Real Time Linux (PREEMPT_RT)

Table of contents

Definitions

- Based on
  - Kernel Recipes 2016 – Understanding a Real-Time System (more than just a kernel) – **Steven Rostedt**, https://www.youtube.com/watch?v=w3yT8zJe0Uw
  - Kernel Recipes 2016 – Real Time Linux, Who need it? (Not you!) – Steven Rostedt, https://www.youtube.com/watch?v=4UY7hQjEW34
- Real Time System – Hard vs Soft
  - **Soft Real Time**: can deal with outliers, tries to be reliable, may have unbounded latency.

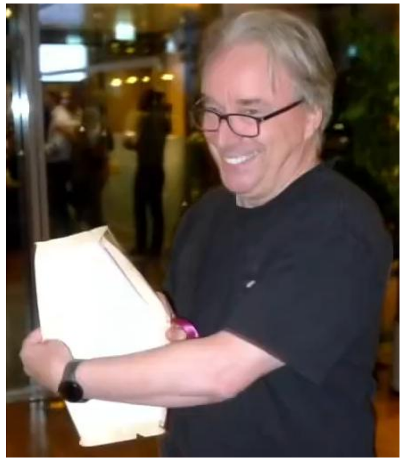
    Examples: video games, video systems, some communication systems.
  - **Hard Real Time**: mathematically provable code, bounded latency.

    Examples: Airplane engine controls, nuclear power plants, Mars Lander, Space shuttle.
- What is PREEMPT_RT then?
  - Hard Real time „designed”.
- Real Time Linux
  - Can not be mathematically proven.
  - Tries to bound all latency (unexpected latency are considered bugs).
  - The design follows that of any hard real time operating system.

History of PREEMPT_RT

- Small group of core developers: Ingo Molnar, Steven Rostedt, Thomas Gleixner, Sebastian A. Siewior, John Ogness.
- Started delivering patches in 2004.
- Merged last brick on a road – the first ever physical pull request (ver. 6.12, September 19, 2024).
- Real time patches: https://lwn.net/Kernel/Index/#Realtime.
- Today almost all required parts of the PREEMPT_RT are part of the common Linux code base.
- Work is continued.

*Realtime kernels are unlikely to become the default, simply because there's some **small performance overhead** from using the realtime config option. But with all the necessary code being part of the mainline kernel, it's certainly possible that some distributions might turn it on by default or make it easier to turn on.*
(https://lwn.net/Articles/990985/)



On September 19, **Thomas Gleixner** delivered the pull request for the realtime preemption enablement patches to **Linus Torvalds** — in printed form, wrapped in gold, with a ribbon, as Torvalds had requested. It was a significant milestone, marking the completion of a project that required 20 years of effort. (https://lwn.net/Articles/990985/)

Who is using PREEMPT_RT over the years?

- PLC – Programmable Logic Controlers.
- Engel Victory for controling the moulding of Lego bricks.
- Keba KeMotion/robotics – painting cars.
- Trumpf TruControl (welding).

Max cycle 150 ms

- Medical devices
- ABS brake controller in a car
- CNC mills and lathes

Max cycle 150 ms

Max cycle 4 ms

Max cycle 2 ms

Realtime Linux (PREEMPT_RT patch)

The **PREEMPT_RT** functionality is about **making the kernel more predictable and reducing latencies** (maintaining low latency) of the kernel to cater to real-time workloads.

The key point of the PREEMPT_RT patch is to **minimize** the amount of kernel code that is **non-preemptible**, while also minimizing the amount of code that must be changed in order to provide this added preemptibility.

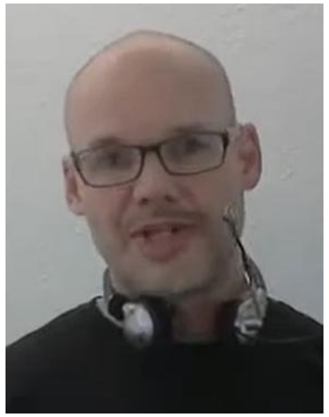
There are three properties that a realtime operating system must have:

- **deterministic scheduling** behavior,
- **interruptibility**,
- and a way to **avoid priority inversion**.

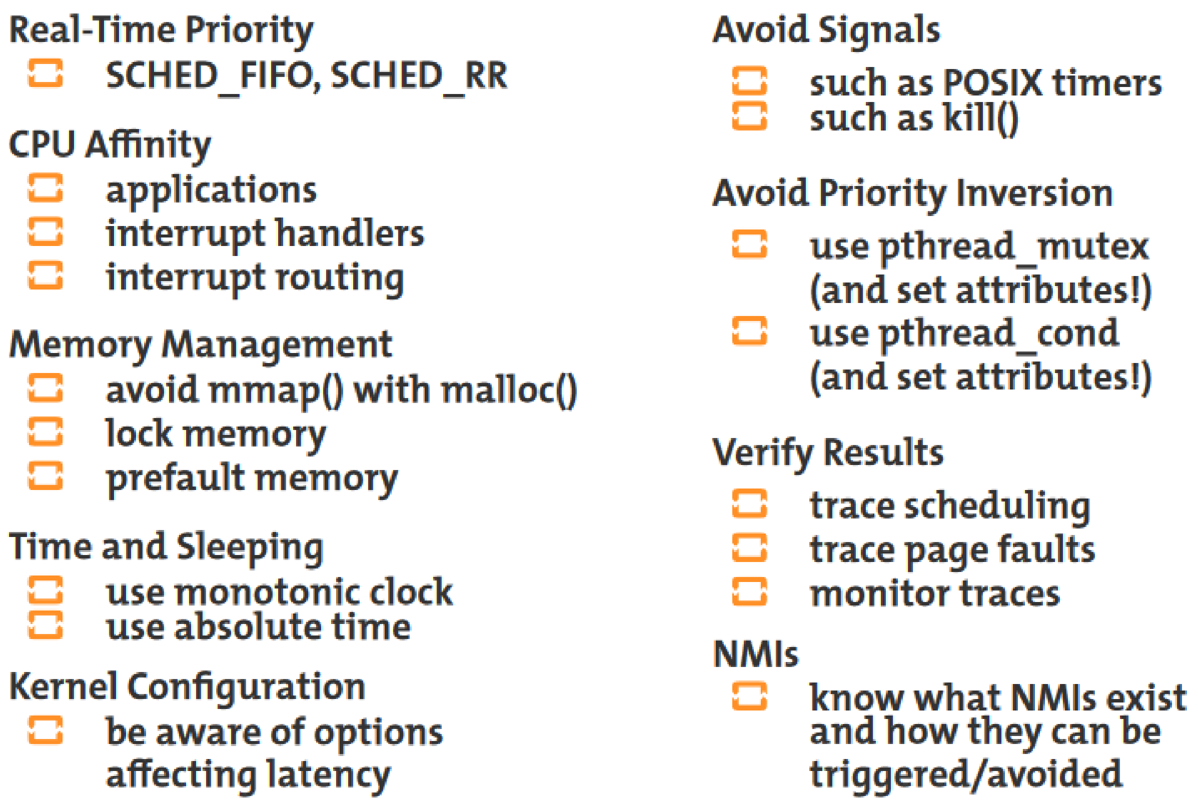
The implementation of many lock types changes in the PREEMPT_RT kernels; in particular, most **spinning locks becoming sleeping locks**.

In the realtime domain, correctness means ***running at the correct time****. The application must wake up within a* ***bounded time limit*** *when there is time-critical work to do. The right timing of tasks is a requirement; things will go wrong if the constraints are not met.*
Developers need to define ***which tasks and applications are time-critical****; a lot of people mistakenly think that all tasks in a realtime system are realtime, while most of them are not.*

Real-Time Checklist & Kernel Configuration



- [A checklist for writing Linux real-time applications](), John Ogness, Embedded Linux Conference Europe, Oct. 2020. (also https://www.youtube.com/watch?v=NrjXEaTSyrw)
- [A real time developer's checklist](), Marta Rybczyńska, Nov 16, 2020.



Real-Time Priority
- ☐ SCHED_FIFO, SCHED_RR

CPU Affinity
- ☐ applications
- ☐ interrupt handlers
- ☐ interrupt routing

Memory Management
- ☐ avoid mmap() with malloc()
- ☐ lock memory
- ☐ prefault memory

Time and Sleeping
- ☐ use monotonic clock
- ☐ use absolute time

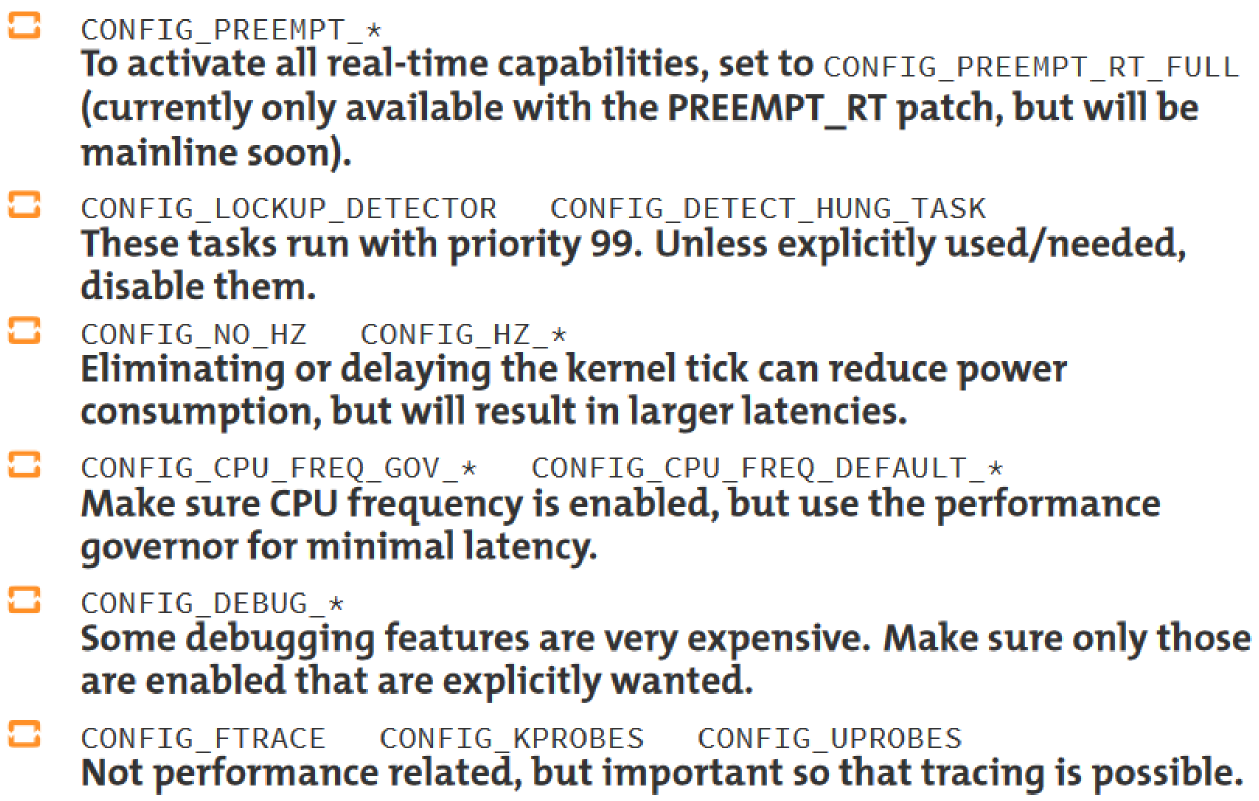
Kernel Configuration
- ☐ be aware of options affecting latency

Avoid Signals
- ☐ such as POSIX timers
- ☐ such as kill()

Avoid Priority Inversion
- ☐ use pthread_mutex (and set attributes!)
- ☐ use pthread_cond (and set attributes!)

Verify Results
- ☐ trace scheduling
- ☐ trace page faults
- ☐ monitor traces

NMIs
- ☐ know what NMIs exist and how they can be triggered/avoided



- ☐ `CONFIG_PREEMPT_*`
  To activate all real-time capabilities, set to `CONFIG_PREEMPT_RT_FULL` (currently only available with the PREEMPT_RT patch, but will be mainline soon).
- ☐ `CONFIG_LOCKUP_DETECTOR` `CONFIG_DETECT_HUNG_TASK`
  These tasks run with priority 99. Unless explicitly used/needed, disable them.
- ☐ `CONFIG_NO_HZ` `CONFIG_HZ_*`
  Eliminating or delaying the kernel tick can reduce power consumption, but will result in larger latencies.
- ☐ `CONFIG_CPU_FREQ_GOV_*` `CONFIG_CPU_FREQ_DEFAULT_*`
  Make sure CPU frequency is enabled, but use the performance governor for minimal latency.
- ☐ `CONFIG_DEBUG_*`
  Some debugging features are very expensive. Make sure only those are enabled that are explicitly wanted.
- ☐ `CONFIG_FTRACE` `CONFIG_KPROBES` `CONFIG_UPROBES`
  Not performance related, but important so that tracing is possible.

What PREEMPT_RT gave to current Linux?

- What does RTOS give us? **Determinism**, determinism, determinism!
  - Repeatability.
  - Reliable results.
  - Known worst case scenarios.
  - Known reaction times.
- What PREEMPT_RT gave to current Linux?
  - NO_HZ and High resolution timers.
  - Generic interrupt design.
  - EDF scheduler (SCHED_DEADLINE).
  - Threaded interrupts.
  - Mutex_lock/unlock (before all sleeping locks were just a semaphore).
  - Priority inheritance (futex).
  - Lockdep – runtime locking correctness validator.
  - Ftrace – the Linux kernel tracer.
  - Printk – prints messages to the kernel log.

Kernel Recipes 2024 – Making the kernel suck less – Steven Rostedt, https://www.youtube.com/watch?v=AjFTVxAU3Vg

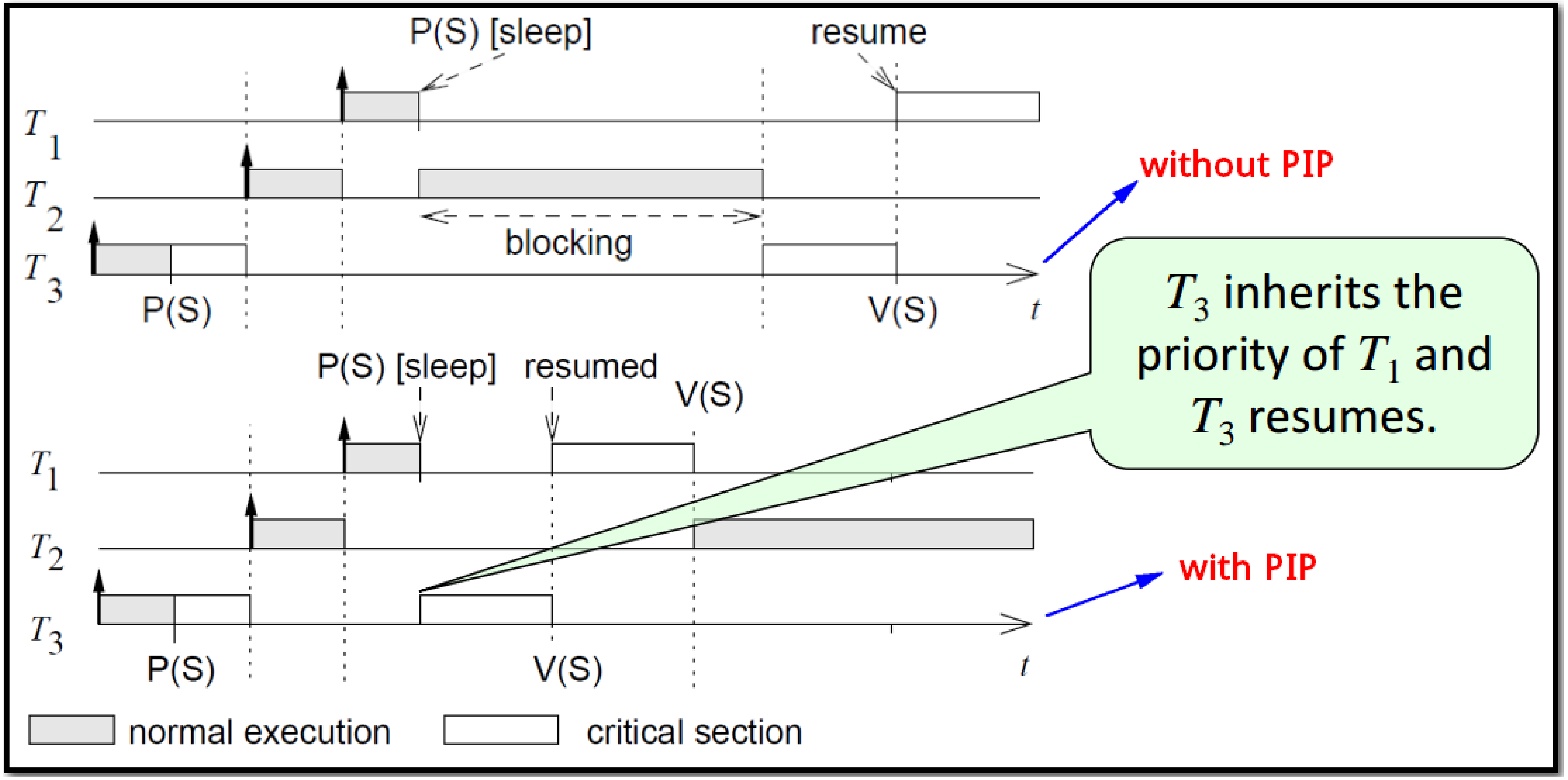
Real time mutexes

Real time mutexes implement **priority inheritence**, and solve the problem of **priority inversion**, which affects **real-time systems**.

Priority inversion is when a **lower priority** process executes while a **higher priority** process wants to run. The example of unbounded priority inversion is where you have three processes, A, B, and C, where A is the **highest** priority process, C is the **lowest**, and B is in **between**. A tries to grab a lock that C owns and must wait and lets C run to release the lock. In the meantime, B executes, and since B is of a higher priority than C, it preempts C, but by doing so, it is in fact preempting A which is a higher priority process.

The problem is solved by **priority inheritance –** process inherits the priority of another process if the other process blocks on a lock owned by the current process. Let's use the previous example. This time, when A blocks on the lock owned by C, C would inherit the priority of A. So now if B becomes runnable, it would not preempt C, since C now has the high priority of A. As soon as C releases the lock, it loses its inherited priority, and A then can continue with the resource that C had.

Priority inversion and priority inheritance



Solutions for Priority Inversion in Real-time Scheduling

Real time mutexes and local locks

Real time mutexes are implemented as a structure **rt_mutex**.

```
struct rt_mutex {
      raw_spinlock_t            wait_lock;
      struct rb_root_cached   waiters;  /*rbtree root to enqueue waiters in priority order; */
      struct task_struct        *owner;
      ...
                                                             Simplified version
};
```

There are functions available:

rt_mutex_init(), **rt_mutex_lock**(), **rt_mutex_unlock**(), **rt_mutex_trylock**().

Local locks in the kernel (from v5.8), Marta Rybczyńska, August 2020.

On non-realtime systems, the acquisition of a local lock simply maps to **disabling preemption** (and possibly **interrupts**).

On real time systems, instead, local locks are actually **sleeping spinlocks**; they **do not disable** either **preemption** or **interrupts**. They are sufficient to serialize access to the resource being protected without increasing latencies in the system as a whole.

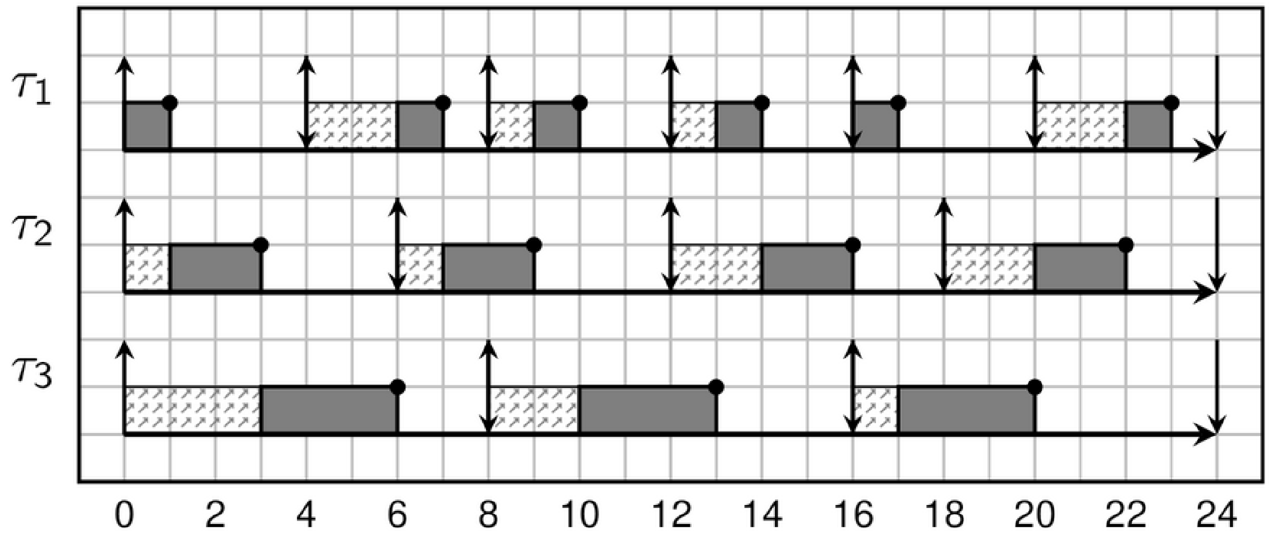
Interface**: local_lock()**, **local_unlock(), local_lock_irq()**, **local_unlock_irq()** etc.

The Deadline scheduler

Worst-case execution time

Task	Runtime (WCET)	Period
T_1	1	4
T_2	2	6
T_3	3	8

https://lwn.net/Articles/743740/



it is not possible to use a fixed-priority scheduler to schedule this task set while meeting every deadline; regardless of the assignment of priorities, one task will not run in time to get its work done.

Deadline scheduling gets away with the notion of process priorities. Instead, processes provide three parameters: runtime, period, and deadline. A SCHED_DEADLINE task is guaranteed to receive "**runtime**" microseconds of execution time every "**period**" microseconds, and these "runtime" microseconds are available within "**deadline**" microseconds from the beginning of the period. The task scheduler uses that information to run the process with the **earliest deadline first** (EDF).

The Deadline scheduler

- /Documentation/echeduler/sched-deadline.rst.
- SCHED_DEADLINE in Wikipedia.
- Deadline scheduling in the Linux kernel, J.Lelli, C. Scordino, L. Abeni, D.Faggioli, Software – Practice& Experience, vol. 46 (6), June 2016.

  Container-Based Real-Time Scheduling in the Linux Kernel, L. Abeni, A. Balsini, T. Cucinotta, EWiLi'18, October 4th, 2018.

- Deadline scheduling part 1 — overview and theory, D. Oliveira, 2018.
- Deadline scheduler part 2 — details and usage, D.Oliveira, 2018.
- https://lwn.net/Kernel/Index/#Realtime-Deadline_scheduling.
- SCHED_DEADLINE desiderata and slightly crazy ideas, D. Oliveira, J. Lelli, DevConf.cz, January 2020.
- Capacity awareness for the deadline scheduler, M. Rybczyńska, May 2020.

  The current implementation of the deadline scheduler does not work well on asymmetric CPU configurations like Arm's big.LITTLE. Dietmar Eggemann posted a patch set to address this problem by adding the notion of **CPU capacity** (the number of instructions that can be executed in a given time) and taking it into account in the admission-control and task-placement algorithms.