02225 DRTS Exercise description (draft)

This document describes the exercise for the 02225 course—Distributed Real-Time Systems (DRTS). The goal of the exercise is to prepare you for the project. You will have to implement a simple simulator and a simple schedulability analysis tool. The exercise considers fixed-priority preemptive scheduling.

Any programming language and any libraries can be used for the implementation. Your programs should read files that contain the following models: (1) *Application model*. The model of the application consists of a set of tasks with constraints. For each task τ_i we know the best-case execution time (BCET) and the worst-case execution time (WCET). (2) *Architecture model*. For the exercise, the architecture is a single processing element, i.e., a single CPU or core. This part of the model can be omitted for the exercise, but it is needed in the project.

1 Very Simple Simulator (VSS)

Fixed-priority preemptive scheduling is presented in the lecture slides and in Chapter 4 of [1]. When the priorities are proportional (monotonic) to the rate of the tasks (the inverse of the period, i.e., $1/T$), this scheduling technique is also called *Rate Monotonic* scheduling (RM).

The simulator takes as input the application consisting of a set of tasks stored in a file and the simulation time. The output is a list of worst-case response times (WCRT) observed during the simulation for each task. A suggestion for implementation is available in Algorithm 1. You may implement it differently if you want, especially the part related to advancing the time.

Algorithm 1 Very Simple Simulator (VSS)

```
 1: n ← number of cycles
 2: initialize_priorities
 3: initialize_jobs
                                                              ▷ simulate
 4: currentTime ← 0
 5: while currentTime ≤ n and there are unfinished jobs do
 6:     ready_list ← get_ready(jobs_list, currentTime)
 7:     current_job ← highest_priority(ready_list)
 8:     if current_job ≠ null then
 9:         Δ ← AdvanceTime()          ▷ could advance the time with 1 time unit
10:         currentTime ← currentTime + Δ
11:         current_job.time ← current_job.time − Δ
12:         if current_job.time ≤ 0 then
13:             current_job.response ← currentTime − current_job.release
14:             remember worst-case response time
15:             remove(current_job, ready_list)
16:         end if
17:     else
18:         Δ ← AdvanceTime()
19:         currentTime ← currentTime + Δ
20:     end if
21: end while
```

Notes:

- If you want to simulate every single step, `AdvanceTime` can simply return 1 time unit each time. If you prefer to skip idle intervals, `AdvanceTime` could jump to the next release. You can decide on its exact implementation.

- Consider how you generate the random values for C_i. Are they uniformly distributed in the interval [BCET, WCET], or did you use another distribution?
- A `ready job` at the time moment `currentTime` is a job that has already been released, i.e., it has τ_i`.`job_j`.release` $\leq$ `currentTime`. The job executing at `currentTime` is the highest-priority job out of those that are ready.
- There is no need to explicitly "stop" or "preempt" a job already on the processor, since at every time instant `currentTime` (or after each `AdvanceTime` call) the simulator checks which job has the highest priority and decrements that job's execution time.
- `remember worst-case response time`: For each task, you have to print out its WCRT. This means that for a task τ_i you have to find the maximum response time over all the simulated jobs $\tau_i.job_j$ of the task.

2 Response-Time Analysis (RTA)

RTA is covered in detail in the lectures and is described in Chapter 4 of [1]. The analysis works under the simplifying assumptions mentioned in Chapter 4 (and listed in the footnote[1]. Note that these assumptions *are not needed* for the simulator you are implementing.

The analysis tool takes as input the application consisting of a set of tasks stored in a file and the priorities for each task. The tool outputs the worst-case response times (WCRTs). The WCRTs can be calculated using Figure 4.17 from [1], also presented in Algorithm 2. Note that the algorithm assumes that the tasks are sorted based on their priority. Make sure your algorithm works for any priorities, not only rate monotonic.

Algorithm 2 Response-Time Analysis (RTA)

function RTA_test(Γ)
for all $\tau_i \in \Gamma$ **do** ▷ in order of decreasing priority
  $I \leftarrow 0$
  repeat
    $R \leftarrow I + C_i$
    if $R > D_i$ **then**
      return UNSCHEDULABLE
    end if
    $I \leftarrow \sum_{j=1}^{i-1} \left\lceil \frac{R}{T_j} \right\rceil \times C_j$
  until R did not change
end for
return SCHEDULABLE

Discussion: How do the worst-case response times that you get with RTA compare to those you get with VSS? How many simulation runs do you need to run VSS to get the same numbers?

References

[1] Giorgio Buttazzo. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer, 2011.

[1] Single processor. Strictly periodic tasks. (There are no variations in arrival times.) All tasks are released as soon as they arrive. The periods are synchronized–all tasks start at the same time. Deadlines are equal to the periods. All tasks are independent. (No precedence or resource constraints.) No task can suspend itself. All overheads in the kernel are assumed to be zero. The tasks do not experience blocking times, e.g., from lower priority tasks.