EL2450 Hybrid and Embedded Control

Lecture 7: Real-time scheduling

- Scheduling periodic and aperiodic tasks
- Schedulability analysis

Today's Goal

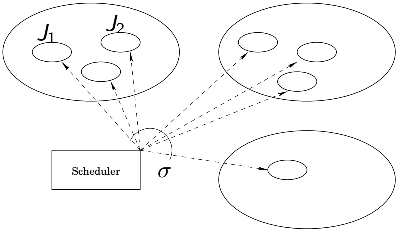
You should be able to model and analyze

- scheduling problems
- earliest deadline first scheduling
- rate monotonic scheduling
- deadline monotonic scheduling
- polling server

Scheduling

For a set of tasks $J = \{J_1, \ldots, J_n\}$, a **schedule** is a map $\sigma : \mathbb{R}^+ \mapsto \{0, 1, \ldots, n\}$ assigning a task at each time instant t:

$$\sigma(t) = \begin{cases} k \neq 0, & \text{CPU should execute } J_k \\ 0, & \text{CPU is idle} \end{cases}$$

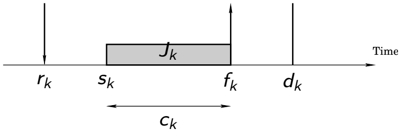


- σ is **feasible** if J can be completed according to specified constraints
- J is **schedulable** if there exists a feasible σ

Timing Constraints

A task J_k can be characterized by the following parameters:

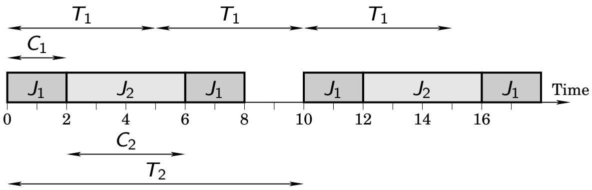
- **Release time** r_k is the time at which J_k becomes ready for execution
- **Computation time** c_k is the time necessary for the CPU to execute J_k
- **Deadline** d_k is the time before which J_k should be completed
- **Start time** s_k is the (actual) time at which J_k starts executing
- **Finishing time** f_k is the (actual) time at which J_k finishes executing



Independent Periodic Tasks

Suppose all tasks J_k are independent and periodic with

- **Period** T_k
- **Worst-case computation time** C_k
- **Relative deadline** D_k (deadline relative to current release time; often $D_k \equiv T_k$)
- **Worst-case response time** R_k (largest time between release and termination)
- **Phase** ϕ_k (release time of the first task instance)



Schedule Length and Feasibility

For independent and periodic tasks J, the length of a schedule σ is equal to

$$\mathrm{lcm}(T_1, \ldots, T_n)$$

σ is feasible if all deadlines are met, i.e.,

$$R_k \leq D_k, \qquad \forall J_k \in J$$

Utilization Factor

The **utilization factor** U of a periodic task set J is the fraction of processor time spent in the execution of the task set. Since C_i/T_i is the fraction for J_i, we have

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

- If $U > 1$, then the task set J is not schedulable
- Even if $U \leq 1$, it might be hard to find a feasible schedule
- U is independent of the scheduling algorithm

Scheduling Problem

The scheduling problem of finding a feasible σ for a set of independent periodic tasks $J = \{J_1, \ldots, J_n\}$ can be formulated as

$$\begin{aligned} \text{Find} \quad & \sigma \\ \text{such that} \quad & R_k \leq D_k \\ & U \leq 1 \end{aligned}$$

We will consider the following potential solutions

- Earliest deadline first scheduling
- Rate monotonic scheduling
- Deadline monotonic scheduling

Earliest Deadline First Scheduling

Earliest deadline first (EDF) scheduling algorithm assigns **dynamic priorities** to the tasks based on their absolute deadlines:

Execute task with shortest time to deadline d_k

Note

- Priorities are set dynamically
- Works also for aperiodic tasks

EDF Schedulability

A set of periodic tasks $J = \{J_1, \ldots, J_n\}$ with $D_k = T_k$, $k = 1, \ldots, n$, is schedulable with EDF if and only if

$$U \leq 1$$

Note

- Processor can be fully utilized with EDF.
- A similar result holds even if $D_k \neq T_k$.
- If J can be scheduled by any algorithm, then EDF can schedule J. Equivalently, for $U > 1$, no algorithm can produce a feasible schedule.

Proof Sketch

(Only if) The total demand of computation time by all tasks between $t = 0$ and $t = T_1 T_2 \dots T_n$ is

$$\frac{T_1 T_2 \dots T_n}{T_1} C_1 + \frac{T_1 T_2 \dots T_n}{T_2} C_2 + \dots + \frac{T_1 T_2 \dots T_n}{T_n} C_n$$

If this exceeds the available processor time $t = T_1 T_2 \dots T_n$, ie,

$$\frac{T_1 T_2 \dots T_n}{T_1} C_1 + \frac{T_1 T_2 \dots T_n}{T_2} C_2 + \dots + \frac{T_1 T_2 \dots T_n}{T_n} C_n > T_1 T_2 \dots T_n$$

or if $\sum_{i=1}^{n} \frac{C_i}{T_i} = U > 1$ then J is not schedulable with EDF (or any other scheduling algorithm).

Proof Sketch

(If) Shown by contradiction. Suppose that $U \leq 1$ and J is not schedulable. Let t_2 be the instant of time-overflow (deadline of an unfulfilled request). Let $[t_1, t_2]$ be the longest interval of continuous utilization, such that only tasks with deadline less than or equal to t_2 are executed in $[t_1, t_2]$. Then, t_1 is the release time of some periodic task. The total computation time demanded by periodic tasks in $[t_1, t_2]$ is

$$C_p(t_1, t_2) = \sum_{r_k \geq t_1, d_k \leq t_2} C_k = \sum_{i=1}^{n} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i$$

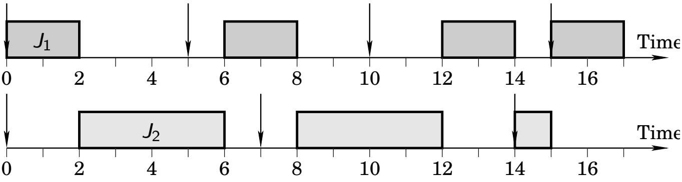
where $\left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor$ is the total number of periods of J_i entirely contained in $[t_1, t_2]$. Then,

$$C_p(t_1, t_2) \leq \sum_{i=1}^{n} \frac{t_2 - t_1}{T_i} C_i = (t_2 - t_1)U$$

Since there is no processor idle period, $C_p(t_1, t_2) > t_2 - t_1$, we get the contradiction $U > 1$.

Example: EDF Scheduling

J_1 has $T_1 = D_1 = 5$, $C_1 = 2$
J_2 has $T_2 = D_2 = 7$, $C_2 = 4$
Since $U = \frac{2}{5} + \frac{4}{7} = 0.97 \leq 1$, the tasks are schedulable with EDF.



Rate Monotonic Scheduling

Rate monotonic (RM) scheduling algorithm assigns **fixed priorities** to tasks, such that $T_i < T_j$ implies that J_i gets higher priority than J_j.

Note

- Provides a way to set fixed priorities for a set of tasks
- Fixed priorities might otherwise often be set heuristically

RM Schedulability

A set of periodic tasks $J = \{J_1, \ldots, J_n\}$ is schedulable with RM if

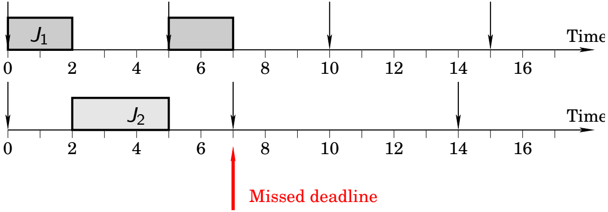
$$U \leq n(2^{1/n} - 1)$$

Note

- Not a necessary condition, so there might exist an RM schedule even if U does not fulfill the inequality
- $n(2^{1/n} - 1) \to \ln 2 \approx 0.69$, as $n \to \infty$, so RM can always schedule J if the total process utilization is less than 0.69
- A maximum utilization of 0.69 is often used as a rule of thumb for RM

Example: RM Scheduling

Try to schedule the previous example with RM. Since $T_1 < T_2$, RM gives higher priority to J_1 than J_2.
RM does not give a feasible schedule!



Note that $U = 0.97 > 2(2^{1/2} - 1) \approx 0.83$

RM is Optimal

If a set of periodic tasks are not schedulable by RM, then the set is not schedulable by any other **fixed priority** scheduling algorithm.

- RM is in this sense the best fixed priority algorithm
- RM is not good when $D_i \ll T_i$ (rare but urgent tasks)

Deadline Monotonic Scheduling

Deadline monotonic (DM) scheduling algorithm assigns **fixed priorities** to tasks, such that $D_i < D_j$ implies that J_i gets higher priority than J_j.

Note

- At any instant, the task with shortest relative deadline is executed
- Fixed priority schedule since relative deadlines are constant
- For tasks with deadlines less than periods
- Works for rare but urgent tasks
- DM=RM if $D_i \equiv T_i$

Worst-Case Response Time Calculation

Suppose the tasks $J_1, \ldots, J_i$ are ordered by decreasing fixed priority. Worst-case response time R_i for J_i is the largest time between release and termination. It can be derived as the smallest positive solution to

$$R_i = C_i + \sum_{j=1}^{i-1} \lceil \frac{R_i}{T_j} \rceil C_j$$

Note

- R_i appears on both sides of the equation
- $\sum_{j=1}^{i-1} \lceil \frac{R_i}{T_j} \rceil C_j$ represents the preemption by higher-priority tasks

Example

Consider tasks (from previous examples):

J_1 has $T_1 = 5$, $C_1 = 2$, high priority

J_2 has $T_2 = 7$, $C_2 = 4$, low priority

Worst-case response times are then $R_1 = 2$ and $R_2 = 8$, because:

$$R_1 = C_1 = 2, \qquad R_2 = C_2 + \left\lceil \frac{R_2}{T_1} \right\rceil C_1 = 4 + \left\lceil \frac{R_2}{5} \right\rceil 2$$

Iterate over R_2^k with $R_2^0 = 0$:

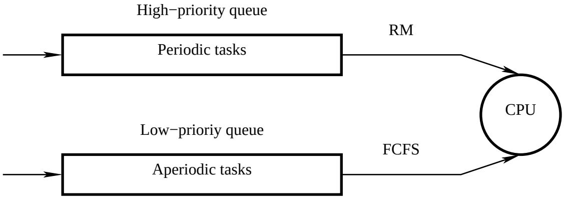
$$R_2^1 = 4 + \left\lceil \frac{R_2^0}{5} \right\rceil 2 = 4, \qquad R_2^2 = 4 + \left\lceil \frac{4}{5} \right\rceil 2 = 6$$

$$R_2^3 = 4 + \left\lceil \frac{6}{5} \right\rceil 2 = 8, \qquad R_2^4 = 4 + \left\lceil \frac{8}{5} \right\rceil 2 = 8 = R_2^3$$

Scheduling Periodic and Aperiodic Tasks Together

Background Scheduling

- Schedule aperiodic tasks in the background (when CPU would be idle)
- May lead to long response time for aperiodic requests



Polling Server Scheduling

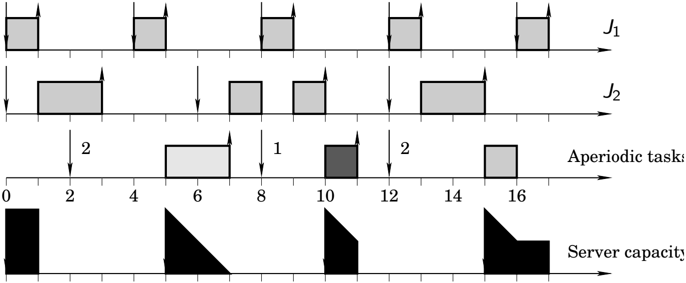
- A *polling server* is a periodic task that serves aperiodic tasks
- Gives guaranteed CPU utilization also for the aperiodic tasks

Polling Server

- A polling server task J_S is characterized by a period T_S and a server capacity C_S, as any other periodic task
- The polling server is scheduled by the algorithm for periodic tasks
- Once activated, the server starts serving the pending aperiodic requests within the limit of its capacity
- Several scheduling strategies possible for the aperiodic requests

Example: RM Scheduling and Polling Server

Periodic task J_1: $T_1 = 4$, $C_1 = 1$
Periodic task J_2: $T_2 = 6$, $C_2 = 2$
Server task J_S: $T_S = 5$, $C_S = 2$



Subtask Scheduling

- It is often suitable to divide tasks into subtasks, e.g., control tasks
- May create dependency, so it is in general harder to design schedule

Control Tasks

Each control task J_k is dived into four subtasks:

J_k^{AD} AD conversion

J_k^{CO} Calculate controller output

J_k^{DA} DA conversion

J_k^{US} Update state

```
nexttime = getCurrentTime();
while (true) {
  AD_conversion();
  calculateOutput();
  DA_conversion();
  updateState();
  nexttime = nexttime + h;
  sleepUntil(nexttime);
}
```

Design Control Task Schedule

- Set $D^{US} = T$ for all tasks
- Minimize D^{CO} for all tasks

Next Lecture

Models of computation

- Discrete-event systems
- Transition systems