5. Periodische Tasks

**Overview**

- Table of some known preemptive scheduling algorithms for periodic tasks:

<table>
<thead>
<tr>
<th></th>
<th>Deadline equals period</th>
<th>Deadline smaller than period</th>
</tr>
</thead>
<tbody>
<tr>
<td>static priority</td>
<td>RM</td>
<td>DM (deadline-monotonic)</td>
</tr>
<tr>
<td>dynamic priority</td>
<td>EDF</td>
<td>EDF*</td>
</tr>
</tbody>
</table>

**Model of periodic tasks**

- **Examples**: sensory data acquisition, low-level servoing, control loops, action planning and system monitoring. When a control application consists of several concurrent periodic tasks with individual timing constraints, the OS has to guarantee that each periodic instance is regularly activated at its proper rate and is completed within its deadline.

- **Definitions**:

  - \( \Gamma \) : denotes a set of periodic tasks
  - \( \tau_i \) : denotes a generic periodic task
  - \( \tau_{i,j} \) : denotes the \( j \)th instance of task \( i \)
  - \( r_{i,j} \), \( s_{i,j} \), \( f_{i,j} \), \( d_{i,j} \) : denotes the release time, start time, finishing time, absolute deadline of the \( j \)th instance of task \( i \)
  - \( \Phi_i \) : phase of task \( i \) (release time of its first instance)
  - \( D_i \) : relative deadline of task \( i \)

- **The following hypotheses are assumed on the tasks**:
  - The instances of a periodic task are regularly activated at a constant rate. The interval \( T_i \) between two consecutive activations is called period. The release times satisfy
    \[
    r_{i,j} = \Phi_i + (j-1)T_i
    \]
  - All instances have the same worst case execution time \( C_i \)
  - All instances of a periodic task have the same relative deadline \( D_i \). Therefore, the absolute deadlines satisfy
    \[
    d_{i,j} = \Phi_i + (j-1)T_i + D_i
    \]
Model of periodic tasks

- The following hypotheses are assumed on the tasks cont’:
  - Often, the relative deadline equals the period $D_i = T_i$ and therefore $d_{i,j} = \Phi_i + jT_i$.
  - All periodic tasks are independent; that is, there are no precedence relations and no resource constraints.
  - No task can suspend itself, for example on I/O operations.
  - All tasks are released as soon as they arrive.
  - All overheads in the OS kernel are assumed to be zero.

Example

- $T_i = \Phi_i + D_i + \tau_i$.

Rate monotonic scheduling (RM)

- Assumptions:
  - Task priorities are assigned to tasks before execution and do not change over time (static priority assignment).
  - RM is intrinsically preemptive: the currently executing task is preempted by a task with higher priority.
  - Deadlines equal the periods $D_i = T_i$.

- Algorithm: Each task is assigned a priority. Tasks with higher request rates (that is with shorter periods) will have higher priorities. Tasks with higher priority interrupt tasks with lower priority.

Optimality: RM is optimal among all fixed-priority assignments in the sense that not other fixed-priority algorithm can schedule a task set that cannot be scheduled by RM.

The proof is done by considering several cases that may occur, but the main ideas are as follows:

- A critical instant for any task occurs whenever the task is released simultaneously with all higher priority tasks. The tasks schedulability can easily be checked at their critical instances. If all tasks are feasible at their critical instants, then the task set is schedulable in any other condition.
- Show that, given two periodic tasks, if the schedule is feasible by an arbitrary priority assignment, then it is also feasible by RM.
- Extend the result to a set of $n$ periodic tasks.
Rate monotonic scheduling (RM)

- **Schedulability analysis:** A set of periodic tasks is schedulable with RM if
  \[ \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n \left( 2^{1/n} - 1 \right) \]
  This condition is sufficient but not necessary (in general). The proof of this condition is rather involved.
- The term \( U = \sum_{i=1}^{n} \frac{C_i}{T_i} \) denotes the processor utilization factor \( U \) which is the fraction of processor time spent in the execution of the task set.

Deadline monotonic scheduling (DM)

- **Assumptions** are as in rate monotonic scheduling, but deadlines may be smaller than the periodic, i.e.
  \( C_i \leq D_i \leq T_i \)
- **Algorithm:** Each task is assigned a priority. Tasks with smaller deadlines will have higher priorities. Tasks with higher priority interrupt tasks with lower priority.
- **Schedulability analysis:** A set of periodic tasks is schedulable with DM if
  \[ \sum_{i=1}^{n} \frac{C_i}{D_i} \leq n \left( 2^{1/n} - 1 \right) \]
  This condition is sufficient but not necessary (in general).

Deadline monotonic scheduling (DM)

- The longest response time \( R_i \) of a periodic task \( i \) is computed, at the critical instant, as the sum of its computation time and the interference due to preemption by higher priority tasks
  \[ R_i = C_i + I_i \]
  Hence, the schedulability test needs to compute the smallest \( R_i \) that satisfies
  \[ R_i = C_i + \sum_{j=1}^{i-1} \left[ \frac{R_j}{T_j} \right] C_j \]
  for all tasks \( i \). Then, \( R_i \leq D_i \) must hold for all tasks \( i \).
- It can be shown that this condition is necessary and sufficient.
Deadline monotonic scheduling (DM)

- The longest response times $R_i$ of the periodic tasks $i$ can be computed iteratively by the following algorithm:

  ```
  Algorithm: DM_guarantee (Γ)
  { for (each $τ_i ∈ Γ$) {
      I = 0;
      do {
          R = I + C_i;
          if (R > D_i) return(UNSCHEDULABLE);
          I = \sum_{j=1}^{i-1} \lfloor R/T_j \rfloor C_j;
      } while (I + C_i > R);
      return(SCHEDULABLE);
  }
  }
  ```

DM example

- **Example:**
  - Task 1: $C_1 = 1; T_1 = 4; D_1 = 3$
  - Task 2: $C_2 = 1; T_2 = 5; D_2 = 4$
  - Task 3: $C_3 = 2; T_3 = 6; D_3 = 5$
  - Task 4: $C_4 = 1; T_4 = 11; D_4 = 10$

- **Algorithm for task 4:**
  - Step 0: $R_4 = 1$
  - Step 1: $R_4 = 5$
  - Step 2: $R_4 = 6$
  - Step 3: $R_4 = 7$
  - Step 4: $R_4 = 9$
  - Step 5: $R_4 = 10$

DM example

- **U = 0.874**
  - $\sum_{i=1}^{n} \frac{C_i}{D_i} = 1.08 > n \left(2^{1/n} - 1\right) = 0.757$

DM example
**EDF scheduling (earliest deadline first)**

- **Assumptions:**
  - dynamic priority assignment
  - intrinsically preemptive
  - \( D_i \leq T_i \)

- **Algorithm:** The currently executing task is preempted whenever another periodic instance with earlier deadline becomes active.
  \[
d_{i,j} = \Phi_i + (j-1)T_i + D_i
\]

- **Optimality:** No other algorithm can schedule a set of periodic tasks if the set that can not be scheduled by EDF.
  - The proof is simple and follows that of the aperiodic case.

**EDF scheduling**

- A necessary and sufficient schedulability test if \( D_i = T_i \):
  - A set of periodic tasks is schedulable with EDF if and only if
    \[
    \sum_{i=1}^{n} \frac{C_i}{T_i} = U \leq 1
    \]
  - The term
    \[
    U = \sum_{i=1}^{n} \frac{C_i}{T_i}
    \]
    denotes the average processor utilization.

**EDF scheduling**

- If the utilization satisfies \( U > 1 \), then there is no valid schedule: The total demand of computation time in interval \( T = T_1 \cdot T_2 \cdot \ldots \cdot T_n \) is
  \[
  \sum_{i=1}^{n} \frac{C_i}{T_i} T = U T > T
  \]
  and therefore, it exceeds the available processor time.

- If the utilization satisfies \( U \leq 1 \), then there a valid schedule (proof by contradiction): Assume that deadline is missed at some time \( t_2 \).

**EDF scheduling**

- Within an interval \([t_1, t_2]\) the total computation time demanded by periodic tasks is bounded by
  \[
  C_p(t_1, t_2) = \sum_{i=1}^{n} \left[ \frac{t_2 - t_1}{T_i} \right] C_i \leq \sum_{i=1}^{n} \frac{t_2 - t_1}{T_i} C_i = (t_2 - t_1)U
  \]

  number of complete periods of task \( I \) in the interval

- Since the deadline at time \( t_2 \) is missed, we must have:
  \[
  t_2 - t_1 < C_p(t_2, t_1) \leq (t_2 - t_1)U \Rightarrow U > 1
  \]
Periodic tasks

Example: 2 tasks, deadline = periods, $U = 97\%$