Deadline-driven scheduling

Michal Sojka

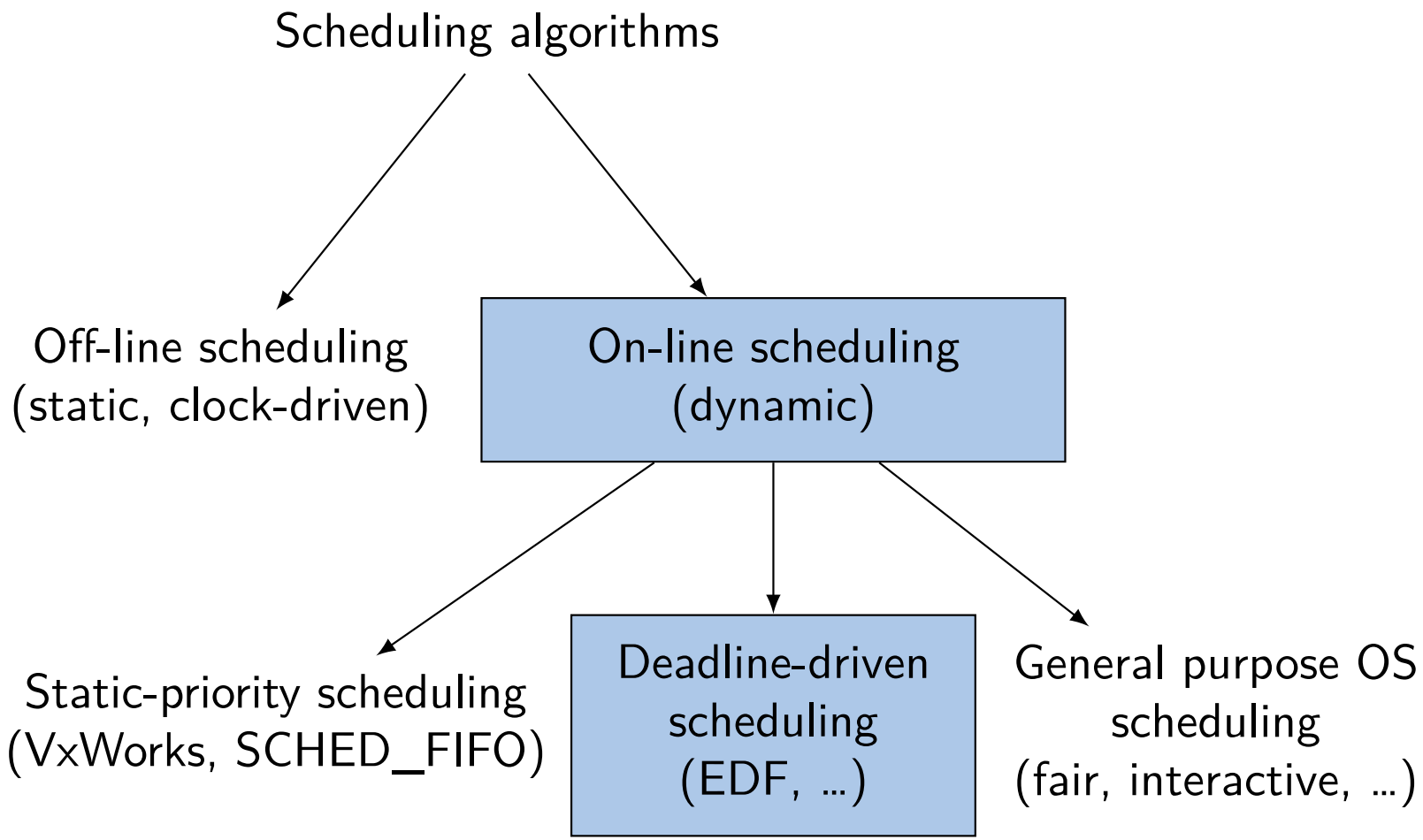
Czech Technical University in Prague,
FEE and CIIRC

November 22, 2023

Some slides are derived from lectures by Steve Goddard and James H. Anderson

Classification of scheduling algorithms

(used in real-time systems)



Note

Terminology

Deadline-driven scheduling is also referred to as dynamic-priority scheduling.

- Why dynamic priority?
  - Historical name coming from implementations on top of fixed-priority schedulers.
  - The term "Deadline driven scheduling" better reflects the nature of the algorithms.
- Differences against fixed-priority scheduling:
  - Different jobs of the same tasks can have assigned different "priority".
  - Infinite number of "priorities"

Outline

Outline

Earliest Deadline First (EDF) scheduling

- Scheduler selects a job with earliest deadline
  (the job with earliest deadline has the "highest priority")
- From the programmers point of view (when dealing with CPU scheduling):
  - On task creation, one specifies relative deadline (or period if $D = T$) rather than the priority.
  - Job completion has to be explicitly announced to the scheduler.
  - Scheduler is able to detect deadline misses and react appropriately (e.g. notify the application).

EDF optimality

Theorem (Liu, Layland)

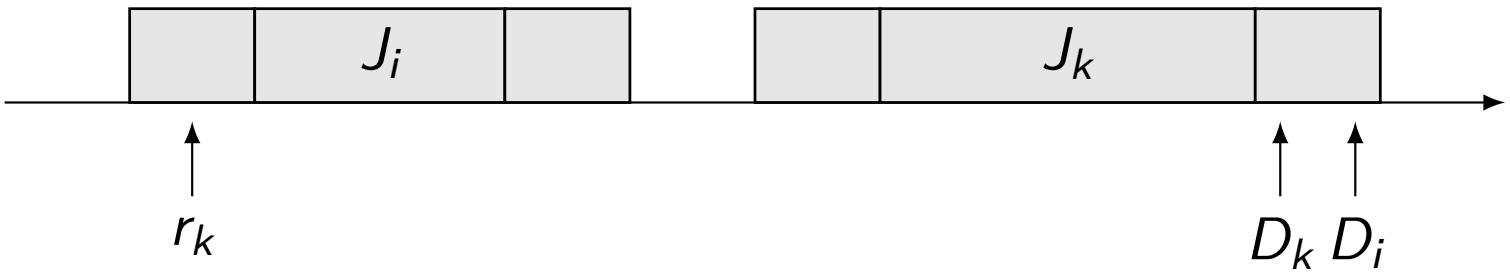
When preemption is allowed and jobs do not contend for resources, the EDF algorithm can produce a feasible schedule of a set $\mathcal{T}$ of independent jobs with arbitrary release times and deadlines on a processor if and only if $\mathcal{T}$ has feasible schedules.

Notes:

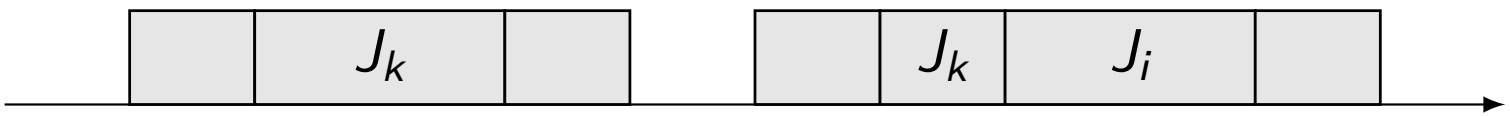
- Applies even if tasks are not periodic.
- If periodic, a task's relative deadline can be less than its period, equal to its period, or greater than its period.
- Stronger claim that RM/DM optimality.

Proof

- We show that any feasible schedule of jobs $\mathcal{T}$ can be systematically transformed to EDF schedule.
- Assume that parts of two jobs J_i a J_k are scheduled in non-EDF order:



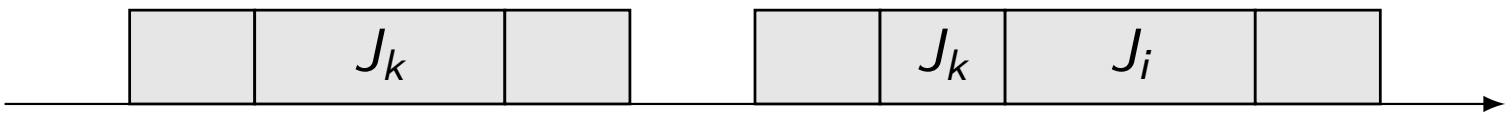
- This can be easily resolved by swapping the jobs:



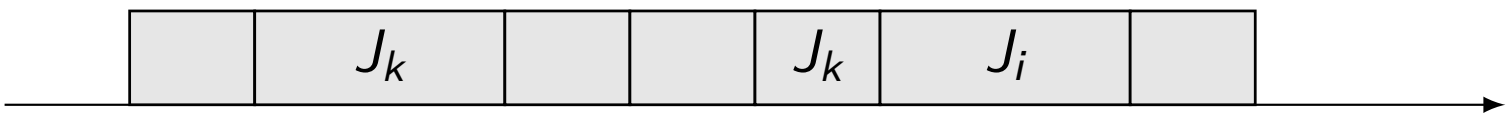
Note that this operation cannot cause deadline miss.

Proof (continued)

- By repeating this step, we get rid of all EDF order violations.
- Resulting schedule may still not be valid EDF schedule, because it can contain "gaps" even when some jobs were ready:



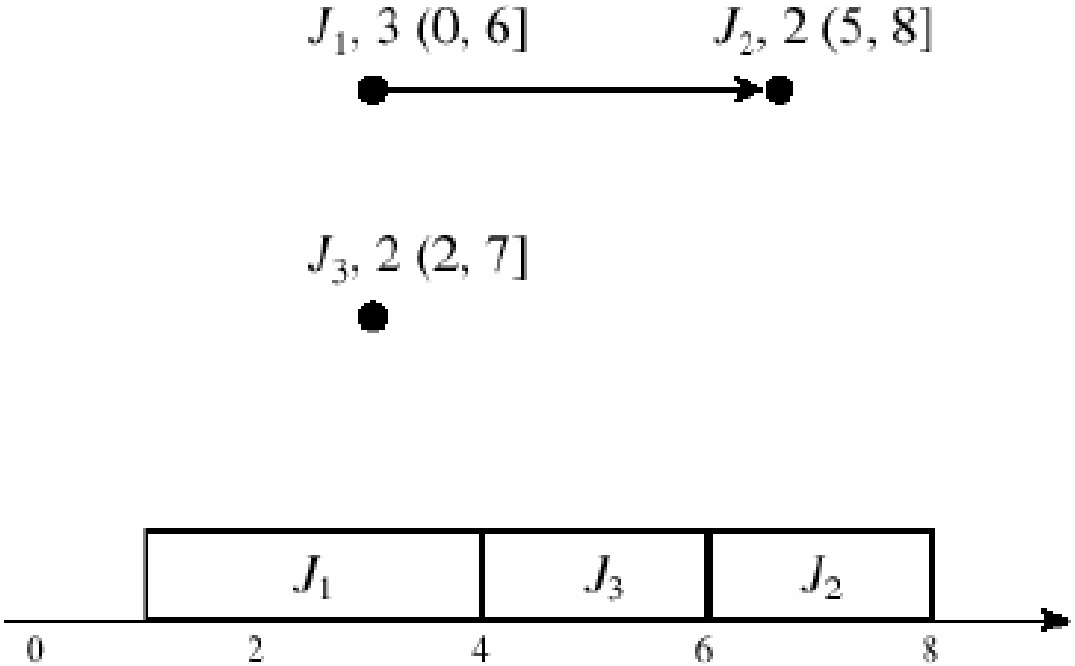
- These gaps can be easily eliminated by shifting the schedule ahead of time:



Outline

Latest Release Time (LRT) algorithm

- It is sufficient when jobs finish right at the deadline.
- We will run them at the latest time possible to not miss deadline.
- EDF with time going backward $\Rightarrow$ the same optimality claim holds.

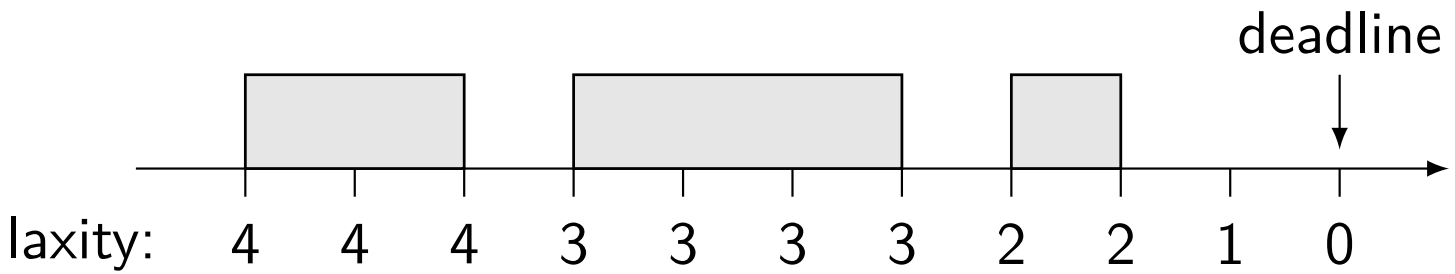


Least Laxity First (LLF) scheduling

Sometimes called "least slack time first" or "minimum laxity first" (Liu).

Definition (slack, laxity)

At any time t the slack (or laxity) of a job with deadline at D is equal $D - t - x$, where x is the remaining portion of the job.



LLF scheduling: The job with the smallest laxity has highest priority at all times.

LLF optimality

Theorem

When preemption is allowed and jobs do not contend for resources, the LLF algorithm can produce a feasible schedule of a set J of independent jobs with arbitrary release time times and deadlines on a processor if and only if J has feasible schedules.

- The proof is similar to that of EDF.

Quiz

Which algorithm would be preferable in practice for CPU scheduling?

A. EDF (earliest deadline first)
B. LLF (least laxity first)

Outline

Utilization-based schedulability test for (preemptive) EDF

CZ: Test rozvrhnutelnosti pro EDF založený na zatížení

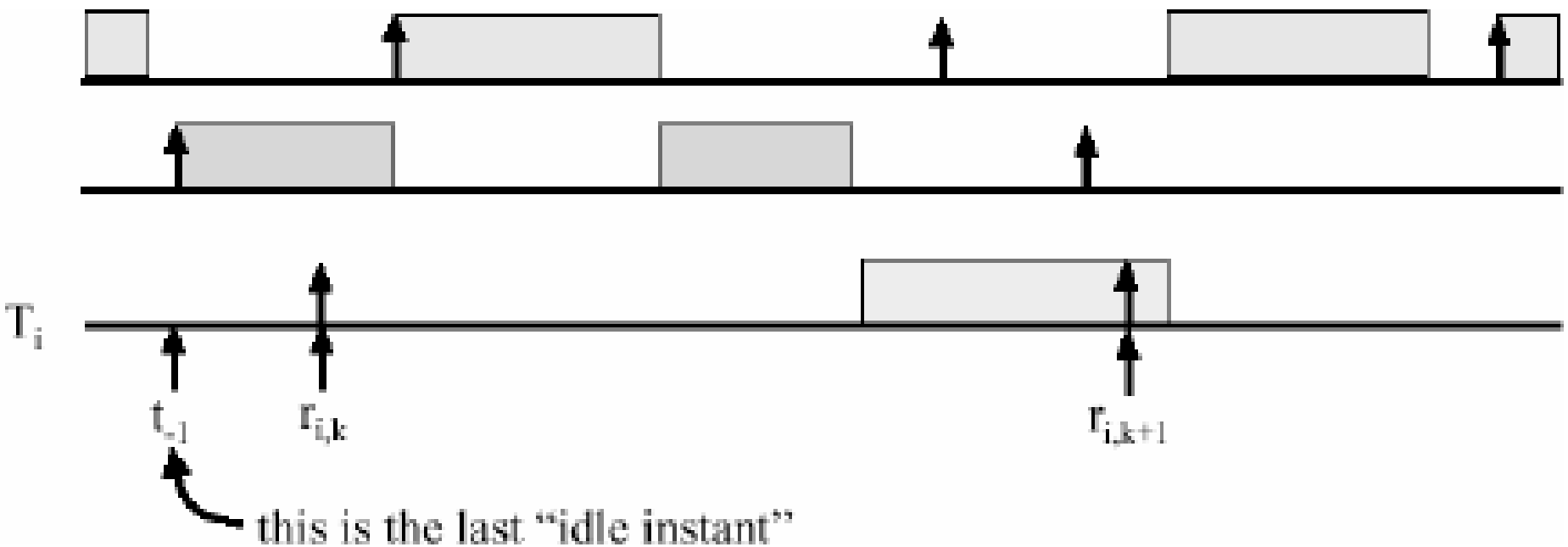
Theorem (Liu, Layland)

A system $\mathcal{T}$ of independent, preemptable, periodic tasks with relative deadlines equal to their periods can be feasibly scheduled (under EDF) on one processor if and only if its total utilization U is at most one.

Proof: Implication in one direction is trivial: If $U > 1$, then it is obvious that some task has to miss its deadline. Let's focus on the opposite direction.

Proof

- We want to show that $U \leq 1 \Rightarrow \mathcal{T}$ is schedulable.
- We will prove equivalent statement (contrapositive)
  $\mathcal{T}$ is not schedulable $\Rightarrow U > 1$.
- Assume that $\mathcal{T}$ is not schedulable. Let $\tau_{i,k}$ is the first job to miss the deadline.



Proof (continued)

As $\tau_{i,k}$ misses the deadline, the demand placed on the processor by jobs with deadline less than $r_{i,k+1}$ in the interval $[t_{-1}, r_{i,k+1})$ is greater than the available processor time in $[t_{-1}, r_{i,k+1}]$. Thus,

$$
\begin{aligned}
r_{i,k+1} - t_{-1} &= \text{available processor time in } [t_{-1}, r_{i,k+1}] < \\
&< \text{demand placed on the processor by jobs with } D \leq r_{i,k+1} = \\
&= \sum_{j=1}^{N} (\text{number of jobs } \tau_j \text{ s } D_{j,x} \leq r_{i,k+1} \text{ and } r_{j,x} \in [t_{-1}, r_{i,k+1}]) C_j \leq \\
&\leq \sum_{j=1}^{N} \left\lfloor \frac{r_{i,k+1} - t_{-1}}{T_j} \right\rfloor \cdot C_j \leq \\
&\leq \sum_{j=1}^{N} \frac{r_{i,k+1} - t_{-1}}{T_j} \cdot C_j
\end{aligned}
$$

Proof (continued)

So we have

$$r_{i,k+1} - t_{-1} < \sum_{j=1}^{N} \frac{r_{i,k+1} - t_{-1}}{T_j} \cdot C_j$$

Canceling $r_{i,k+1} - t_{-1}$ yields

$$1 < \sum_{j=1}^{N} \frac{C_j}{T_j},$$

i.e.

$$1 < U,$$

This completes the proof.

Outline

EDF with deadlines $<$ period

- If deadlines are less than periods then $U < 1$ is no longer sufficient schedulability condition.
- This is easy to see. Consider two tasks such that, for both, $C_i = 1$ a $T_i = 2$. If both have deadlines at 1.0, then the system is not schedulable, even though $U = 1$.
- For these kinds of systems, we work with densities instead of utilizations.

Definition

The density of task τ_k is defined as $\delta_k = \dfrac{C_k}{\min(D_k, T_k)}$.

The density of the system is defined as $\Delta = \sum_{k=1,\ldots,N} \delta_k$.

EDF with deadlines $<$ period (continued)

Theorem

System $\mathcal{T}$ of independent, preemptible, periodic tasks can be feasibly scheduled on one processor if its density is at most one.

- Proof is similar to the one for the previous theorem.
- **Note:** This gives only sufficient condition.
- The next expression is called EDF schedulability condition.

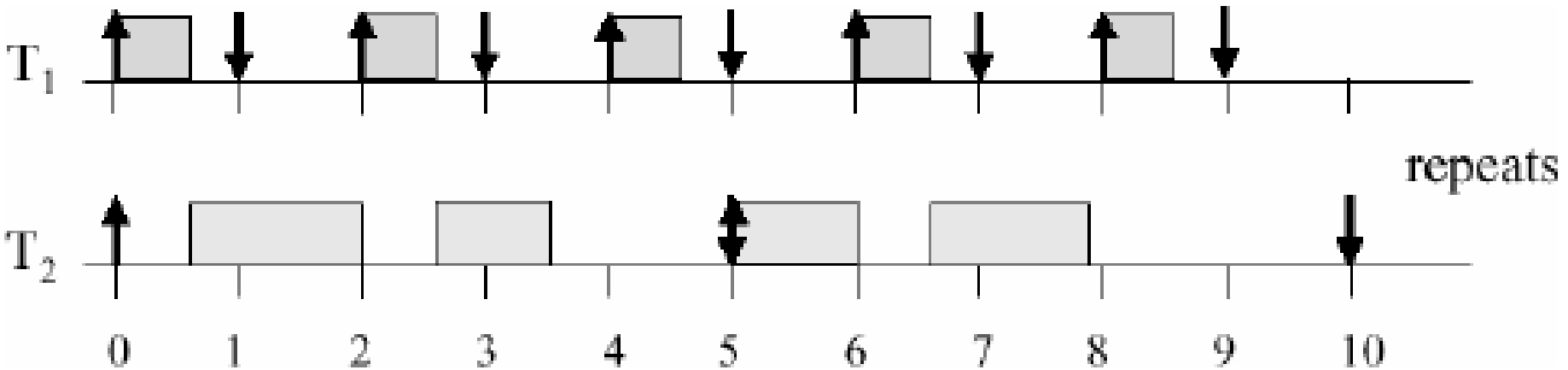
$$\sum_{k=1}^{n} \frac{C_k}{\min(D_k, T_k)} \leq 1$$

Proof on non-tightness

Next example shows, why $\Delta > 1$ does not imply non-feasibility.

Example

Consider tasks $\tau_1 = (2, 0.6, 1)$ and $\tau_2 = (5, 2.3)$.
$\Delta = 0.6/1 + 2.3/5 = 1.06$. Yet, the tasks can be feasibly scheduled by EDF:



EDF in existing systems

- Real-Time threads[1] in MAC OS X are reported to use a variant of EDF.
- SCHED_DEADLINE scheduler in Linux:
  - In mainline kernel since 3.14 (January 2014)
  - Developed at SSSA, Pisa, Italy
  - SCHED_DEADLINE tasks are scheduled before SCHED_FIFO tasks (have higher priority)
  - SCHED_DEADLINE tasks have guaranteed bandwidth (see scheduling servers lecture)

[1] `http://developer.apple.com/library/mac/#documentation/Darwin/Conceptual/KernelProgramming/scheduler/scheduler.html`

SCHED_DEADLINE example (in Linux)

```
void *run_deadline(void *data) {
        struct sched_attr attr;

        attr.size = sizeof(attr);
        attr.sched_flags = attr.sched_nice = attr.sched_priority = 0;

        /* This creates a 10ms/30ms reservation */
        attr.sched_policy = SCHED_DEADLINE;
        attr.sched_runtime = 10 * 1000 * 1000;
        attr.sched_period = attr.sched_deadline = 30 * 1000 * 1000;

        sched_setattr(0, &attr, 0);

        while (!done) { ... }
}
int main (int argc, char **argv) {
        pthread_t thread;
        pthread_create(&thread, NULL, run_deadline, NULL);
        // ...
}
```

Outline

Comparison of EDF and FPS (RM)

Giorgio C. Buttazzo: Rate Monotonic vs. EDF: Judgment Day

https://support.dce.felk.cvut.cz/psr/prednasky/3/rtsj05-rmedf.pdf

- Implementation: Comparable or "it depends"
- Overhead: EDF typically exhibits less number of preemptions
  $\Rightarrow$ lower overhead
- Overrun behavior ($U > 1$):
  - Permanent: EDF – automatic "period rescaling",
    RM – complete blocking of lower priority tasks
  - Transient: Task overrun can cause deadline miss of
    - EDF: arbitrary task
    - RM: only lower priority task

    If we don't know which task will overrun, the result is the same.
- Jitter and latency: RM has no jitter only for the highest-priority task. In overall comparison, EDF provides better results (smaller release-time jitter a smaller input-output latency)
- Resource reservation: Simpler in case of EDF (see future lecture)

Outline

Multiprocessor Scheduling

- EDF scheduling for multiple processors is not optimal.

Multiprocessor Scheduling

- EDF scheduling for multiple processors is not optimal.

- So far we have talked about uni-processor scheduling.
- Nowadays, multi-core CPUs are common.
- Possible approaches to real-time multiprocessor scheduling:
  - **Partitioned scheduling** – each CPU is scheduled independently of the others (Linux, most RTOSes)
  - **Global scheduling** – single scheduler for all CPUs
  - **Clustered scheduling** – mixture of the above