An EDF Scheduling Class for the Linux Kernel

Dario Faggioli
faggioli@evidence.eu.com

Claudio Scordino
claudio@evidence.eu.com
"The next project will be SCHED_EDF (Earliest Deadline First scheduling) to bring full deadline scheduling to the linux kernel."

from linux-2.6/Documentation/scheduler/sched-rt-group.txt
What is EDF?

• Earliest Deadline First (EDF) is a dynamic-priority real-time scheduling algorithm.

• At each instant, the system schedules the (ready) task(s) having earliest deadline(s).

![Diagram of EDF scheduling](image)
EDF and “cheating” tasks

• What if
  – We are not able to estimate accurately task execution times?
  – A task executes more than it declares?
EDF and “cheating” tasks

• What if
  – We are not able to estimate accurately task execution times?
  – A task executes more than it declares?

“cheating” task missing its deadline... quite bad..

non “cheating” task missing its deadline because of some other “cheating” task... TOO BAD!!
EDF + Temporal Isolation

• Each task is assigned a runtime and a (relative) deadline
  – The meaning is that the task is guaranteed a CPU share equal to runtime every deadline
  – The ratio runtime/deadline is called task's "bandwidth"

• The CPU bandwidth guaranteed to a task doesn't depend on what happens in the rest of the system
  – If a task tries to execute more than its bandwidth, it is slowed down (i.e., blocked “for a while”) to not affect other tasks
  – Meeting a task deadline only depends on its own behaviour
  – Buggy or misbehaving tasks can't monopolize the CPU
Example of an EDF Program

nrun = 0;
<set_EDF_scheduling_policy(deadline, runtime)>
<interval = get_current_time()>
while(!shutdown) {
    print("task cycle %d
", nrun);

    /*
     * Actual computation
     */

    nrun++;
    <end_of_EDF_instance()>
    <interval = calculate_next_period(period)
    <sleep_until(interval)>
nrun = 0;
<set_EDF_scheduling_policy(deadline, runtime)>
<interval = get_current_time()>
while(!shutdown) {
    print("task cycle %d\n", nrun);
    /*
     * Actual computation
     */
    nrun++;
    <end_of_EDF_instance()>
    <interval = calculate_next_period(period)>
    <sleep_until(interval)>
}
Example of an EDF Program

nrun = 0;
<set_EDF_scheduling_policy(deadline, runtime)>
<interval = get_current_time()>
while(!shutdown) {
    print("task cycle %d\n", nrun);
    /*
     * Actual computation
     */
    nrun++;
    <end_of_EDF_instance()>
    <interval = calculate_next_period(period)>
    <sleep_until(interval)>
}

new absolute deadline assigned: wakeup_time + deadline

Actual execution time... is it <= runtime?
Current Linux scheduler

- Modular scheduler, composed by two **scheduling classes**:
  - sched_fair for SCHED_NORMAL (best-effort) tasks: proportional share algorithm with focus on fairness (CFS)
  - sched_rt for SCHED_FIFO/RR tasks: POSIX real-time compliant (fixed-priority) scheduler

- Control Groups (cgroups)
  - groups tasks together
  - allows hierarchical scheduling/CPU sharing
Current Linux scheduler (cont'd)

- Fair scheduler
- Kind of resource reservation

SCHED_NORMAL Task

SCHED_RR Task

SCHED_FIFO Task

sched_fair
  - Fair scheduler
  - Kind of resource reservation

sched_rt
  - High-priority scheduler

Linux scheduler
Do we need a NEW scheduling class?

- Using sched_fair...
  - real-timing guarantees are hard to meet:
    - difficult to specify a task must execute for 20msec within 100msec
    - even when there is enough spare CPU time, tasks may experience deadline misses (test case on the website)
    - time between consecutive activations of a task is not deterministic and cannot be bound since it depends on the number of tasks running

- Using sched_rt...
  - we can achieve real-time guaranteed behaviour, but:
    - smaller maximum bandwidth to exploit than EDF
    - good predictability, but smaller flexibility than EDF
About SCHED_EDF

- We added such NEW scheduling class
  - SCHED_EDF (but we are open to new names)
  - Earliest Deadline First (EDF) scheduling order
Details about SCHED_EDF (1)

• Tracking Linux development
  – patch on top of Linus' current git
  – patch on top of sched-devel
  – porting to preempt-rt
• Early (too early? ;-P) released on LKML
• Supports x86 and ARM platforms
• It is the highest-priority scheduling class
  (following the comments we got on LKML)
• Works SMP/Multicore systems:
  – Partitioned-EDF, as a starting point
  – Global-EDF planned with push & pull migration logic
    (as in sched_rt)
Details about SCHED_EDF (2)

• Deals with periodic, sporadic or aperiodic tasks
  – a task can inform the kernel an
    (periodic/sporadic?)
    “instance” just ended
  – a task can ask to be notified about deadline
    misses and/or runtime overruns (following the
    comments we got on LKML)

• Integrated with control groups to:
  – hierarchical admission test
  – CPU isolation/system partitioning enforcement (cpusets)
SCHED_EDF: Open Issues

• If an EDF task forks, the child:
  – (if !sched_reset_on_fork) receives NO bandwidth, and its parent has to sched_setscheduler_ex() it (following the comments we got on LKML)

• Critical Sections:
  – very simple (and wrong!) deadline inheritance mechanism...
  – more smart and sophisticated solution needed, e.g., BWI/PEP (we're investigating on this!)
The SCHED_EDF Task API

#define SCHED_EDF 6

#define SCHED_EDF_WANT_MISS_SIGNAL 1
#define SCHED_EDF_WANT_ORUN_SIGNAL 2

struct sched_param_ex {
    int sched_priority;
    struct timespec sched_edf_deadline;
    struct timespec sched_edf_runtime;
    int sched_flags
};

int sched_setscheduler_ex (pid_t pid, int policy, struct sched_param_ex *param);
int sched_getparam_ex(pid_t pid, struct sched_param_ex *param_ex);
int sched_setparam_ex(pid_t pid, struct sched_param_ex *param_ex);
The SCHED_EDF Bandwidth Interface

• System wide EDF-bandwidth:
  • /proc/sys/kernel/sched_edf_period_us
  • /proc/sys/kernel/sched_edf_runtime_us

• Per-group EDF-bandwidth
  • /cgroups/<group>/cpu.edf_period_us
  • /cgroups/<group>/cpu.edf_runtime_us
Example of an EDF Program

```c
sched_edfp.sched_deadline = us_to_timespec(deadline);
sched_edfp.sched_runtime = us_to_timespec(wcet);

sched_setscheduler_ex(0, SCHED_EDF, &sched_edfp);
glock_gettime(CLOCK_MONOTONIC, &interval);
while (!shutdown) {
    printf("task %d: running cycle %d\n", gettid(), nrun);

    /*
     * Actual computation
     */

    nrun++;
    sched_yield();

    timespec_add_us(&interval, period)
    clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &interval, NULL);
}
```
Trying this out!

• Web site: http://www.evidence.eu.com/sched_edf.html
  – Documentation, git & patches

• Cloning from us:
  – git clone \git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git
  – cd linux-2.6
  – git pull git://feanor.sssup.it/sched-edf.git sched-edf

• Modified version of schedtool:
  – git clone git://feanor.sssup.it/schedtool-edf.git
A Simple SCHED_EDF Example:

• Simple yes program with
  – Period = 100000us
  – Budget = 20000us

• Commands:
  – git clone git://feanor.sssup.it/schedtool-edf.git
  – cd schedtool-edf
  – make
  – ./schedtool -E -d 100000 -b 20000 -a 0 -e yes
A simple example: trace

- Trace obtained using:
  - ftrace kernel infrastructure (sched_switch)
  - analyze program
  - GtkWave viewer
- Meaning of the lines:
  - Green "low" lines: task execution
  - Yellow "high" lines: task not runnable
  - Red boxes: task ready but not running
Thank you for listening!

Questions?