Linux EDF scheduling: SCHED_DEADLINE

Corso di
Progetto e Sviluppo di
Sistemi in Tempo Reale

Marcello Cinque
Daniele Ottaviano

SCHED_DEADLINE

- Summary
  - SCHED_DEADLINE: rationale
  - Basic functioning
  - Use in Linux
- References
  - https://www.kernel.org/doc/html/latest/scheduler/sched-deadline.html
  - Linux manpages

It is possible to schedule tasks with EDF in Linux?

- Since the Linux Kernel version 3.14, the SCHED_DEADLINE policy is available to schedule tasks with EDF
- It is a Linux-specific policy, not yet included in RT-POSIX
  - So, not yet available in pthreads functions
- The policy can be set to tasks using
  - `chrt` from the command line
  - or scheduling system calls in the code

SCHED_DEADLINE: rationale

- Regular tasks (`SCHED_OTHER`) are scheduled in background with respect to real-time ones
- Real-time tasks can starve other applications in case of bugs or bad programming
- Example: this task if scheduled as `SCHED_FIFO` with high priority can make a CPU core unstable

```
void bad_bad_task( ) {
  while (1);
}
```

SCHED_DEADLINE: rationale

- Linux implements a rt-throttling protection mechanism that throttles bad tasks (removes them from the CPU)
  - Not clear how this interferes with real-time guarantees…
  - Can we use something more theoretically founded?
- What about aperiodic servers?

SCHED_DEADLINE: key idea

- Augment the basic EDF algorithm with a bandwidth reservation mechanism (specifically, the Constant Bandwidth Server – CBS) to isolate the behavior of real-time tasks between each other.
- The CBS coupled with EDF solves two problems:
  - Enables more appropriate “rt throttling” performed when the task finishes its budget, allowing feasibility analysis and removing interference in case of bad programming
  - Assigns dynamic deadlines to tasks, even when having periodic behavior
    - The kernel should assign deadlines to the individual jobs, but it is actually not aware of jobs! It only sees the task that voluntary “sleeps” until next period!

SCHED_DEADLINE: parameters

- Each task is bound to a CBS characterized by three parameters:
  - ***runtime***, ***period***, and ***deadline***
- Each SCHED_DEADLINE task should receive «*runtime*» microseconds of execution time every «*period*» and the allotted «*runtime*» is available within «*deadline*» microseconds from the start of the period
- The state of each task is described by a ***scheduling deadline*** (the current deadline d_s of its CBS) and a ***remaining runtime*** (the current budget c_s of its CBS)

SCHED_DEADLINE: rules

- When a SCHED_DEADLINE task wakes up (becomes ready for execution), the scheduler checks if:

$$\frac{remaining\ runtime}{scheduling\ deadline\ -current\ time} > \frac{runtime}{period}$$

- if the scheduling deadline is smaller than the current time, or above condition is verified, the scheduling deadline and the remaining runtime are re-initialized as
  - $scheduling\ deadline = current\ time + deadline$
  - $remaining\ runtime = runtime$
- otherwise, the scheduling deadline and the remaining runtime are left unchanged

SCHED_DEADLINE: rules

- When a task executes for an amount of time t, its *remaining runtime* is decreased as:
  - $remaining\ runtime = remaining\ runtime - t$

  (technically, the runtime is decreased at every tick, or when the task is preempted)

- When the remaining runtime becomes 0, the task is “throttled” and cannot be scheduled until its scheduling deadline. The “replenishment time” for this task is set to be equal to the current *scheduling deadline* (implementing a sort of hard CBS behavior)

- When the current time is equal to the replenishment time of a throttled task, the parameters are updated as:
  - $scheduling\ deadline = scheduling\ deadline + period$
  - $remaining\ runtime = remaining\ runtime + runtime$

SCHED_DEADLINE: how to set the parameters?

- Depends on the task model
- For classical Liu and Layland periodic (C_i, T_i) tasks, it has to be:

$$runtime \geq C_i \quad \text{and} \quad period \leq T_i$$

- For sporadic tasks with (C_i, D_i, T_i), where T_i is the minimum interarrival time and $D_i < T_i$, it has to be:

$$runtime \geq C_i \quad \text{and} \quad deadline = D_i \quad \text{and} \quad D_i < period \leq T_i$$

- SCHED_DEADLINE can be used also for aperiodic tasks and self-suspending tasks (so, basically arbitrary tasks but with a guaranteed *runtime/period* bandwith, assuring isolation)

Setting SCHED_DEADLINE with `chrt`

- Using `chrt` with `-d` or `--deadline` flag it is possible to assign the SCHED_DEADLINE scheduling class to a new or an existing task
- Scheduling parameters can be set with the following flags
  - `-T, --sched-runtime nanoseconds` to set the *runtime*
  - `-P, --sched-period nanoseconds` to set the *period*
  - `-D, --sched-deadline nanoseconds` to set the *deadline*
  - priority must be set to 0 (meaningless for SCHED_DEADLINE)
- Example:
  - `sudo chrt -d -T 1000000 -P 10000000 -D 5000000 0 ./task` launches «task» with SCHED_DEADLINE and the specified parameters

Setting SCHED_DEADLINE in the code

- Using `sched_gettattr` and `sched_settattr` primitives:

```
struct sched_attr attr;
sched_getattr(0, &attr, sizeof(attr), 0);
attr.sched_policy = SCHED_DEADLINE;
attr.sched_runtime = 500 * 1000;                    //ns
attr.sched_deadline = 5 * 1000 * 1000;              //ns
attr.sched_period = 5 * 1000 * 1000;                //ns
sched_setattr(0, &attr, 0);
```

Note: these primitives are still not available in sched.h on some distributions; they need a wrapper (see `sched_attributes.h` in the examples)

Setting SCHED_DEADLINE in the code

```
struct sched_attr {
        u32 size;               /* Size of this structure */
        u32 sched_policy;       /* Policy (SCHED_*) */
        u64 sched_flags;        /* Flags */
        s32 sched_nice;         /* Nice value (SCHED_OTHER, SCHED_BATCH) */
        u32 sched_priority;     /* Static priority (SCHED_FIFO, SCHED_RR) */
        /* Remaining fields are for SCHED_DEADLINE */
        u64 sched_runtime;        /* nanoseconds */
        u64 sched_deadline;        /* nanoseconds */
        u64 sched_period; /* nanoseconds */
};
```