Introduction to Real-Time Systems

Solutions to final exam May 26, 2018 (version 20180526)

PROBLEM 1

a) False: Deadline inversion occurs in EDF scheduling when a higher-priority task (with a short deadline) cannot execute (because another task holds a resource that the higher-priority task needs) and a lower-priority task (with longer deadline) 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: 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 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) True: If we know that the task set is not schedulable then a *sufficient* test must have resulted in the outcome 'False'. This is because, for sufficient tests, the outcome 'True' always means that the task set is 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.
- 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 task 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 `Calculate(x)` is derived based on three cases of the value of parameter x.

Case 1: $x = 0$:

$$\begin{aligned}&WCET(Calculate(x=0))\\&=\{Declare,t\}+\{Compare,x=0\}+\{Assign,t\}+\{Return,t\}\\&=1+2+1+2=6\end{aligned}$$

Case 2: $x = 1$:

$$\begin{aligned}&WCET(Calculate(x=1))\\&=\{Declare,t\}+\{Compare,x=0\}+\{Compare,x=1\}+\\&\{Assign,t\}+\{Return,t\}\\&=1+2+2+1+2=8\end{aligned}$$

Case 3: $x > 1$:

$$\begin{aligned}&WCET(Calculate(x>1))\\&=\{Declare,t\}+\{Compare,x=0\}+\{Compare,x=1\}+\\&\{Sub,x-1\}+\{Call,Calculate(x-1)\}+WCET(Calculate(x-1))+\\&\{Multiply,x*Calculate(x-1)\}+\{Assign,t\}+\{Return,t\}\\&=1+2+2+3+2+5+1+2+WCET(Calculate(x-1))\\&=18+WCET(Calculate(x-1))\end{aligned}$$

The WCET of `Respond()` is derived as follows:

$$\begin{aligned}&WCET(Respond())\\&=\{Declare,c\}+\{Declare,r\}+\{Assign,c\}+\\&\{Call,Calculate(c)\}+WCET(Calculate(c))+\{Shift,Calculate(c)\}+\\&\{Add,c\}+\{Assign,r\}+\{Assign,Outport\}\\&=1+1+1+2+2+3+1+1+WCET(Calculate(c))\\&=12+WCET(Calculate(c))\end{aligned}$$

Therefore, for the largest input port value $c = 6$, the WCET of `Respond()` is

$$\begin{aligned}&WCET(Respond())\\&=12+WCET(Calculate(6))=12+18+WCET(Calculate(5))\\&=12+2\times18+WCET(Calculate(4))=12+3\times18+WCET(Calculate(3))\\&=12+4\times18+WCET(Calculate(2))=12+5\times18+WCET(Calculate(1))\\&=12+5\times18+8=110\mu\text{s}\end{aligned}$$

Since the deadline of `Respond()` is 80 μs, the deadline is not met.

b) The largest input port value for which the WCET of `Respond()` does not exceed the deadline of 80 μs is $c = 4$:

$$\begin{aligned}&WCET(Respond)\\&=12+WCET(Calculate(4))=12+18+WCET(Calculate(3))\\&=12+2\times18+WCET(Calculate(2))=12+3\times18+WCET(Calculate(1))\\&=12+3\times18+8=74\mu\text{s}\end{aligned}$$

Thus, the largest acceptable input port data range is $[0, +4]$

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(105), MSEC(30), self, T1, 0);
}

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

    SEND(MSEC(70), MSEC(25), self, T2, 0);
}

void kickoff(PeriodicTask *self, int u) {
    SEND(MSEC(0), MSEC(30), &ptask1, T1, 0);
    SEND(MSEC(15), MSEC(25), &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) TinyTimber uses the Deadline Inheritance Protocol, combined with deadlock detection via the return value of the SYNC call.

PROBLEM 5

We start by observing that task τ_1 has a first arrival time that differs from that of the other tasks. This means that the use of a utilization-based or response-time-based schedulability test may become overly pessimistic IF there exists no point in time in the schedule where all tasks arrive at the same time. This, in turn, could mean that, should the test fail, the task set could potentially still be schedulable.

Our first candidate method for schedulability analysis is Liu and Layland's classic utilization-based test. For three tasks, the schedulability bound is $U_{lub} = 3(2^{1/3} - 1) \approx 0.780$. Unfortunately, the accumulated task utilization, $U = 1/3 + 3/8 + 1/4 \approx 0.958$, significantly exceeds the guarantee bound, and the test (being only sufficient) does not provide any useful information.

Our second candidate method is response-time analysis. Since task periods are required by the analysis, we begin by deriving the period of each task: $T_i = C_i/U_i = C_i \cdot (U_i)^{-1}$

$T_1 = C_1 \cdot (U_1)^{-1} = 0.2C \cdot 3 = 0.6C$

$T_2 = C_2 \cdot (U_2)^{-1} = 0.3C \cdot 8/3 = 0.8C$

$T_3 = C_3 \cdot (U_3)^{-1} = 0.3C \cdot 4 = 1.2C$

Assuming RM scheduling, task τ_1 has highest priority (shortest period) and task τ_3 has lowest priority. We then calculate the response time of each task and compare it against the corresponding task deadline:

$R_1 = C_1 = 0.2C < D_1 = T_1 = 0.6C$.

$R_2 = C_2 + \lceil \frac{R_2}{T_1} \rceil \cdot C_1$. Assume that $R_2^0 = C_2 = 0.3C$:

$R_2^1 = 0.3C + \lceil \frac{0.3C}{0.6C} \rceil \cdot 0.2C = 0.3C + 1 \cdot 0.2C = 0.5C$

$R_2^2 = 0.3C + \lceil \frac{0.5C}{0.6C} \rceil \cdot 0.2T = 0.3C + 1 \cdot 0.2C = 0.5C < D_2 = T_2 = 0.8C$

$R_3 = C_3 + \lceil \frac{R_3}{T_1} \rceil \cdot C_1 + \lceil \frac{R_3}{T_2} \rceil \cdot C_2$. Assume that $R_3^0 = C_3 = 0.3C$:

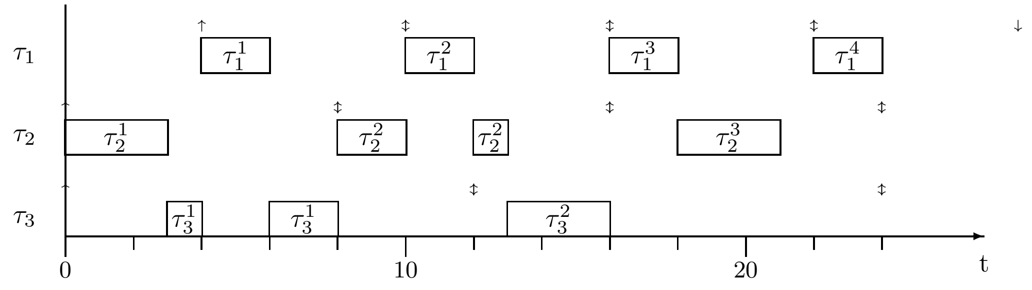
$R_3^1 = 0.3C + \lceil \frac{0.3C}{0.6C} \rceil \cdot 0.2C + \lceil \frac{0.3C}{0.8C} \rceil \cdot 0.3C = 0.3C + 1 \cdot 0.2C + 1 \cdot 0.3C = 0.8C$

$R_3^2 = 0.3C + \lceil \frac{0.8C}{0.6C} \rceil \cdot 0.2C + \lceil \frac{0.8C}{0.8C} \rceil \cdot 0.3C = 0.3C + 2 \cdot 0.2C + 1 \cdot 0.3C = 1.0C$

$R_3^3 = 0.3C + \lceil \frac{1.0C}{0.6C} \rceil \cdot 0.2C + \lceil \frac{1.0C}{0.8C} \rceil \cdot 0.3C = 0.3C + 2 \cdot 0.2C + 2 \cdot 0.3C = 1.3C > D_3 = T_3 = 1.2C$

The response-time analysis thus indicates that the worst-case response-time for τ_3 exceeds the task deadline. However, by observing the given periods and offsets (see time diagram below), we can see that there does NOT exist a point in time where all tasks arrive at the same time. This means that the worst-case response-time for τ_3 calculated by the response-time analysis will in fact never occur, and that it still is possible that τ_3 will meet its deadline. We will find out by using hyper-period analysis.

By simulating the RM schedule for one hyper period[1] we will see that <u>all tasks indeed meet their deadlines</u>. The hyper period for the given task set is LCM = 2.4C. Let C = 10 to get LCM = 24:



[1]Although we have an asynchronous task set, all task executions are completed within the first hyper period. This means that we only need to check schedulability within that interval.

PROBLEM 6

a) Perform processor-demand analysis:

First, determine LCM of the task periods: LCM$\{T_1, T_2, T_3\}$ = LCM$\{8, 16, 32\} = 32$.

Then, derive the set K of control points: $K_1 = \{4, 12, 20, 28\}$, $K_2 = \{12, 28\}$ and $K_3 = \{30\}$ which gives us $K = K_1 \cup K_2 \cup K_3 = \{4, 12, 20, 28, 30\}$.

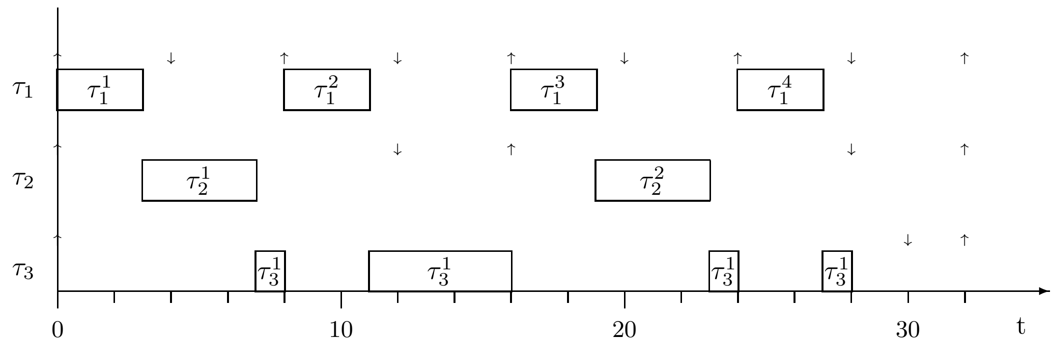
Schedulability analysis now gives us:

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	$(\lfloor\frac{(4-4)}{8}\rfloor + 1) \cdot 3 = 3$	$(\lfloor\frac{(4-12)}{16}\rfloor + 1) \cdot 4 = 0$	$(\lfloor\frac{(4-30)}{32}\rfloor + 1) \cdot 8 = 0$	3	OK
12	$(\lfloor\frac{(12-4)}{8}\rfloor + 1) \cdot 3 = 6$	$(\lfloor\frac{(12-12)}{16}\rfloor + 1) \cdot 4 = 4$	$(\lfloor\frac{(12-30)}{32}\rfloor + 1) \cdot 8 = 0$	10	OK
20	$(\lfloor\frac{(20-4)}{8}\rfloor + 1) \cdot 3 = 9$	$(\lfloor\frac{(20-12)}{16}\rfloor + 1) \cdot 4 = 4$	$(\lfloor\frac{(20-30)}{32}\rfloor + 1) \cdot 8 = 0$	13	OK
28	$(\lfloor\frac{(28-4)}{8}\rfloor + 1) \cdot 3 = 12$	$(\lfloor\frac{(28-12)}{16}\rfloor + 1) \cdot 4 = 8$	$(\lfloor\frac{(28-30)}{32}\rfloor + 1) \cdot 8 = 0$	20	OK
30	$(\lfloor\frac{(30-4)}{8}\rfloor + 1) \cdot 3 = 12$	$(\lfloor\frac{(30-12)}{16}\rfloor + 1) \cdot 4 = 8$	$(\lfloor\frac{(30-30)}{32}\rfloor + 1) \cdot 8 = 8$	28	OK

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

b) From sub-problem a): LCM$\{8, 16, 32\} = 32$.

A simulation of the tasks using EDF scheduling in the interval $[0, LCM]$ gives the following timing diagram. We see that, also here, all task meet their deadlines.

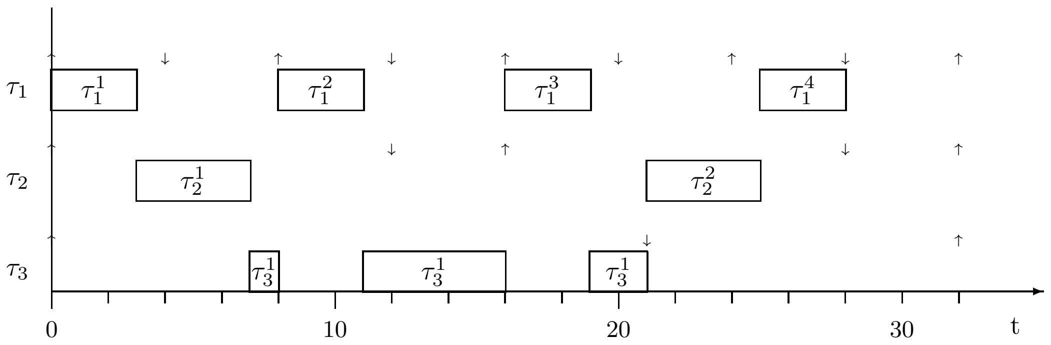


c) Based on the timing diagram in sub-problem b) it is obvious that task τ_3 may decrease its deadline to $D_3 = 28$ without any task missing its deadline. Re-applying the processor-demand analysis after merging the new D_3 with the already-existing control point at $L = 28$ (and removing $L = 30$) verifies this:

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	$(\lfloor\frac{(4-4)}{8}\rfloor + 1) \cdot 3 = 3$	$(\lfloor\frac{(4-12)}{16}\rfloor + 1) \cdot 4 = 0$	$(\lfloor\frac{(4-28)}{32}\rfloor + 1) \cdot 8 = 0$	3	OK
12	$(\lfloor\frac{(12-4)}{8}\rfloor + 1) \cdot 3 = 6$	$(\lfloor\frac{(12-12)}{16}\rfloor + 1) \cdot 4 = 4$	$(\lfloor\frac{(12-28)}{32}\rfloor + 1) \cdot 8 = 0$	10	OK
20	$(\lfloor\frac{(20-4)}{8}\rfloor + 1) \cdot 3 = 9$	$(\lfloor\frac{(20-12)}{16}\rfloor + 1) \cdot 4 = 4$	$(\lfloor\frac{(20-28)}{32}\rfloor + 1) \cdot 8 = 0$	13	OK
28	$(\lfloor\frac{(28-4)}{8}\rfloor + 1) \cdot 3 = 12$	$(\lfloor\frac{(28-12)}{16}\rfloor + 1) \cdot 4 = 8$	$(\lfloor\frac{(28-28)}{32}\rfloor + 1) \cdot 8 = 8$	28	OK

This is, however, not the smallest possible value of D_3 that still makes the task set schedulable.

It is, in fact, possible for task τ_3 to decrease its deadline to $D_3 = 21$ without any task missing its deadline, as can be seen in the timing diagram below:



Re-applying the original processor-demand analysis with the new control point at $L = 21$ (replacing $L = 30$) verifies this:

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	$(\lfloor\frac{(4-4)}{8}\rfloor + 1) \cdot 3 = 3$	$(\lfloor\frac{(4-12)}{16}\rfloor + 1) \cdot 4 = 0$	$(\lfloor\frac{(4-21)}{32}\rfloor + 1) \cdot 8 = 0$	3	OK
12	$(\lfloor\frac{(12-4)}{8}\rfloor + 1) \cdot 3 = 6$	$(\lfloor\frac{(12-12)}{16}\rfloor + 1) \cdot 4 = 4$	$(\lfloor\frac{(12-21)}{32}\rfloor + 1) \cdot 8 = 0$	10	OK
20	$(\lfloor\frac{(20-4)}{8}\rfloor + 1) \cdot 3 = 9$	$(\lfloor\frac{(20-12)}{16}\rfloor + 1) \cdot 4 = 4$	$(\lfloor\frac{(20-21)}{32}\rfloor + 1) \cdot 8 = 0$	13	OK
21	$(\lfloor\frac{(21-4)}{8}\rfloor + 1) \cdot 3 = 9$	$(\lfloor\frac{(21-12)}{16}\rfloor + 1) \cdot 4 = 4$	$(\lfloor\frac{(21-21)}{32}\rfloor + 1) \cdot 8 = 8$	21	OK
28	$(\lfloor\frac{(28-4)}{8}\rfloor + 1) \cdot 3 = 12$	$(\lfloor\frac{(28-12)}{16}\rfloor + 1) \cdot 4 = 8$	$(\lfloor\frac{(28-21)}{32}\rfloor + 1) \cdot 8 = 8$	28	OK

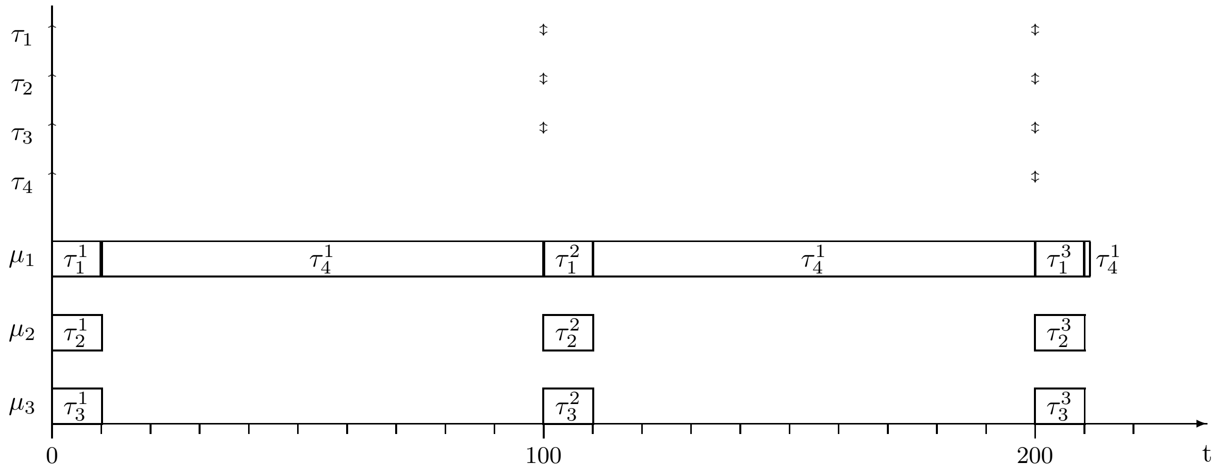
It is not possible to further decrease D_3. For example, choosing $D_3 = 20$ would cause either τ_1 or τ_3 to miss its deadline. The last processor-demand analysis above verifies this: there is only $20 - 13 = 7$ time units of slack available in control point $L = 20$, but adding one instance of τ_3 would require $C_3 = 8$ time units.

PROBLEM 7

a) Since rate-monotonic (RM) scheduling is used, the task priorities are as follows:

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

We generate a multiprocessor schedule with tasks τ_1, τ_2 and τ_3 (having the highest priorities) running on one processor each. Task τ_4 is scheduled in the remaining time slots according to the following diagram (covering the first execution of τ_4):



We observe that the first instance of task τ_4 completes its execution at $t = 211$ on processor μ_1, thereby missing its deadline at $t = T_4 = 200$. This happens despite there being significant processor capacity available on processors μ_2 and μ_3. A clear case of Dhall's effect!

b) We begin by calculating the utilization U_i for each task:

	C_i	T_i	U_i
τ_1	10	100	0.1
τ_2	10	100	0.1
τ_3	10	100	0.1
τ_4	141	200	0.705
τ_5	141	200	0.705
τ_6	141	200	0.705

Then, number the three processors μ_1, μ_2 and μ_3.

According to the RMFF partitioning algorithm the tasks should be assigned to the processors in the following (RM) order: $\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6$.

Tasks τ_1, τ_2, and τ_3 can all be assigned to processor μ_1, since

$U_1 + U_2 + U_3 = 0.1 + 0.1 + 0.1 = 0.3 < \text{U}_{\text{RM}(3)} = 3 \cdot (2^{1/3} - 1) \approx 0.780$

Task τ_4 cannot assigned to μ_1, since

$U_1 + U_2 + U_3 + U_4 = 0.3 + 0.705 > 1.0$

Task τ_4 can be assigned to μ_2, since there are no task assigned to that processor.

Task τ_5 cannot assigned to μ_1, since

$U_1 + U_2 + U_3 + U_5 = 0.3 + 0.705 > 1.0$

Task τ_5 cannot assigned to μ_2, since

$U_4 + U_5 = 0.705 + 0.705 > 1.0$

Task τ_5 can be assigned to μ_3, since there are no task assigned to that processor.

Task τ_6 cannot assigned to μ_1, since

$U_1 + U_2 + U_3 + U_6 = 0.3 + 0.705 > 1.0$

Task τ_6 cannot assigned to μ_2, since

$U_4 + U_6 = 0.705 + 0.705 > 1.0$

Task τ_6 cannot assigned to μ_3, since

$U_5 + U_6 = 0.705 + 0.705 > 1.0$

Task τ_6 can, consequently, not be assigned to any processor. This means that the given task set cannot be scheduled using the RMFF algorithm.

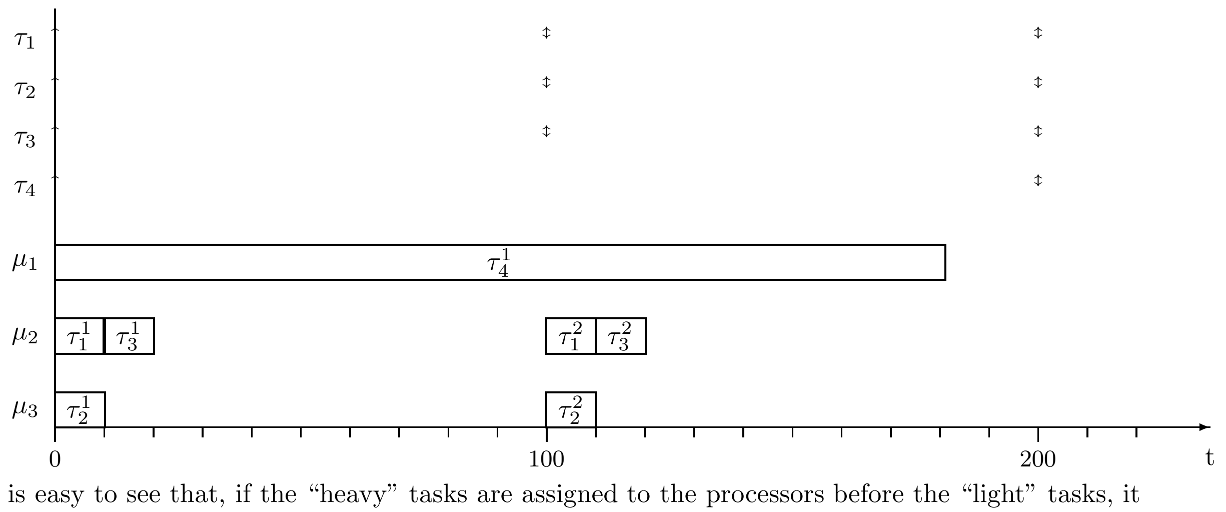
c) The common property of the task sets in sub-problems a) and b) is that, with the RM approach, lowest priority is given to the tasks that have the highest computational requirements in relation to their periods, *i.e.* the "heavy" tasks in the task set. The way the set of highest-priority tasks is constructed the "heavy" task will not be able to utilize the available processor capacity in the best way, and thereby a "heavy" task will either miss its deadline (as in sub-problem a) or not be assigned to any processor (as in sub-problem b).

d) If task priorities are given according to the rate-monotonic utilization-separation (RM-US) approach it may be possible to circumvent Dhall's effect, since that approach gives highest priority to "heavy" tasks in the task set. In order to make sure that task deadlines are met using the RM-US approach we need to verify that the total task utilization does not exceed the guarantee bound for RM-US.

The total utilization of the task set is $U_{\text{Total}} = U_1 + U_2 + U_3 + U_4 = 0.3 + 0.905 = 1.205$

The guarantee bound for RM-US is $U_{\text{RM}-\text{US}} = \frac{m^2}{3m-2}$. Since $m = 3$, $U_{\text{RM}-\text{US}} = 9/7 \approx 1.285$.

Consequently, by using the RM-US approach, all task deadlines for the task set in sub-problem a) will be met since $1.205 < 1.285$. The successful schedule in the hyper period $[0, 200]$ can be seen in the timing diagram below.



e) It is easy to see that, if the "heavy" tasks are assigned to the processors before the "light" tasks, it is possible to find a task-to-processor assignment for which RM-priority scheduling will meet all task deadlines.

An example of one such assignment is where each processor contains one "heavy" and one "light" task. The task utilization on each processor is then $0.1 + 0.705 = 0.805 < \text{U}_{\text{RM}(2)} = 2 \cdot (2^{1/2} - 1) \approx 0.828$, which means that all task deadlines are met on each processor.

Consequently, there exists a task-to-processor assignment for the task set in sub-problem b), such that all task deadlines will be met when task priorities are given according to the RM policy.