Contribution ID: **18** Type: **not specified**

Core Scheduling: Taming Hyper-Threads to be secure

Monday, 24 August 2020 09:00 (45 minutes)

The core idea behind core scheduling is to have SMT (Simultaneous Multi Threading) on and make sure that only trusted applications run concurrently on the hardware threads of a core. If there is no group of trusting applications runnable on the core, we need to make sure that remaining hardware threads are idle while applications run in isolation on the core. While doing so, we should also consider the performance aspects of the system. Theoretically it is impossible to reach the same level of performance where all hardware threads are allowed to run any runnable application. But if the performance of core scheduling is worse than or the same as that without SMT, we do not gain anything from this feature other than added complexity in the scheduler. So the idea is to achieve a considerable boost in performance compared to SMT turned off for the majority of production workloads.

This talk is continuation of the core scheduling talk and micro-conference at LPC 2019. We would like to discuss the progress made in the last year and the newly identified use-cases of this feature.

Progress has been made in the performance aspects of core scheduling. Couple of patches addressing the load balancing issues with core scheduling, have improved the performance. And stability issues in v5 have been addressed as well.

One area of criticism was that the patches were not addressing all cases where untrusted tasks can run in parallel. Interrupts are one scenario where the kernel runs on a cpu in parallel with a user task on the sibling. While two user tasks running on the core could be trusted, when an interrupt arrives on one cpu, the situation changes. Kernel starts running in interrupt context and the kernel cannot trust the user task running on the other sibling cpu. A prototype fix has been developed to fix this case. One gap that still exists is the syscall boundary. Addressing the syscall issue would be a big hit to performance, and we would like to discuss possible ways to fix it without hurting performance.

Lastly, we would also like to discuss the APIs for exposing this feature to userland. As of now, we use CPU controller CGroups. During the last LPC, we had discussed this in the presentation, but we had not decided on any final APIs yet. ChromeOS has a prototype which uses prctl(2) to enable the core scheduling feature. We would like to discuss possible approaches suitable for all use cases to use the core scheduling feature.

I agree to abide by the anti-harassment policy

I agree

Primary authors: REMANAN PILLAI, Vineeth (DigitalOcean); DESFOSSEZ, Julien (DigitalOcean)

Co-author: FERNANDES, Joel

Presenters: REMANAN PILLAI, Vineeth (DigitalOcean); DESFOSSEZ, Julien (DigitalOcean); FERNANDES, Joel

Session Classification: LPC Refereed Track

Track Classification: LPC Refereed Track (Closed)