Contribution ID: **114** Type: **not specified**

Core Scheduling: Taming Hyper-Threads to be secure

Monday 9 September 2019 10:45 (45 minutes)

Last couple of years, we have witnessed an onslaught of vulnerabilities in the design and architecture of cpus. It is interesting and surprising to note that the vulnerabilities are mainly targeting the features designed to improve the performance of cpus - most notable being the hyperthreading(smt). While some of the vulnerabilities could be mitigated in software and cpu microcodes, couple of others didn't have any satisfiable mitigation other than making sure that smt is off and every context switch needed to flush the cache to clear the data used by the task that is being switched out. Turning smt off is not a viable alternative to many production scenarios like cloud environment where you lose a considerable amount of computing power by turning off smt. To address this, there have been community efforts to keep smt on while trying to make sure that non-trusting applications are never run concurrently in the hyperthreads of the core, they have been widely called as core scheduling.

This talk is about the development, testing and profiling efforts of core scheduling in the community. There were multiple proof of concepts - while differing in the design, ultimately trying to make sure that only mutually trusted applications run concurrently on the core. We discuss the design, implementation and performance of the POCs. We also discuss the profiling attempts to understand the correctness and performance of the patches - various powerful kernel features that we leveraged to get the most time sensitive data from the kernel to understand the effect of scheduler with the core scheduling feature. We plan to conclude with a brief discussion of the future directions of core scheduling.

The core idea about core scheduling is to have smt on and make sure that only trusted applications run concurrently on siblings of a core. If there are no group of trusting applications runnable on the core, we need to make sure that remaining siblings should idle while applications run in isolation on the core. This should also consider the performance aspects of the system. Theoretically it is impossible to reach the same level of performance where the cores are allowed to any runnable applications. But if the performance of core scheduling is worse than or same as the smt off situation, we do not gain anything from this feature other than the added complexity in the scheduler. So the idea is to achieve a considerable boost in performance compared to smt-off for the majority of production workloads.

Security boundary is another aspect of critical importance in core scheduling. What should be considered as a trust boundary? Should it be at the user/group level, process level or thread level? Should kernel be considered trusty by applications or vice-versa? With virtualization and nested virtualization in picture, this gets even more complicated. But answers to most of these questions are environment and workload dependent and hence these are implemented as policies rather than hardcoding in the code. And then arises the question - how the policies should be implemented? Kernel has a variety of mechanisms to implement these kind of policies and the proof of concepts posted upstream mainly uses cgroups. This talk also discusses other viable options for implementing the policies.

I agree to abide by the anti-harassment policy

Yes

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

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

Session Classification: LPC Refereed Track