Using EDF in Linux: SCHED_DEADLINE

Luca Abeni
luca.abeni@unitn.it

Using Fixed Priorities in Linux

- SCHED_FIFO and SCHED_RR use fixed priorities
  - They can be used for real-time tasks, to implement RM and DM
  - Real-time tasks have priority over non real-time (SCHED_OTHER) tasks
- The difference between the two policies is visible when more tasks have the same priority
  - In real-time applications, try to avoid multiple tasks with the same priority

Setting the Scheduling Policy

```
int sched_get_priority_max(int policy);
int sched_get_priority_min(int policy);

int sched_setscheduler(pid_t pid, int policy,
                       const struct sched_param *param);
int sched_setparam(pid_t pid,
                   const struct sched_param *param);
```

- If `pid == 0`, then the parameters of the running task are changed
- The only meaningful field of `struct sched_param` is `sched_priority`

Problems with Real-Time Priorities

- In general, “regular” (`SCHED_OTHER`) tasks are scheduled in background respect to real-time ones
- Real-time tasks can / starve other applications
- Example: the following task scheduled at high priority can make a CPU / core unusable

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

  - Real-time computation have to be limited (use real-time priorities only when **really needed**!)
  - Using real-time priorities requires root privileges (or part of them!)

Real-Time Throttling

- A “bad” rt task can make a CPU / core unusable...
- ...Linux provides the *real-time throttling* mechanism
  - How does real-time throttling interfere with real-time guarantees?
  - Given a priority assignment, a taskset is guaranteed all the deadlines if no throttling mechanism is used...
  - ...But, what happens in case of throttling?
- Very useful idea, but something more “theoretically founded” might be needed...

What About EDF?

- Can EDF (or similar) be supported in Linux?
- Problem: the kernel is not aware of tasks deadlines...
- ...But deadlines are needed to schedule the tasks
  - EDF schedules tasks based on absolute deadlines
- So, a more advanced API is needed...

EDF on a real OS

- More advanced API:
  - Assign relative deadlines D_i to the tasks...
  - A *runtime* and a *period* are also needed
- Moreover, $d_{i,j} = r_{i,j} + D_i$...
  - ...However, how can the scheduler know $r_{i,j}$?
  - The scheduler is not aware of jobs...
- To use EDF, the scheduler must know when a job starts / finishes
  - Modify applications, or guess...

Tasks and Jobs... And Scheduling Deadlines!

- Applications must be modified to signal the beginning / end of a job (some kind of `startjob()` / `endjob()` system call)...
- ...Or the scheduler can assume that a new job arrives each time a task wakes up!
- Alternative:assign dynamic *scheduling deadlines*
  - Scheduling deadline d_i^s: assigned by the kernel
  - If the scheduling deadline d_i^s matches the absolute deadline $d_{i,j}$ of a job, then the scheduler can respect $d_{i,j}$!!!

CBS: The Basic Idea

- Constant Bandwidth Server (CBS): algorithm to assign a dynamic scheduling deadline d_i^s to a task τ_i
- Based on the *Resource Reservation* paradigm
  - Task τ_i is periodically reserved a *maximum runtime* Q_i every *reservation period* P_i
- Temporal isolation between tasks
  - The worst case finishing time for a task does not depend on the other tasks...
  - ...Because the task is guaranteed to receive its reserved time

CBS: Some More Details

- Based on CPU reservations (Q_i, P_i)
  - If τ_i tries to execute for more than Q_i every P_i, the algorithm decreases its priority, or throttles it
  - τ_i has the same CPU utilisation of a task with WCET Q_i and period P_i
- Q_i/P_i: fraction of CPU time reserved to τ_i
- EDF on the scheduling deadlines $\Rightarrow$ τ_i is guaranteed to receive Q_i time units every P_i if $\sum_j Q_j/P_j \leq 1$!!!

CBS: Schedulability

- EDF $\rightarrow$ easy to guarantee the respect of scheduling deadlines
  - Only on uni-processor / partitioned systems...
- M CPUs/cores with global scheduling: if $\sum_j Q_j/P_j \leq M$ each task is guaranteed to receive Q_i every P_i with a maximum delay

CBS vs Other Reservation Algorithms

- The CBS is based on EDF
  - Assigns scheduling deadlines d_i^s
  - EDF on $d_i^s \Rightarrow$ optimal on UP
- The CBS allows to serve *non periodic tasks*
  - Some reservation-based schedulers have problems with aperiodic job arrivals - due to the (in)famous “deferrable server problem”
  - Explicit support for aperiodic tasks (see the rule for assigning deadlines when a task wakes up)
- Allows to support “self-suspending” tasks

CBS: the Algorithm - 1

- Each task τ_i is associated a scheduling deadline d_i^s and a current runtime q_i
  - Both initialised to 0 when the task is created
- When a task wakes up:
  - Check if the current scheduling deadline can be used ($d_i^s > t$ and $q_i/(d_i^s - t) < Q_i/P_i$)
    - If not, $d_i^s = t + P_i$, $q_i = Q_i$

CBS: the Algorithm - 2

- When τ_i executes for a time δ, $q_i = q_i - \delta$
- When $q_i = 0$, τ_i cannot be scheduled (until time d_i^s)
  - At time d_i^s, $d_i^s = d_i^s + P_i$ and $q_i = q_i + Q_i$

SCHED_DEADLINE - 1

- New SCHED_DEADLINE scheduling policy
  - Foreground respect to all of the other policies
- Uses the CBS to assign scheduling deadline to SCHED_DEADLINE tasks
  - Assign a (maximum) runtime Q_i and a (reservation) period P_i to every SCHED_DEADLINE task
  - Additional parameter: relative deadline D_i
  - The “check if the current scheduling deadline can be used” rule is used at task wake-up

SCHED_DEADLINE - 2

- Once the CBS has been used to assign scheduling deadlines to tasks...
- ...Use EDF (based on scheduling deadlines) to schedule them
- What about multiple CPUs?
  - Both global EDF and partitioned EDF are possible
  - Configurable through the `cpuset` mechanism

Using SCHED_DEADLINE

- Linux provides a (non standard) API for using `SCHED_DEADLINE`, but...
- ...How to dimension the scheduling parameters?
  - (Maximum) runtime Q_i
  - (Reservation) period P_i
- Obviously, it must be

$$\sum_i \frac{Q_i}{P_i} \leq M$$

  - The kernel can do this admission control

Assigning Runtime and Period

- Temporal isolation
  - Each task can be guaranteed independently from the others
- `SCHED_DEADLINE` can be used to serve both hard real-time and soft real-time tasks!
  - The scheduling parameters must be assigned according to the kind of task
  - Hard schedulability property or stochastic analysis

Assigning Parameters to Hard Tasks

- Hard Schedulability property
  - If $Q_i \geq C_i$ and $P_i \leq T_i$ (maximum runtime larger than WCET, and server period smaller than task period)...
  - ...Then the scheduling deadlines are equal to the jobs' deadlines!!!
- All deadlines are guaranteed to be respected (on UP / partitioned systems), or an upper bound for the tardiness is provided (if global scheduling is used)!!!
- Hard real-time tasks need partitioned scheduling!

What About Soft Real-Time?

- What happens if $Q_i < C_i$, or $P_i > T_i$?
  - $\frac{Q_i}{P_i}$ must be larger than $\overline{c_i}/\overline{t_i}$
  - ...Otherwise, $d_i^s \rightarrow \infty$ and there will be no control on the task's response times

- Possible to do some stochastic analysis
  - Given $\overline{c_i} < Q_i < C_i$, $T_i = nP_i$, and the probability distributions of execution and inter-arrival times
  - ...It is possible to find the probability distribution of the response times (and the probability to miss a deadline)!

Changing Parameters...

- Tasks' parameters (execution and inter-arrival times) can change during the tasks lifetime... So, how to dimension Q_i and P_i?
- Short-term variations: CPU reclaiming mechanisms (GRUB, ...)
  - If a job does not consume all of the runtime Q_i, maybe the residual runtime can be used by other tasks...
- Long-term variations: adaptive reservations
  - Generally "slower", can be implemented by a user-space daemon