CHAPTER 2

INTRODUCTION TO REAL-TIME SYSTEMS

Definition 1 (***Hard real-time***)

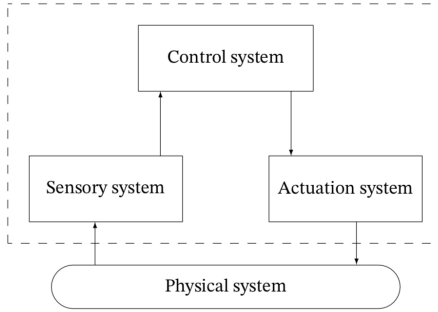
Absolutely no deadline can be missed (it could have health, financial or ecological consequences).

Definition 2 (***Firm real-time***)

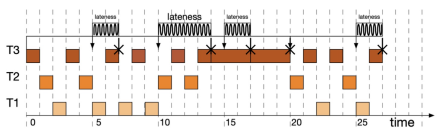
Ideally no deadline should be missed. However, some deadline misses may be tolerated, in which case we define a Quality of Service (QoS). For example, in video conferencing or live streaming, the system must ensure that video frames are transmitted and displayed within a specific time frame to maintain a smooth and uninterrupted video stream. Any delay in transmitting or displaying video frames can result in jitter, buffering, or even loss of data.

Definition 3 (***Soft real-time***)

Deadlines can be missed.

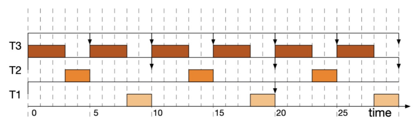


Round-Robin [24]. The Round Robin illustrated in Figure 2.2 is a technique where each task periodically receives an equal share of the processing resource — e.g. the CPU — in order to provide a good *average* process response time. The key idea is to assign time slices in equal portions to each process in a *circular order*. This technique is a typical scheduling scheme for most general purpose operating systems (unix, linux, windows). Figure 2.2 shows that in our set of tasks, the Round Robin is deadly for τ_3.



Rate Monotonic [18]. Rate Monotonic is a preemptive scheduling algorithm used in Real-Time Operating System (RTOS), where each task τ receives a static priority P_τ. This priority is assigned to the task by looking at its periodicity: the shorter the period, the higher the priority. In our example system, τ_3 receives the highest priority, τ_2 the middle priority and τ_1 the lowest priority. The schedule produced is shown in Figure 2.3. In our example, we observe that with the RM scheduler, all tasks are processed on time. Note that the schedule repeats from time marker 20 (the least common multiple of the task's periods).

Task	Execution time	Period (& deadline)	Load
τ_1	4	20	20%
τ_2	2	10	20%
τ_3	3	5	60%
		Total	100%



Definition 4 (***Real-time systems***)

Real-time systems *are defined as systems in which the correctness of the system depends not only on the logical result of computations, but also on the time at which the results are produced.*

Definition 5 (***Job***)

A real-time job *is characterized by the tuple $\langle a, e, d \rangle$, where:*

- *a corresponds to the release time.*
- *e corresponds to the maximal execution time (which is known as the worst-case execution time — WCET).*
- *d corresponds to the* absolute *deadline with the interpretation that in the interval $[a, d)$ the job must receive e CPU units.*

Definition 6 (***Active job***)

A job is said to be active *from its release time to its completion time. In other words an* active *job is a job already released but not completed.*

Definition 7 (***Periodic task***)

A periodic task *τ_i is characterized by the tuple $\langle O_i, C_i, D_i, T_i \rangle$, where:*

- O_i (the offset): *corresponds to the release time of the* first *job of the task.*
- C_i (the maximal computation time): *corresponds to the worst-case execution requirement of the task (WCET).*
- D_i (the *relative* deadline): *corresponds to the time-delay between a job release and its corresponding deadline. A job released at time t must be completed before or at time $t + D_i$.*
- T_i (the period): *corresponds to the* exact *duration between two consecutive task releases.*

Tasks vs. jobs

A *task*:

- is an offline concept;
- has parameters known at design time (e.g.: the *relative* deadline);
- is recurring;
- induces a (theoretically) infinite set of jobs
- there is a fixed number (n) of tasks in the system.

A *Job*:

- is a runtime concept;
- has parameters only known at runtime (e.g. the *absolute* deadline);
- is not recurring (executed once);
- can be executed;
- exists independently of the notion of task
- there is a potentially infinite number of jobs.

Definition 8 (***Sporadic task***)

Sporadic tasks *are similar to periodic tasks, the only difference being that the period of a sporadic task denotes the* minimum *inter-arrival time instead of the exact one. An example example of the way this is represented graphically is given in Figure 2.6.*

Definition 9 (***System utilisation***)

$$U(\tau) \stackrel{\text{def}}{=} \sum_{i=1}^{n} U(\tau_i) = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

Regarding the deadlines

- *Implicit-deadline:* the deadline corresponds to the period, i.e., $D_i = T_i; \forall i$. Each job must finish before the next release of the task.
- *Constrained-deadline:* the deadline is explicit and not greater than the period, i.e. $D_i \leqslant T_i,\ \forall i$.
- *Arbitrary-deadline,* the general case: no constraint exists between the period and the deadline.

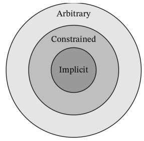
Definition 10 (*Preemptive*)

A scheduler can be preemptive, *meaning that it is capable of forcibly removing processes from a CPU when it decides to allocate that CPU to another process.*

- *Fixed Task Priority (FTP):* a fixed and unique priority is assigned to each task at design time. During system execution, the priority of each job is inherited from its task. An example is given in Figure 2.7. The highest priority task (τ_1) is executed *asap* without preemption, while task τ_2 can progress only when task τ_1 is not active. The task τ_2 can be preempted, this is the case for instance at time instant 5, the second job of τ_2 is resumed at time instant 7.
- *Fixed Job Priority (FJP):* a fixed and unique priority is assigned to each *job* during system execution. Several jobs of a same task may have distinct priorities.
- *Dynamic Priority (DP):* no restrictions are placed on the priorities that may be assigned to jobs. A job may have different priority levels over the duration of its execution.

Definition 12 (*Earliest Deadline First (EDF)*)

EDF is an FJP (Fixed Job Priority) scheduler, where the priority is based on the absolute *job deadline: The lower the absolute deadline, the higher the priority.*



Regarding the offsets

- *Synchronous:* the first job of every task is released simultaneously at time origin, i.e., $O_i = 0,\ \forall i$.
- *Asynchronous,* the general case: no constraint exists on the offsets.

Unless we explicitly state the opposite, we make the following assumptions in the framework of uniprocessor systems.

- The time space is discrete;
- We consider hard real-time systems;
- We consider preemptive tasks/jobs;
- Preemption delays are negligible;

Definition 11 (*Job response time*)

The response time of a job is the completion time minus the release time.

Lemma 14 (*A necessary condition*)

Let τ be a periodic task-set, then $U(\tau) \leqslant 1$ is a necessary *condition for system schedulability.*