

Two constraints on kernel tracers

1

Tracers are core code: they should be **built-in code only**, they have no justification for being kernel modules.

Paraphrasing core kernel developers

2

“If people need two tracing models, then the other one will be out of tree. It’s that simple.

So as far as I’m concerned, this discussion is not a discussion. Either there’s a way to merge things incrementally with SHARED infrastructure, or there isn’t.”

Linus Torvalds, LKML, July 2025

https://lore.kernel.org/lkml/CAHk-=wiT9Cz+EbbuKozqiu7DnZQ7ftAWSmGf-xy_CdhJPCsNSg@mail.gmail.com/

Combined: a collaboration inhibitor

- It is more efficient to work **outside of common tracer code** if that code cannot be used by an out-of-tree kernel tracer.
- It actively **prevents** incrementally contributing LTTng facilities into common tracer code **while keeping those facilities usable by the LTTng tracer**.
- Making perf, ftrace or eBPF use those facilities may justify integrating those contributions, but **does not justify exporting the required symbols** to GPL modules.

What is doable is an *all-or-nothing* tracer ABI swap, never incremental.

20 years of Linux kernel contributions

Atomic and local atomic headers

- Generalization (~2007)

asm goto / static keys

- Active involvement in the original desiderata leading to GCC support of asm goto

Tracing subsystem

- Tracepoint implementation
- Faultable system call tracepoints
- Participation in the SFrame kernel vs userspace integration effort (stack walk)
- Reviewer

RCU subsystem

- Membarrier
- Hazard pointers (ongoing)
- Reviewer

Scheduler

- Restartable Sequences
- Concurrency IDs

Memory management

- Hierarchical per-CPU RSS accounting (ongoing)