Contribution ID: **178** Type: **not specified**

Making SCHED_DEADLINE safe for kernel kthreads

Monday, 9 September 2019 16:00 (30 minutes)

Dmitry Vyukov's testing work identified some (ab)uses of sched_setattr() that can result in SCHED_DEADLINE tasks starving RCU's kthreads for extended time periods, not millisecond, not seconds, not minutes, not even hours, but days. Given that RCU CPU stall warnings are issued whenever an RCU grace period fails to complete within a few tens of seconds, the system did not suffer silently. Although one could argue that people should avoid abusing sched_setattr(), people are human and humans make mistakes. Responding to simple mistakes with RCU CPU stall warnings is all well and good, but a more severe case could OOM the system, which is a particularly unhelpful error message.

It would be better if the system were capable of operating reasonably despite such abuse. Several approaches have been suggested.

First, sched_setattr() could recognize parameter settings that put kthreads at risk and refuse to honor those settings. This approach of course requires that we identify precisely what combinations of sched_setattr() parameters settings are risky, especially given that there are likely to be parameter settings that are both risky and highly useful.

Second, in theory, RCU could detect this situation and take the "dueling banjos" approach of increasing its priority as needed to get the CPU time that its kthreads need to operate correctly. However, the required amount of CPU time can vary greatly depending on the workload. Furthermore, non-RCU kthreads also need some amount of CPU time, and replicating "dueling banjos" across all such Linux-kernel subsystems seems both wasteful and error-prone. Finally, experience has shown that setting RCU's kthreads to real-time priorities significantly harms performance by increasing context-switch rates.

Third, stress testing could be limited to non-risky regimes, such that kthreads get CPU time every 5-40 seconds, depending on configuration and experience. People needing risky parameter settings could then test the settings that they actually need, and also take responsibility for ensuring that kthreads get the CPU time that they need. (This of course includes per-CPU kthreads!)

Fourth, bandwidth throttling could treat tasks in other scheduling classes as an aggregate group having a reasonable aggregate deadline and CPU budget. This has the advantage of allowing "abusive" testing to proceed, which allows people requiring risky parameter settings to rely on this testing. Additionally, it avoids complex progress checking and priority setting on the part of many kthreads throughout the system. However, if this was an easy choice, the SCHED_DEADLINE developers would likely have selected it. For example, it is necessary to determine what might be a "reasonable" aggregate deadline and CPU budget. Reserving 5% seems quite generous, and RCU's grace-period kthread would optimally like a deadline in the milliseconds, but would do reasonably well with many tens of milliseconds, and absolutely needs a few seconds. However, for CONFIG_RCU_NOCB_CPU=y, the RCU's callback-offload kthreads might well need a full CPU each! (This happens when the CPU being offloaded generates a high rate of callbacks.)

The goal of this proposal is therefore to generate face-to-face discussion, hopefully resulting in a good and sufficient solution to this problem.

I agree to abide by the anti-harassment policy

Yes

Primary author: MCKENNEY, Paul (IBM Linux Technology Center)

Presenter: MCKENNEY, Paul (IBM Linux Technology Center)

Session Classification: Scheduler MC