Real-time Concepts for Embedded Systems
WT 18/19
Assignment 3

Submission Deadline: Sunday, 02.12.2018, 23:55

- *Submit the solution in PDF via Ilias (only one solution per team).*
- *Respect the submission guidelines (see Ilias).*

1 Network Flow Graph [8 points]

a) [3 points] Figure 1 shows a set of jobs with their precedence constraints. The values in the paranthesis above each job are their given release times and deadlines. Fill in the empty boxes with the effective release times and deadlines for each job. (Note: 0.5 points for each correct box)

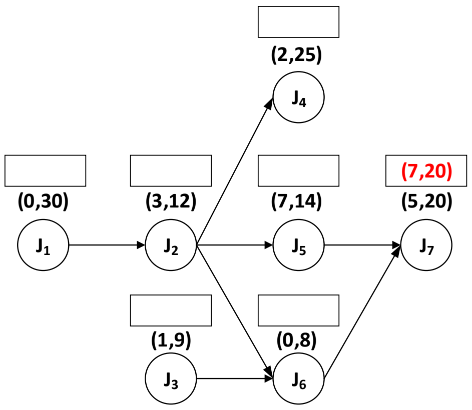

Figure 1: Effective Release time and Deadline

b) i. [3 points] Draw a network-flow graph that we can use to find a preemptive cyclic schedule of the periodic tasks $T_1 = (3, 1)$, $T_2 = (5, 1, 7)$, $T_3 = (5, 2, 7)$. Annotate the edges with their assigned flows and capacities. Using the concept of max-flow discussed in the lecture, check whether the task set is schedulable or not. Justify your answer. Note: Show the schedule by drawing a timing diagram.
Grading scheme: 2 points for the annotations and 1 point for the correct answer and justification.

ii. [2 points] Consider the same task set from the previous subpart. (i.e. $T_1 = (3, 1)$, $T_2 = (5, 1, 7)$, $T_3 = (5, 2, 7)$). Now the tasks T_2 and T_3 have the following precedence constraint between them.

$$Pri(T_3) > Pri(T_2) \tag{1}$$

Now check whether the schedule generated in i) satisfies this precedence constraint. Justify your answer.
Note: Support your answer by drawing a timing digram for the generated schedule.

2 Aperiodic and sporadic tasks [5 points]

In this question, we deal with scheduling of aperiodic and sporadic tasks.

a) [2 points] Consider the given frame-based schedule depicted in Figure 2. Each frame is 6 time units long, and the schedule consists of 8 frames in total. The figure shows the schedule that spans for 48 time units. The schedule is split into two rows with the first row starting at time 0 and the second row starting at time 24. Given in Table 1 are preemptable sporadic tasks that are to be scheduled to the existing schedule. Decide with the help of the acceptance test, whether a sporadic task can be scheduled. Give the schedule including the sporadic tasks.

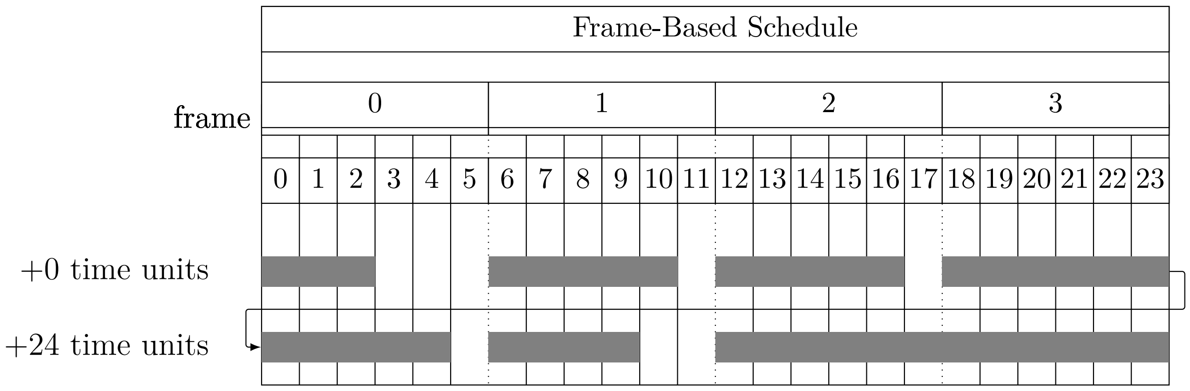

Figure 2: Frame-based schedule with periodic tasks depicted in grey.

Table 1: Sporadic Task set

Sporadic Task i	Release time r_i	deadline d_i	execution time e_i
S 1	4	7	2
S 2	5	27	5
S 3	6	31	2
S 4	10	35	2
S 5	20	35	2

b) [3 points] Consider a scenario with periodic tasks and aperiodic tasks. The hyper-period for the period tasks is 42 time units, and the sum of execution times of all periodic tasks is 27. We have the following information given in Table 2 about the aperiodic tasks, which occur in the system. What is the average response time W of any aperiodic task in this scenario? Explain your calculation.

Table 2: Aperiodic Task set

Task i	$\mathrm{E}[\beta_i]$	$\mathrm{E}[\beta_i^2]$	λ_i
Task 1	2.5	8.25	$\frac{1}{31}$
Task 2	3.375	12.625	$\frac{1}{45}$
Task 3	2.25	6	$\frac{1}{54}$
Task 4	2.5	8	$\frac{1}{32}$

3 Rate Monotonic Scheduling and Deadline Monotonic Scheduling [16 points]

a) [2 points] A task set $\mathbb{T}_1 = \{A, B, C, D\}$ where the corresponding computation times and periods of the tasks are given in Table 3. Check whether the task set can be scheduled on a uniprocessor system using the Rate Monotonic Scheduling algorithm. (Note: Assume all the tasks have zero phasing.)

Table 3: Task Set $\mathbb{T}_1$

Task	**Computation Time**	**Period/Deadline**
A	4	15
B	4	20
C	4	30
D	4	60

b) [5 points] Consider the same task set as in the first part, except that the computation times of Tasks C and D are changed from 4 to 8 as given in Table 4. Now check the schedulability of the new task set for Rate Monotonic Scheduling using the various tests given in the lecture. Show all your calculations for the performed tests.

Table 4: Task Set $\mathbb{T}_2$

Task	**Computation Time**	**Period/Deadline**
A	4	15
B	4	20
C	8	30
D	8	60

c) [4 points] According to the domain experts, suppose that task D is the most important of all the tasks. (i.e. $Pri(D) > Pri(A) > Pri(B) > Pri(C)$). Now using the concept of *Period Transformation* discussed in the lecture, modify the task set such that it is schedulable and check the schedulability of the modified task set for Rate Monotonic Scheduling. Note: Show the schedule by drawing a timing diagram.

d) [5 points] Consider the same task set from part b). In addition to the computation times, both of tasks C and D now have blocking times of 4 and 8 time units respectively, as given in Table 5. Now check the schedulability of the new task set $\mathbb{T}_3$ for Rate Monotonic Scheduling using the various tests given in the lecture. Show all your calculations for the performed tests.

Table 5: Task Set $\mathbb{T}_3$

Task	**Computation Time**	**Period/Deadline**	**Blocking Time**
A	4	15	0
B	4	20	0
C	8	30	4
D	8	60	8

4 Cyclic Scheduling with Integer Linear Programming (ILP) [16 points]

In this question, we will deal with cyclic scheduling, focusing on how compute a static schedule table. The computation of the static schedule table can be done offline. Thus this scenario enables the utlization of computationally-expensive methods to compute an "optimal" schedule.

In this question, we consider the following scenario: You are given a task set $\mathbb{T}$ consisting of different tasks T_i, with $i \in \{1..N\}$. In this scenario, tasks **cannot** be preempted. The tasks may have different execution times, but all tasks have the same period p, i.e. there is one job of each task executed in each cycle. In this scenario, every task has an effective release time *relative to the start of the period* and an effective deadline *relative to the start of the period*. These effective release times and deadlines can originate from inter-task dependencies, such as precedence constraints which were discussed in the lecture.

To formulate the scheduling problem for this scenario an Integer Linear Program (ILP) together with an ILP solver is used to compute the schedule. We use **zimpl** to model the scheduling problem and **SCIP** as ILP solver (download at `https://scip.zib.de`).

The zimpl-executable can be downloaded from `https://zimpl.zib.de`. The zimpl executable parses the description of an ILP in the ZIMPL language and generates the input (an ILP instance) for an ILP solver from that description. Information about modeling an ILP with ZIMPL can be found at `https://zimpl.zib.de/download/zimpl.pdf`. ZIMPL also allows the separation of ILP model and the problem instance, i.e. it allows us to write a generic model for—in our case—a scheduling problem which we can "compile" to concrete ILP instances for different task sets.

The provided zip-file contains three files: `scheduler.zpl`, `period.dat` and `taskset.lst`. In the file `taskset.lst`, each line contains the properties of one task $T_i \in \mathbb{T}$ with the following formating:

$$i \qquad r_i \qquad d_i \qquad e_i$$

with i the index of T_i, r_i the effective release time of T_i relative to the start of the period, d_i the effective deadline of T_i relative to the start of the period and e_i the execution time of T_i. The file `period.dat` contains the length of the period.

Whereas the previous two files contain the ILP parameters, `scheduler.zpl` is a skeleton of the ILP model in the ZIMPL language. In this task, you have to extend the file `scheduler.zpl` as described in the remainder of this question. The statements in provided file `scheduler.zpl` import the parameters from the parameter files (`taskset.lst`,`period.dat`) and define some variables as follows:

Predefined parameters:

`Ti`	: $\{0, 1, 2, \ldots\}$	set of task indices
`Tr[i]`	: $[r_0, r_1, r_2, \ldots]$	array of relative release times
`Td[i]`	: $[r_0, r_1, r_2, \ldots]$	array of relative deadlines
`Td[i]`	: $[r_0, r_1, r_2, \ldots]$	array of execution times
	`p`	period of all tasks

Predefined variables:

`var ts[<i> in Ti]` : $\begin{Bmatrix} ts_1 \\ ts_2 \\ \vdots \end{Bmatrix}$ where ts_i denotes the start of scheduled execution of T_i

`var te[<i> in Ti]` : $\begin{Bmatrix} te_1 \\ te_2 \\ \vdots \end{Bmatrix}$ where te_i denotes the end of scheduled execution of T_i

The ILP model in the file `scheduler.zpl` shall be completed according to the instructions below, and finally solved with an ILP solver.

Please make sure to comment all your additions in the file `scheduler.zpl` with verbose comments that explain what each statement is intended to do and why it is necessary.

a) [1 point] Add the following constraints in ZIMPL to `scheduler.zpl`:

$$\forall T_i : te_i - ts_i = e_i$$

These constraints ensures that task T_i is scheduled to execute for e_i time units.

b) [1 point] Add constraints to `scheduler.zpl` that ensure that every task is scheduled after its relative release time and before its relative deadline.

c) [4 points] Add constraints (and variables, if necessary) that ensure that only one task is scheduled for execution at any time, i.e. given T_a and T_b, neither should T_a start while T_b is scheduled for execution, nor vice versa. In this exercise, there is **no priority** assigned to the periodic tasks. Therefore, if there are any two tasks T_a and T_b, where the interval between r_a and d_a intersects with the interval between r_b and d_b, such that both tasks could be executed in the intersecting time interval, we do not know the order of these two tasks in the final schedule a-priori.

Hint: To express conditional constraints, you can use either the "bigM"-approach or the ZIMPL language construct `vif`.

d) [6 points] Add constraints, variables and objective that maximize the number of tasks, which are scheduled with no gap inbetween. Two tasks T_a and T_b are scheduled without gap, if either $te_a = ts_b$ or $te_b = ts_a$.

e) [4 points] Build the ILP instance and solve it. You build the ILP instance by executing the command `zimpl scheduler.zpl` on the command line (e.g. CMD, Powershell, bash) in the directory where the file `scheduler.zpl` is stored. This will "compile" the ILP model and the parameters into a text file (with suffix ".lp") which contains the ILP instance in the LP format. LP format is a common format that is understood by most ILP solvers.

If you use SCIP, you can solve this ILP instance directly by executing the command `scip -c "read scheduler.lp" -c "optimize" -c "write solution out.txt" -c quit`. This commands writes the solution to the file `out.txt`.

If you solved question part d): How many tasks can scheduled without gap for the given task set? Give the schedule.

Alternative: If you could not solve question part d): Add the necessary constraints, variables and objective function, that maximize the slack, i.e. the interval at the end of the period, where no periodic tasks are scheduled. How large is this interval? Give the schedule.