Timerlat tracer

The timerlat tracer aims to help the preemptive kernel developers to find sources of wakeup latencies of real-time threads. Like cyclictest, the tracer sets a periodic timer that wakes up a thread. The thread then computes a *wakeup latency* value as the difference between the *current time* and the *absolute time* that the timer was set to expire. The main goal of timerlat is tracing in such a way to help kernel developers.

Usage

Write the ASCII text "timerlat" into the current_tracer file of the tracing system (generally mounted at /sys/kernel/tracing).

For example:

```
[root@f32 ~]# cd /sys/kernel/tracing/
[root@f32 tracing]# echo timerlat > current_tracer
```

It is possible to follow the trace by reading the trace trace file:

```
[root@f32 tracing]# cat trace
# tracer: timerlat
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            || /
#                            ||||             ACTIVATION
#         TASK-PID      CPU# ||||   TIMESTAMP    ID            CONTEXT                 LATENCY
#            | |         |   ||||      |         |                  |                       |
        <idle>-0       [000] d.h1    54.029328: #1     context    irq timer_latency       932 ns
         <...>-867     [000] ....    54.029339: #1     context thread timer_latency     11700 ns
        <idle>-0       [001] dNh1    54.029346: #1     context    irq timer_latency      2833 ns
         <...>-868     [001] ....    54.029353: #1     context thread timer_latency      9820 ns
        <idle>-0       [000] d.h1    54.030328: #2     context    irq timer_latency       769 ns
         <...>-867     [000] ....    54.030330: #2     context thread timer_latency      3070 ns
        <idle>-0       [001] d.h1    54.030344: #2     context    irq timer_latency       935 ns
         <...>-868     [001] ....    54.030347: #2     context thread timer_latency      4351 ns
```

The tracer creates a per-cpu kernel thread with real-time priority that prints two lines at every activation. The first is the *timer latency* observed at the *hardirq* context before the activation of the thread. The second is the *timer latency* observed by the thread. The ACTIVATION ID field serves to relate the *irq* execution to its respective *thread* execution.

The *irq*/*thread* splitting is important to clarify in which context the unexpected high value is coming from. The *irq* context can be delayed by hardware-related actions, such as SMIs, NMIs, IRQs, or by thread masking interrupts. Once the timer happens, the delay can also be influenced by blocking caused by threads. For example, by postponing the scheduler execution via preempt_disable(), scheduler execution, or masking interrupts. Threads can also be delayed by the interference from other threads and IRQs.

Tracer options

The timerlat tracer is built on top of osnoise tracer. So its configuration is also done in the osnoise/ config directory. The timerlat configs are:

- cpus: CPUs at which a timerlat thread will execute.
- timerlat_period_us: the period of the timerlat thread.
- stop_tracing_us: stop the system tracing if a timer latency at the *irq* context higher than the configured value happens. Writing 0 disables this option.
- stop_tracing_total_us: stop the system tracing if a timer latency at the *thread* context is higher than the configured value happens. Writing 0 disables this option.
- print_stack: save the stack of the IRQ occurrence, and print it after the *thread context* event".

timerlat and osnoise

The timerlat can also take advantage of the osnoise: traceevents. For example:

```
  [root@f32 ~]# cd /sys/kernel/tracing/
  [root@f32 tracing]# echo timerlat > current_tracer
  [root@f32 tracing]# echo 1 > events/osnoise/enable
  [root@f32 tracing]# echo 25 > osnoise/stop_tracing_total_us
  [root@f32 tracing]# tail -10 trace
       cc1-87882   [005] d..h...   548.771078: #402268 context    irq timer_latency     13585 ns
       cc1-87882   [005] dNLh1..   548.771082: irq_noise: local_timer:236 start 548.771077442 duration 7597 ns
       cc1-87882   [005] dNLh2..   548.771099: irq_noise: qxl:21 start 548.771085017 duration 7139 ns
       cc1-87882   [005] d...3..   548.771102: thread_noise:      cc1:87882 start 548.771078243 duration 9909
timerlat/5-1035    [005] .......   548.771104: #402268 context thread timer_latency     39960 ns
```

In this case, the root cause of the timer latency does not point to a single cause but to multiple ones. Firstly, the timer IRQ was delayed for 13 us, which may point to a long IRQ disabled section (see IRQ stacktrace section). Then the timer interrupt that wakes up the timerlat thread took 7597 ns, and the qxl:21 device IRQ took 7139 ns. Finally, the cc1 thread noise took 9909 ns of time before the context switch. Such pieces of evidence are useful for the developer to use other tracing methods to figure out how to debug and optimize the system.

It is worth mentioning that the *duration* values reported by the osnoise: events are *net* values. For example, the thread_noise does not include the duration of the overhead caused by the IRQ execution (which indeed accounted for 12736 ns). But the values reported by the timerlat tracer (timerlat_latency) are *gross* values.

The art below illustrates a CPU timeline and how the timerlat tracer observes it at the top and the osnoise: events at the bottom. Each "-" in the timelines means circa 1 us, and the time moves ==>:

```
      External     timer irq                   thread
       clock        latency                    latency
       event        13585 ns                   39960 ns
         |             ^                         ^
         v             |                         |
         |-------------|                         |
         |-------------+-------------------------|
                       ^                         ^
  ========================================================================
                    [tmr irq]  [dev irq]
  [another thread...^       v..^       v.......][timerlat/ thread]  <-- CPU timeline
  =========================================================================
                    |-------|  |-------|
                            |--^       v-------|
                            |          |       |
                            |          |       + thread_noise: 9909 ns
                            |          +-> irq_noise: 6139 ns
                            +-> irq_noise: 7597 ns
```

IRQ stacktrace

The osnoise/print_stack option is helpful for the cases in which a thread noise causes the major factor for the timer latency, because of preempt or irq disabled. For example:

```
        [root@f32 tracing]# echo 500 > osnoise/stop_tracing_total_us
        [root@f32 tracing]# echo 500 > osnoise/print_stack
        [root@f32 tracing]# echo timerlat > current_tracer
        [root@f32 tracing]# tail -21 per_cpu/cpu7/trace
          insmod-1026    [007] dN.h1..   200.201948: irq_noise: local_timer:236 start 200.201939376 duration 787
          insmod-1026    [007] d..h1..   200.202587: #29800 context    irq timer_latency      1616 ns
          insmod-1026    [007] dN.h2..   200.202598: irq_noise: local_timer:236 start 200.202586162 duration 118
          insmod-1026    [007] dN.h3..   200.202947: irq_noise: local_timer:236 start 200.202939174 duration 731
          insmod-1026    [007] d...3..   200.203444: thread_noise:   insmod:1026 start 200.202586933 duration 83
      timerlat/7-1001    [007] .......   200.203445: #29800 context thread timer_latency    859978 ns
      timerlat/7-1001    [007] ....1..   200.203446: <stack trace>
  => timerlat_irq
  => __hrtimer_run_queues
  => hrtimer_interrupt
  => __sysvec_apic_timer_interrupt
  => asm_call_irq_on_stack
  => sysvec_apic_timer_interrupt
  => asm_sysvec_apic_timer_interrupt
  => delay_tsc
  => dummy_load_1ms_pd_init
  => do_one_initcall
  => do_init_module
  => __do_sys_finit_module
  => do_syscall_64
  => entry_SYSCALL_64_after_hwframe
```

In this case, it is possible to see that the thread added the highest contribution to the *timer latency* and the stack trace, saved during the timerlat IRQ handler, points to a function named dummy_load_1ms_pd_init, which had the following code (on purpose):

```
static int __init dummy_load_1ms_pd_init(void)
{
        preempt_disable();
        mdelay(1);
        preempt_enable();
        return 0;

}
```