



# Compiler-Based Context Analysis

## Compile-Time Lock Safety for the Linux Kernel

---

**Marco Elver**  
Google

Bart Van Assche  
Google

Linux Plumbers Conference · October 2026



LINUX  
PLUMBERS  
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

# **What is Context Analysis?**

---

# The Problem: Kernel Synchronization at Scale

- The kernel has a vast number of synchronization primitives; locking remains one of the hardest areas of kernel development.
- Bugs range from deadlocks to data races and **memory-safety bugs** (e.g. racy use-after-free from missing locks or RCU misuse).

## Lockdep & KCSAN

### Dynamic analysis.

- Lockdep: deadlocks, ordering.
- KCSAN: data races.
- **Coverage:** only sees paths tests or fuzzers execute.

## Sparse `__context__`

### Static, but shallow.

- 1,700+ warnings (defconfig), >5,200 (allmodconfig) — mostly false positives.
- **Ingo Molnar (v7.0 pull):** *“actual locking bugs found ... is also rather ... sparse: **only 3 such commits in the last 3 years.**”*

## Rust for Linux

### Type-safe abstractions.

- Encodes lock guards in types (Mutex<T>).
- **Coverage:** Does not benefit C, which is still the majority of the kernel.

**Goal:** catch locking bugs at compile time — before they are committed — in code that opts in.

# What Context Analysis Is

- **A C language extension:** a *capability system* layered on C's type system. Code acquires and releases user-definable “**context locks**”; data and functions declare which contexts they require.
- **Trade-off: a stricter dialect of kernel C.** Some valid but convoluted patterns (e.g. uncorrelated conditional acquire/release) are rejected.
- **Statically checked by compiler:** Based on Clang's Thread Safety Analysis (`-Wthread-safety`, **Clang 23+**);<sup>1</sup> per-function, at compile time → **zero runtime cost**.
- **vs. Rust:** Rust encodes locking rules in ownership types (`Mutex<T>`); Context Analysis can incrementally be enabled and brings compile-time checking of locking rules to existing C.
- **Upstream since v7.0:** Merged in v7.0, but several improvements since then. Recommend working with at least v7.3 and Clang 23 (for stable kernel use).

---

<sup>1</sup>Widely used in large C++ codebases (e.g. Chromium, Abseil) but making it work for C and the kernel needed new Clang features.

# Quick Usage Guide & Supported Primitives

## Kconfig & Makefile Opt-In

```
# Kconfig (Clang 23+; default y)
CONFIG_WARN_CONTEXT_ANALYSIS=y
# Tree-wide (experimental)
CONFIG_WARN_CONTEXT_ANALYSIS_ALL=y
```

```
# Makefile: per-object or directory
CONTEXT_ANALYSIS_mymodule.o := y
CONTEXT_ANALYSIS := y
```

## Annotation Cheat Sheet

|                     |                                                                                       |
|---------------------|---------------------------------------------------------------------------------------|
| <b>Data / Ptr</b>   | <b><code>__guarded_by</code>, <code>__pt_guarded_by</code></b>                        |
| <b>Precondition</b> | <b><code>__must_hold[_shared]</code>, <code>__must_not_hold</code></b>                |
| <b>Transition</b>   | <b><code>__acquires</code>, <code>__releases</code>, <code>__cond_acquires</code></b> |
| <b>Escape</b>       | <b><code>context_unsafe(...)</code>, <code>__context_unsafe</code></b>                |
| <b>Assertion</b>    | <b><code>lockdep_assert_held()</code></b><br>(runtime check; compiler assumes held)   |

## Zero-warnings policy

Once enabled for a subsystem, new warnings are regressions.

## Supported Synchronization Primitives (15+)

`raw_spinlock_t`, `spinlock_t`, `rwlock_t`, `mutex`, `rt_mutex`, `seqlock_t`, `bit_spinlock`, `RCU` (`rcu`, `_sched`, `_bh`), `SRCU`, `rw_semaphore`, `local_lock_t`/`local_trylock_t`, `ww_mutex`/`ww_acquire_ctx`, `kref` helpers.

## Real Code: kernel/kcov.c (Enabled Since v7.0)

```
1  struct kcov {
2      /* The lock protects mode, size, area and t. */
3      spinlock_t      lock;
4      enum kcov_mode   mode __guarded_by(&lock);
5      unsigned int     size __guarded_by(&lock);
6      void             *area __guarded_by(&lock);
7      struct task_struct *t __guarded_by(&lock);
8      ...
9  };
10
11 static void kcov_reset(struct kcov *kcov) __must_hold(&kcov->lock)
12 {
13     kcov->t = NULL;
14     kcov->mode = KCOV_MODE_INIT;
15     ...
16 }
```

- The comment “*The lock protects mode, size, area and t*” becomes a **compiler-enforced invariant**.

# Compiler Diagnostics in Action (kcov\_mmap( ))

Clang 23.1.0 output (CONFIG\_WARN\_CONTEXT\_ANALYSIS=y, CONFIG\_WERROR=y) — **Bug A is real; Bugs B & C injected:**

## Bug A (real, fixed in 6556fde265a7, v7.0): Guarded field read after unlock (TOCTOU)

```
spin_lock_irqsave(&kcov->lock, flags); ...
+ area = kcov->area;
spin_unlock_irqrestore(&kcov->lock, flags); ...
- page = vmalloc_to_page(kcov->area + off);
+ page = vmalloc_to_page(area + off);
(original code) kernel/kcov.c:526:32: error: reading variable 'area' requires holding spinlock '&kcov::lock'
```

## Bug B (injected): Error path returns with lock still held

```
spin_lock_irqsave(&kcov->lock, flags);
+ if (!kcov->size) return -EINVAL;
kernel/kcov.c:539:1: error: spinlock 'vma->vm_file->private_data->lock' is not held on every path through here
kernel/kcov.c:515:2: note: spinlock acquired here
```

## Bug C (injected): Call \_\_must\_hold function without holding lock

```
spin_unlock_irqrestore(&kcov->lock, flags);
+ kcov_reset(kcov);
kernel/kcov.c:524:2: error: calling function 'kcov_reset' requires holding spinlock 'vma->vm_file->private_data->lock'
```

**Where is it used today?**

---

## Adoption Timeline (v7.0 → v7.3-rc4)

| Release  | Newly Enabled Subsystems / Files                                                                                                   | Contributors                                                                     |
|----------|------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------|
| v7.0     | mm/kfence, kcov, kcsan, lib/stackdepot, lib/rhashtable, security/tomoyo, crypto/, sched (core.c, fair.c)                           | Marco Elver (initial series)                                                     |
| v7.1     | kernel/locking (mutex, rtmutex_api, ww_rt_mutex, rwsem), kernel/futex                                                              | Peter Zijlstra                                                                   |
| v7.2     | block/, drivers/ata; aspeed-lpc-snoop (with bug fix)                                                                               | Bart Van Assche, Karthikeyan KS                                                  |
| v7.3-rc4 | arch/s390 + drivers/s390 (16 dirs), Bluetooth (5 dirs), nvme/host, SCSI core (23 objs + device_handler), UFS, lib/raid (raid6/xor) | Heiko Carstens, Pauli Virtanen, Nilay Shroff, Bart Van Assche, Christoph Hellwig |

- **Totals (as of v7.3-rc4): 34 directory-level + 34 per-object** enables across ~45 Makefiles.
- Started in core kernel subsystems and expanded to major storage/network drivers and a full architecture (s390) within 3 releases (**16 enabling commits in v7.3 alone**).

# Annotation Growth (v6.19 → v7.3-rc4)

## Tree-Wide Annotation Counts

| Annotation                                   | v6.19     | v7.3-rc4         |
|----------------------------------------------|-----------|------------------|
| <b>__guarded_by</b> / <b>__pt_guarded_by</b> | 0         | <b>88 / 17</b>   |
| <b>__must_hold</b> (+shared)                 | 240       | <b>471</b> (+65) |
| <b>__cond_acquires</b> (+shared)             | 8         | <b>96</b> (+16)  |
| <b>__acquires</b> / <b>__releases</b>        | 496 / 578 | <b>614 / 688</b> |

## Momentum Across Releases

- Commits mentioning context analysis:  
46 (7.0) → 14 (7.1) → 10 (7.2) → **50** (7.3-rc4)

## From Abandoned to Enforced Contracts

Legacy Sparse **\_\_acquires**, **\_\_releases**, and **\_\_must\_hold** annotations used to be largely decorative. Clang now enforces them — including verifying that a function's **definition** actually satisfies its annotations (e.g. actually acquires what **\_\_acquires** promises).

# Bugs Found with Tree-Wide Analysis

- **Bart Van Assche** concurrently annotated the *entire* kernel tree:  
[PATCH 00/62] Bug fixes and refactoring patches related to locking (Feb 2026, 160 files, +695/−443).  
(More on this approach later.)

## Quote from Cover Letter & Bart Van Assche (May 2026)

*“Annotating all source files in the kernel tree with lock context annotations led to the discovery of a significant number of locking bugs.” — ≈40 locking bugs, mostly in error paths.*

- **Breakdown of the 62-patch series** (DRM, net, btrfs, f2fs, cachefiles, PCI, SCSI, KVM, input, hwmon):
  - ~24 explicit bug fixes & ~12 “easier to analyze” control-flow refactors.
  - Tree-wide mutex\_lock\_interruptible() return-check cleanups.
  - ~25 patches making dma\_resv\_lock() return-value checks mandatory.
- **Dominant bug class:** **error paths** — the exact code dynamic tests and fuzzers rarely reach.

# Prominent Fixes & Verification Workflow

| Commit / Subsystem                           | Bug since                 | Bug Class                                                           |
|----------------------------------------------|---------------------------|---------------------------------------------------------------------|
| aa0ae1c35f7b <b>ata: libata</b> <sup>†</sup> | <b>v2.6.37</b> (2010)     | ata_exec_internal() drops & reacquires eh_mutex caller doesn't hold |
| 2b31e86387e6 <b>drbd</b>                     | <b>v4.5</b>               | rcu_read_unlock() without matching rcu_read_lock()                  |
| 7adaaee5edd3 <b>synaptics-rmi4</b>           | <b>v4.9</b> (Cc: stable)  | Error path unlocks a mutex it never locked                          |
| 72c6df8f284b <b>wifi: wlcore</b>             | <b>v4.19</b>              | Unlocks a mutex that is not held                                    |
| 0ded1f36ba40 <b>ice</b>                      | <b>v5.6</b>               | goto skips mutex_lock() but still unlocks on exit                   |
| b2840e33127c <b>btrfs</b>                    | <b>v6.17</b> (Cc: stable) | Missing rcu_read_unlock() on loop error exit                        |

All commit messages state “detected by the Clang thread-safety analyzer/checker”, except <sup>†</sup>: found while annotating eh\_mutex for the drivers/ata enablement (not explicitly credited).

## Workflow: Fix → Annotate → Enforce (bugs found by other means, locked in by the analysis)

- **aspeed-lpc-snoop** (1acef6d85bfd, since v4.19): kfifo data race → usercopy overflow BUG; fixed with spinlock + **\_\_guarded\_by** + CONTEXT\_ANALYSIS enabled for the driver.
- **Bluetooth L2CAP**: syzbot race fixed (2b66c83ff175); b80de2cbb1c8 added **\_\_guarded\_by(&proto\_lock)** + **\_\_must\_hold** to verify.

**How did we get here?**

---

# History & Prior Attempts

- **2008–2012 (GCC annotalysis):** Le-Chun Wu et al. (Google). Terminated in 2012 (GIMPLE lowering artifacts, stopped tracking trunk, “estimated number of users is zero”); ported to Clang (Caitlin Sadowski, DeLesley Hutchins, 2011+).
- **2016–2019 (LKML):** Sparse guarded\_by RFC (2016); retracted md patch (2019). Jann Horn summarized the kernel hurdles: init/free paths, RCU-or-mutex readers, conditional locking via return values.
- **2019–2020 (ClangBuiltLinux — Lukas Bulwahn, Himanshu Jha):**

| Primitive | Annotations | Silenced | Warnings (defconfig) | Top False-Positive Classes              |
|-----------|-------------|----------|----------------------|-----------------------------------------|
| mutex     | 208         | 98       | ~150                 | Conditional locks (56), aliasing (21),  |
| spinlock  | 77          | 108      | 281                  | return values (4), special patterns (4) |

- “We actually never annotated variables with `__guarded_by(<lock>)`” (C struct members unsupported before 2024).
- “Patches with `__no_thread_safety_analysis` are not going to be accepted...” — conclusion: “Not the most promising results though...” (171 tracker issues still open)

## What Changed in 2024–2026

1. Clang gained C struct-member guarded\_by (Jun 2024) · 2. We modified Clang to fix aliasing, reentrancy, multi-lock guards, fn pointers · 3. Per-subsystem opt-in · 4. Dropped Sparse compatibility.

# History Timeline (2024 → 2026)

| When         | Milestone                                                                                                              |
|--------------|------------------------------------------------------------------------------------------------------------------------|
| Jun 2024     | Clang supports <b>annotating struct fields in C</b> (LLVM #20777), removing the biggest blocker.                       |
| Sep 2024     | Marco starts design investigation and prototype (incremental opt-in).                                                  |
| Feb 6, 2025  | <b>RFC v1</b> (“Capability Analysis”); Bart posts <b>tree-wide mutex-only RFC the same day</b> .                       |
| Mar 2025     | <b>v2</b> , LWN article, Clang requirements list. PeterZ: <i>“this is all awesome”</i> / <i>“falls over”</i> on sched. |
| Apr–Sep 2025 | Upstream Clang work: reentrant capabilities, basic alias analysis.                                                     |
| Sep 18, 2025 | <b>v3</b> posted; Linus Torvalds objects to “capability” naming, endorses the concept.                                 |
| Nov 2025     | <b>v4</b> : renamed to <b>“Context Analysis”</b> (“context guard”).                                                    |
| Dec 19, 2025 | <b>v5</b> : “context lock”, full <linux/cleanup.h> scoped guards (36 patches, +2523/–609).                             |
| Jan–Feb 2026 | Applied to tip/locking/core (Jan 5); merged by Linus for <b>Linux 7.0</b> (Feb 10, 0923fd0419a1).                      |
| May 2026     | Minimum compiler raised to <b>Clang 23</b> for new kernel-requested features.                                          |

# Two Strategies: Breadth vs. Depth

## A. Tree-Wide, All-or-Nothing

- **Bart's Feb 2025 RFC:** mutex-only, *"a massive patch touching over 800 files"* (LWN).
- **Risk:** high upfront consensus needed; powerful tools have died before because of annotation and maintenance burden (e.g. KTSAN).
- **Strength:** broad bug hunting across drivers.

## B. Selective & Incremental (Merged)

- **This series:** 15+ primitives, opt-in per subsystem (`CONTEXT_ANALYSIS := y`).
- **Philosophy:** many small, low-risk steps with zero warnings beat a single high-risk tree-wide conversion.
- **Strength:** clean merge path + high signal-to-noise ratio for maintainers.

## Two Complementary Sides

**Strategy B** provided maintainable infrastructure which was merged, but requires patience to realize full impact; **Strategy A** (whole-tree) found  $\approx 40$  bugs and proved it works.

# Making Clang fit the Kernel

**Why kernel C is harder than C++** (few RAII-wrapped mutex types): bespoke C dialect, 15+ primitives plus IRQ/preempt disabling as implicit locking, pervasive pointer passing, vtable-style ops structs. [Post-RFC Clang requirements list](#) made it clear changes are needed.

| # | Kernel Requirement                                                     | Upstream Clang Change                 | LLVM |
|---|------------------------------------------------------------------------|---------------------------------------|------|
| 1 | Passing pointers to guarded data to helpers                            | -Wthread-safety-pointer (#127396)     | 21   |
| 2 | Re-entrant contexts (RCU, preempt_disable)                             | reentrant_capability (#137133)        | 21   |
| 3 | Locks reached via local pointers ( <i>"Fix the analyzer instead"</i> ) | Basic <b>alias analysis</b> (#142955) | 22   |
| 4 | Sharded locks: write needs <i>all</i> , read needs <i>any</i> (PeterZ) | Multi-arg guarded_by (#186838)        | 23   |
| 5 | Ops structs / callbacks (kill_sb, blk_holder_ops; Christoph, Bart)     | Attributes on fn pointers (#191187)   | 23   |
| 6 | int x __guarded_by(&t->lock) in C structs                              | Implicit member access fix (#194457)  | 23   |

- **28 upstream LLVM TSA commits** (Feb 2025 – Aug 2026), including **7 major features** + backports to release/23.x.

Aaron Ballman (Clang maintainer): *"the Linux kernel use case is an important one"*.

- **Kernel-side macro bridges** where compiler changes weren't practical:
  - Locks inside returned pointers: **\_\_acquire\_ret(call, expr)** · Conditional locking with error codes: **\_\_cond\_acquires(ret, x)**
- **Clang 22 → 23 bump**: stacking macro workarounds in kernel headers *"doesn't scale"*.

# Kernel vs. Clang

## <linux/cleanup.h> & Init Guards

- Scoped `guard()` / `scoped_guard()` supported cleanly since v5.
- Scoped **lock init guards** (e.g. `guard(spinlock_init)`) contain **guarded object** initialization (at PeterZ's request) — large init functions can be analyzed (unlike C++ ctors).

## Escape Hatches (use sparingly!)

- **`__context_unsafe("comment")`**: function attribute to disable analysis.
- **`context_unsafe(expr)`**: wrap arbitrary expression (disables warnings for `expr`).
- `data_race(expr)` for intentional data races (disables warnings for `expr`).
- Header suppressions; **`__no_context_analysis`** reserved for context lock implementations.

## Unexpected False Positives

- `per_cpu()` / `RELOC_HIDE` hiding pointer identity (fixed by simplification).
- Arrays of locks (LLVM #148551).
- Statement expressions in try-lock conditions (LLVM #209330).

## Compile-Time Cost

Negligible build impact: **+0.08% mean** compile-time delta on a Linux 7.2 benchmark (LLVM #212615).

# The Naming Saga

Thread Safety Analysis (Clang) → Capability Analysis (v1-v3) → **Context Analysis** (v4+)

## Linus Torvalds on v3 (Sep 18, 2025)

*“So this looks really interesting, but I absolutely **\*hate\*** the new ‘capability’ name. [...] We have existing and traditional — and very very different — meaning of ‘capabilities’ in the kernel...”*

*“Sparse called it ‘context analysis’ [...] at least the ‘context’ part of the name is I think somewhat sane. Because it’s about making decisions based on the context the code runs in. [...]”*

*“Please? **Because other than this naming issue, I think this really is a good idea.**”*

- **Follow-on naming puzzle:** what do you call the object that establishes a context?
- “Context guard” in v4 clashed with `cleanup.h`’s `guard()` → v5 settled on **“context lock”**.

**Where Next?**

---

## Towards Tree-Wide Coverage

1. Incrementally enable more subsystems (`CONTEXT_ANALYSIS := y`).
2. Add selective opt-outs (`CONTEXT_ANALYSIS := n`) where `WARN_CONTEXT_ANALYSIS_ALL` complains.
3. Once `WARN_CONTEXT_ANALYSIS_ALL` builds cleanly, make it the default.
4. Remove redundant `CONTEXT_ANALYSIS := y` lines.

*“Even if we get e.g. 80% coverage, we’re able to prevent several classes of locking bugs from ever entering the kernel in 80% of the code. I’d take that any day over 0% coverage.”*

## Open Questions

- **More contexts:** Preemptible and IRQ contexts (`PREEMPT_LOCK`, `IRQ_LOCK?`) and annotating the explicit (`preempt_disable/enable()`, ...) and implicit transitions (~2 months work).
- **Open compiler problem:** conditional locking by return-value range (`ERR_PTR` / negative `errno`). Needs range/expression matching (~2–3 months LLVM work).
- **AI patching & review (syzbot, Sashiko):** annotations are locking rules the compiler has validated; can agents use them effectively to understand locking code more reliably?

# Takeaways & Call to Action

## 1. **Available since v7.0:**

Context analysis statically checks locking rules, zero runtime cost, negligible compile-time overhead, has already found decade-old kernel locking bugs.

## 2. **Enable it for your subsystem:**

Add `CONTEXT_ANALYSIS := y` (and use Clang 23+), annotate important data with `__guarded_by` and helpers with `__must_hold`, and keep the build warning-free.

## 3. **Feedback welcome:**

Without strong feedback, we wouldn't have gotten this far — more feedback will help keep improving context analysis.

## Documentation & Resources

- Kernel docs: <https://docs.kernel.org/dev-tools/context-analysis.html>
- Clang docs: <https://clang.llvm.org/docs/ThreadSafetyAnalysis.html>