Deadline scheduling on Linux and why it hasn't happened yet.

Peter Zijlstra
Red Hat
peterz@infradead.org

What is Real-Time?

- ~~Very Fast~~
- ~~Low Latency~~
- Determinism;
  - being able to accurately predict what is going to happen
  - Stop the saw when your hand is near, not maybe a bit late

What's wrong with FIFO?

- It is deterministic, very simple scheduling rule:
  - The task with the highest priority runs
- However, priorities don't always map well to our problems.
- Priorities don't provide isolation in a usable fashion.

Problem with Priorities

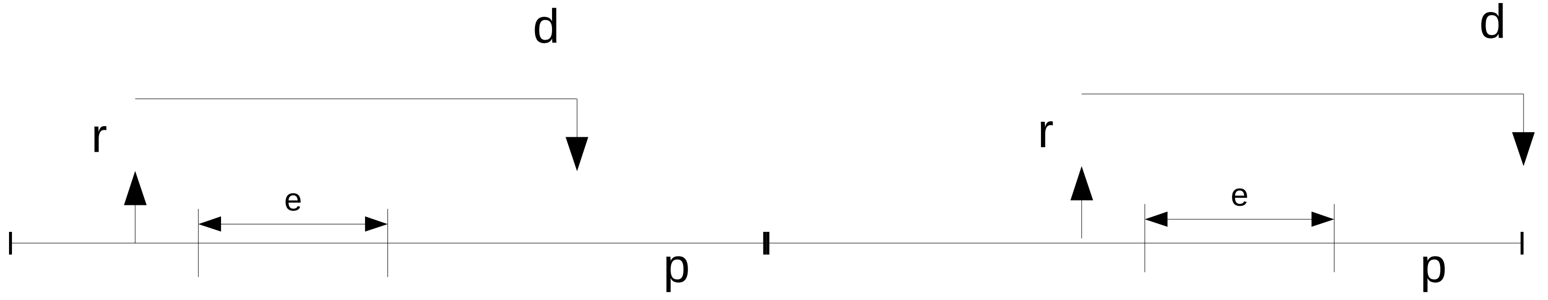
- Is charging the battery a low or high priority task?
  - The priority is inv. proportional to the charge level
- How do you map priorities onto two disjoint applications?

Operating systems and Resource Management

- The goal of operating systems is to abstract and manage resources
- Provide isolation between users
- Does the abstraction match the problem domain
- SCHED_FIFO → fail!

What is this Deadline stuff I keep hearing about?

- A different (task) model for Real-Time applications
- Each task t_i has:
  - WCET - e_i
  - Deadline - d_i
  - Period - p_i
- Where: e_i ≤ d_i ≤ p_i



So what about that?

- It allows you to specify when something is supposed to be done (deadline), how long it'll take to do it (wcet) and how often you'll need it (period).
- Allows the operating system to know how much pending work there is, and reject new jobs if it sees it can't meet expectations – Admission.
- Provides Isolation between workloads.
- Maps better to most problems.

How do you schedule such a task-set?

- Earliest Deadline First (EDF)
  - Schedule the task who's deadline will expire first.
  - Runs work as soon as possible.
- Least Laxity First (LLF)
  - Schedule the task that has least room to still make its deadline.
  - Runs work as late as possible.

- Variety of other creative ways.

If its so neat, why don't we have it?

- More complicated task model
  - 3 variables to specify instead of 1
    - While providing the deadline is often easy providing the WCET is a rather difficult problem on its own.
- This little detail called SMP
- Another pesky detail commonly know as Priority Inversion.

Utilization, Schedulability and Admittance (EDF-UP)

- Utilization $u_i = e_i/p_i$
- Schedulability, the full task set is schedulable:
  - $U = \text{Sum } u_i \leq 1$
- Admittance, reject jobs when the above would be violated.

Utilization, Schedulability and Admittance (GEDF-SMP)

- Utilization limit: $U = (m + 1) / 2$
  - Which gives: $m \gg 1$, $U \sim 50\%$
- Schedulability & Admittance, more complex and interesting.

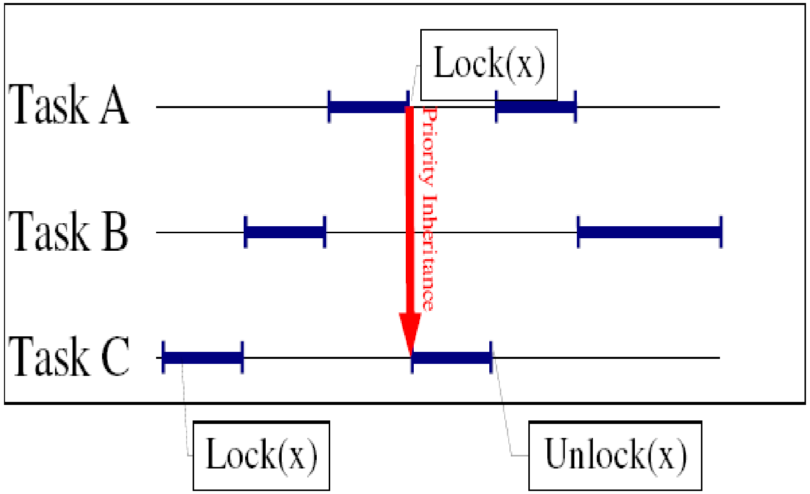
Partitioning

- Too much work, 32-cores is not uncommon
- Not always possible, imagine 3 jobs of u_i = 60% on 2 cores.
- Its possible to reduce a global algorithm to partitions, but not the other way around, which suggests its the wrong abstraction.

Soft Real-Time

- Bounded Tardiness
- GEDF up to U=m
- Can run in Hard-RT idle time
- Needs co-operation for schedulability/admission.

Priority Inversion



- Let A, B and C be a high, med. and low priority task.
- A blocks on a lock held by C.
- B preempts C, and can delay A indefinitely

Priority Inheritance

- Let the lock owner inherit the highest prio of its block list.
- Bound to static priority scheduling

Generalized Inheritance

- Apply the scheduling function to the block-list.
- Turns the block list into a runqueue
- Turns the block chain into a recursive scheduler
- Turns the cost of PI into the cost of the scheduling function – O(log(n))

Deadline Inheritance

- EDF selects tasks on earliest deadline.
- So lock contention can get an earlier deadline stuck behind a later one.
- Have the lock owner inherit the earliest deadline of the block list.

Bandwidth Inheritance

- Bandwidth enforcement
- Lock owner without bandwidth
- Consume the bandwidth of the task that donated its deadline

Proxy Execution

- Turn the Inheritance problem up-side-down
- Leave blocked tasks on the runqueue
- Chain/Proxy blocked tasks to the lock owner

Proxy Execution vs SMP

- If tasks A and B, running on different CPUs, both block on C, then it could happen that both CPUs end up running C → badness.
- 'migrate' all blocked tasks to the owner's CPU.
  - Reduces the problem to UP
  - O(n) migration overhead

Proxy Execution vs IO

- A blocked on B, which is blocked on IO.
  - Need to take A off the runqueue – O(n)
  - Needs a 'wait-list' to put A back when B gets put back

Deadlines and cgroups

- Control utilization
- Hierarchical accounting
- No need for hierarchical scheduling

Deadline and POSIX

- Outside the SPEC, life is good, you get to make the rules
- SCHED_DEADLINE/SCHED_SOFT above SCHED_FIFO
- New interfaces..
  - struct sched_param_ex
  - sys_sched_setscheduler_ex()
  - sys_sched_setparam_ex()
  - sys_sched_getparam_ex()

The end!

Questions?