TECHNISCHE UNIVERSITEIT DELFT
Faculteit Elektrotechniek, Wiskunde en Informatica

IN4343 – Real Time Systems

June 29th 2017, from 09:00 to 12:00

Koen Langendoen

Question:	1	2	3	4	5	6	Total
Points:	10	15	10	25	20	10	90
Score:							

- This is a closed book exam
- You may use a **simple** calculator only (i.e. graphical calculators are not permitted)
- Write your answers with a black or blue pen, not with a pencil
- **Always justify your answers, unless stated otherwise**

- The exam covers the following material:
  (a) chapters 1-6, 8-9 of the book "Hard Real-Time Computing Systems (3rd ed)" by G. Buttazzo
  (b) the paper "The Worst-Case Execution-Time Problem" by Wilhelm et al. (except Section 6)
  (c) the paper "Transforming Execution-Time Boundable Code into Temporally Predictable Code" by P. Puschner
  (d) the paper "Best-case response times and jitter analysis of real-time tasks" by R.J. Bril, E.F.M. Steffens, and W.F.J. Verhaegh

Liu and Layland (LL) bound		$U_{lub}^{RM} = n(2^{1/n} - 1)$
Hyperbolic (HB) bound		$\prod_{i=1}^{n}(U_i + 1) \leq 2$
Response Time Analysis		$WR_i = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{WR_i + AJ_k}{T_k} \right\rceil C_k$ $BR_i = C_i + \sum_{k=1}^{i-1} \left(\left\lceil \frac{BR_i - AJ_k}{T_k} \right\rceil - 1 \right)^{+} C_k$ $w^{+} = \max(w, 0)$
Processor Demand		$g(t_1, t_2) = \sum_{r_i \geq t_1}^{d_i \leq t_2} C_i \qquad g(0, L) = \sum_{i=1}^{n} \left\lfloor \frac{L + T_i - D_i}{T_i} \right\rfloor C_i$
	schedulability	$\forall L \in D, \quad g(0, L) \leq L$ $D = \{d_k \mid d_k \leq \min(H, \max(D_{max}, L^*))\}$ $H = lcm(T_1, \ldots, T_n)$ $L^* = \frac{\sum_{i=1}^{n}(T_i - D_i)U_i}{1 - U}$
Polling Server	schedulability	$U_{lub}^{RM+PS} = U_s + n \left[\left(\frac{2}{U_s + 1} \right)^{1/n} - 1 \right]$ $\prod_{i=1}^{n}(U_i + 1) \leq \frac{2}{U_s + 1}$
	response time	$R_a = C_a + \Delta_a + F_a(T_s - C_s)$ $\Delta_a = \left\lceil \frac{r_a}{T_s} \right\rceil T_s - r_a \qquad F_a = \left\lceil \frac{C_a}{C_s} \right\rceil - 1$
Deferrable Server	schedulability	$U_{lub}^{RM+DS} = U_s + n \left[\left(\frac{U_s + 2}{2U_s + 1} \right)^{1/n} - 1 \right]$ $\prod_{i=1}^{n}(U_i + 1) \leq \frac{U_s + 2}{2U_s + 1}$
	response time	$R_a = C_a + \Delta_a - C_0 + F_a(T_s - C_s)$ $C_0 = \min(C_s(r_a), \Delta_a)$ $\Delta_a = \left\lceil \frac{r_a}{T_s} \right\rceil T_s - r_a \qquad F_a = \left\lceil \frac{C_a - C_0}{C_s} \right\rceil - 1$
NP scheduling	level-i busy period	$L_i = B_i + \sum_{h=1}^{i} \left\lceil \frac{L_i}{T_h} \right\rceil C_h \qquad N_i = \left\lceil \frac{L_i}{T_i} \right\rceil$ $B_i = \max_{j>i}\{C_j\}$
	response time	$s_{ik} = B_i + (k-1)C_i + \sum_{h=1}^{i-1} \left(\left\lfloor \frac{s_{ik}}{T_h} \right\rfloor + 1 \right) C_h$ $R_{ik} = (s_{ik} + C_i) - (k-1)T_i$ $R_i = \max_{k \in [1, N_i]}\{R_{ik}\}$
Elastic Model	utilization	$\forall i \quad U_i = U_{i0} - (U_0 - U_d)\frac{E_i}{E_S} \qquad$ where $E_S = \sum_{i=1}^{n} E_i$

Question 1 [10 points]

To avoid the intricacies of determining the WCET of a task, the code can be transformed into a single-path equivalent utilizing a predicated execution model [Puschner:2002]. Consider a simple microcontroller without caches, execution pipelines, branch prediction units, etc. A naive approach towards handling unbounded loops is to transform them into fixed loops as follows:

```
while (cond) {
    body
}
```

$\Rightarrow$

```
for (int i = 0; i<MAX_LB; i++) {
    if (cond) {
        body;
    }
}
```

Next, the if-rule will be applied to turn the body of the for loop into predicated instructions.

(a) [2 points] Explain the importance of choice of the `MAX_LB` constant.

Solution: The price of making the loop deterministic is that it always has to run for `MAX_LB` iterations (even if the condition is false on entrance), so the larger the bound, the more the overhead. Setting it too low, however, will lead to incorrect result, so being conservative is the only "safe"option.

(b) [3 points] Explain the problem of the naive translation, that is, why the two code fragments can result in different behavior.

Solution: The condition `cond` may contain a side effect, so executing it in the "extra" loops required to make it deterministic may result in erroneous behavior.

(c) [5 points] Provide a proper translation scheme and explain how the problem of the naive version is addressed.

Solution:
We introduce a boolean flag to ensure that the condition is no longer evaluated once it has become false.

```
while (cond) {
    body
}
```

$\Rightarrow$

```
int done = 0;
for (int i = 0; i<MAX_LB; i++) {
    if (!done && cond) {
        body;
    } else {
        done = 1;
    }
}
```

Question 2 [15 points]

The Earliest Due Date (EDD) is an "old" scheduling policy for handling aperiodic tasks [Jackson:1955]. It is known for being optimal in the sense of minimizing the maximum lateness.

(a) [3 points] Explain how EDD works.

Solution: EDD sorts the tasks (who all arrive at the same time) according to their relative deadlines, and schedules them from earliest to latest.

(b) [5 points] Proof the optimality of EDD. Hint: use an interchange argument.

Solution: See slide 26 'EDD Optimality' from the lecture on aperiodic scheduling.

EDD can be regarded as a special case of the "modern" Earliest Deadline First policy [Horn:1974].

(c) [3 points] Explain the differences between EDD and EDF in terms of the constraints they impose on the task set.

Solution: Using Graham's notation (slide 8, lecture on aperiodic scheduling) EDD is classified as $1|sync|L_{max}$ and EDF as $1|preem.|L_{max}$. EDD thus assumes that all tasks arrive at the same time. EDF assumes that tasks can be preempted.

(d) [4 points] Argue that EDD is a special case of EDF.

Solution: When scheduling a synchronous task set the notion of relative (EDD) and absolute deadlines (EDF) is the same. When EDF schedules a synchronous task set preemption does not occur as it will start with the task with the earliest deadline and run that to completion, then the next, and so on, just like EDD would.

Question 3 [10 points]

	C_i	T_i
τ_1	1	3
τ_2	1	4
τ_3	1	5

(i)

	C_i	T_i
τ_1	1	3
τ_2	1	4
τ_3	1	6

(ii)

	C_i	T_i
τ_1	1	3
τ_2	1	4
τ_3	2	5

(iii)

	C_i	T_i
τ_1	1	3
τ_2	1	4
τ_3	2	6

(iv)

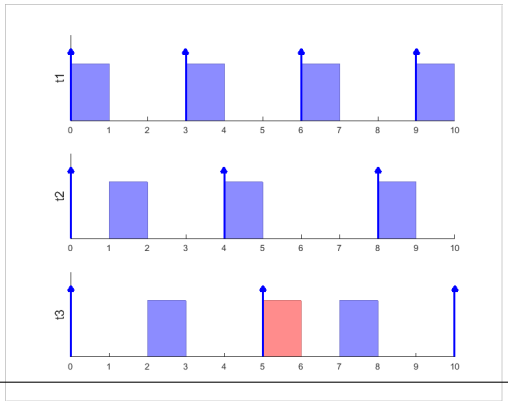
(a) [8 points] Which task sets are feasible under Rate Monotonic scheduling?

Solution:

(i) Feasible as HB $= (1 + 1/3) * (1 + 1/4) * (1 + 1/5) = 4/3 * 5/4 * 6/5 = 2$

(ii) Feasible as HB will be lower than for (i), which was already fine.

(iii) Unfeasible as follows from the critical instant



(iv) Feasible as follows from (iii) that showed a Lateness of 1 by task τ_3.

(b) 2 points Which task sets are feasible under Earliest Deadline First scheduling?

Solution: Tasks sets (i), (ii), and (iv) are feasible as RM can do them and EDF is optimal. Task set (iii) is also feasible as follows from the processor utilization ($1/3 + 1/4 + 2/5 = 59/60$) being less than 1.

Question 4 [25 points]

Consider the following task set:

	C_i	D_i	T_i
τ_1	2	5	6
τ_2	3	6	7
τ_3	5	15	21

(a) 10 points Use the processor demand criterion to demonstrate that the above task set is unfeasible under EDF scheduling.

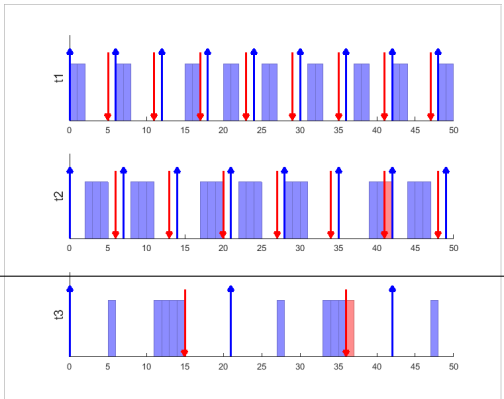
Solution: Processor utilization is 1 (!), so L^* can't be computed and we need to go as far as the Hyper period (42).

	demand
g(0,5)	2
g(0,6)	5
g(0,11)	7
g(0,13)	10
g(0,15)	15
g(0,17)	17
g(0,20)	20
g(0,23)	22
g(0,27)	25
g(0,29)	27
g(0,34)	30
g(0,35)	32
g(0,36)	37

Task τ_3 (which must complete by $t = 36$) misses its deadline!

(b) 5 points Determine the worst-case response times of the three tasks. *Hint: draw a time line.*

Solution:



Task	Response time
τ_1	5
τ_2	7
τ_3	16

Alternative solution is to schedule task τ_2 at time $t = 36$ allowing it to finish within its deadline, but then task τ_1 incurs a deadline violation.

Task	Response time
τ_1	6
τ_2	6
τ_3	16

The feasibility (under EDF) of a task set depends on the phase offsets of the tasks involved. The basic principle of the processor demand criterion still holds, but the demand side has to be adapted. Following the 2017-04-13 exam, an intuitive approach would be to account for the offset in the numerator of the demand as follows:

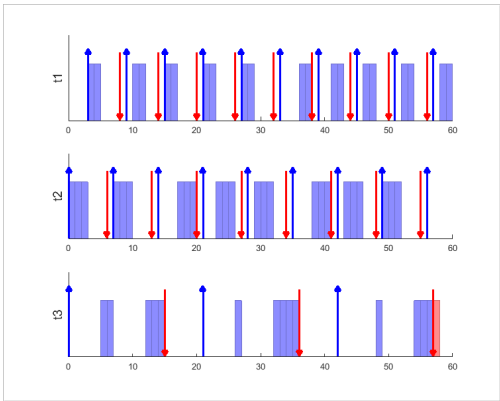
$$g'(0, L) = \sum_{i=1}^{n} \left\lfloor \left(\frac{L - \phi_i + T_i - D_i}{T_i} \right)^+ \right\rfloor C_i$$

(c) [2 points] Give a rationale for subtracting the offset from the interval length L.

Solution: The offset makes a task start "late" so by subtracting it the demand jumps exactly offset-late as needed. (Adding the offset would make the demand jump early.)

(d) [5 points] Draw a timeline to determine the first deadline violation (at $t_{violaton}$) when τ_1 has an offset of 3 time units.

Solution:



Note that the schedule at $t = 21$ has all tasks start from "scratch" as in (a), so the violation will occur 36 time units later.

$t_{violation} = 21 + 36 = 57$

(e) [3 points] Compute the revised processor demand at the time of the violation, that is, compute $g'(0, t_{violation})$. What can you conclude?

Solution: $g'(0, t_{violation}) = g'(0, 57) = 57$ which implies that there is **no** deadline violation at that time. This contradicts the timeline, so the revised demand criteria g' must be wrong! (The problem is that at time $t = 20$ there is no available task, so that slot goes by wasted, creating a problem later at $t = 57$; the idle time must somehow be accounted for.)

Question 5 [20 points]

Consider the following periodic tasks and aperiodic jobs:

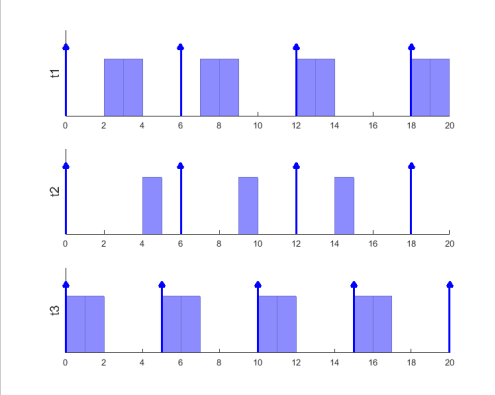
	C_i	T_i
τ_1	2	6
τ_2	1	6

	a_i	C_i
J_1	6	2
J_2	10	2

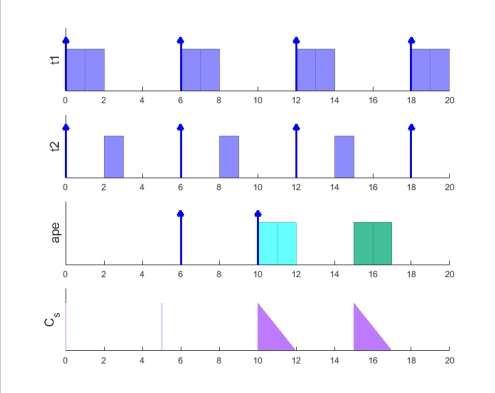
In the case of RM scheduling we may resort to fixed priority servers to schedule the jobs. Assume the servers are configured as $< C_s = 2, T_s = 5 >$.

(a) [6 points] Can a **polling server** be used without compromising the feasibility of the periodic tasks? Determine the response times of the two jobs.

Solution: The periodic tasks are fine if adding the polling server still results in a feasible RM schedule. Util $= 0.9 \leq 1$, but HB $= 4/3 * 7/6 * 7/5 = 196/90 > 2$, so we have to resort to the critical instant to show feasibility.



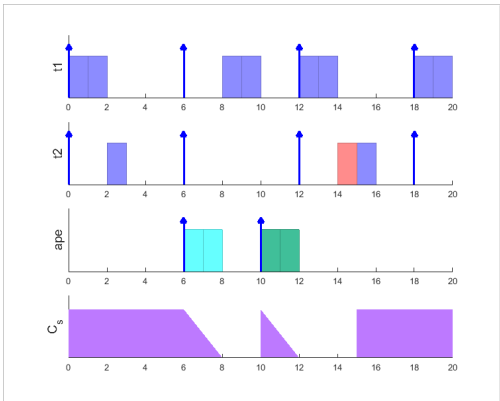
The response times ($J_1 = 6, J_2 = 7$) follow from the timeline:



(b) [6 points] Can a **deferable server** be used without compromising the feasibility of the periodic tasks? Determine the response times of the two jobs.

Solution: Compared to a polling server we now face the issue of jitter as the deferable server schedules a task as soon as it gets the chance. We can compute the worst case response time of task τ_2 factoring in a jitter of $5 - 2 = 3$ for the server iterating towards a fixed point of 8: $R_2 = 1 + \lceil \frac{8+3}{5} \rceil + \lceil \frac{8}{6} \rceil 2 = 1 + 3 + 4 = 8$. Thus, the configuration with a polling server may (depending on the actual job arrivals) compromise the timely execution of task τ_2. (The worst case response time of task τ_1 can be iterative computed as $R_1 = 2 + \lceil \frac{4+3}{5} \rceil = 4$.)

The response times ($J_1 = 2, J_2 = 2$) follow from the timeline (notice the deadline miss of task τ_2):

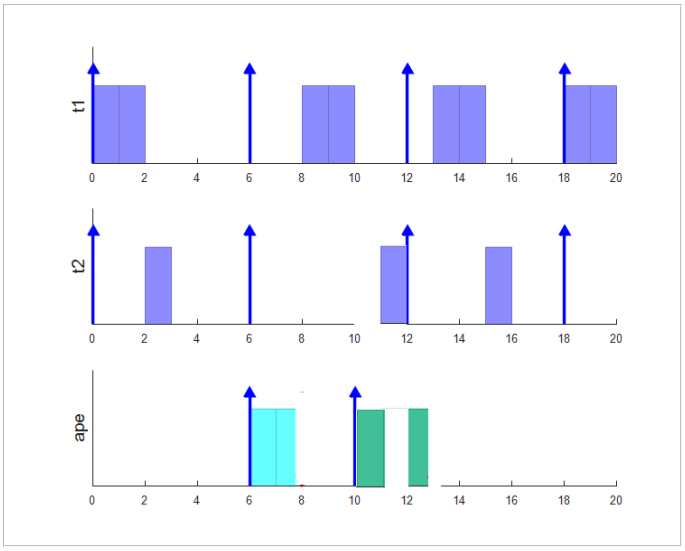


(c) [4 points] **Slack stealing** is the most advanced fixed-priority scheduler as it is capable of utilizing every available cycle for executing aperiodic jobs. Yet, it is not optimal. Explain what the issue is; give an example.

Solution: Slack stealing cannot look into the future and has to make a decision as soon as a job becomes available. See slide 41 'Scheduling dilemma' from the lecture on fixed priority servers for an example.

(d) [4 points] Determine the response times of the two jobs when slack stealing is applied.

Solution: The response times ($J_1 = 2, J_2 = 3$) follow from the timeline



Question 6 [10 points]

Robust EDF is part of the class of value-based scheduling policies designed to handle overruns. Consider the following task set:

	a_i	WC_i	AC_i	d_i	V_i
τ_1	4	3	2	9	7
τ_2	3	5	3	12	10
τ_3	1	5	5	13	5
τ_4	0	3	3	16	2
τ_5	2	4	4	17	3

where WC_i denotes the worst-case execution time, and AC_i the actual execution time.

(a) [5 points] Describe how Robust EDF handles (temporary) overloads.

Solution: Robust EDF maintains an additional reject queue. When an overload is detected, the least valued task(s) is placed in the reject queue. When the load drops below 1, then the highest valued task(s) in the reject queue that "fits" (i.e. new schedule is feasible) is put back into the ready queue.

(b) [5 points] Determine the value acquired by Robust EDF on the given task set.

Solution: The task set is taken straight from slides 64-66 of the lecture on 'Handling overload'.
At time $t = 4$ the system becomes overloaded and task τ_3 is rejected. At time $t = 8$ the system becomes underloaded and task τ_3 can be recovered. The total value acquired is the sum of the values of all tasks, which equates to 27.