Real-time system (RTS) notes

parameters:

- release time (when the task may start executing)
- deadline (when the task must finish executing)
- period (the minimum time between release times of the same task)
- worst case execution time (how long does it take, at most, to finish a task in a specific hardware, if executing by itself, ie, no interference)
- job or instance: an invocation of a task, typically corresponding to a period
- utilization = wcet/period (loosely interpreted as percentage of CPU a task uses)

Hard-real-time (MUST) and soft-real-time (should) finish by their respective task deadlines

Metrics/criteria for RTSs

Define tardiness = deadline – finish time
HRT: complete all tasks before deadline (ie, tardiness <= 0)
SRT: minimize "tardiness" over all tasks (eg, average tardiness, maximum tardiness)

In a **preemptive single-core system with n independent tasks**, Earliest Deadline First scheduling guarantees that all tasks will meet deadlines iff (if and only if) the sum of utilizations <= 1.

For k cores, Global EDF (all tasks are considered at every preemption point, and k tasks dispatched) **may** be very inefficient. Assume task set where N-1 tasks have wcet = 2e and period = 1-e, where e is a very small constant, and one task has wcet = 1-e and period 1. under EDF, what happens if there are N-1 cores/cpus to execute this task set? What is the utilization of this task set? If N grows very large, what happens to utilization? If there are N cores/cpus, all tasks will finish by their deadlines.
Partitioned EDF (tasks are assigned to cores/processors upon arrival and each runs EDF) does as well as bin-packing algorithms.
What if all tasks have the same period, what can be said?

For tasks with dependency in a preemptive single-core system, that is, can only start task 2 when finished task 1, if there is a dependency between 1 and 2 (like a pipeline or a DAG). ONE algorithm to transform this DAG into a set of independent tasks is to assign deadlines to each node/task so that they execute in the order required. So, if 1-2 is a dependency, task1 must have a deadline shorter than task2 (so that EDF will pick 1 before 2).
What if there are 2 branches (1-2, 1-3), does it matter which task gets the smaller deadline? Can they get the same deadline?

For tasks with dependency in a preemptive multi-core or multi-processor system, taking into account the transmission of data from task to subsequent task is a concern. If tasks are in different cores/processors/nodes, there are different communication costs. Also, if cores/processors/nodes have different architectures/capabilities, the wcet of a task will be different depending on where the task will run. One transformation that can be done is to consider a link between tasks as a "task", so that the mapping can be done more uniformly.
One approach for determining allocation of tasks to processors is brute force, which is not feasible. Therefore heuristics are developed to allocate tasks to different processors.

Communication issues when tasks are distributed abound, since there is a requirement to deliver the messages by a specific deadline. A common way to schedule messages in a shared medium (like wireless) is to use TDMA, which reserves slots for tasks to transmit (and tasks only transmit in their

reserved slot). For single hop transmissions, the system can determine when the message is received (same slot as it is sent) with TDMA. For multi-hop transmissions, to be able to guarantee delivery by the deadline, the system needs to know the route and needs to take into account how to schedule slots for every link in the route. Typically, in TDMA, the slots are a little larger than needed to accommodate clock skew and synchronization.
An alternative scheme is to use a token, whereby only the node that has the token can transmit.

Enforcement: recall that all the schemes above are collaborative. That is, tasks do not violate their assumptions (eg, amount of data to transmit, wcet). Therefore, it is a good strategy to create mechanisms for enforcing the assumptions (e.g., the OS interrupts the task after the wcet) or mechanisms of fault tolerance if enforcement is not possible (not possible for a node to restrict the wireless transmission of another node), such as using extra resources (e.g,, a different channel).

Other requirements exist, such as SWEPT: size, weight, energy, power, time. The last one is about RTSs, but all requirements need to be to taken into account in a embedded system, in a cyber-physical system, in a mobile IOT (internet of things) system. Some of these do not have strict requirements and therefore can use SRT principles, or even non-real-time (eg, if tasks do not finish before deadlines, it's inconvenient, but not a big deal)