Scheduling a-periodic tasks with periodic tasks (a-periodic servers)

- Execute the periodic tasks according to your scheduling algorithm
- When an a-periodic task arrives, it is put in an “a-periodic tasks queue”
- Have a server whose job is to execute tasks from the a-periodic queue

  - *Background server*: executes only when the periodic task queue is empty

  - *Polling server*: a task, J_s , with a maximum capacity (execution time) c_s , and period T_s . The capacity is replenished at the beginning of every T_s . If the capacity is not used when the server is scheduled to run, it is wasted.

  - *Deferrable server*: same as polling server, except that when there are no a-periodic tasks to run by the server when it is scheduled to run, a periodic task runs and the unused capacity of the server is deferred to be used at a later time within the current period.

a-periodic servers (feasibility tests)

- *Background server*: make sure that the periodic tasks, J_1 , … , J_n , meet their deadlines. The a-periodic tasks are served on a best-effort basis.

- *Polling server*: make sure that the periodic tasks, J_1 , … , J_n , and J_s , meet their deadlines. The a-periodic tasks are served at a rate of c_s time units every T_s time units.

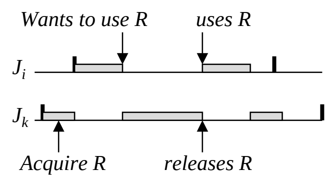
- *Deferrable server*: interferes with the regular schedule (say RMS) because the feasibility test assumes that a task runs when it is scheduled. A given test has been developed assuming RMS scheduling and assuming that the server has the highest priority (the shortest period).

$$\sum_{i=1}^{n} c_i \leq U_s + n\left(\left(\frac{U_s + 2}{2U_s + 1}\right)^{1/n} - 1\right)$$

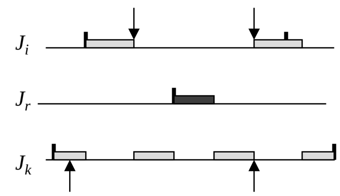
The priority inversion phenomenon in RMS

Assuming that tasks are ordered by priorities and that both J_i and J_k use some shared resource, *R* (ex. use semaphores to execute critical sections).

- If $k > i$ and J_k acquires *R* before J_i
- Then J_i preempt J_k and start execution
- Then J_i request *R*
- J_i blocks (because of mutual exclusion)
- J_k start execution although it has lower priority than J_i .
- J_i can execute only if J_k use releases *R*
- Hence, J_i may miss its deadline.



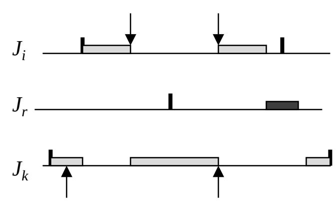
- The issue is aggravated if some other task J_r , $i < r < k$, preempt J_k thus delaying J_i further.



The priority inheritance protocol

If $i < k$, then when J_i is blocked because of a resource *R* held by J_k ,

- It transfer its priority to J_k ,
- J_k runs at the priority of J_i until it releases *R* (inherit the priority)
- when J_k releases *R*, it returns to its own priority.
- A task that inherit multiple priorities, runs at the highest priority and when it releases a resource, it only relinquishes that priority



- Priority inheritance is transitive. That is, If J_k inherits a priority from J_i , and then is blocked because of a resource help by J_u , then J_u inherits the priority of J_i .

> A job *J* can be blocked for at most *min(n,m)* critical sections, where *n* is the number of lower priority jobs that could block *J* and *m* is the number of distinct semaphores that could block *J*

Schedulability of the priority inheritance protocol

Add a blocking factor to the RMS analysis. Let B_i be the maximum blocking that J_i can experience.

- For each i, J_i will meet its deadline if $\sum_{k=1}^{i} \frac{c_k}{T_k} + \frac{B_i}{T_i} \le i\left(2^{1/i} - 1\right)$

- May use the time domain analysis (or response time analysis, after adding the blocking time. Specifically, the response time R_i for J_i should be less than T_i , where R_i is obtained from

$$c_i + B_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil c_k = R_i$$

A priority ceiling protocol limits B_i to one critical section by

- assigning a ceiling to each semaphore guarding a critical section (ceiling = highest priority of any task that can acquire the semaphore), and
- not allowing a job to acquire a semaphore at a time *t* unless its priority is higher than the ceilings of all the semaphores active at *t*.

Dealing with overload

- Worst case execution is very pessimistic
- May accept more tasks than what the system can guarantee, and then deal with the problem when it occurs

  - Assign a value to each task, and when you detect an overload, drop the task with the least values, or
  - Use a scheduler which maximizes the total value of the system.

  - Use a scheduler that guarantees that at most *m* out of every *k* instances of a task will miss the deadlines. Each task may have a different *(m,k).*

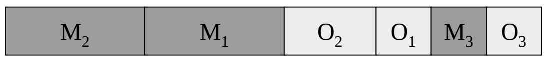
  - Trade precision for timeliness. In some applications, approximate but timely result may be acceptable (multimedia, image processing, real-time decision making …). This model is called the imprecise computation model.

  - Have multiple versions of each task with different precisions and computation requirements

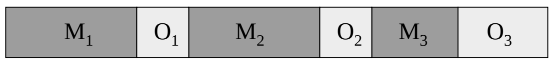
The imprecise computation model

Each task, T_i ,is divided to:

- A mandatory part, M_i , which should execute before the deadline
- An optional part, O_i, which may or may not execute, and may be interrupted.
- Each optional part carries a reward if it executes before the deadline
- Traditional worst-case execution time: $c_i = m_i + o_i$
- Two task models can be considered
  - Independent tasks



  - Chains (a restricted form of dependent tasks)



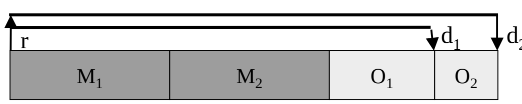
In either cases, the optional part should execute after the mandatory part.

Timing Constraints

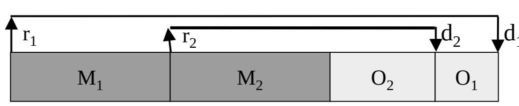
Frame-based systems

With single deadlines

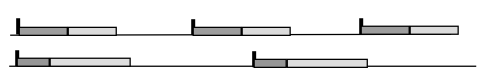
Identical ready times



Arbitrary ready times and deadlines



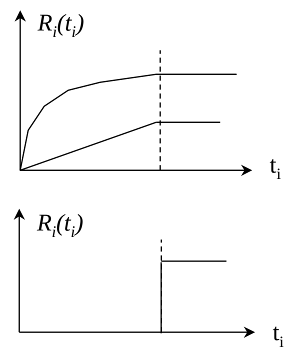
Periodic tasks



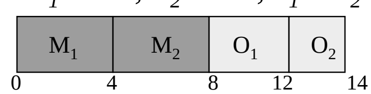
Aim: Produce a schedule with maximum total reward.

Reward Functions

- A non-decreasing *reward function* is associated with the optional execution, quantifying the refinement.
  - most commonly used functions are linear or concave
  - May have a step function (hard to analyze)



Example: $R_1 = 5\ t$, $R_2 = 3\ t$, $o_1 = o_2 = 4$, $d_1 = d_2 = 14.$



Reward = 26

Maximizing the total reward Reward

- Assume *n* aperiodic tasks with zero ready times and single deadline, *d*, (frame based system). Hence, the problem is to find the optional execution times, , t_1, …, , t_n, such that to maximize

$$\text{maximize} \quad \sum_{i=1}^{n} R_i\,(t_i)$$

$$\text{subject to} \quad \sum_{i=1}^{n} t_i \ {}_{\min} \leq \quad d - \sum_{i=1}^{n} m_i$$

$$\text{and} \qquad 0 \ \leq \ t_i \ \leq \ o_i$$

- The optimization problem without the upper and lower bounds on t_i can be easily solved using Lagrange multiplier techniques. The addition of the bound complicated the problem slightly.
- In his Ph.D. dissertation, Hakan Aydin solved the problem of maximizing the reward for periodic tasks.