Some of the answers to these questions can be found using the lecture slides, the recommended textbooks or other sources (question marked with *). Some questions may have more than one possible answer, or be more or less open for discussion. Note that no answers (solutions) will be given to these questions, but if help is needed the assistants will be available to answer questions. The concepts marked with yellow are important and should be fully understood.

Embedded Operating Systems

1. What are the characteristics and requirements of an embedded (real-time) system?
2. What are the differences between hard and soft real-time constraints?
3. Discuss the properties of an embedded OS's process management, memory management, and scheduling using AUTOSAR OS as an example.
4. What kind of events in an event triggered OS do you know?
5. Earliest deadline first (EDF) scheduling (non pre-emptive case)

Task $task_i$	Ready time r_i	Deadline d_i	Execution time Δe_i
1	0	7	1
2	0	13	5
3	0	7	2
4	0	5	4

Table 1: Task set for EDF exercise

In EDF scheduling, the processor is assigned to the task with the earliest deadline. If several tasks share the same deadline, they can be scheduled in any order. EDF can be applied for static and dynamic scheduling, to both, preemptive and non-preemptive tasks.

In the non-preemptive case, the algorithm is as follows. TS denotes the task set (i.e., all tasks to be run), TL the task list (i.e., scheduled tasks), and $c[TL[k]]$ the completion time of the last task that has been scheduled so far.

Apply algorithm EDFnp to the task set given in Table 1.

```
algorithm EDFnp

 method main (TS)
   TS = sort_deadline(TS); // sort TS by deadline
   EDF_schedule(∅, TS);

 method EDF_schedule (TL, TS)
```

```
  forall i in TS
   if (feasible (TL, i)) // feasibility check for task i
     TL = append (TL, i);

method feasible (TL, i)
  // check for overlappings; returns true / false
  return MAX (c[TL[k]], r[i]) + Δe[i] ≤ d[i];
```

6. Least slack time (LST) scheduling (also referred to as least laxity first) is described below. Apply pre-emptive LST scheduling to the task set given in Table 2.

Task $task_i$	Ready time r_i	Deadline d_i	Execution time Δe_i
1	1	6	2
2	5	9	2
3	4	16	5
4	6	12	4

Table 2: Task set for LST exercise

```
algorithm LST

 method main ()
  for each time t until t ≤ MAX(d_i)
   e_i^rem = Δ e_i − e_i^passed; // calculate remaining execution time
   slack_i = d_i − t − e_i^rem; // calculate slack time
   adjust(task_i, slack_i);

 method adjust (task, slacktime)
   // no pseudo code given here, informal description only:
   // re-assign task priority based on the task's slack time,
   // the task with the smallest slack time has the highest priority
```