IPI Deferral

Valentin Schneider <vschneid@redhat.com>

LPC 2024

Red Hat

Context

- CPU Isolation, NOHZ_FULL, RCU_NOCB…
  - Single userspace task on **isolated** CPU
  - No (voluntary) kernel entry
- Some **IPIs** still end up hitting the **isolated** CPU
  - `text_poke_sync()` (**static keys** & friends)
  - `vunmap()`'s `flush_tlb_kernel_range()` (freeing / unmapping)
- Deferral concept: IPI **doesn't concern userspace**?
  - Don't send it
  - Execute related callback ASAP upon kernel entry

Progress so far

- **Tracepoints** for IPIs & remote callbacks (v6.4)
  - trace_ipi_send_{cpu,cpumask}
  - trace_csd_queue_cpu
  - trace_csd_function_{entry, exit}
- Free extra: use ftrace synthetic events + histograms to compute **CSD delivery time** [1]
- Ftrace tweaks to **filter by cpumask** (v6.6)

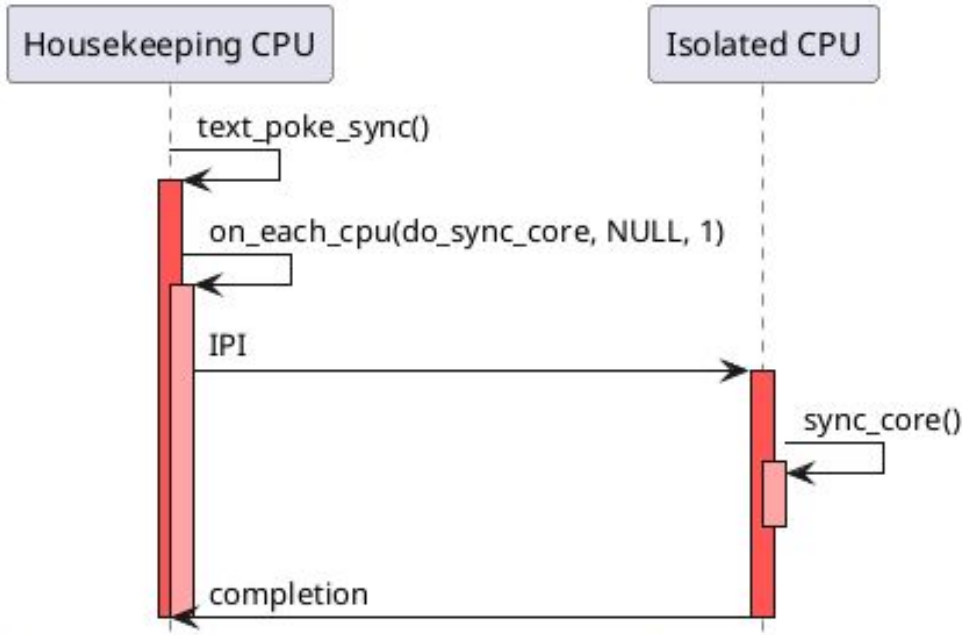
```
trace-cmd record -e 'sched_switch' -f "CPU & CPUS{$ISOLATED_CPUS}"        \
                 -e 'sched_wakeup' -f "target_cpu & CPUS{$ISOLATED_CPUS}"  \
                 -e 'ipi_send_cpu' -f "cpu & CPUS{$ISOLATED_CPUS}"         \
                 -e 'ipi_send_cpumask' -f "cpumask & CPUS{$ISOLATED_CPUS}" \
                 hackbench
```


[1]: https://lore.kernel.org/lkml/xhsmh4jn8y8vt.mognet@vschneid.remote.csb/

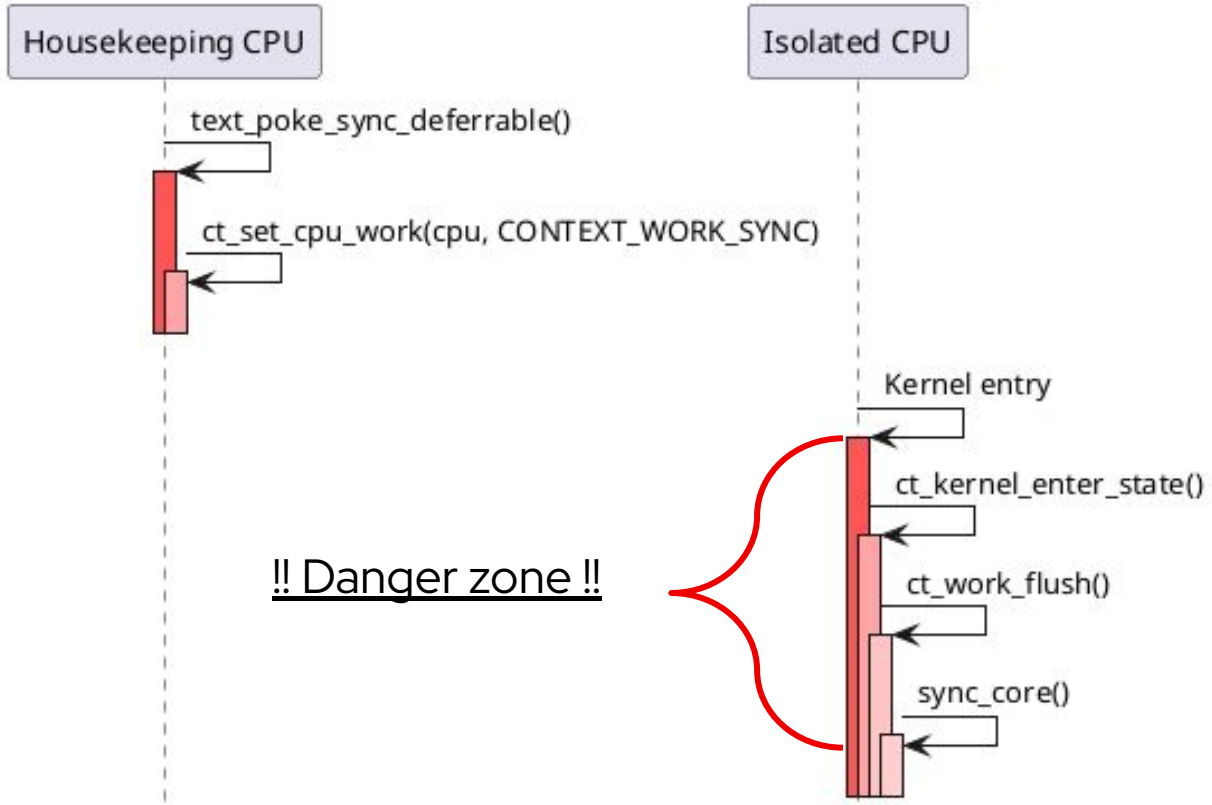
Deferral vs early entry code

- Deferred operation is /!\ not **immediately** executed upon kernel entry /!\

Current approach



Deferral



Instruction patching vs early entry code

- Danger zone => static key in early entry text

- Early kernel entry≈**noinstr**
- Leverage **objtool**, warn about static keys used in .**noinstr** regions

- Some non-issue ones, **__ro_after_init**
- Two problematic keys stand out:
  - **mds_idle_clear**; x86 mitigation, flipped at SMT hotplug
  - **__sched_clock_stable**; flipped by **mark_tsc_unstable()**, called by a lot of **__init** functions but also runtime ones (e.g. loading KVM module)
- Can we just let the IPI through and blame the user?

TLB flush vs early entry code

- Danger zone => accessing vmap'd addresses in early entry code

- CONFIG_VMAP_STACK
  - No in-flight stack changes between fork() and exit()/put_task_struct()
  - Not a problem? (cue for one of you to disagree)

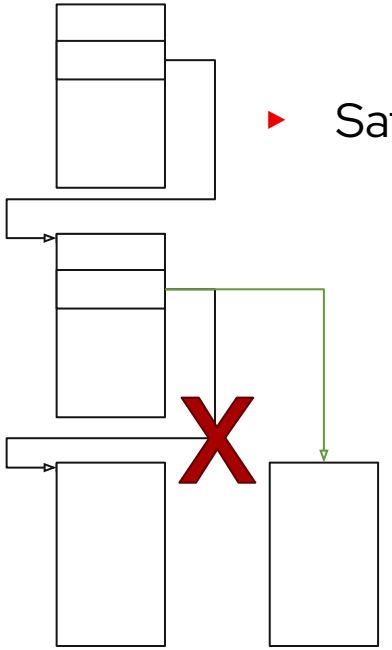
- Something to track vmap'd addresses in .noinstr regions?

TLB flush vs x86 being annoying

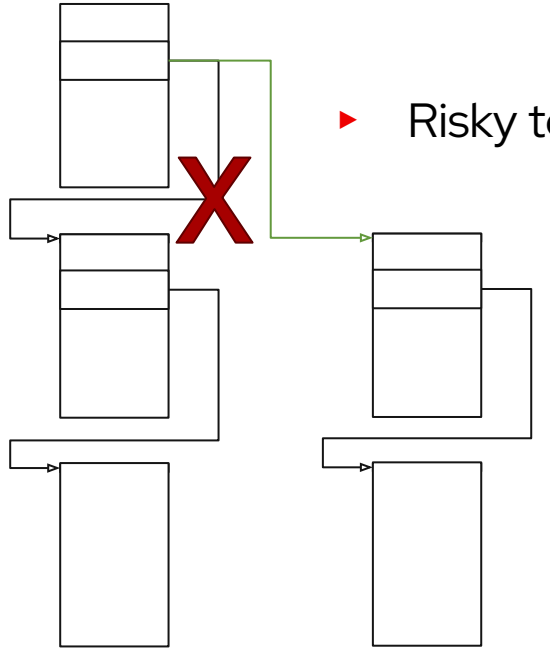
- Paging structure cache, Intel SDM volume 3, 4.10.3
  - CPU can cache “any part of the paging hierarchy”
  - Cached entries can be accessed speculatively

- Pointed out by Nadav Amit & Dave Hansen [1]

- **vunmap()** does not free page-table-pages, which negates the risk… **For now**



- Safe-ish to defer



- Risky to defer

[1]: https://lore.kernel.org/all/188AEA79-10E6-4DFF-86F4-FE624FD1880F@vmware.com/

Red Hat

Thank you!


linkedin.com/company/red-hat


youtube.com/user/RedHatVideos


facebook.com/redhatinc


twitter.com/RedHat

Red Hat