Reducing OS jitter due to per-cpu kthreads

This document lists per-CPU kthreads in the Linux kernel and presents options to control their OS jitter. Note that non-per-CPU kthreads are not listed here. To reduce OS jitter from non-per-CPU kthreads, bind them to a "housekeeping" CPU dedicated to such work.

References

- Documentation/core-api/irq/irq-affinity.rst: Binding interrupts to sets of CPUs.
- Documentation/admin-guide/cgroup-v1: Using cgroups to bind tasks to sets of CPUs.
- man taskset: Using the taskset command to bind tasks to sets of CPUs.
- man sched_setaffinity: Using the sched_setaffinity() system call to bind tasks to sets of CPUs.
- /sys/devices/system/cpu/cpuN/online: Control CPU N's hotplug state, writing "0" to offline and "1" to online.
- In order to locate kernel-generated OS jitter on CPU N:

  cd /sys/kernel/debug/tracing echo 1 > max_graph_depth # Increase the "1" for more detail echo function_graph > current_tracer # run workload cat per_cpu/cpuN/trace

kthreads

Name:
: ehca_comp/%u

Purpose:
: Periodically process Infiniband-related work.

To reduce its OS jitter, do any of the following:

1. Don't use eHCA Infiniband hardware, instead choosing hardware that does not require per-CPU kthreads. This will prevent these kthreads from being created in the first place. (This will work for most people, as this hardware, though important, is relatively old and is produced in relatively low unit volumes.)
2. Do all eHCA-Infiniband-related work on other CPUs, including interrupts.
3. Rework the eHCA driver so that its per-CPU kthreads are provisioned only on selected CPUs.

Name:
: irq/%d-%s

Purpose:
: Handle threaded interrupts.

To reduce its OS jitter, do the following:

1. Use irq affinity to force the irq threads to execute on some other CPU.

Name:
: kcmtpd_ctr_%d

Purpose:
: Handle Bluetooth work.

To reduce its OS jitter, do one of the following:

1. Don't use Bluetooth, in which case these kthreads won't be created in the first place.
2. Use irq affinity to force Bluetooth-related interrupts to occur on some other CPU and furthermore initiate all Bluetooth activity on some other CPU.

Name:
: ksoftirqd/%u

Purpose:
: Execute softirq handlers when threaded or when under heavy load.

To reduce its OS jitter, each softirq vector must be handled separately as follows:

TIMER_SOFTIRQ

Do all of the following:

1. To the extent possible, keep the CPU out of the kernel when it is non-idle, for example, by avoiding system calls and by forcing both kernel threads and interrupts to execute elsewhere.
2. Build with CONFIG_HOTPLUG_CPU=y. After boot completes, force the CPU offline, then bring it back online. This forces recurring timers to migrate elsewhere. If you are concerned with multiple CPUs, force them all offline before bringing

the first one back online. Once you have onlined the CPUs in question, do not offline any other CPUs, because doing so could force the timer back onto one of the CPUs in question.

NET_TX_SOFTIRQ and NET_RX_SOFTIRQ

Do all of the following:

1. Force networking interrupts onto other CPUs.
2. Initiate any network I/O on other CPUs.
3. Once your application has started, prevent CPU-hotplug operations from being initiated from tasks that might run on the CPU to be de-jittered. (It is OK to force this CPU offline and then bring it back online before you start your application.)

BLOCK_SOFTIRQ

Do all of the following:

1. Force block-device interrupts onto some other CPU.
2. Initiate any block I/O on other CPUs.
3. Once your application has started, prevent CPU-hotplug operations from being initiated from tasks that might run on the CPU to be de-jittered. (It is OK to force this CPU offline and then bring it back online before you start your application.)

IRQ_POLL_SOFTIRQ

Do all of the following:

1. Force block-device interrupts onto some other CPU.
2. Initiate any block I/O and block-I/O polling on other CPUs.
3. Once your application has started, prevent CPU-hotplug operations from being initiated from tasks that might run on the CPU to be de-jittered. (It is OK to force this CPU offline and then bring it back online before you start your application.)

TASKLET_SOFTIRQ

Do one or more of the following:

1. Avoid use of drivers that use tasklets. (Such drivers will contain calls to things like tasklet_schedule().)
2. Convert all drivers that you must use from tasklets to workqueues.
3. Force interrupts for drivers using tasklets onto other CPUs, and also do I/O involving these drivers on other CPUs.

SCHED_SOFTIRQ

Do all of the following:

1. Avoid sending scheduler IPIs to the CPU to be de-jittered, for example, ensure that at most one runnable kthread is present on that CPU. If a thread that expects to run on the de-jittered CPU awakens, the scheduler will send an IPI that can result in a subsequent SCHED_SOFTIRQ.
2. CONFIG_NO_HZ_FULL=y and ensure that the CPU to be de-jittered is marked as an adaptive-ticks CPU using the "nohz_full=" boot parameter. This reduces the number of scheduler-clock interrupts that the de-jittered CPU receives, minimizing its chances of being selected to do the load balancing work that runs in SCHED_SOFTIRQ context.
3. To the extent possible, keep the CPU out of the kernel when it is non-idle, for example, by avoiding system calls and by forcing both kernel threads and interrupts to execute elsewhere. This further reduces the number of scheduler-clock interrupts received by the de-jittered CPU.

HRTIMER_SOFTIRQ

Do all of the following:

1. To the extent possible, keep the CPU out of the kernel when it is non-idle. For example, avoid system calls and force both kernel threads and interrupts to execute elsewhere.
2. Build with CONFIG_HOTPLUG_CPU=y. Once boot completes, force the CPU offline, then bring it back online. This forces recurring timers to migrate elsewhere. If you are concerned with multiple CPUs, force them all offline before bringing the first one back online. Once you have onlined the CPUs in question, do not offline any other CPUs, because doing so could force the timer back onto one of the CPUs in question.

RCU_SOFTIRQ

Do at least one of the following:

1. Offload callbacks and keep the CPU in either dyntick-idle or adaptive-ticks state by doing all of the following:
    a. CONFIG_NO_HZ_FULL=y and ensure that the CPU to be de-jittered is marked as an adaptive-ticks CPU using the "nohz_full=" boot parameter. Bind the rcuo kthreads to housekeeping CPUs, which can tolerate OS jitter.
    b. To the extent possible, keep the CPU out of the kernel when it is non-idle, for example, by avoiding system calls

and by forcing both kernel threads and interrupts to execute elsewhere.

2. Enable RCU to do its processing remotely via dyntick-idle by doing all of the following:
    a. Build with CONFIG_NO_HZ=y.
    b. Ensure that the CPU goes idle frequently, allowing other CPUs to detect that it has passed through an RCU quiescent state. If the kernel is built with CONFIG_NO_HZ_FULL=y, userspace execution also allows other CPUs to detect that the CPU in question has passed through a quiescent state.
    c. To the extent possible, keep the CPU out of the kernel when it is non-idle, for example, by avoiding system calls and by forcing both kernel threads and interrupts to execute elsewhere.

Name:

kworker/%u:%d%s (cpu, id, priority)

Purpose:

Execute workqueue requests

To reduce its OS jitter, do any of the following:

1. Run your workload at a real-time priority, which will allow preempting the kworker daemons.
2. A given workqueue can be made visible in the sysfs filesystem by passing the WQ_SYSFS to that workqueue's alloc_workqueue(). Such a workqueue can be confined to a given subset of the CPUs using the `/sys/devices/virtual/workqueue/*/cpumask` sysfs files. The set of WQ_SYSFS workqueues can be displayed using "ls /sys/devices/virtual/workqueue". That said, the workqueues maintainer would like to caution people against indiscriminately sprinkling WQ_SYSFS across all the workqueues. The reason for caution is that it is easy to add WQ_SYSFS, but because sysfs is part of the formal user/kernel API, it can be nearly impossible to remove it, even if its addition was a mistake.
3. Do any of the following needed to avoid jitter that your application cannot tolerate:
    a. Build your kernel with CONFIG_SLUB=y rather than CONFIG_SLAB=y, thus avoiding the slab allocator's periodic use of each CPU's workqueues to run its cache_reap() function.
    b. Avoid using oprofile, thus avoiding OS jitter from wq_sync_buffer().
    c. Limit your CPU frequency so that a CPU-frequency governor is not required, possibly enlisting the aid of special heatsinks or other cooling technologies. If done correctly, and if you CPU architecture permits, you should be able to build your kernel with CONFIG_CPU_FREQ=n to avoid the CPU-frequency governor periodically running on each CPU, including cs_dbs_timer() and od_dbs_timer().

        WARNING: Please check your CPU specifications to make sure that this is safe on your particular system.
    d. As of v3.18, Christoph Lameter's on-demand vmstat workers commit prevents OS jitter due to vmstat_update() on CONFIG_SMP=y systems. Before v3.18, is not possible to entirely get rid of the OS jitter, but you can decrease its frequency by writing a large value to /proc/sys/vm/stat_interval. The default value is HZ, for an interval of one second. Of course, larger values will make your virtual-memory statistics update more slowly. Of course, you can also run your workload at a real-time priority, thus preempting vmstat_update(), but if your workload is CPU-bound, this is a bad idea. However, there is an RFC patch from Christoph Lameter (based on an earlier one from Gilad Ben-Yossef) that reduces or even eliminates vmstat overhead for some workloads at https://lore.kernel.org/r/00000140e9dfd6bd-40db3d4f-c1be-434f-8132-7820f81bb586-000000@email.amazonses.com.
    e. If running on high-end powerpc servers, build with CONFIG_PPC_RTAS_DAEMON=n. This prevents the RTAS daemon from running on each CPU every second or so. (This will require editing Kconfig files and will defeat this platform's RAS functionality.) This avoids jitter due to the rtas_event_scan() function. WARNING: Please check your CPU specifications to make sure that this is safe on your particular system.
    f. If running on Cell Processor, build your kernel with CBE_CPUFREQ_SPU_GOVERNOR=n to avoid OS jitter from spu_gov_work(). WARNING: Please check your CPU specifications to make sure that this is safe on your particular system.
    g. If running on PowerMAC, build your kernel with CONFIG_PMAC_RACKMETER=n to disable the CPU-meter, avoiding OS jitter from rackmeter_do_timer().

Name:

rcuc/%u

Purpose:

Execute RCU callbacks in CONFIG_RCU_BOOST=y kernels.

To reduce its OS jitter, do at least one of the following:

1. Build the kernel with CONFIG_PREEMPT=n. This prevents these kthreads from being created in the first place, and also obviates the need for RCU priority boosting. This approach is feasible for workloads that do not require high degrees of responsiveness.
2. Build the kernel with CONFIG_RCU_BOOST=n. This prevents these kthreads from being created in the first place. This approach is feasible only if your workload never requires RCU priority boosting, for example, if you ensure frequent idle time on all CPUs that might execute within the kernel.

3. Build with CONFIG_RCU_NOCB_CPU=y and boot with the rcu_nocbs= boot parameter offloading RCU callbacks from all CPUs susceptible to OS jitter. This approach prevents the rcuc/%u kthreads from having any work to do, so that they are never awakened.
4. Ensure that the CPU never enters the kernel, and, in particular, avoid initiating any CPU hotplug operations on this CPU. This is another way of preventing any callbacks from being queued on the CPU, again preventing the rcuc/%u kthreads from having any work to do.

Name:

rcuop/%d and rcuos/%d

Purpose:

Offload RCU callbacks from the corresponding CPU.

To reduce its OS jitter, do at least one of the following:

1. Use affinity, cgroups, or other mechanism to force these kthreads to execute on some other CPU.
2. Build with CONFIG_RCU_NOCB_CPU=n, which will prevent these kthreads from being created in the first place. However, please note that this will not eliminate OS jitter, but will instead shift it to RCU_SOFTIRQ.