EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science

Examination of Real-time Systems (2IMN20)
on Wednesday, July 5^{th} 2017, 18.00h-21.00h.

First read the entire examination. There are 5 exercises in total. Grades are included between parentheses at all parts and sum up to 10 points. *Motivate all your answers.* Good luck!

1. *Reference model*

   (a) (0.5) Explain the notions *jitter* and *drift* in your own words.
   Answer: *Jitter* is the fluctuation in, for example, *activation*, *start*, *computation*, *finalization*, or *response times* of a task. For real-time systems, jitter shall typically be *bounded*. Drift is defined as *unbounded* jitter, i.e. the jitter may become arbitrary large.

   (b) (0.5) Give at least 3 types of jitter.
   Answer: Types may include (*i*) activation jitter, (*ii*) start jitter, (*iii*) computation jitter, (*iv*) finalization jitter, and (*v*) response jitter.

   (c) (0.5) Explain the difference between periodic tasks with jitter and elastic tasks in your own words.
   Answer: Elastic tasks have an unbounded activation jitter, whereas periodic tasks with jitter have a bounded activation jitter.

   (d) (0.5) Give an example illustrating the need for a constraint on jitter for a controlling system.
   Answer: In general: to guarantee control performance of real-time tasks.
   Control theory typically assumes strictly periodic sensing and actuation. Start-jitter and finalization jitter of a task may cause sensing (upon start) and actuation (upon finalization) to fluctuate, i.e. neither will be strictly periodic anymore. Although jitter has (at least partially) been addressed in control theory, jitter typically has a negative influence on the control performance of real-time tasks. As a result, a system may have constraints on the sensing- and actuation jitter of a controlling system to guarantee (a minimal) control performance.
   Note that sporadic and elastic tasks experience *drift* (unbounded jitter) in their activation times. Arguments about deadline misses due to jitter are therefore not accepted without sufficient justification/motivation.

2. *Response time analysis*

   (a) (0.5) Describe at least 2 reasons for best-case response time analysis.
   Answer: to determine (*i*) whether or not a task meets its best-case deadline, (*ii*) a bound on the response jitter and finalization jitter of a task, e.g. to check *a*) whether or not the task meets its control performance or to determine *b*) the activation jitter of a task or message triggered by the task.

   (b) (0.5) An equation to determine the best-case response time of a task τ_i is given by equation (1).

$$x = BC_i + \sum_{j<i} \left(\left\lceil \frac{x - AJ_j}{BT_j} \right\rceil - 1 \right)^+ \cdot BC_j \tag{1}$$

   Give at least 4 conditions that need to hold such that equation (1) can be applied.
   Answer: Potential conditions include: (*i*) Tasks are scheduled by fixed-priority preemptive scheduling (FPPS); (*ii*) tasks do not suspend themselves; (*iii*) tasks have

unique priorities, and the task with the lower the subscript the higher the priority; (*iv*) scheduling and context switching overheads are ignored; (*v*) a job of τ_i is not started before its previous job has completed; (*vi*) the sum of worst-case deadline D_i and the activation jitter of τ_i is at most equal to the minimal inter-arrival time WT_i of τ_i, i.e. $WD_i + AJ_i \leq WT_i$.

(c) (0.5) Which solution of (1) represents the best-case response time of task τ_i?
Answer: The *largest* (positive) solution.

(d) (0.5) Is it possible to use equation (1) also for sporadic tasks and elastic tasks? If yes, explain how the terms BT_j and AJ_j in (1) shall be used. If no, explain why.
Answer: Yes. For a sporadic task τ_j, $BT_j = \infty$. For an elastic task, BT_j is part of the characteristics of a task, hence given. For both a sporadic as well as an elastic task τ_j, $AJ_j = 0$.
See also `RTS.B5-Analysis-2-FPPS` exercise 4 "Lifting assumptions".

(e) (1.0) Prove that ι_i in equation (2) represents a proper initial value to start the iterative procedure to find the best-case response time BR_i of task τ_i when $\sum_{j<i} BU_j < 1$, where BU_j represents the best-case utilization $\frac{BC_j}{BT_j}$ of task τ_j.

$$\iota_i = \frac{BC_i}{1 - \sum_{j<i} BU_j}. \tag{2}$$

Answer: The best-case response time BR_i of task τ_i is the largest positive solution of (1), and the iterative procedure therefore starts with an upper-bound. Hence, we have to prove that $BR_i \leq \iota_i$. To this end, we derive

$$\begin{aligned}
BR_i &= BC_i + \sum_{j<i} \left(\left\lceil \frac{BR_i - AJ_j}{BT_j} \right\rceil - 1 \right)^+ \cdot BC_j && (3)\\
&\leq BC_i + \sum_{j<i} \left(\left\lceil \frac{BR_i}{BT_j} \right\rceil - 1 \right) \cdot BC_j && (4)\\
&\leq \{\lceil x \rceil - 1 \leq x\} BC_i + \sum_{j<i} \frac{BR_i}{BT_j} \cdot BC_j && (5)\\
&= BC_i + BR_i \cdot \sum_{j<i} BU_j. && (6)
\end{aligned}$$

Hence, for $\sum_{j<i} BU_j < 1$, we get $BR_i \leq \frac{BC_i}{1-\sum_{j<i} BU_j}$.
Intuitively, the term $1 - \sum_{j<i} BU_j$ represents the *average* fraction of the time (based on best-case values of higher priority tasks) available to task τ_i. The best-case fraction is higher than or equal to the average fraction. By dividing the best-case computation time BC_i by that average fraction, we therefore get a value that is higher than or equal to the best-case response time BR_i of τ_i.
Finally note that a similar question (for worst-case response times) was asked in the exam of August 30^{th}, 2006.

(f) (1.0) Consider five tasks satisfying the requirements for applying equation (1), with characteristics as given below.

	BT	BD	AJ	BC
τ_1	5	1	1	1
τ_2	8	2	0	2
τ_3	9	3	0	3
τ_4	12	1	0	1
τ_5	44	8	0	4

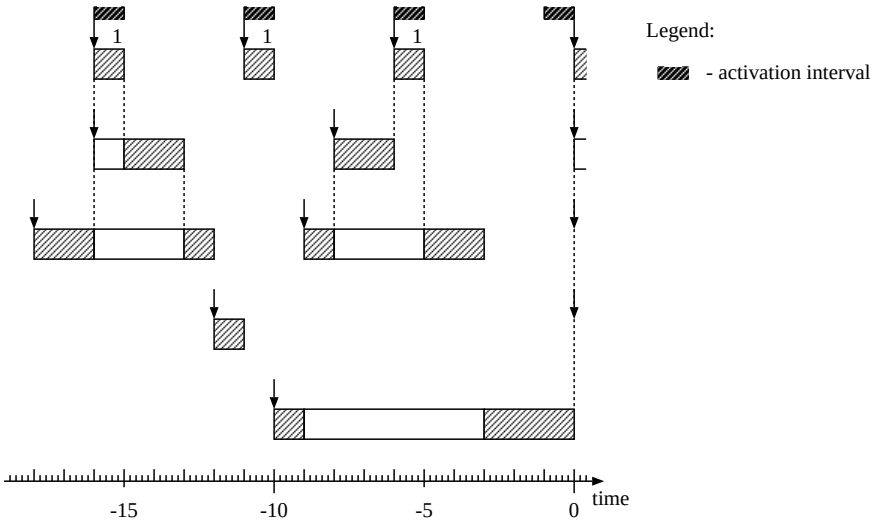

Figure 1: Timeline with an optimal instant for task τ_5.

Determine the best-case response time BR_5 of task τ_5 using a timeline.
Answer: See Figure 1. Note that $BR_3 = 10$.

(g) (0.5) Determine the best-case response time BR_5 of task τ_5 using equation (1).
Hint: $\iota_5 < 31$.
Answer $BR_5 = 10$.

3. *Servers*
Consider a (fixed-priority) polling server, characterized by a period T_{PS}, a capacity C_{PS}, and a phasing ϕ_{PS}.

(a) (0.5) Describe how the server shall be incorporated in the worst-case and best-case response time analysis.
Answer: See `RTS.B4-Polices-2-FP-servers` exercise 3.

(b) (0.5) Let the server be scheduled at the highest priority in a system. Suppose an aperiodic request A of $2 \times C_{\text{PS}}$ arrives at time a_A. What is the best-case and worst-case response time of that aperiodic request when it is handled by the server, assuming no pending load of aperiodic requests at time a_A and no arrival of aperiodic requests at a_A other than A?
Answer: The best-case occurs when the aperiodic request arrives at $a_A = \phi_{\text{PS}} + k \times T_{\text{PS}} + \epsilon$, i.e. at the arrival of the server, and the entire capacity of the polling server is therefore still available at time a_A. The best-case response time BR_A of the request is subsequently identical to $T_{\text{PS}} + C_{\text{PS}}$.

The worst-case happens when the aperiodic request arrives at $a_A = \phi_{\text{PS}} + k \times T_{\text{PS}} + \epsilon$, i.e. an infinitesimal time ϵ *after* an activation of the server, and the entire capacity of the server has been depleted at time a_A. The worst-case response time WR_A of the request is now identical to $2 \times T_{\text{PS}} + C_{\text{PS}} - \epsilon$.

Answer: Note that a similar question for a deferrable server was asked in the exam of 2017 April 17^{th}.

4. *Resource Access Protocols*

(a) (0.5) The stack resource policy (SRP) allows tasks to share a single stack. Does the priority inheritance protocol (PIP) also allow tasks to share a single stack? If yes explain why, if not explain why not.

Answer: No. Sharing a single stack does not allow tasks to *interleave* their executions, e.g. task τ_i starts execution, is preempted by task τ_j, which allows task τ_i to continue before it completes. In such a situation, a single shared stack would cause task τ_i to overwrite the data of task τ_j which is still on the stack. Unfortunately, PIP may cause interleaved executions when tasks τ_i and τ_j share a mutually exclusive resource other than the processor.

Note that the fact that SRP uses *resource ceilings* and PIP is not a proper argument. As a counter example, PCP also uses resource ceilings, but does not allow a single shared stack, simply because PCP also allows tasks to have interleaved executions.

(b) (1.0) Construct an example for PIP such that the priority of a task is higher than its original priority *before* releasing a resource and the priority is not changed upon releasing the resource.

Answer: Consider three tasks τ_1, τ_2, and τ_3 and two shared resources R_1 and R_2. Let τ_1 and τ_3 share R_1 and τ_2 and τ_3 share R_2. Moreover let τ_3 first request R_1 and subsequently R_2 in a nested fashion. When τ_3 is preempted by τ_1 *before* it accesses R_2, and τ_1 is blocked on R_1, the priority of τ_3 will be raised to the priority of τ_1, and it will request, access, and release R_2 at that priority.

Note: The example can be simplified to two tasks, by having R_2 also be used by τ_1.

5. *Implementing periodic tasks*
There are several ways periodic tasks can be implemented, e.g. either by the programmer using generic primitives of an RTOS (such as timers) or via dedicated primitives provided by the RTOS.

   (a) (0.5) Describe the 4 types of timers that could be provided by an RTOS and give an example for each of them.

   (b) (0.5) Describe 2 dedicated primitives for implementing periodic tasks that could be provided by an RTOS.

   Answers: See `RTS.C7-Periodic Tasks`.