Priority-Driven Scheduling of Periodic Tasks
- Chapter 6 –
(Dynamic Priority (2))

Summary

- All of the above schedulability check works only under limited conditions
  - Preemptable at any time
  - Context switch overhead is negligible
  - Scheduling decision is made immediately upon jobs release and completion
- Practical Issues
  - What if the deadline is earlier than the period?
  - What if there is a non-preemptable code section (e.g., system call)?
  - What if the context switch overhead is not negligible?
  - Tick scheduling?
  - Precedence constraints

EDF with deadlines less than periods

- Quiz: can you find an easy but sufficient schedulability condition for a task set with deadlines less than periods?

$$D_i < p_i$$

$$\sum_{j=1}^{n} \frac{e_j + p_j - D_j}{p_j} \le 1; \text{ increase execution time}$$

$$\sum_{j=1}^{n} \frac{e_j}{D_j} \le 1; \text{ decrease period}$$

Non-preemptable code section

- a non-preemptable code section (NPS) of a low priority task ***blocks*** high priority task
  - How to take this into account in utilization bound check?

Theorem : Only job J_k with longer relative deadline D_k can block a job J_i with shorter relative deadline D_i

$$b_i = \max_{j=i+1}^{n} NPS_j \ /^* D_i < D_k \text{ if } i < k \ ^*/$$

$$\sum_{j=1}^{n} \frac{e_j}{p_j} + \frac{b_i}{p_i} \le 1; \text{ task by task check (only for task } i)$$

$$\sum_{j=1}^{n} \frac{e_j}{p_j} + \max_{j=1}^{n} \frac{b_j}{p_j} \le 1; \text{single check (for all tasks)}$$

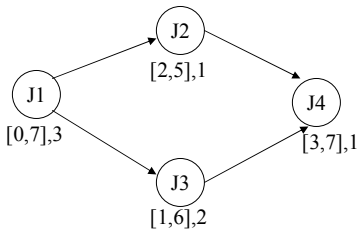
Earlier deadline and Non-preemptable code section

$$\sum_{j=1}^{n} \frac{e_j}{\min(D_j, p_j)} + \frac{b_i}{\min(D_i, p_i)} \le 1;\ \text{task by task check (only for task } i\text{)}$$

$$\sum_{j=1}^{n} \frac{e_j}{\min(D_j, p_j)} + \max_{j=1}^{n} \frac{b_j}{\min(D_i, p_j)} \le 1;\ \text{single check (for all tasks)}$$

Precedence constraints

- Precedence relation says you can't start until your predecessor has done.
- Basically, each task cannot start before its predecessors and cannot be preempted by its successors

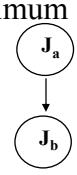


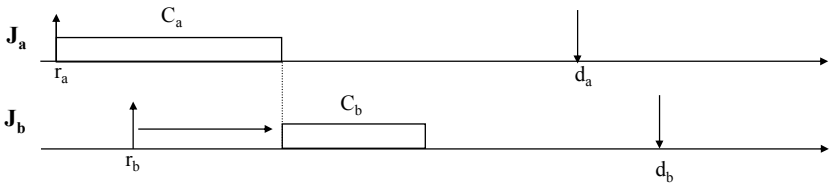
Precedence constraints

- There exists an elegant way to handle precedence constraints:
  - The basic idea is to transform a set Γ of dependent tasks into a set Γ^* of independent tasks by an adequate modification of timing parameters
  - Then, tasks are scheduled by EDF algorithm

Effective Release Times

- Modification of the release times:
  - A job's start time cannot be earlier than the minimum finishing time of its predecessors
  - Hence, $r_b^* = max\ (r_b, r_a + C_a)$

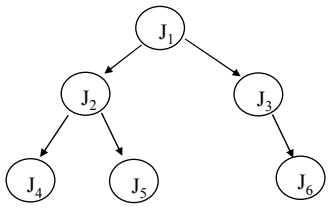




Effective Release Times

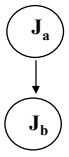
- The algorithm that modifies the release times:

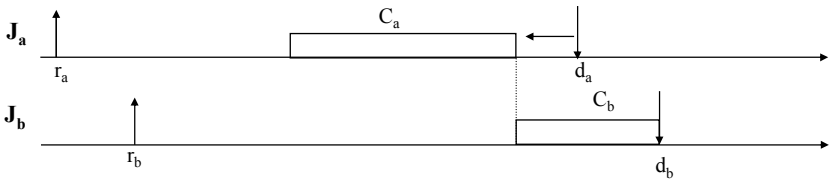
1. For any initial node of the precedence graph, set $r_i^* = r_i$
2. Select a task T_i such that its release time has not been modified but the release time of any immediate predecessors T_h have been modified. If no such task exists, exit
3. Set $r_i^* = max\,[r_i, max(r_h^* + C_h)]$
4. Return to step 2



Effective Deadlines

- Modification of the deadlines:
  - A job's finishing time cannot be later than the maximum start time of its successors
  - Hence, $d_a^* = min\,(d_a, d_b - C_b)$

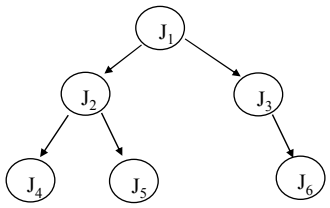




Effective Deadlines

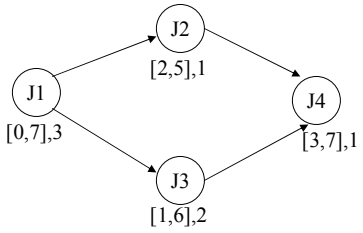
- The algorithm that modifies the deadlines:

1. For any terminal node of the precedence graph, set $d_i^* = d_i$
2. Select a task T_i such that its deadline has not been modified but the deadline of any immediate successor T_k have been modified. If no such task exists, exit
3. Set $d_i^* = min\ [d_i, min(d_k^* - C_k)]$
4. Return to step 2



Effective Release Times & Deadlines

- Given a set of arbitrary release times and deadlines, if we revise them according to these two rules, the resulting deadlines and release times are said to be effective release times and effective deadlines.



Effective Release Times & Deadlines

- The swapping trick still works with effective release times and deadlines in a single CPU with preemptive scheduling.
- To do EDF under general precedence relations
  - Get effective release times and deadlines
  - Assign priorities according to EDF
- EDF keeps its optimality under precedence constraints when tasks are preemptable

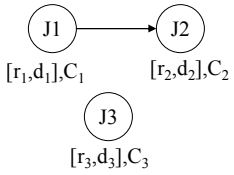
Optimality of Effective EDF under precedence constraints

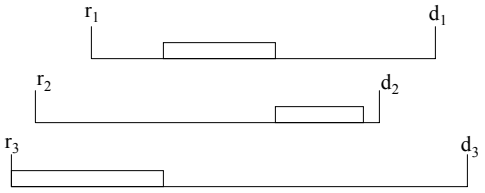
- Theorem: *If there exists a feasible schedule (meet all release times, deadlines, precedence constraints), effective EDF schedule (EDF schedule with effective release times and deadlines) is also feasible*

Note 1: effective EDF does not care about precedence constraints!

Note 2: To show effective EDF schedule is feasible, we have to show all release times, deadlines and precedence constraints are met.

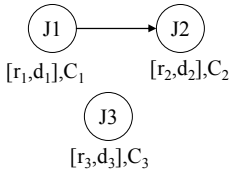
How to prove the optimality?

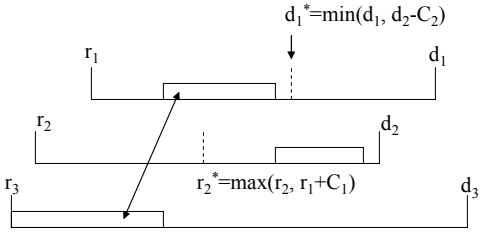




A feasible schedule

How to prove the optimality?

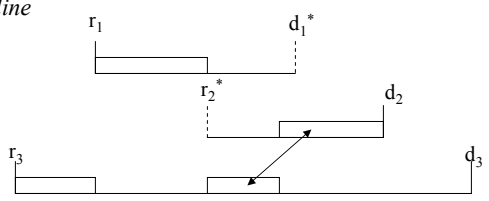




A feasible schedule

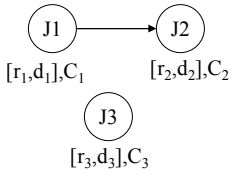
Swapping does not violate any deadline

Why? _____

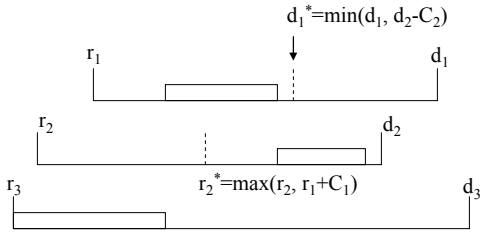


After 1st swapping

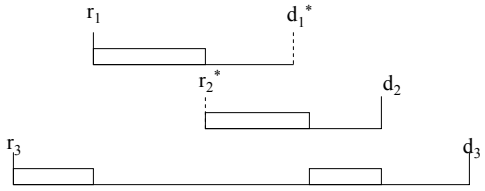
How to prove the optimality?



Precedence constraints are automatically met –why?

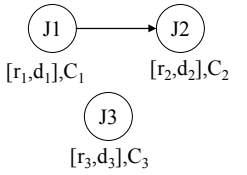


A feasible schedule

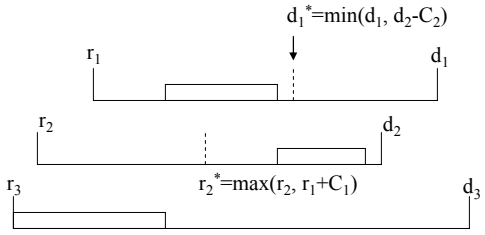


The effective EDF schedule

How to prove the optimality?



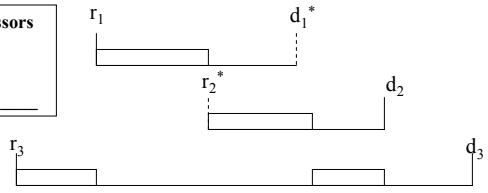
Precedence constraints are automatically met –why?



A feasible schedule

All predecessor can start earlier than successors since _____

Any successor CANNOT start before the completion of all predecessor because _____



The resulting schedule after swapping is feasible

effective EDF schedule is feasible!

Static Priority Scheduling vs. EDF

- Which one is more popular in the real-time market?
- You can't beat 100% schedulability, can you? Strangely enough, static priority scheduling algorithm with schedulability significantly less than 100% has taken over the world of real-time computing. Why?

- What prevents EDF becoming popular?
  - 1) EDF has small marketing budget and it loses the marketing war. Sometimes inferior technology wins, isn't it?
  - 2) There is a dark side of EDF.

The Stability Problem

There is a dark side (serious problem) with EDF.

- When a system becomes overloaded and not all the tasks can meet their deadlines, it is important to keep meeting deadlines of critical tasks. This is an easy problem for fixed priority scheduling. Unfortunately, there is no low complexity solution for this problem under EDF since each job's priority is changing and it is expensive to keep track the execution states during runtime.

- Fortunately, in certain applications, the penalty of missing a deadline can be lessened by application level measures, for example, dropping a B frame in video is ok. Even in feedback control, there are new results that "soften" the deadline. So EDF will become more popular.