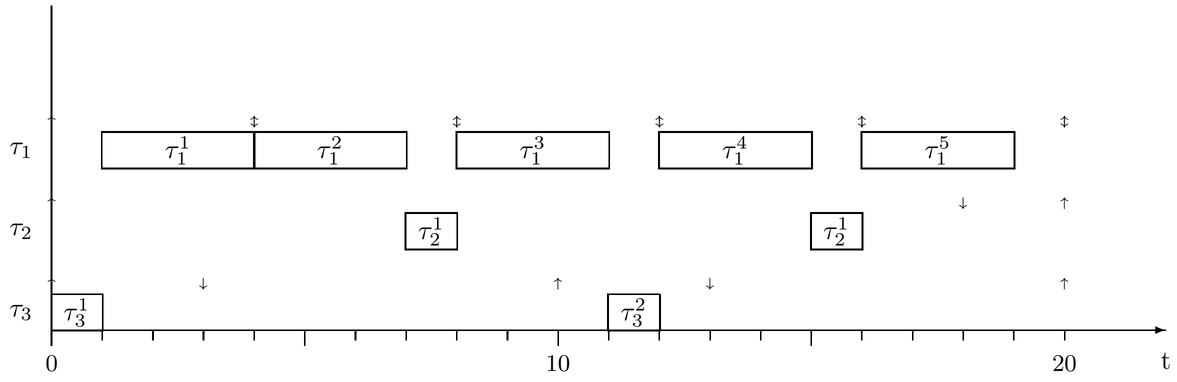
Schedulability analysis now gives us:

L	$N_1^L \cdot C_1$	$N_2^L \cdot C_2$	$N_3^L \cdot C_3$	$C_P(0, L)$	$C_P(0, L) \leq L$
3	$(\lfloor\frac{(3-4)}{4}\rfloor + 1) \cdot 3 = 0$	$(\lfloor\frac{(3-18)}{20}\rfloor + 1) \cdot 2 = 0$	$(\lfloor\frac{(3-3)}{10}\rfloor + 1) \cdot 1 = 1$	1	OK
4	$(\lfloor\frac{(4-4)}{4}\rfloor + 1) \cdot 3 = 3$	$(\lfloor\frac{(4-18)}{20}\rfloor + 1) \cdot 2 = 0$	$(\lfloor\frac{(4-3)}{10}\rfloor + 1) \cdot 1 = 1$	4	OK
8	$(\lfloor\frac{(8-4)}{4}\rfloor + 1) \cdot 3 = 6$	$(\lfloor\frac{(8-18)}{20}\rfloor + 1) \cdot 2 = 0$	$(\lfloor\frac{(8-3)}{10}\rfloor + 1) \cdot 1 = 1$	7	OK
12	$(\lfloor\frac{(12-4)}{4}\rfloor + 1) \cdot 3 = 9$	$(\lfloor\frac{(12-18)}{20}\rfloor + 1) \cdot 2 = 0$	$(\lfloor\frac{(12-3)}{10}\rfloor + 1) \cdot 1 = 1$	10	OK
13	$(\lfloor\frac{(13-4)}{4}\rfloor + 1) \cdot 3 = 9$	$(\lfloor\frac{(13-18)}{20}\rfloor + 1) \cdot 2 = 0$	$(\lfloor\frac{(13-3)}{10}\rfloor + 1) \cdot 1 = 2$	11	OK
16	$(\lfloor\frac{(16-4)}{4}\rfloor + 1) \cdot 3 = 12$	$(\lfloor\frac{(16-18)}{20}\rfloor + 1) \cdot 2 = 0$	$(\lfloor\frac{(16-3)}{10}\rfloor + 1) \cdot 1 = 2$	14	OK
18	$(\lfloor\frac{(18-4)}{4}\rfloor + 1) \cdot 3 = 12$	$(\lfloor\frac{(18-18)}{20}\rfloor + 1) \cdot 2 = 2$	$(\lfloor\frac{(18-3)}{10}\rfloor + 1) \cdot 1 = 2$	16	OK
20	$(\lfloor\frac{(20-4)}{4}\rfloor + 1) \cdot 3 = 15$	$(\lfloor\frac{(20-18)}{20}\rfloor + 1) \cdot 2 = 2$	$(\lfloor\frac{(20-3)}{10}\rfloor + 1) \cdot 1 = 2$	19	OK

The processor demand in each strategic time interval never exceeds the length of the interval, so all tasks meet their deadlines.

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

A simulation of the tasks using EDF scheduling in the interval $[0, LCM]$ gives the following timing diagram.

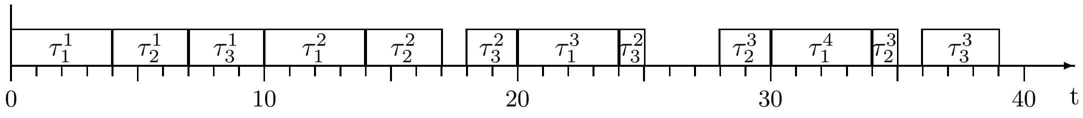


PROBLEM 6

a) The execution pattern in the timing diagram indicates that the two tasks have different periods. Since RM scheduling is used the same task must have highest priority and therefore always execute without preemption at each arrival. The timing diagram reveals that this task is τ_1 and that its period is $T_1 = 10$. The period of τ_2 must then be longer than 10 in order to follow the correct priority assignment. The gap in the task execution between time points 24 and 28 indicates that $T_2 = 14$.

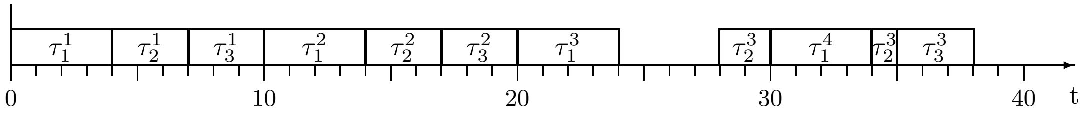
b) The utilization $U = C_1/T_1 + C_2/T_2 = 4/10 + 6/14 \approx 0.83$. Note that the two tasks critically utilize the processor even though U is significantly less than 1. This is a characteristic property of RM.

c) We generate a new schedule with task τ_1, task τ_2 (with new execution time) and the the new task τ_3 assuming $T_3 = 18$:



It is clear from the new schedule that the period of τ_3 must be shorter than 18, since it is now possible to slightly increase the execution time of any of the tasks without causing a deadline to be missed (that is, the schedule does not critically utilize the processor).

We therefore generate a new schedule with assuming $T_3 = 17$:



The new schedule with $T_3 = 17$ does critically utilize the processor, since it is no longer possible to slightly increase the execution time of any of the tasks without causing the deadline for the first instance of τ_3 to be missed.

The utilization of this task set is $U = 4/10 + 3/14 + 3/17 \approx 0.79$.

This task set with $T_3 = 17$ is the solution to our problem, since any other task set will either (i) critically utilize the processor but have a utilization higher than 0.79 (which happens when $10 \le T_3 < 17$) or (ii) make the schedule infeasible (which happens when $T_3 < 10$).

d) What we have seen in sub-problems b) and c) are cases where task sets of size $n = 2$ and $n = 3$ critically utilize the processor at the utilizations $U \approx 0.83$ and $U \approx 0.79$, respectively. If we now consider that we are using RM scheduling it is interesting to note that Liu & Layland's feasibility bound for RM, $n(2^{1/n} - 1)$, evaluates to values that are very close to our derived values for $n = 2$ and $n = 3$. What we have done in this problem can therefore be seen as a simplified proof for Liu & Layland's feasibility test for systems with size $n = 2$ and $n = 3$. Of course, the original proof uses stricter mathematical notation, more general value domains (non-integer values are allowed) and also derives general expressions for the relations between task period and execution times. For example, the task set for $n = 3$ that exactly corresponds to Liu & Layland's proof is as follows: $T_1 = 10$, $T_2 = 2^{1/3} \cdot T_1 \approx 12.6$, $T_3 = 2^{2/3} \cdot T_1 \approx 15.9$, $C_1 = T_2 - T_1 \approx 2.6$, $C_2 = T_3 - T_2 \approx 3.3$, $C_3 = 2T_1 - T_3 \approx 4.1$.

In conclusion, we could expect a similar correspondance to Liu & Layland's feasibility bound when we do the analysis after adding a fourth (a fifth, a sixth etc) task to the task set.

PROBLEM 7

a) The Oh & Baker utilization guarantee bound for partitioned scheduling on a multiprocessor system is known to be no less than $m(2^{1/2} - 1)$, where m is the number of processors.

b) The utilization guarantee bound for any multiprocessor scheduling algorithm (partitioned or global) with static task priorities can not exceed $0.5m$, where m is the number of processors [Andersson et al., 2001].

c) First we compute the utilization of the tasks.

	C_i	T_i	U_i
τ_1	10	50	0.2
τ_2	?	200	$C_2/200$
τ_3	8	20	0.4
τ_4	7	10	0.7
τ_5	5	30	0.167
τ_6	40	100	0.4

Tasks in RMFF are assigned to the processors in their increasing order of periods. Therefore, the order of assigning the tasks (from left to right) is as follows:

$$\tau_4, \tau_3, \tau_5, \tau_1, \tau_6, \tau_2$$

Since task τ_2 will be assigned the last, we can follow the RMFF to assign the five tasks $\tau_4, \tau_3, \tau_5, \tau_1, \tau_6$ on $m = 3$ processor without knowing the value of C_2.

Denote the three processors by μ_1, μ_2, and μ_3.

By following the RMFF algorithm (Lecture 14), the five tasks are assigned as follows:

$\mu_1 : \tau_4$	total utilization=0.7
$\mu_2 : \tau_3, \tau_5, \tau_1$	total utilization= $0.4 + 0.167 + 0.2 = 0.767$
$\mu_3 : \tau_6$	total utilization=0.4

Since μ_3 is the most lightly loaded processor, task τ_2 needs to be assigned to μ_3 so that its C_2 value is maximized. Since $U_6 = 0.4$ and the Liu and Layland bound for uniprocessor rate-monotonic scheduling for two tasks is $2 \cdot (2^{\frac{1}{2}-1})$, task τ_2 must satisfy the following for μ_3:

$$U_6 + C_2/T_2 \leq 2 \cdot (2^{\frac{1}{2}} - 1)$$

Since $U_6 = 0.4$ and $T_2 = 200$, we have

$$0.4 + C_2/200 \leq 2 \cdot (2^{\frac{1}{2}} - 1)$$

which implies $C_2 \leq 85.685$. Therefore, the maximum value of C_2 is 85.685. Since C_2 is an integer, we have $C_2 = 85$.

d) First we compute the utilization of the tasks.

	C_i	T_i	U_i
τ_1	10	50	0.2
τ_2	2	?	$2/T_2$
τ_3	8	20	0.4
τ_4	7	10	0.7
τ_5	50	300	0.167
τ_6	20	100	0.2

Since T_2 is an integer and $C_2 \leq T_2$, we have to consider $T_2 = 2, 3, 4, ...$

Using the RMFF algorithm (Lecture 14), we find that if $T_2 = 2$ or $T_2 = 3$, then all the tasks cannot be assigned to $m = 3$ processors such that all the deadlines are met. If $T_2 = 4$, then the order of assigning the six tasks (from left to right) is as follows:

$$\tau_2, \tau_4, \tau_3, \tau_1, \tau_6, \tau_5$$

Denote the three processors by μ_1, μ_2, and μ_3.

The utilization of the tasks, where $T_2 = 4$, is given as follows

	C_i	T_i	U_i
τ_1	10	50	0.2
τ_2	2	4	0.5
τ_3	8	20	0.4
τ_4	7	10	0.7
τ_5	50	300	0.167
τ_6	20	100	0.2

By following the RMFF algorithm (Lecture 14), the six tasks are assigned on $m = 3$ processors as follows:

$\mu_1 : \tau_2, \tau_1$	total utilization$=0.5 + 0.2$
$\mu_2 : \tau_4$	total utilization$= 0.7$
$\mu_3 : \tau_3, \tau_6, \tau_5$	total utilization$=0.4 + 0.2 + 167 = 0.767$

The minimum value of T_2 is 4.