

Understanding differences in results between timerlat and cyclictst

Dissecting the timer wake-up cycle

Tomas Glozar, John Kacur, Luis Goncalves (Red Hat)

Introduction

- both tools measure (cyclical) *timer latency* (serving also as an example of an arbitrary IRQ-driven workload)
 - set up high resolution timer to wake up process at time X
 - process wakes up at time X+Y, measure and record Y
- difference: cyclicttest measures the timer's userspace-side, timerlat is kernel-side (optionally with kthread instead of process)
 - *clock_nanosleep* vs timerlat-specific *hrtimer* (directly or through a tracer file descriptor)
 - **one** vs **three** latency numbers (IRQ, thread, user)
- cyclicttest is a part of rt-tests, timerlat is a part of the tracing subsystem in the Linux kernel
- **rtla** is the userspace frontend to the timerlat (and osnoise) tracers

The problem

In a way, rtla/timerlat is a **better replacement** for cyclictst: it provides more tracing capabilities by the virtue of being in-kernel.

But – is timerlat fit to drop-in replace cyclictst? For example, timerlat (in user mode) tends to report **larger** latency numbers than cyclictst.

Is this **tracing overhead**, different **point of measurement**, or something else?

Hint: it's all above, kind of.

Let's compare the timer cycle in close detail.

Timer-based workload cycle: cyclicttest vs timerlat

–	event	sync point	avg latency (c/t/diff, us) ¹			context transition	diff explanation
1	timer expiry	T+0	0.000	0.000	–	–	
2	early IRQ entry		0.222	0.221	-0.001	idle → IRQ	identical, HW dependent
3	IRQ handler in	timerlat IRQ latency	1.274	1.423	+0.149		<i>osnoise:irq_entry</i>
4	thread wake-up		1.291	1.576	+0.285		<i>osnoise:timerlat_sample</i>
5	IRQ handler out	(rtla IRQ duration)	1.854	2.158	+0.303		
6	scheduler switch	timerlat thread latency	3.630	4.017	+0.387	IRQ → task	<i>osnoise:irq_exit</i>
7	wait syscall exit	cyclicttest latency	4.788	5.920	+1.132	kernel → user	between timerlat points
8	wait syscall entry	timerlat user latency	4.907	5.999	+1.091	user → kernel	clock_gettime effect
9	timer start		5.325	6.627	+1.301		
10	scheduler switch		7.136	8.978	+1.841	task → idle	

¹ measured with Intel Processor Trace, with RTLA_NO_BPF=1 for rtla-timerlat; ~1 us of perf overhead to measure the set timer expiry time

References

- cyclicttest:
<https://git.kernel.org/pub/scm/utils/rt-tests/rt-tests.git/tree/src/cyclicttest>
- timerlat tracer:
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/kernel/trace/trace_osnoise.c
- rtla:
<https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/tools/tracing/rtla/>
- Tool used to process the perf traces: <https://gitlab.com/tglozar/timerpt>
 - incomplete (supports only one thread, noise detection is limited)
 - AI-generated (Claude), annotations verified on a small sample

Extra slides (detailed description)

1. Timer expiry time

The set time where the cyclictst/timerlat timer is scheduled to expire.

Reference point for measurement (T+0).

Under standard conditions on an isolated core, the CPU is in idle at this point.

That may mean *polling* idle (to minimize wake-up latency) or a C-state, depending on system settings.

Measured using perf tracepoints (hrtimer_start/hrtimer_expiry_entry).

2. Early IRQ entry

Architecture-specific IRQ backing the timer firing (APIC in this case).

Transition: idle to IRQ context

Noise:

Other IRQs or non-blockable interrupts (NMI, SMI) may delay the timer IRQ (applies to all points here).

Latency penalty through timekeeper blocking here may be mistaken for hardware/firmware noise.

Negative noise possible through tailing on another timer's expiry.

3. Timer handler entry

Once the early IRQ entry is processed, the kernel enters the timer callback function.

Measurement point: timerlat IRQ latency

Noise:

Latency penalty through timekeeper blocking might also occur here.

Timerlat traces this through the osnoise IRQ entry tracepoint.

4. Measurement thread wakeup

The timer callback function wakes up the measurement workload thread.

Noise:

Scheduler.

5. IRQ handler exit

The timer expiry function and subsequently the IRQ handler itself exit.

Measurement point: timerlat auto-analysis IRQ duration.

Noise:

IRQ exit tail events.

6. Scheduler switch to measurement thread

At the IRQ exit tail, the scheduler switches from the idle task onto the measurement thread.

Measurement point: timerlat thread latency (kernel side).

Noise:

Scheduler.

7. Measurement threads exits wait syscall

Kernel space to user space transition after the thread wakes up.

Measurement point: cyclicttest latency (userspace side).

Noise:

Work scheduled for syscall exit.

8. Measurement thread enters next syscall

User space to kernel space transition for the next timer cycle.

For timerlat, there is only one syscall (*read* of timerlat fd) for the entire cycle; cyclicttest uses two: *clock_nanosleep* and *clock_gettime*.

Measurement point: timerlat user latency (kernel side).

Noise:

syscall entry.

9. Timer is re-armed for the next cycle

Either timerlat's *hrtimer* or the clock_nanosleep *hrtimer* is re-armed.

Noise:

Any noise here or in the next state is not counted into the measurement, unless it rolls over into the next measurement period.

10. Scheduler switch to idle

The scheduler switches the measurement thread out and the CPU enters an idle state.

Noise:

IRQ noise might occur here as well, wrapping over to the next cycle.