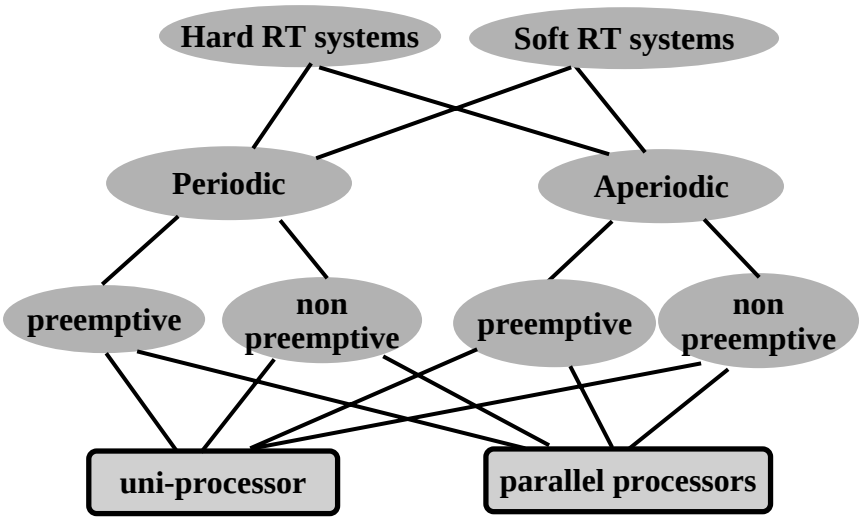
Introduction to Real Time systems



Hard Real Time system ⟹ guarantee deadlines

- To guarantee deadlines, we need to know worst case execution times
- Predictability: need to know if deadlines may be missed

Soft Real Time system ⟹ try to meet deadlines

- If a deadline is missed, there is a penalty
- Provides statistical guarantees (probabilistic analysis)
- Need to know the statistical distribution of execution times

Applications:

Safety critical systems, control and command systems, robotics, Communication, multimedia

A-periodic (sporadic) systems

- Each job, J_i , (task or process)
  - is released (arrives) at a time r_i ,
  - is characterized by a worst case execution time c_i ,
  - has an absolute deadline d_i , by which it has to finish execution
  - Sometimes a relative deadline, $D_i = d_i - r_i$, is used
- Static systems: the release times of the jobs are known before execution
- Dynamic systems: job arrivals are not know before execution
- The response time of job J_i is defined by f_i = completion time of J_i - r_i

Static scheduling:

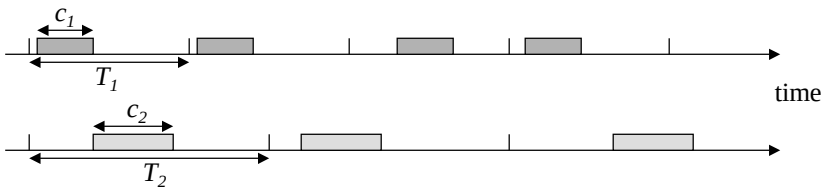
Given a set of jobs, $\{J_i , i=1,\ldots,n\}$, where $J_i = (r_i , c_i , d_i)$, construct a schedule (template or calendar) in which each job meets its deadline (*a feasible schedule*)

Dynamic scheduling:

At given instances during execution, determine the jobs to run next. Again, the goal is that each job meets its deadline (*feasibility*)

Periodic systems

- Each task (job), J_i , is released periodically and is characterized by
  - an invocation period T_i ,
  - a worst case execution time c_i ,
  - a relative deadline D_i . Usually $D_i = T_i$,
- An instance of the task is released at the beginning of its period and, if $D_i = T_i$, should complete execution by the end of the period.
- In some system, the first instance of a task J_i , is released at time ϕ_i (a phase). We will assume that $\phi_i = 0$.
- Specifically, the k^{th} instance of J_i , namely $J_{i,k}$, is released at time $\phi_i + (k-1) T_i$, and should complete by time $\phi_i + k T_i$. Here, $k=1, 2, \ldots$



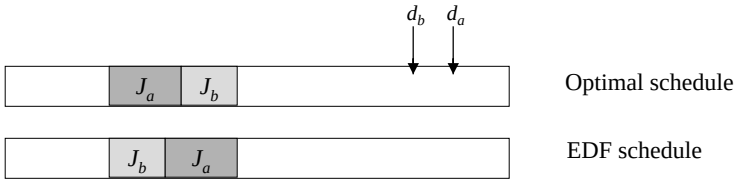
- Given a set of tasks $\{J_i , i=1,\ldots,n\}$, where $J_i = (T_i , c_i)$, a hyper period is the least common multiple of $T_i , i=1,\ldots,n$.

Scheduling a-periodic tasks with identical ready times

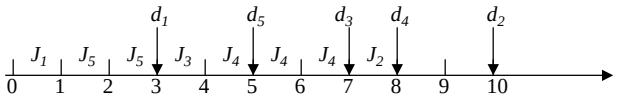
Earliest deadline first, EDF, (sometimes called or Earliest Due date first. EDD) is optimal in the sense that:

- If there is a feasible schedule, then EDF produces a feasible schedule
- EDF produces a schedule with the shortest completion time

Proof of optimality is based on an *interchange argument*.



Example: 5 tasks (0,1,3), (0,1,10), (0,1,7), (0,3,8), (0,2,5)



Scheduling a-periodic tasks with identical ready times

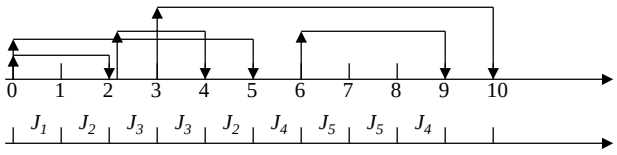
- Note that pre-emption is not needed for optimality,
- Given n jobs, $\{J_i, i=1,\ldots,n\}$, EDF will produce a feasible schedule if

$$\sum_{k=1}^{i} c_k \leq d_i \quad , \quad i = 1, \cdots, n$$

This can be checked in *O(n)* time, but need *O(n log n)* time to sort the tasks by the deadlines.

Can also apply EDF if ready times are not identical.

Example: 5 tasks (0,1,2), (0,2,5), (2,2,4), (3,2,10), (6,2,9)



Note that pre-emption is used.

Scheduling a-periodic tasks with non-identical ready times

- EDF is optimal if pre-emption is allowed – can be proved using the interchange argument.
- Optimality applies for both static and dynamic scheduling
- Can construct the schedule in *O(n log n)* time.
- Given *n* jobs, $\{J_i, i=1,\ldots,n\}$, EDF will produce a feasible schedule if, at any time *t*,

$$\sum_{k=1}^{i} c_k(t) \leq d_i \quad , \quad i = 1, \quad \cdots \quad , n$$

Where $c_k(t)$ is the remaining execution time of task *k* at time *t*. Note that the above check needs to be done only at release times (there are *n* of them).

If pre-emption is not allowed, then EDF is not optimal

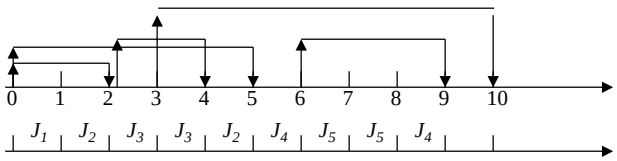
Example: 2 tasks, (0,4,7) and (1,2,3)

without pre-emption the problem is NP hard.

The least laxity (slack) first algorithm

- The laxity of a task at a given time is the maximum time its execution can be delayed before it is sure to miss its deadline.
  Slack(t) = d – t – c(t), where *c(t)* is the remaining execution time
- LLF is an optimal scheduling algorithm for a-periodic tasks

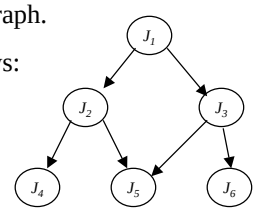
Example: 5 tasks (0,1,2), (0,2,5), (2,2,4), (3,2,10), (6,2,9)
Initial laxities = 1,3,*,*,*
Laxities at time 2 = *,2,0,*,*
Laxities at time 3 = *,1,0,5,*



LLF may result in a large number of preemptions – example (0,3,6), (0,3,6) -- and requires a knowledge of the execution time.

Scheduling a-periodic tasks with precedence constraints

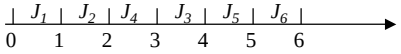
- Precedence constraints are given by a dependence graph.
- Change the release times and the deadlines as follows:
  If $J_i \rightarrow J_k$, then $r_k \geq r_i + c_i$ and $d_i \leq d_k - c_k$
- This modification can be done in $O(n^2)$
- Schedule the modified task set using EDF.



If there exists an EDF schedule for the modified task set, then the original task set is schedulable. If not, then the original task set is not schedulable.

Example: 6 tasks, (0,1,2), (0,1,5), (3,1,4), (0,1,3) (0,1,5) and (0,1,6)

- We can also use the "*latest deadline algorithm*" to build the schedule backward (starting from the leaves).

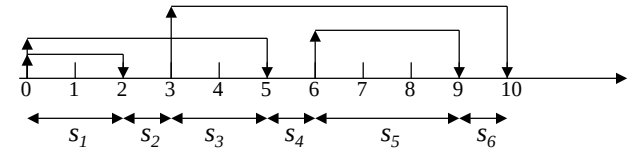


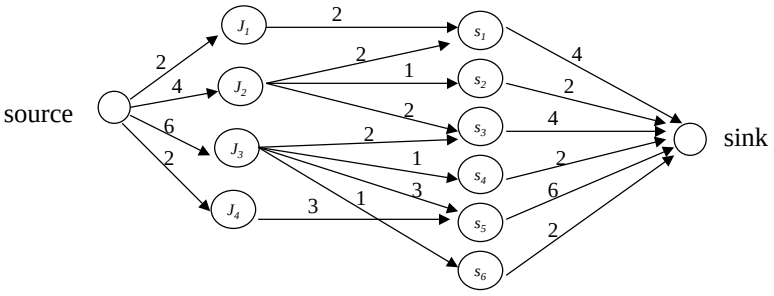
Scheduling a-periodic tasks on multiprocessors

- EDF is not optimal even with pre-emption.
  Example: 3 tasks, (0,1,2), (0,1,2) and (0,3,3) on two processors
- To find a schedule, transform the problem of scheduling $\{J_i, i=1,\ldots,n\}$, on P processors into a network flow problem as follows:
  - Divide the time line into time segments, where a time segment is a maximal interval that does not include any arrival or deadline,
  - Create a node, J_i, for each job
  - Create a node for each time segment, s_k. There is at most $2n$ segments
  - Create a source node and a sink node
  - Create an edge from the source to each J_i with a weight c_i
  - Create an edge from each s_k to the sink with a weight equal to the length of the time interval multiplied by P.
  - Create an edge from each J_i, to each s_k if J_i can execute in s_k. The weight of the edge is the length of s_k.
- The solution of the maximum flow problem in the network corresponds to a schedule.

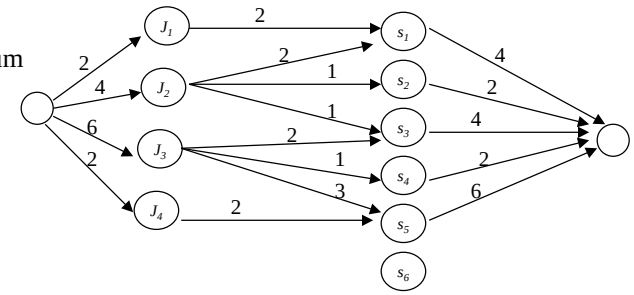
Scheduling a-periodic tasks on multiprocessors

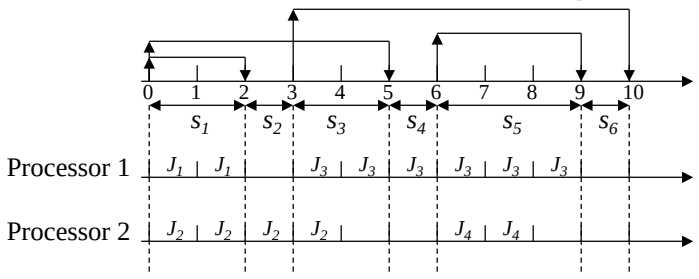
Example: 4 tasks (0,2,2), (0,4,5), (3,6,10), (6,2,9) on two processors





Solution of the maximum flow problem:





Exercise: repeat the solution with (2,5,8) added to the 4 tasks.