Run-time interrupts latency detection in real-time systems

Julien Desfossez
Michel Dagenais

December 2015
École Polytechnique de Montreal

Latency-tracker

- Kernel module to track down latency problems at run-time
- Simple API that can be called from anywhere in the kernel (tracepoints, kprobes, netfilter hooks, hardcoded in other module or the kernel tree source code)
- Keep track of entry/exit events and calls a callback if the delay between the two events is higher than a threshold

Usage

```
tracker = latency_tracker_create(threshold,
timeout, callback);


latency_tracker_event_in(tracker, key);

....

latency_tracker_event_out(tracker, key);
```

If the delay between the event_in and event_out for the same key is higher than “threshold”, the callback function is called.

The timeout parameter allows to launch the callback if the event_out takes too long to arrive (off-CPU profiling).

Implemented use-cases

- Block layer latency
  - Delay between block request issue and complete
- Wake-up latency
  - Delay between sched_wakeup and sched_switch
- Network latency
- IRQ handler latency
- System call latency
  - Delay between the entry and exit of a system call
- Offcpu latency
  - How long a process has been scheduled out

Performance optimizations

- Controlled memory allocation
- Lock-less per-cpu RCU free-list
- Out-of-context reallocation of memory if needed/enabled
- Kernel-ported lock-less userspace-rcu hashtable
- Custom call_rcu thread to avoid the variable side-effects of the built-in one

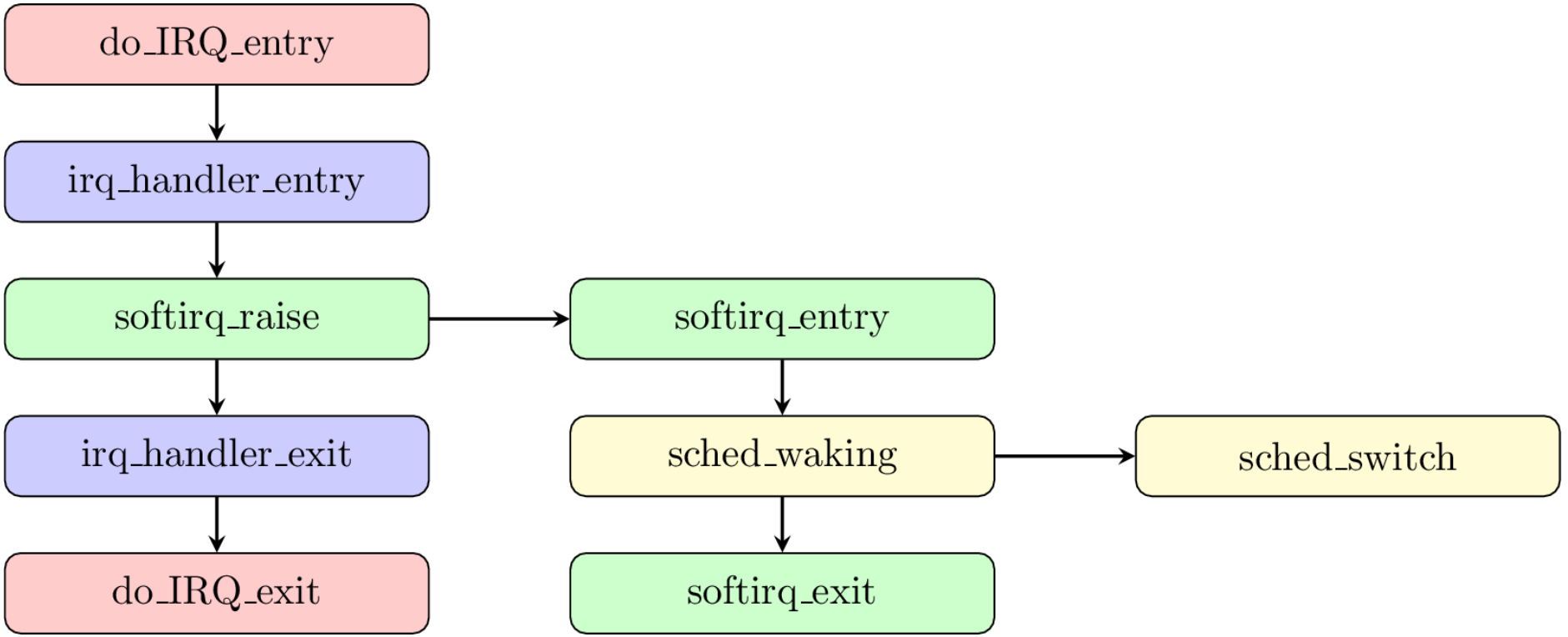
Tracking interrupts latency

- Start tracking when the kernel receives the interrupt
- Compute the delay up to the moment when:
  - The target task get scheduled in
  - The target task informs the kernel it finished its work
  - The target task goes back to waiting for the next interrupt
- Launch a user-defined action on high latency

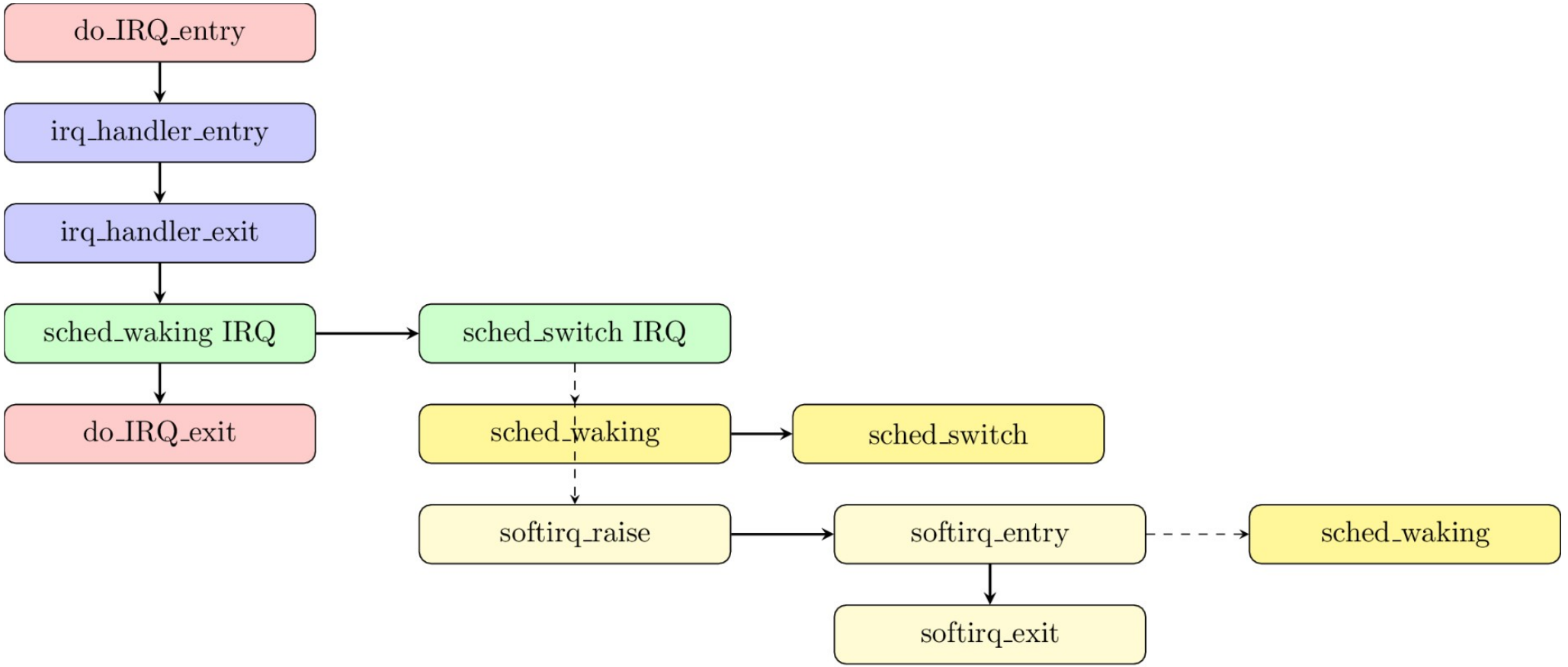
Tracking interrupts latency

- Work with the two main workloads:
  - periodic (timers)
  - aperiodic (hardware interrupts)

Interrupts critical path in the mainline kernel



Interrupts critical path in the PREEMPT_RT kernel



PREEMPT_RT hardware interrupts critical path

Tracking state evolution

- The relevant information is known while processing the chain
- Not a single matching entry/exit key
- Need to make the state changes in real-time
- Filter based on the target use-case:
  - IRQ number
  - SoftIRQ number
  - Target PID/Procname
  - Only real-time priority tasks

Online critical tree

- Tracking an interrupt up to the point where a user-space task starts to run is usually a chain (no branches)
- But if we track an interrupt until the target task completes its work, there can be a lot of branches
- Each call to sched_waking or softirq_raise creates a new branch in the chain

Online critical tree

- We stop the tracking when one chain matches all the criteria
- We only know which one at the end
- So we need to track everything and cleanup as soon as possible to limit the overhead

Demos

- User inputs
- Jack realtime sound server

Overhead

- Early measurements
- 740ns per state change for keeping the state
- 6 state changes --> 4.4µs
- Additional overhead for keeping the textual breakdown

Install it

```
apt-get install git gcc make
linux-headers-generic

git clone
https://github.com/jdesfossez/late
ncy_tracker.git

cd latency_tracker

make
```

Questions ?