Contribution ID: **88** Type: **not specified**

Hazard pointers in Linux kernel

Wednesday, 18 September 2024 17:45 (45 minutes)

Reference counting in Linux kernel is often implemented using conditional atomic increment/decrement operations on a counter. These atomic operations can become a scalability bottleneck with increasing numbers of CPUs. The RCURef patch series 1 and Nginx refcount scalability issues 2 are recent examples where the refcount bottleneck significantly degraded application throughput.

Per-CPU refcounting 2 avoids memory contention by distributing refcount operations across CPUs. However, this is not free: on a 64-bit system, the per-object space overhead for per-CPU refcounting is 72 bytes plus eight additional bytes per CPU.

The hazard-pointers technique 3 dynamically distributes refcounting, and is especially useful in cases where reference counters are acquired conditionally, for example, via using kref_get_unless_zero(). It can greatly improve scalability, resulting in userspace use [4,5] and also inclusion into the C++26 standard 6.

Moreover, hazard pointers can be significantly more space-efficient than per-CPU refcounting. For large numbers of objects on a 64-bit system, only 16 bytes is required per object, which is a great savings compared to 72 bytes plus eight more bytes per CPU for per-CPU refcounting.

Of course, there are advantages to per-CPU refcounting, for example, given large numbers of tasks, each having a long-lived reference to one of a small number of objects. On a 64-bit system, the current hazard-pointers prototype incurs a per-task space overhead of 128 bytes. In contrast, per-CPU refcounting incurs no per-task space overhead whatsoever.

Thus, hazard pointers is most likely to be the right tool for the job in cases of high contention on reference counters protecting large numbers of objects.

In this talk, we will present the design 7 and implementation of hazard pointers, including Linux-kernel-specific challenges. We will also present examples of hazard-pointers usage, performance results and comparison to other techniques, including RCU and Sleepable-RCU.

Primary authors: FENG, Boqun (Microsoft); UPADHYAY, Neeraj (AMD); MCKENNEY, Paul (Meta)

Presenters: FENG, Boqun (Microsoft); UPADHYAY, Neeraj (AMD); MCKENNEY, Paul (Meta)

Session Classification: LPC Refereed Track

Track Classification: LPC Refereed Track