Real-Time Systems

Solutions to final exam March 16, 2020 (version 20200316)

PROBLEM 1

a) Liu & Layland's utilization test for RM compares the accumulated utilization U of a given task set against a bound U_{RM} that only depends on the number of tasks. If $U \leq U_{\mathrm{RM}}$ the task set is schedulable. The utilization of each task depends on the period T_i of as follows: $U_i = C_i/T_i$. This means that, if the task period increases, the task utilization will be reduced. Assume that a given periodic task set is known to pass Liu & Layland's utilization test for RM. If one of the tasks in the task set would not be strictly periodic, but instead sporadic, interarrival times between subsequent instances of the task could be longer (but never shorter) than the period assumed in the periodic case. In other words, the sporadic task could have a lower (but never higher) utilization than in the periodic case, and the same would apply for the accumulated utilisation U. Consequently, the sporadic task set would also pass Liu & Layland's utilization test for RM.

b) It is obvious that Liu & Layland's utilization test for RM cannot be used since $U = C_1/T_1 + C_2/T_2 = 0.4 + (15\,C_1)/(10\,T_1) = 0.4 + 1.5 \cdot C_1/T_1 = 0.4 + 0.6 = 1.0$ is greater than the utilization bound for RM for two tasks, $U_{\mathrm{RM}} = 0.828$. It it therefore necessary to use some type of exact analysis method.

Response-time analysis:.

$C_1 = 0.4\,T_1$

$C_2 = 15\,C_1 = 15 \cdot 0.4\,T_1 = 6\,T_1$

Task τ_1 is the highest-priority task since $T_1 < T_2$.

$R_1 = C_1 = 0.4\,T_1 \leq T_1$.

$R_2 = C_2 + \lceil \frac{R_2}{T_1} \rceil \cdot C_1$. Assume that $R_2^0 = C_2 = 6\,T_1$:

$R_2^1 = 6\,T_1 + \lceil \frac{6\,T_1}{T_1} \rceil \cdot 0.4\,T_1 = 6\,T_1 + 6 \cdot 0.4\,T_1 = 6\,T_1 + 2.4\,T_1 = 8.4\,T_1$

$R_2^2 = 6\,T_1 + \lceil \frac{8.4\,T_1}{T_1} \rceil \cdot 0.4\,T_1 = 6\,T_1 + 9 \cdot 0.4\,T_1 = 6\,T_1 + 3.6\,T_1 = 9.6\,T_1$

$R_2^3 = 6\,T_1 + \lceil \frac{9.6\,T_1}{T_1} \rceil \cdot 0.4\,T_1 = 6\,T_1 + 10 \cdot 0.4\,T_1 = 6\,T_1 + 4\,T_1 = 10\,T_1$

$R_2^4 = 6\,T_1 + \lceil \frac{10\,T_1}{T_1} \rceil \cdot 0.4\,T_1 = 6\,T_1 + 10 \cdot 0.4\,T_1 = 6\,T_1 + 4\,T_1 = 10\,T_1 \leq T_2$

Or, use the following reasoning that takes advantage of the fact that there are only two tasks in the system. It is clear that task τ_1 will meet its deadlines since it is the highest-priority task and $C_i < T_i$. From the calculation above we know that the processor is 100% busy executing the tasks. We also know that the hyper period (least-common-multiple of the task periods) is $10\,T_1$. Since task τ_1 will execute 10 times within the hyper period, and the processor is never idle, the completion time of τ_2 will be delayed exactly $10\,C_1 = 10 \cdot 0.4\,T_1 = 4\,T_1$ time units. And since $C_2 = 15\,C_1 = 15 \cdot 0.4\,T_1 = 6\,T_1$ the completion time of τ_2 will be $(6+4)\,T_1 = 10\,T_1$, which means that also τ_2 meets its deadlines.

PROBLEM 2

a) While the RM and EDF priority-assignment policies both make similar assumptions regarding the task model, they differ in the way they affect how the tasks are sorted in the ready queue in the run-time system. Depending on the system load the outcome will differ, as shown below:

Under normal load conditions it can be shown that there exist task sets that can be scheduled with dynamic priorities but cannot be scheduled with static priorities. Thus, EDF dominates RM in terms of schedulability.

Under overload, however, not all task will be able to meet their deadlines. In such situations it can be shown that more tasks may miss their deadlines if scheduled with dynamic priorities than if they are scheduled with static priorities. To that end, RM dominates EDF when the performance goal is to maximise the number of tasks that meet their deadlines.

b) Yes, the task set is still schedulable even if it becomes asynchronous. The reason is that the synchronous case is known to be the worst-case scenario (the critical instant) on a uniprocessor, that is, where the task response times are maximized. An asynchronous case then means that one or more tasks may have shorter response times than in the critical instant case.

c) The following conditions apply for pseudo-polynomial time complexity of an NP-complete algorithm:

- The algorithm must be a number problem, that is, the largest number (parameter value) in a problem instance cannot be bounded by the input length (size) of the problem.
- The time complexity of the number problem can be shown to be a polynomial-time function of both the input length and the largest number.

Assume a task set with n tasks. For task τ_i to be schedulable its WCET C_i must not exceed its deadline D_i. Consequently, C_i and D_i are both bounded above by the period T_i, which means that $\max_{\forall i}(T_i)$ is the largest number in the problem. Since each T_i can be chosen arbitrarily $\max_{\forall i}(T_i)$ is not bounded by n (the size of the problem).

Calculating the response-time for task τ_i requires at most D_i iterations of the algorithm. Each D_i is in turn bounded above by T_i. The time it takes to calculate the response times for <u>all</u> tasks is then bounded above by $n \cdot \max_{\forall i}(T_i)$, which is a polynomial-time function of both the input length and the largest number. Consequently, response-time analysis has pseudo-polynomial time complexity.

PROBLEM 3

a) The WCET of `Brake` is dependent on the WCET of function `Calculate`.

WCET of "Calculate":

$WCET(Calculate(x, y)) =$
$\{Declare, result\} + \{Assign, result\} + (b + 1) \cdot \{Compare, x \neq 0\} +$
$b \cdot (\{And, x \mathbin{\&} 1\} + \{Compare, x \neq 0\} + \{Add, result + y\} + \{Assign, result\} +$
$\{Shift, x\} + \{Assign, x\} + \{Shift, y\} + \{Assign, y\}) + \{Return, result\} =$
$1 + 1 + (b + 1) \cdot 2 + b \cdot (1 + 2 + 3 + 1 + 3 + 1 + 3 + 1) + 2 = b \cdot 17 + 6$

The number of `while`-loops, b, is controlled by the value of parameter x. Since `Calculate` is called with values read from the input ports, the value of x can be at most $15_{10} = 1111_2$. The loop runs as long as x is not zero, and shifts x one step to the right for each iteration: $1111_2, 0111_2, 0011_2, 0001_2$, and 0000_2. Hence, the loop can run at most $b = 4$ iterations.

$WCET(Calculate(x, y)) = 4 \cdot 17 + 6 = 74\ \mu$s

WCET of "Brake":

$WCET(Brake) =$
$\{Declare, v\} + \{Declare, p\} + \{Declare, f\} + \{Assign, v\} + \{Assign, p\} +$
$\{Call, Calculate(v, p)\} + WCET(Calculate(v, p)) + \{Assign, f\} + \{Assign, Outport\_Force\} =$
$1 + 1 + 1 + 1 + 1 + 7 + WCET(Calculate(v, p)) + 1 + 1 =$
$14 + WCET(Calculate(v, p)) = 14 + 74 = 88\ \mu$s

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

b) As shown in sub-problem a) the deadline is missed by just 1 μs. We therefore need to find a way to reduce the WCET of `Calculate` by at least that amount. To that end, we notice that the `if`-statement inside the `while`-loop in `Calculate` runs one time for every bit of value '1' in the binary representation of x. When $x = 15$, four bits in the binary representation of x have the value '1'. If one of these bits instead had the value '0', the WCET of `Calculate` would be reduced by 4 μs (the cost of one assignment and one add statement). We choose to set the least-significant bit of x to '0', as this will lead to the least reduction of the largest input port value (to $14_{10} = 1110_2$).

$WCET(Brake) = 88 - 4 = 84\ \mu$s

With the data input range $[1, 14]$ for `Inport_Velocity` the deadline is met!

c) We calculate WCET for the new version of `Calculate`:

$WCET(Calculate(x, y)) =$
$\{Declare, result\} + \{Assign, result\} + (b + 1) \cdot \{Compare, x \neq 0\} +$
$b \cdot (\{And, x \mathbin{\&} 1\} + \{Multiply, y * (x \mathbin{\&} 1)\} + \{Add, result + y * (x \mathbin{\&} 1)\} + \{Assign, result\} +$
$\{Shift, x\} + \{Assign, x\} + \{Shift, y\} + \{Assign, y\}) + \{Return, result\} =$
$1 + 1 + (b + 1) \cdot 2 + b \cdot (1 + 5 + 3 + 1 + 3 + 1 + 3 + 1) + 2 = b \cdot 20 + 6$

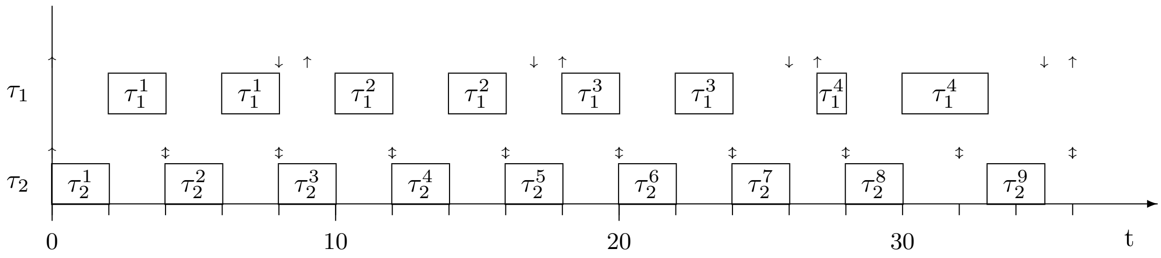
The original input port data range $[1, 15]$ leads to at most $b = 4$ iterations of the `while`-loop.

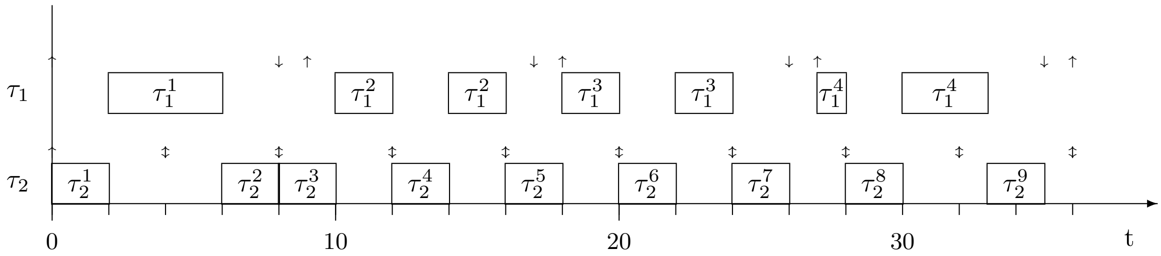
$WCET(Calculate(x, y)) = 4 \cdot 20 + 6 = 86\ \mu$s

Each execution of the `while`-loop now takes 3 μs longer time, leading to a total WCET increase of 12 μs. Hence, the new version is not a good replacement from a timing perspective.

PROBLEM 4

a) The timing diagram should look like one of the following:





b) The design flaws are the following (see comments in corrected code):

```
Object ptask = initObject();
Object ptask2 = initObject(); // separate object needed for second task

void T1(Object *self, int u) {
    Action4ms();

    SEND(MSEC(9), MSEC(8), self, T1, 0); // period of 9ms + deadline of 8ms needed
}

void T2(Object *self, int u) {
    Action2ms();

    SEND(MSEC(4), MSEC(4), self, T2, 0); // deadline of 4ms needed
}

void kickoff(Object *self, int u) {
    BEFORE(MSEC(8),&ptask, T1, 0);   // deadline of 8ms needed
    BEFORE(MSEC(4),&ptask2, T2, 0);  // deadline of 4ms needed
}

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

PROBLEM 5

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

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

We can now determine the ceiling priority for each resource:

$\mathrm{ceil}\{R_a\} = \max\{\mathrm{M,H,L}\} = \mathrm{H}$ (since τ_1, τ_2 or τ_3 may lock the semaphore)

$\mathrm{ceil}\{R_b\} = \max\{\mathrm{M,H}\} = \mathrm{H}$ (since τ_1 or τ_2 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:

$B_1 = H_{3,a} = 3$ (since τ_1 may be blocked by τ_3, who locks a resource whose ceiling priority is higher than or equal to the priority of τ_1)

$B_2 = \max\{H_{1,a}, H_{1,b}, H_{3,a}\} = \max\{5, 1, 3\} = 5$ (since τ_2 may be blocked by τ_1 or τ_3 who lock semaphores whose ceiling priorities are higher than or equal to the priority of τ_2)

$B_3 = 0$ (since τ_3 has lowest priority of all tasks, and thereby per definition cannot be subject to blocking)

We can now calculate the task response times, and check whether they are less than or equal to the corresponding deadline:

$R_2 = C_2 + B_2 = 6 + 5 = 11 \leq D_2 = 17.$

$R_1 = C_1 + B_1 + \lceil \frac{R_1}{T_2} \rceil \cdot C_2$. Assume that $R_1^0 = C_1 = 7$:

$R_1^1 = 7 + 3 + \lceil \frac{7}{28} \rceil \cdot 6 = 7 + 3 + 1 \cdot 6 = 16$

$R_1^2 = 7 + 3 + \lceil \frac{16}{28} \rceil \cdot 6 = 7 + 3 + 1 \cdot 6 = 16 \leq D_1 = 19$

$R_3 = C_3 + \lceil \frac{R_3}{T_1} \rceil \cdot C_1 + \lceil \frac{R_3}{T_2} \rceil \cdot C_2$. Assume that $R_3^0 = C_3 = 6$:

$R_3^1 = 6 + \lceil \frac{6}{19} \rceil \cdot 7 + \lceil \frac{6}{28} \rceil \cdot 6 = 6 + 1 \cdot 7 + 1 \cdot 6 = 19$

$R_3^2 = 6 + \lceil \frac{19}{19} \rceil \cdot 7 + \lceil \frac{19}{28} \rceil \cdot 6 = 6 + 1 \cdot 7 + 1 \cdot 6 = 19 \leq D_3 = 25$

All tasks meet their deadlines, so the task set is schedulable.

b) We notice from sub-problem a) that we could let $D_3 = 19$, without causing task τ_3 to miss its deadline, assuming that the original priority ordering is maintained. However, to find out whether D_3 can be even smaller, we also need to investigate a scenario where τ_3 is assigned a higher priority.

If we assume that $D_3 < 17$, the static task priorities will be as follows:

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

Since the task priorities have changed we need to recalculate the ceiling priorities:

$\mathrm{ceil}\{R_a\} = \max\{\mathrm{L,M,H}\} = \mathrm{H}$ (since τ_1, τ_2 or τ_3 may lock the resource)

$\mathrm{ceil}\{R_b\} = \max\{\mathrm{L,M}\} = \mathrm{M}$ (since τ_1 or τ_2 may lock the resource)

Since the ceiling priorities have changed we need to recalculate the blocking factor B_i:

$B_1 = 0$ (since τ_1 has lowest priority of all tasks, and thereby per definition cannot be subject to blocking)

$B_2 = \max\{H_{1,a}, H_{1,b}\} = \max\{5, 1\} = 5$ (since τ_2 may be blocked by τ_1, who locks a resource whose ceiling priority is higher than or equal to the priority of τ_2)

$B_3 = \max\{H_{1,a}, H_{2,a}\} = \max\{5, 1\} = 5$ (since τ_3 may be blocked by τ_1 or τ_2 who lock semaphores whose ceiling priorities are higher than or equal to the priority of τ_3)

Since both task and ceiling priorities have changed we need to redo the response-time analysis:

$R_1 = C_1 + \lceil \frac{R_1}{T_2} \rceil \cdot C_2 + \lceil \frac{R_1}{T_3} \rceil \cdot C_3$. Assume that $R_1^0 = C_1 = 7$:

$R_1^1 = 7 + \lceil \frac{7}{28} \rceil \cdot 6 + \lceil \frac{7}{30} \rceil \cdot 6 = 7 + 1 \cdot 6 + 1 \cdot 6 = 19$

$R_1^2 = 7 + \lceil \frac{19}{28} \rceil \cdot 6 + \lceil \frac{19}{30} \rceil \cdot 6 = 7 + 1 \cdot 6 + 1 \cdot 6 = 19 \leq D_1 = 19$

$R_2 = C_2 + B_2 + \lceil \frac{R_2}{T_3} \rceil \cdot C_3$. Assume that $R_2^0 = C_2 = 6$:

$R_2^1 = 6 + 5 + \lceil \frac{6}{30} \rceil \cdot 6 = 6 + 5 + 1 \cdot 6 = 17$

$R_2^2 = 6 + 5 + \lceil \frac{17}{30} \rceil \cdot 6 = 6 + 5 + 1 \cdot 6 = 17 \leq D_2 = 17$

$R_3 = C_3 + B_3 = 6 + 5 = 11$.

Thus, all tasks will meet their deadlines, and the task set will be schedulable, if $D_3 = 11$.

PROBLEM 6

a) Perform processor-demand analysis:

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

Then, derive the set K of control points: $K_1 = \{5, 13, 21, 29\}$, $K_2 = \{13, 29\}$ and $K_3 = \{19\}$ which gives us $K = K_1 \cup K_2 \cup K_3 = \{5, 13, 19, 21, 29\}$.

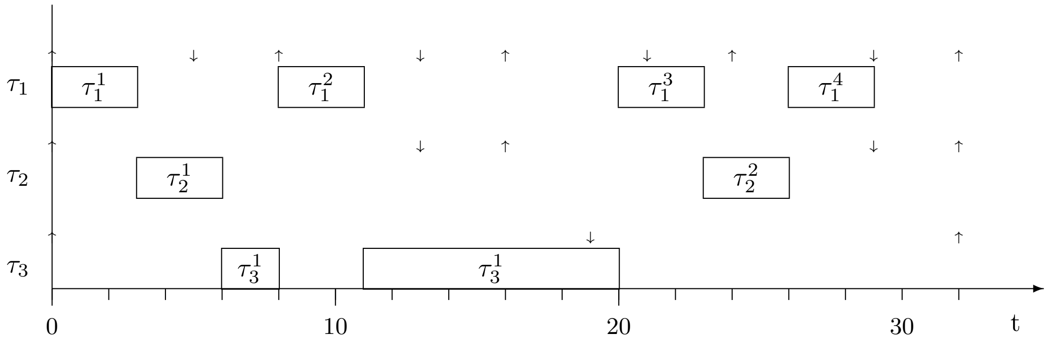
Schedulability analysis now gives us:

L	$N_1^L \cdot C_1$	$N_2^L \cdot C_2$	$N_3^L \cdot C_3$	$C_P(0,L)$	$C_P(0,L) \leq L$
5	$(\lfloor \frac{(5-5)}{8} \rfloor + 1) \cdot 3 = 3$	$(\lfloor \frac{(5-13)}{16} \rfloor + 1) \cdot 3 = 0$	$(\lfloor \frac{(5-19)}{32} \rfloor + 1) \cdot 11 = 0$	3	OK
13	$(\lfloor \frac{(13-5)}{8} \rfloor + 1) \cdot 3 = 6$	$(\lfloor \frac{(13-13)}{16} \rfloor + 1) \cdot 3 = 3$	$(\lfloor \frac{(13-19)}{32} \rfloor + 1) \cdot 11 = 0$	9	OK
19	$(\lfloor \frac{(19-5)}{8} \rfloor + 1) \cdot 3 = 6$	$(\lfloor \frac{(19-13)}{16} \rfloor + 1) \cdot 3 = 3$	$(\lfloor \frac{(19-19)}{32} \rfloor + 1) \cdot 11 = 11$	20	FAIL
21	$(\lfloor \frac{(21-5)}{8} \rfloor + 1) \cdot 3 = 9$	$(\lfloor \frac{(21-13)}{16} \rfloor + 1) \cdot 3 = 3$	$(\lfloor \frac{(21-19)}{32} \rfloor + 1) \cdot 11 = 11$	23	FAIL
29	$(\lfloor \frac{(29-5)}{8} \rfloor + 1) \cdot 3 = 12$	$(\lfloor \frac{(29-13)}{16} \rfloor + 1) \cdot 3 = 6$	$(\lfloor \frac{(29-19)}{32} \rfloor + 1) \cdot 11 = 11$	29	OK

The processor demand in time intervals $L = 19$ and $L = 21$ exceeds the length of the interval. Hence, not all tasks will meet their deadlines.

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

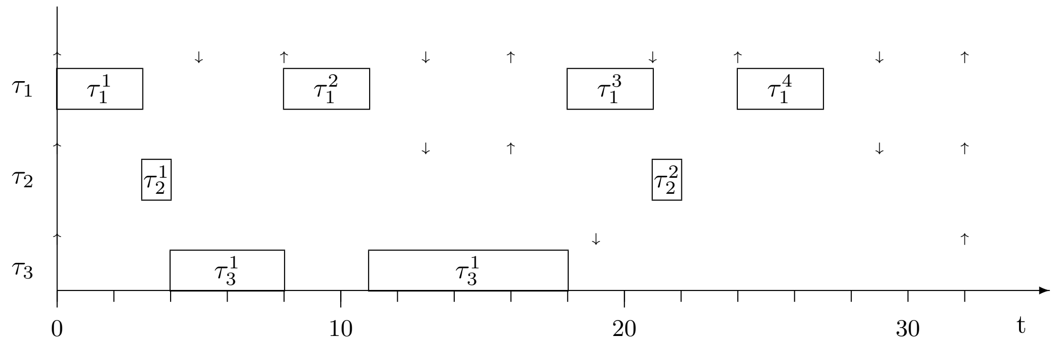
A simulation of the tasks using EDF scheduling in the interval $[0, LCM]$ gives the following timing diagram. We see that, also here, some tasks will not meet their deadlines. More specifically, task τ_3 misses its deadline at time $t = 19$ and task τ_1 misses its deadline at time $t = 21$.



c) Based on the analysis in sub-problem a) we see that there is a processor demand surplus of $23-21 = 2$ time units in control point $L = 21$. Since task τ_2 contributes with one instance at that control point, we need decrease its WCET by 2 time units to $C_2 = 1$. Re-doing the processor-demand analysis with the new value of C_2 verifies that the task set is now schedulable:

L	$N_1^L \cdot C_1$	$N_2^L \cdot C_2$	$N_3^L \cdot C_3$	$C_P(0,L)$	$C_P(0,L) \leq L$
5	$(\lfloor \frac{(5-5)}{8} \rfloor + 1) \cdot 3 = 3$	$(\lfloor \frac{(5-13)}{16} \rfloor + 1) \cdot 1 = 0$	$(\lfloor \frac{(5-19)}{32} \rfloor + 1) \cdot 11 = 0$	3	OK
13	$(\lfloor \frac{(13-5)}{8} \rfloor + 1) \cdot 3 = 6$	$(\lfloor \frac{(13-13)}{16} \rfloor + 1) \cdot 1 = 1$	$(\lfloor \frac{(13-19)}{32} \rfloor + 1) \cdot 11 = 0$	7	OK
19	$(\lfloor \frac{(19-5)}{8} \rfloor + 1) \cdot 3 = 6$	$(\lfloor \frac{(19-13)}{16} \rfloor + 1) \cdot 1 = 1$	$(\lfloor \frac{(19-19)}{32} \rfloor + 1) \cdot 11 = 11$	18	OK
21	$(\lfloor \frac{(21-5)}{8} \rfloor + 1) \cdot 3 = 9$	$(\lfloor \frac{(21-13)}{16} \rfloor + 1) \cdot 1 = 1$	$(\lfloor \frac{(21-19)}{32} \rfloor + 1) \cdot 11 = 11$	21	OK
29	$(\lfloor \frac{(29-5)}{8} \rfloor + 1) \cdot 3 = 12$	$(\lfloor \frac{(29-13)}{16} \rfloor + 1) \cdot 1 = 2$	$(\lfloor \frac{(29-19)}{32} \rfloor + 1) \cdot 11 = 11$	25	OK

An updated timing diagram verifies that all tasks now meet their deadlines.

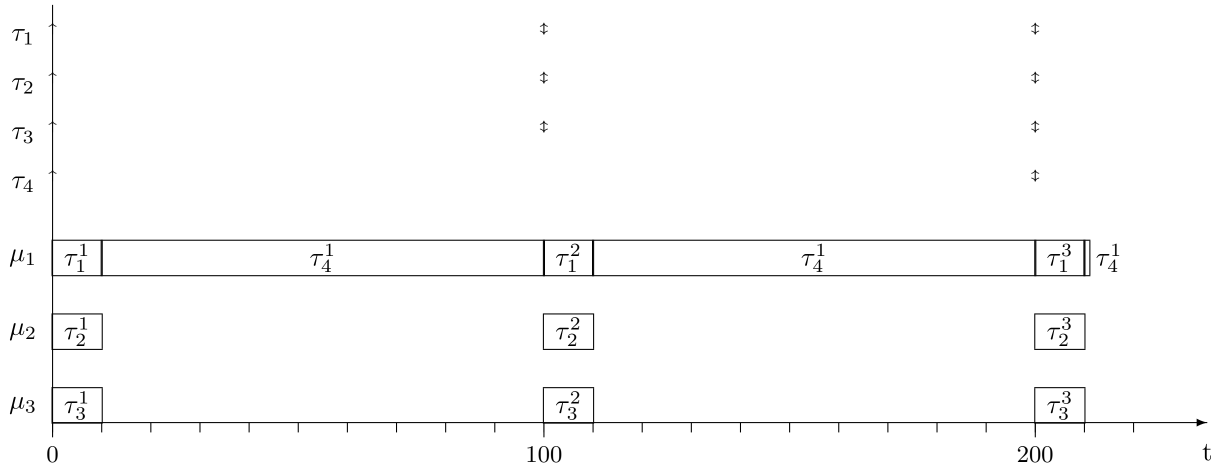


PROBLEM 7

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):



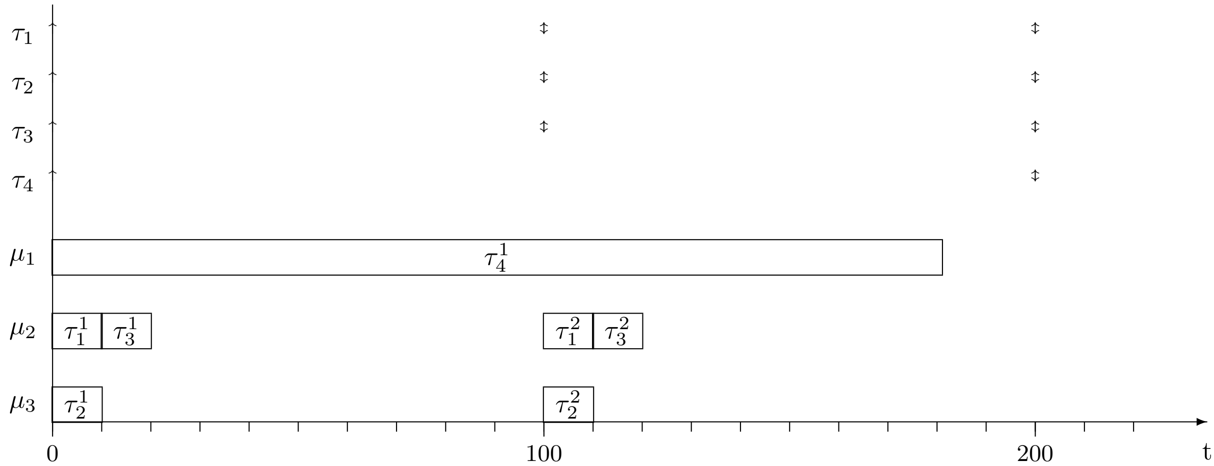
We observe that the first instance of task τ_4 completes its execution at $t = 211$ on processor μ_1, thereby missing its deadline at $t = T_4 = 200$. This happens despite there being significant processor capacity available on processors μ_2 and μ_3. A clear case of Dhall's effect!

b) 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.



c) 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

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

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.

d) 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 $0.1 + 0.705 = 0.805 < \mathrm{U}_{\mathrm{RM}(2)} = 2 \cdot (2^{1/2} - 1) \approx 0.828$, 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.