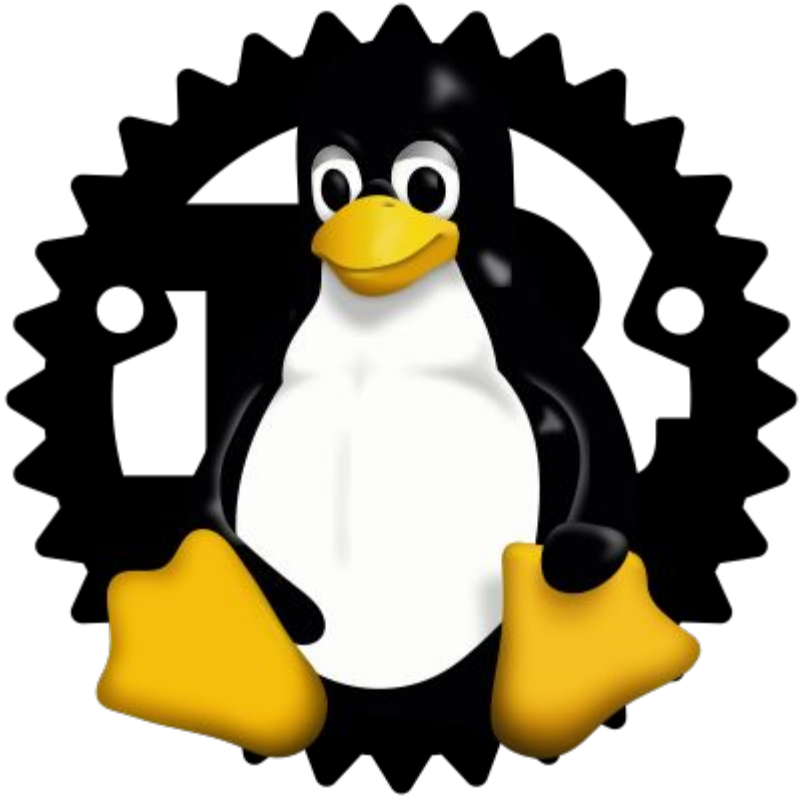


Rust for Linux

Miguel Ojeda

ojeda@kernel.org

Latest news

Linux in perspective

One can think of Linux as a project with lots of subprojects.

In the v6.10 cycle (9 weeks), around:

- 13,312 non-merge commits were pulled.
- 1,918 developers contributed (242 made their first contribution).
- 203 companies supported work in the kernel.
- 1,800 unique kernel maintainers are listed.

It is everywhere, e.g. 4 billion Android devices, 100+ million servers.

— https://lwn.net/Articles/981559/
— Greg KH at KubeCon + CloudNativeCon Europe 2025

Rust for Linux in perspective

~2,500 patch series.

~300 submitters.

Plus ~700 PRs early on in GitHub.

Plus the work in other projects, e.g. upstream Rust.

~100 PRs to Linus Torvalds.

~20 Rust-related entries in `MAINTAINERS`.

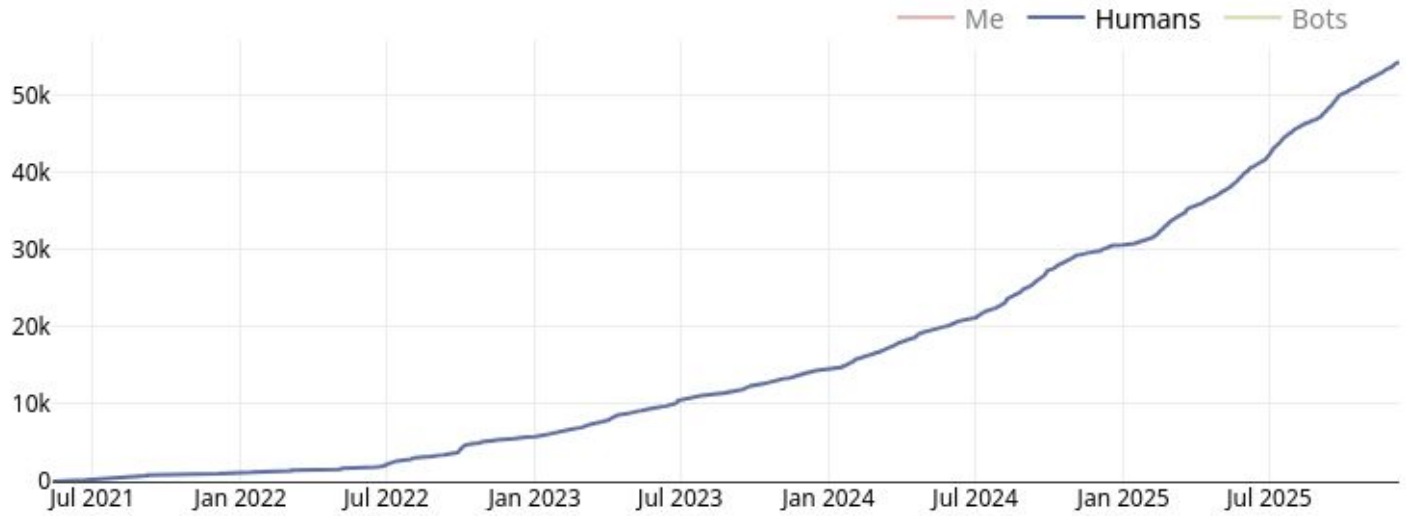
~30 maintainers/reviewers.

~7 active core team members.

Growing Community

~54k messages sent in the Zulip instance (i.e. chat).

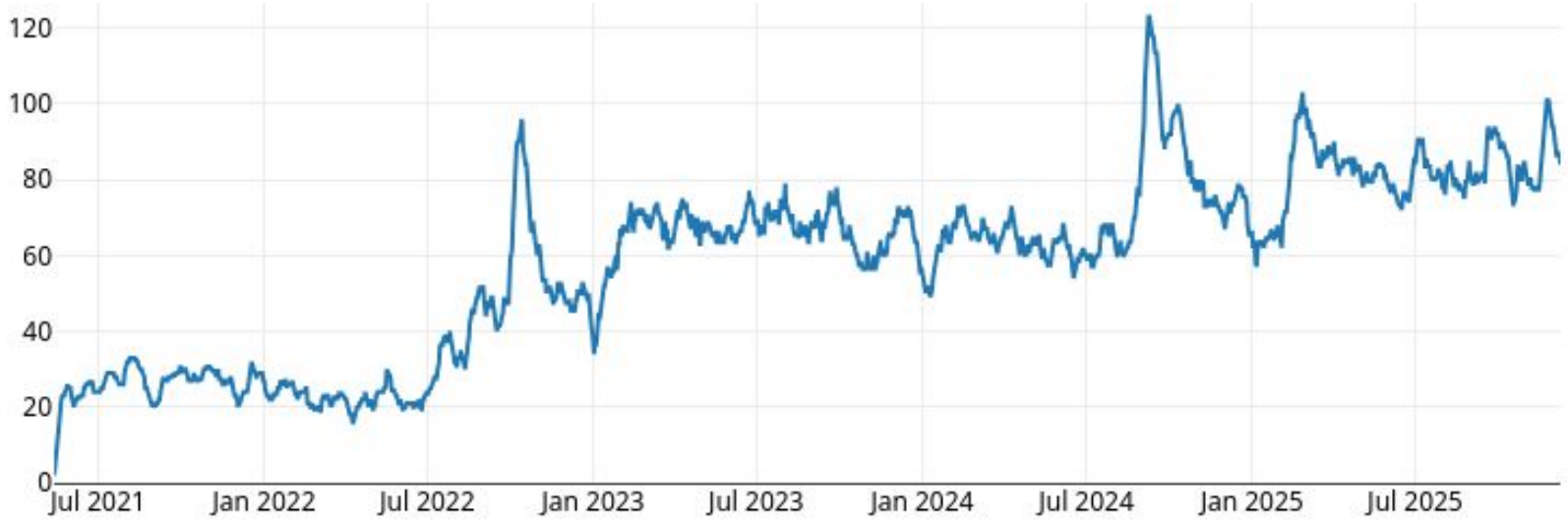
From ~30k a year ago.



— https://rust-for-linux.zulipchat.com/stats

Growing Community

Active users — spikes are due to news/events.



— https://rust-for-linux.zulipchat.com/stats

Who uses Rust for Linux?

Upstreamed users:

- AMCC QT2025 PHY driver.
- Android Binder driver.
- ASIX PHY driver (first “Rust reference driver”).
- DRM Panic QR code generator.
- Nova GPU driver.
- Null Block driver.
- Tyr GPU driver.

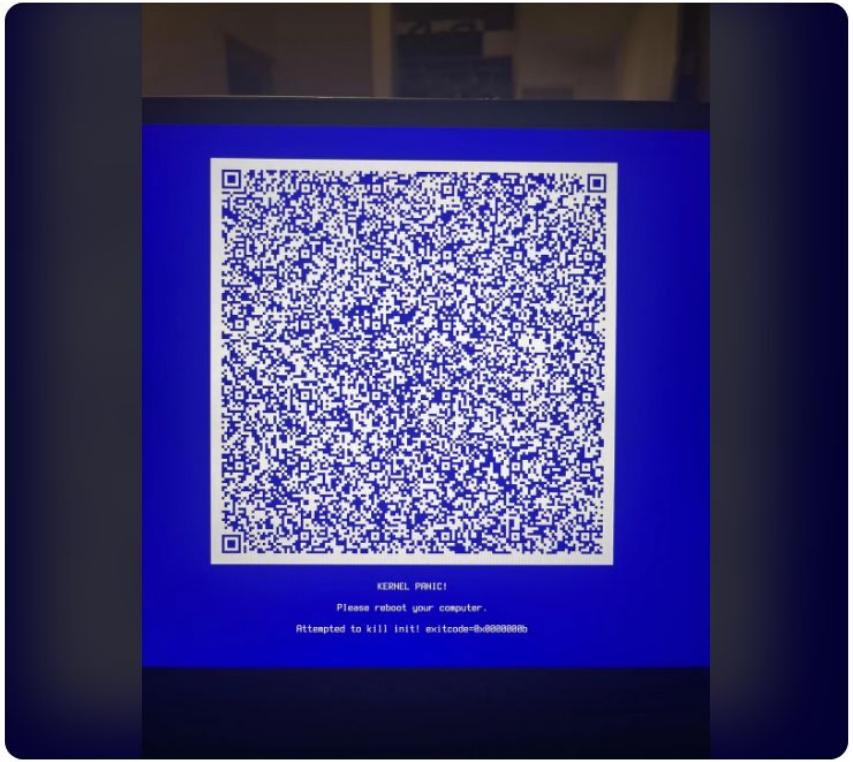
— https://rust-for-linux.com‘s “Users — in mainline” section in the menu
— https://rust-for-linux.com/rust-reference-drivers

Who uses Rust for Linux?

Upstreamed users:

- AMCC QT2025 PHY driver.
- Android Binder driver.
- ASIX PHY driver (first “Rust reference driver”).
- **DRM Panic QR code generator**.
- Nova GPU driver.
- Null Block driver.
- Tyr GPU driver.

— https://rust-for-linux.com‘s “Users — in mainline” section in the menu
— https://rust-for-linux.com/rust-reference-drivers



316

79 Comments

Extension_Ad_370 • 8d ago

from the panic log it looks like your drive is getting corrupted

you can try running fschk on the drive (/dev/nvme0n1p2) but its also possible that the drive is nearly dead

115

Who uses Rust for Linux?

Upstreamed users:

- AMCC QT2025 PHY driver.
- Android Binder driver.
- ASIX PHY driver (first “Rust reference driver”).
- DRM Panic QR code generator.
- **Nova GPU driver**.
- Null Block driver.
- Tyr GPU driver.

— https://rust-for-linux.com‘s “Users — in mainline” section in the menu
— https://rust-for-linux.com/rust-reference-drivers

Nova GPU Driver

Nova is a driver for GSP (GPU system processor) based Nvidia GPUs. It is intended to become the successor of Nouveau as the mainline driver for Nvidia (GSP) GPUs in Linux.

It will support all Nvidia GPUs beginning with the GeForce RTX20 (Turing family) series and newer.

Contact

Available communication channels are:

- The mailing list: nouveau@lists.freedesktop.org
- IRC: #nouveau on OFTC
- Zulip Chat

— https://rust-for-linux.com/nova-gpu-driver

Nova

VBIOS support (FWSEC ucode extraction).

Falcon HAL abstraction.

Initial phases of firmware boot sequence (GSP and Falcon).

Full GSP boot sequence (v6.18).

GSP message queue work and firmware abstraction in progress (v6.19).

— Danilo Krummrich
— https://rust-for-linux.com/nova-gpu-driver

Who uses Rust for Linux?

Upstreamed users:

- AMCC QT2025 PHY driver.
- Android Binder driver.
- ASIX PHY driver (first “Rust reference driver”).
- DRM Panic QR code generator.
- Nova GPU driver.
- Null Block driver.
- **Tyr GPU driver**.

— https://rust-for-linux.com‘s “Users — in mainline” section in the menu
— https://rust-for-linux.com/rust-reference-drivers

Tyr GPU Driver

What is Tyr?

Tyr is a new Rust-based DRM driver for CSF-based Arm Mali GPUs. It is a port of Panthor — a driver written in C for the same hardware — and written as a joint effort between Collabora, Arm and Google engineers.

Tyr aims to eventually implement the same userspace API offered by Panthor for compatibility reasons, so that it can be used as a drop-in replacement in our Vulkan driver, called PanVK. In any case, we foresee Panthor being used — and of course supported — for a relatively long time, as it is a mature driver with a large adoption in the ecosystem. It will probably take a couple of years for Tyr to fully pick up.

— https://rust-for-linux.com/tyr-gpu-driver

Who uses Rust for Linux?

Upstreamed users:

- AMCC QT2025 PHY driver.
- **Android Binder driver**.
- ASIX PHY driver (first “Rust reference driver”).
- DRM Panic QR code generator.
- Nova GPU driver.
- Null Block driver.
- Tyr GPU driver.

— https://rust-for-linux.com‘s “Users — in mainline” section in the menu
— https://rust-for-linux.com/rust-reference-drivers

Android Binder Driver

This project is an effort to rewrite Android's Binder kernel driver in Rust.

The driver was merged into Linux kernel version v6.18-rc1.

Motivation

Binder is one of the most security and performance critical components of Android. Android isolates apps from each other and the system by assigning each app a unique user ID (UID). This is called "application sandboxing", and is a fundamental tenet of the Android Platform Security Model.

The majority of inter-process communication (IPC) on Android goes through Binder. Thus, memory unsafety vulnerabilities are especially critical when they happen in the Binder driver.

— https://rust-for-linux.com/android-binder-driver

Who uses Rust for Linux?

Users targeting upstream:

- Android `ashmem` (“Anonymous Shared Memory Subsystem for Android”).
- Apple AGX GPU driver.
- NVMe driver.

...and other efforts, e.g. tarfs, erofs, PuzzleFS, codec libraries, regulator driver, DSI panel driver...

— https://rust-for-linux.com‘s “Users — outside mainline” section in the menu

Who uses Rust for Linux?

Users targeting upstream:

Android `ashmem` (“Anonymous Shared Memory Subsystem for Android”).

Apple AGX GPU driver.

NVMe driver.

...and other efforts, e.g. tarfs, erofs, PuzzleFS, codec libraries, regulator driver, DSI panel driver...

— https://rust-for-linux.com‘s “Users — outside mainline” section in the menu

Who uses Rust for Linux?

"**Android's 6.12** Linux kernel is
our **first kernel with Rust** support enabled and
our **first production Rust driver**."

⇒ **Millions of devices**

running Rust for Linux in the coming months.

— https://rust-for-linux.com/android-%60ashmem%60
— https://security.googleblog.com/2025/11/rust-in-android-move-fast-fix-things.html

Who uses Rust for Linux?

Rust has been **enabled in the Debian kernel**.

debian/config/config

```
@@ -6372,6 +6380,7 @@ CONFIG_KALLSYMS=y
6372  6380   CONFIG_PERF_EVENTS=y
6373  6381   # CONFIG_DEBUG_PERF_USE_VMALLOC is not set
6374  6382   CONFIG_PROFILING=y
      6383 + CONFIG_RUST=y
6375  6384
6376  6385   ##
6377  6386   ## file: kernel/Kconfig.hz
```

— https://salsa.debian.org/kernel-team/linux/-/merge_requests/1615

Who uses Rust for Linux?

Rust has been **enabled in the Debian kernel**.

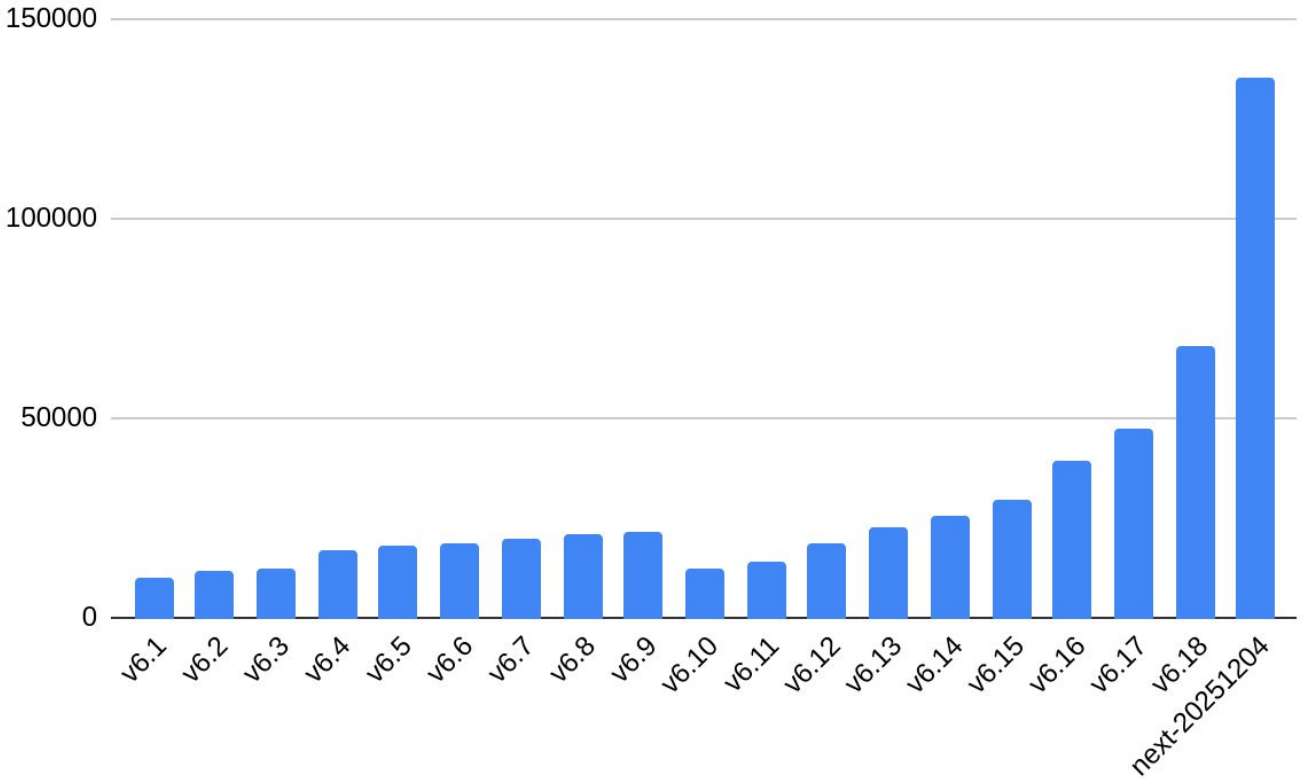
It should become available in **Sid** in the coming weeks.

Eventually, in **Forky** (next Stable).

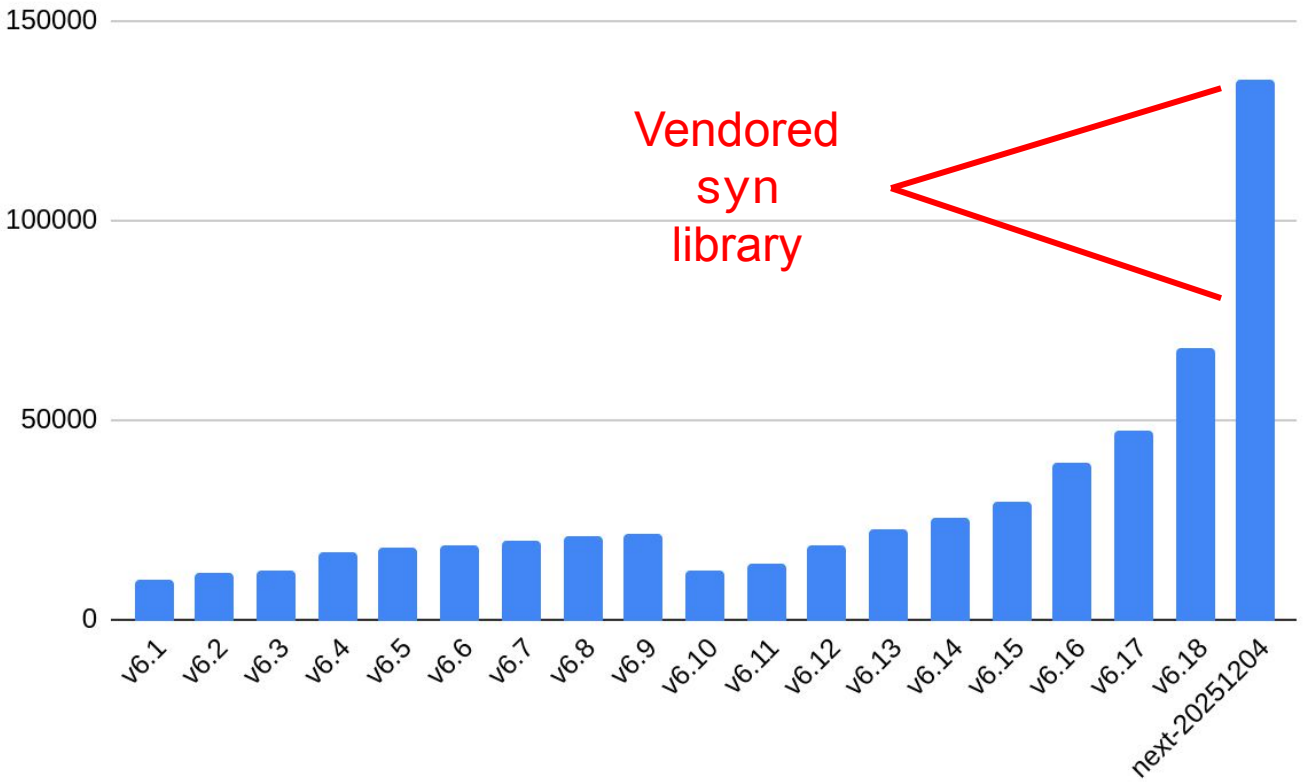
And through stable-backports (opt-in), likely for **Trixie** (current Stable).

— https://salsa.debian.org/kernel-team/linux/-/merge_requests/1615

Lines of Rust code in Linux



Lines of Rust code in Linux



What is syn?

"Syn is a parsing library for **parsing a stream of Rust tokens into a syntax tree** of Rust source code.

Currently this library is geared toward use in Rust **procedural macros**, but contains some APIs that may be useful more generally."

The **most downloaded Rust crate** (crates.io).

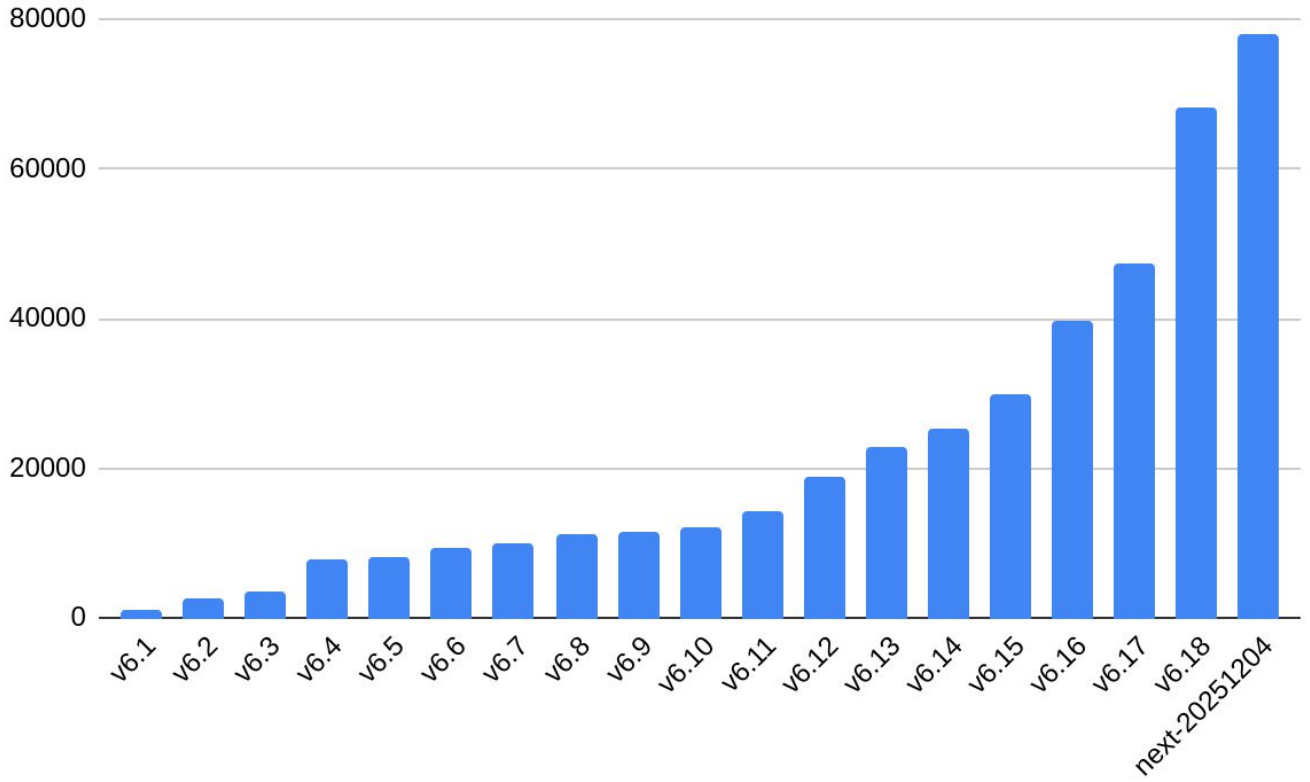
Used by the Rust compiler itself as well.

Allows us to **greatly simplify writing complex macros**, e.g. `pin-init`.

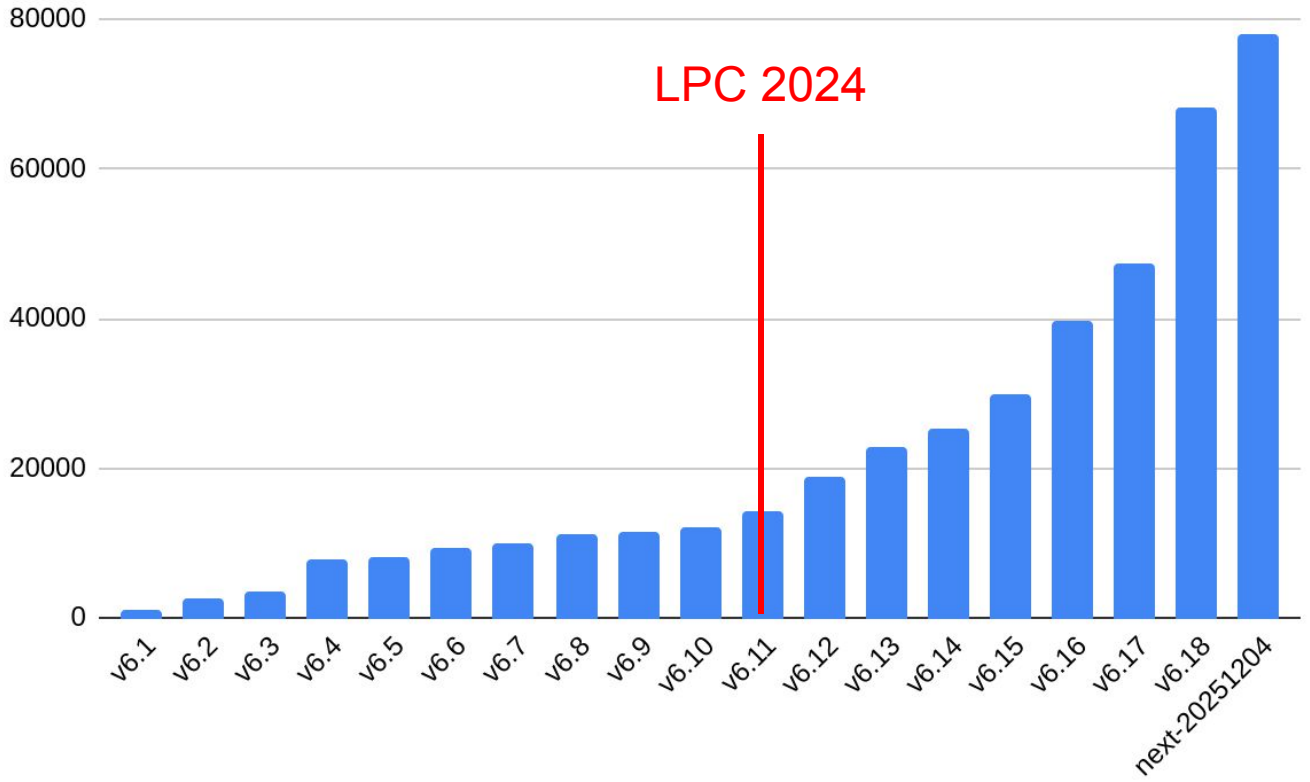
We will also use it in the `macros` crate.

— https://crates.io/crates/syn
— https://crates.io/crates?sort=downloads
— https://rust-for-linux.com/third-party-crates

Lines of Rust code in Linux (no vendored code)

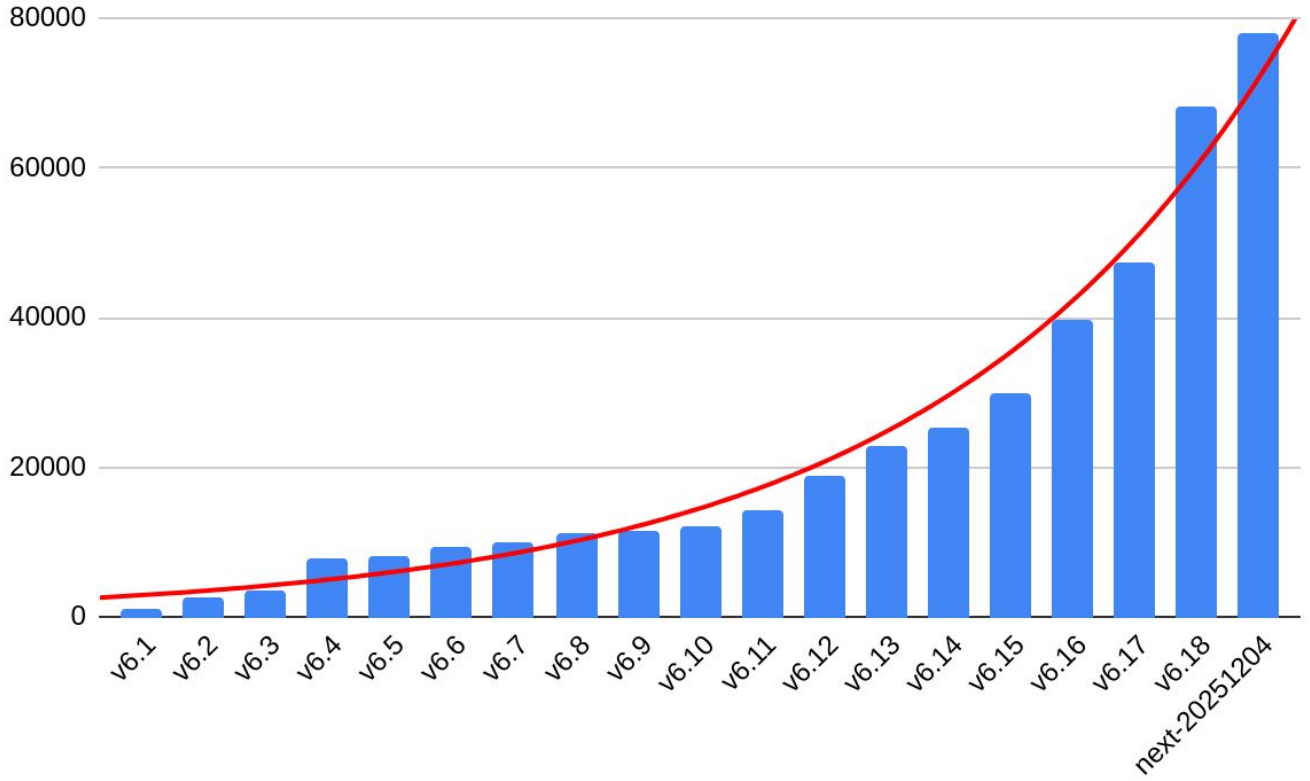


Lines of Rust code in Linux (no vendored code)



Lines of Rust code in Linux (no vendored code)

Lines of Rust code in Linux (no vendored code)



Latest developments (“What those lines are spent on”)

6.12 (LTS): KCFI, KASAN and SCS support, `MITIGATION_*` and `objtool` support, `RUSTC_VERSION`, helpers split, `list` module (`ListArc`, `AtomicTracker`, `ListLinks`, `List`, `Iter`, `Cursor`, `ListArcField`), `rbtree` module (`RBTree`, `RBTreeNode`, `RBTreeNodeReservation`, `Iter`, `IterMut`, `Cursor`), https://rust.docs.kernel.org, Trevor joins, AMCC QT2025 PHY driver...

...

6.15: ARMv7 architecture, hrtimer, dma, driver core updates, pci and platform busses, `nova-core` skeleton driver, `pin-init` as a standalone crate, `#[export]` macro to check signatures for functions called from C by name, additions to `list`/`str`/`alloc`/`module`/`firmware`...

6.16: `#[test]` KUnit support, XArray, mm/vma/mmap, driver core and pci device bindings, `Delta` and `Instant`, pin-init `Wrapper<T>` and `MaybeZeroable`, UAPI for the Asahi driver, type coercion for `Box` and new methods for `Vec`, hrtimer subsystem expanded to timekeeping...

6.17: `warn_on!`, workqueue delayed work items, major driver-core update (e.g. generic device / driver devres infrastructure, firmware node and device properties, ACPI device ID, generic I/O map support...), `bits` module, `UserPtr` newtype, dma improvements, generalized `Instant` and `HrTimer`, `fsleep`, `Results` are pin-initializers, safe `zeroed`, preparations for stdlib `CStr`, enabled Clippy lints to avoid `as` casts...

6.18/RFCs/WIP: LKMM generic atomics, `syn` support (vendored), `module!` parameters, `Alignment` and `Bounded` types, `Integer` trait, replacing custom `CStr` with `core::ffi::CStr`, driver-core (device IRQ handler, sg_table and scatterlist, DebugFS), per-CPU variables, I/O polling, iov_iter, port io, V4L2, I2C, HID, PWM, zpool, better UML support, `Untrusted`, safety standard, codecs, new build system (`kernel` split), MIPS, shrinker abstraction...

Collaboration with Rust

Since early 2024, **regular meetings** between Rust and Rust for Linux.

Rust for Linux has been a **flagship Rust Project goal**.

2024H2: “Resolve the biggest blockers to Linux building on stable Rust”.

2025H1: “Stabilize tooling needed by Rust for Linux”.

Including language, library, compiler, CI...

— https://blog.rust-lang.org/2024/08/12/Project-goals.html
— https://rust-lang.github.io/rust-project-goals/2024h2/rfl_stable.html
— https://rust-lang.github.io/rust-project-goals/2025h1/rfl.html

Collaboration with Rust

We are **very close to Linux using only stable language features**.

- Only 2 features remain: `arbitrary_self_types` and `derive_coerce_pointee`.
- Of those, only one is allowed to be used globally within the kernel.

We are also making **very good progress on compiler features too**, e.g.

- Rust 1.88 stabilized `-Cdwarf-version` and added `-Zsanitize-kcfi-arity`.
- Rust 1.89 added `-Zretpoline{,-external-thunk}` and improved `--output-format=doctest`.

— https://github.com/rust-for-linux/linux/issues/2

Collaboration with Rust

For **2025H2**, two ongoing goals:

- “Getting Rust for Linux into stable Rust: **compiler features**”
- “Getting Rust for Linux into stable Rust: **language features**”

In addition:

- “Design a language feature to solve **Field Projections**”
- “**In-place initialization**”
- “`build-std`”

— https://rust-lang.github.io/rust-project-goals/2025h2/Rust-for-Linux-compiler.html
— https://rust-lang.github.io/rust-project-goals/2025h2/Rust-for-Linux-language.html

Collaboration with Rust

Field projections project goal:

Big boost in development speed since it got accepted as its own project goal.

A few new people have started to contribute: notably Tyler and Nadrieril.

There also is interest from t-libs for this feature for `ArcRef`.

Groundbreaking approach by Nadrieril: virtual places, which allows a more coherent view of the feature.

> Enables better interactions with pattern matching, existing place operations and syntax.

Still lots of things to do.

— Benno Lossin
— https://github.com/rust-lang/rust-project-goals/issues/390
— https://rust-lang.github.io/rust-project-goals/2025h2/field-projections.html

Collaboration with Rust

In-place initialization project goal:

Many discussions:

Which of the different designs is the best? Or, how can we merge them in the best way?

Out-pointers and how to do init-proofs in a non-obstructive way?

Improvements for init-expressions/stuff that transfers to pin-init:

For example the support of arbitrary `Try` types.

Analyzed interactions with effects (so `async`, `try` etc), makes design harder, as it aims for maximal compatibility, needs more time.

Still needs much design work to be done in order to combine the proposals.

— Benno Lossin
— https://github.com/rust-lang/rust-project-goals/issues/395
— https://rust-lang.github.io/rust-project-goals/2025h2/in-place-initialization.html

Collaboration with Rust

`rustfmt` and the **trailing `//` comment** for imports

Since v6.18, we apply this “hack” to get the formatting we want in stable Rust.

This is needed to avoid conflicts on import “blocks”.

```
use crate::{
    example1,
    example2::{
        example3,
        example4, //
    },
    example5,
    example6::example7, //
};
```

— https://docs.kernel.org/rust/coding-guidelines.html#imports

Collaboration with Rust

- `rustfmt` and the **trailing `//` comment** for imports
  - Upstream Rust created a **new sub-team of contributors** for `rustfmt` to try to clear work and get to features.
  - From Manish Goregaokar:
    - “Yes, currently we're working on clearing up the backlog of maintenance tasks, and then perhaps some feature work can be done. There's a couple challenges around the import granularity issue that need to be carefully thought out.”

Collaboration with Rust

"Rust-For-Linux is getting **first class support**"

(within **Clippy** — the Rust linter)

— Alejandra González, in Kangrejos 2025
— https://kangrejos.com/2025/Clippy%20status%20update%20for%20Rust-for-Linux.pdf

Collaboration with Rust

Great ongoing **collaboration with upstream `rustdoc`**, e.g.

- The JSON output format to extract doctests for our KUnit integration.

  `--output-format=doctest`

  Replaces the hack I originally wrote based on the unstable `--test-builder` and `--no-run` features.

- The hidden/private items toggle.

- Backport needed to avoid a kernel workaround.

Thanks to Guillaume Gomez and his team!

— https://github.com/Rust-for-Linux/linux/issues/2
— https://github.com/Rust-for-Linux/linux/issues/350

Linux in Rust's and `bindgen`'s CI

It has been more than a year that **every Rust PR now build-tests the Linux kernel**.

The goal is to avoid unintentional changes to Rust that break the kernel.

Thus, in general, apart from intentional changes, the upcoming Rust compiler versions should generally work.

`bindgen` has also included Linux in its CI since right after the last LPC.

It has worked well so far throughout the year.

— https://rust-for-linux.com/rust-version-policy
— https://rustc-dev-guide.rust-lang.org/tests/ecosystem-test-jobs/rust-for-linux.html

rustc_codegen_gcc

What has been **done in 2025**:

- Preparation work to eventually get **rustup distribution** (still ongoing).
  - We now compile our **GCC fork in Rust CI**.
- Initial support for **non-default ABIs**.
- Fixed issues with **LTO**.
- Cross-language LTO.
- Run most UI tests for cg_gcc in Rust CI.
- Progress to compile Rust to **new platforms**.
  - We were able to compile a (non-working) rustc for m68k.

— Antoni Boucher
— https://blog.antoyo.xyz

rustc_codegen_gcc

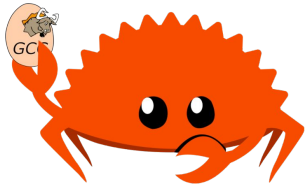
What still needs **to be done**:

- Have rustup distribution.
- Improve the support for debug info.
- Improve the support for unwinding.
  - Mostly working, but not working correctly is some cases.
- Finish implementing function and variable attributes.
  - Most of them are supported, but some of them are currently not supported.
- SIMD for other architectures than x86-64.

Some distributions like Gentoo have started to experiment with packaging the backend too.

— Antoni Boucher
— https://blog.antoyo.xyz

gccrs



From gccrs' October 2025 Monthly report:

> "We have **started experimenting with compiling Rust-for-Linux**, as planned in our last monthly reports, and we are happy to report **very few issues** apart from the ones related to requiring a functional `core` crate. We then **created a new Rust-for-Linux milestone** to gather all the missing functionality required for compiling the Rust parts of the kernel, and are still collecting issues."

From Arthur Cohen:

> "We are making **very good progress on compiling the kernel** :) We'll have **more specific information at the beginning of next year**."

— https://rust-gcc.github.io/2025/11/17/2025-10-monthly-report.html
— https://kangrejos.com/2025/From%20GCC%20to%20Rust%20for%20Linux.pdf

Coccinelle For Rust

CoccinelleForRust is used to perform large-scale transformations in Rust repositories by generalizing repetitive refactorings into *Semantic Patches.*

The following shows an example change

```
@@
expression y;
identifier i;
@@

-let i =
+let mut i =
    y.collect_vec();
...

+i.reverse();
-apply_merge(i);
+new_apply_merge(i, EDITION_ARG);
```

Semantic patch

```
@@ -1,5 +1,6 @@
 fn main() {
-    let vcloned = v.collect_vec();
+    let mut vcloned = v.collect_vec();

-    apply_merge(vcloned);
+    vcloned.reverse()
+    new_apply_merge(vcloned, EDITION_ARG);
 }
```

Target Rust file

For more details and examples please visit:-https://rust-for-linux.com/coccinelle-for-rust

Now tackle more complex tasks with scripting!

— Tathagata Roy and Julia Lawall

— Tutorial at https://kangrejos.com/2025/Coccinelle%20for%20Rust.pdf

Kangrejos

The Rust for Linux Workshop

An event where people involved in the Rust for Linux discussions can meet in a single place before LPC.

https://kangrejos.com

https://lwn.net/Archives/ConferenceIndex/#Kangrejos


Kangrejos 2025, Oviedo, Spain

What does the community think about Rust for Linux?

FOSDEM keynote with many quotes from kernel maintainers and others:

> “In my mind, the Rust for Linux project has already achieved an important goal: **proving that Rust is indeed a viable and desirable language for kernel development**. I think that the discussion on whether we should go that way has run its course; as they say, it's all over but the shouting. Of course, this is the kernel community, so **we should expect a fair amount more shouting still**. **This work is important for the long-term viability of Linux, and I am glad that it is succeeding.**”
>
> — Jonathan Corbet
> *LWN executive editor and kernel maintainer*

— https://fosdem.org/2025/schedule/event/fosdem-2025-6507-rust-for-linux/

All these news just led to...

The (successful) end of the kernel Rust experiment

[Posted December 10, 2025 by corbet]

The topic of the Rust experiment was just discussed at the annual Maintainers Summit. The consensus among the assembled developers is that Rust in the kernel is no longer experimental — it is now a core part of the kernel and is here to stay. So the "experimental" tag will be coming off. Congratulations are in order for all of the Rust for Linux team.

(Stay tuned for details in our Maintainers Summit coverage.)

rust: conclude the Rust experiment

The Rust support was merged in v6.1 into mainline in order to help determine whether Rust as a language was suitable for the kernel, i.e. worth the tradeoffs, technically, procedurally and socially.

At the 2025 Linux Kernel Maintainers Summit, the experiment has just been deemed concluded.

Thus remove the section — it was not fully true already anyway, since there are already uses of Rust in production out there, some well-known Linux distributions enable it and in some months it will be in millions of devices via Android.

Obviously, this does not mean that everything works for every kernel configuration, architecture, toolchain etc., or that there won't be new issues. There is still a ton of work to do in all areas, from the kernel to upstream Rust, GCC and other projects. And, in fact, certain combinations (such as the mixed GCC+LLVM builds and the upcoming GCC support) are still quite experimental but getting there.

But the experiment is done, i.e. Rust is here to stay.

I hope this signals commitment from the kernel to companies and other entities to invest more into it, e.g. into giving time to their kernel developers to train themselves in Rust.

Thanks to the many kernel maintainers that gave the project their support and patience throughout these years, and to the many other developers, whether in the kernel or in other projects, that have made this possible. I had a long list of 173 names in the credits of the original pull that merged the support into the kernel, and now such a list would be way longer, so I will not even try to compose one, but again, thanks a lot, everybody.

Signed-off-by: Miguel Ojeda <ojeda@kernel.org>

What now?

What I am expecting for the next year

Linux **not using any unstable language features**.

Progress on **new features** the kernel may need, such as field projections.

Most **compiler features** stabilized.

Millions of users, due to Android 16 6.12 `ashmem`.

Real users trying out **complex drivers** like Tyr and Nova.

`rustc_codegen_gcc` in the Rust CI building Rust for Linux.

gccrs building the `kernel` crate and a kernel module for the first time.

...and, of course, more upstreamed users of Rust in the mainline kernel.

Rust minimum version upgrade

When do we want to upgrade the minimum supported Rust version in the kernel?

Proposal:

To follow Debian Stable's Rust version.

For instance, that would mean upgrading soon to Trixie's Rust 1.85 (2025-02-20).

From the current 1.78 (2024-05-02).

Thus, upgrade the minimum every ~2 years (a few months after Debian gets released).

It would still be a smaller window than C.

But it is a lot of versions nevertheless, given the release cadence compared to C.

We can always increase the window to "2 Debians" later on.

Build system

Aiming to get it merged in the first cycles after the LTS.

LPC is a chance to discuss it with Kbuild and KUnit.

That gives us the better part of a year to iron it out until the next LTS.

We could consider backporting too, if needed.

There are not many user-facing changes, after all.

Crate rust_driver_platform

Source

Settings Help Summary

Rust Platform driver sample. ACPI match table test

This demonstrates how to test an ACPI-based Rust platform driver using QEMU with a custom SSDT.

Steps:

1. **Create an SSDT source file** (`ssdt.dsl`) with the following content:

```
DefinitionBlock ("", "SSDT", 2, "TEST", "VIRTACPI", 0x00000001)
{
    Scope (\_SB)
    {
        Device (T432)
        {
            Name (_HID, "LNUXBEEF")  // ACPI hardware ID to match
            Name (_UID, 1)
            Name (_STA, 0x0F)        // Device present, enabled
            Name (_CRS, ResourceTemplate ()
            {
                Memory32Fixed (ReadWrite, 0xFED00000, 0x1000)
            })
        }
    }
}
```

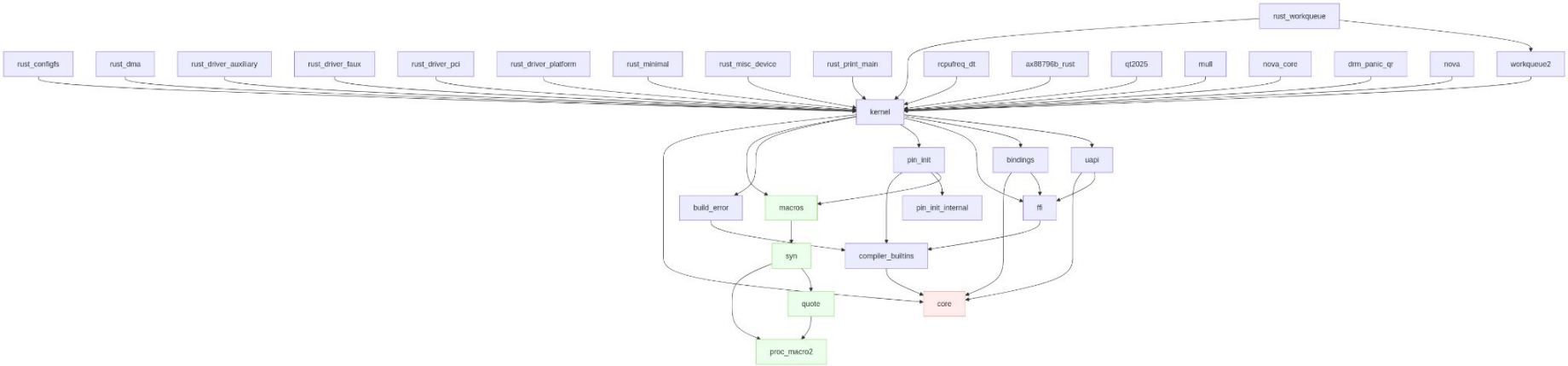
2. **Compile the table**:

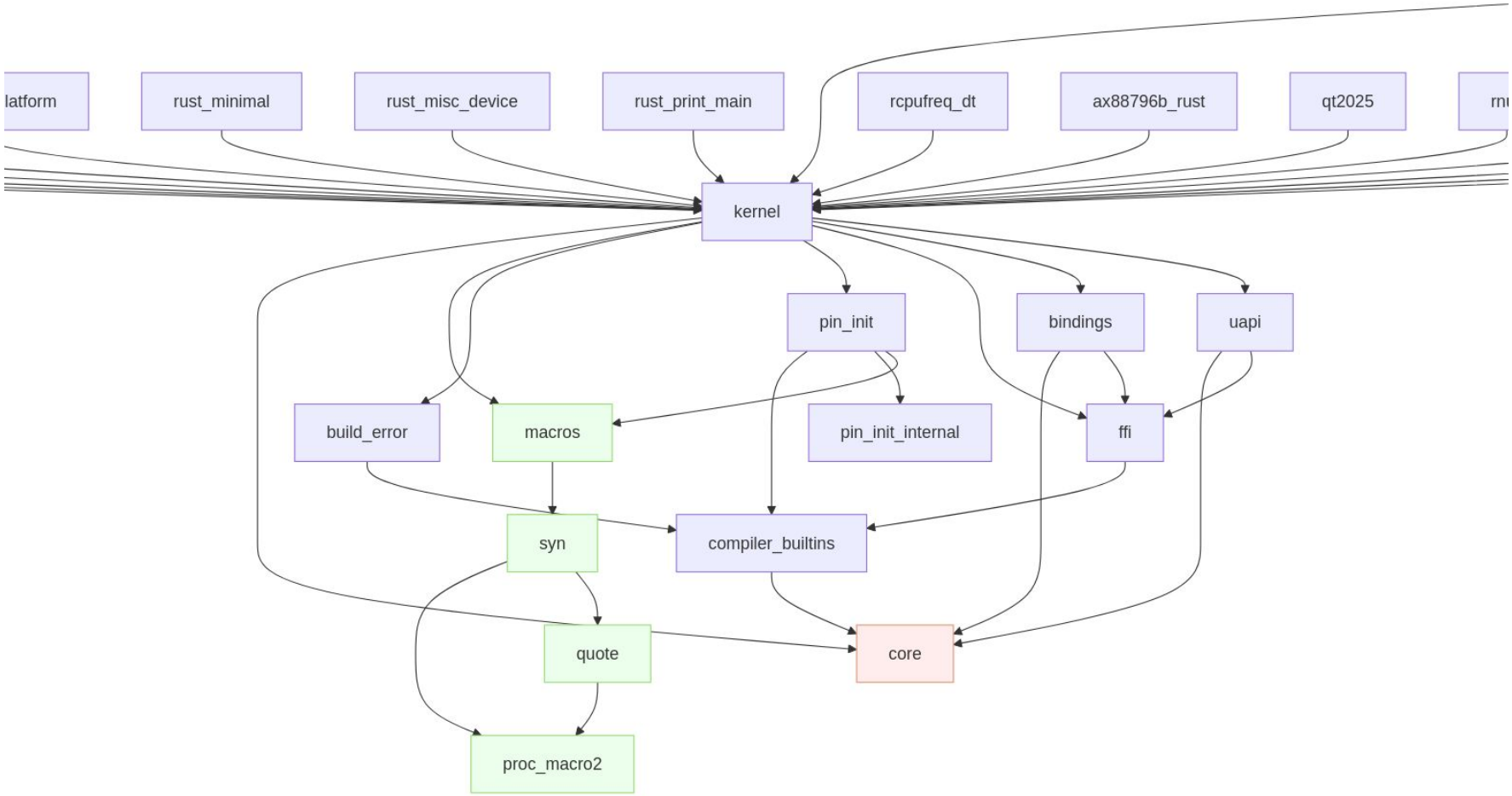
```
iasl -tc ssdt.dsl
```

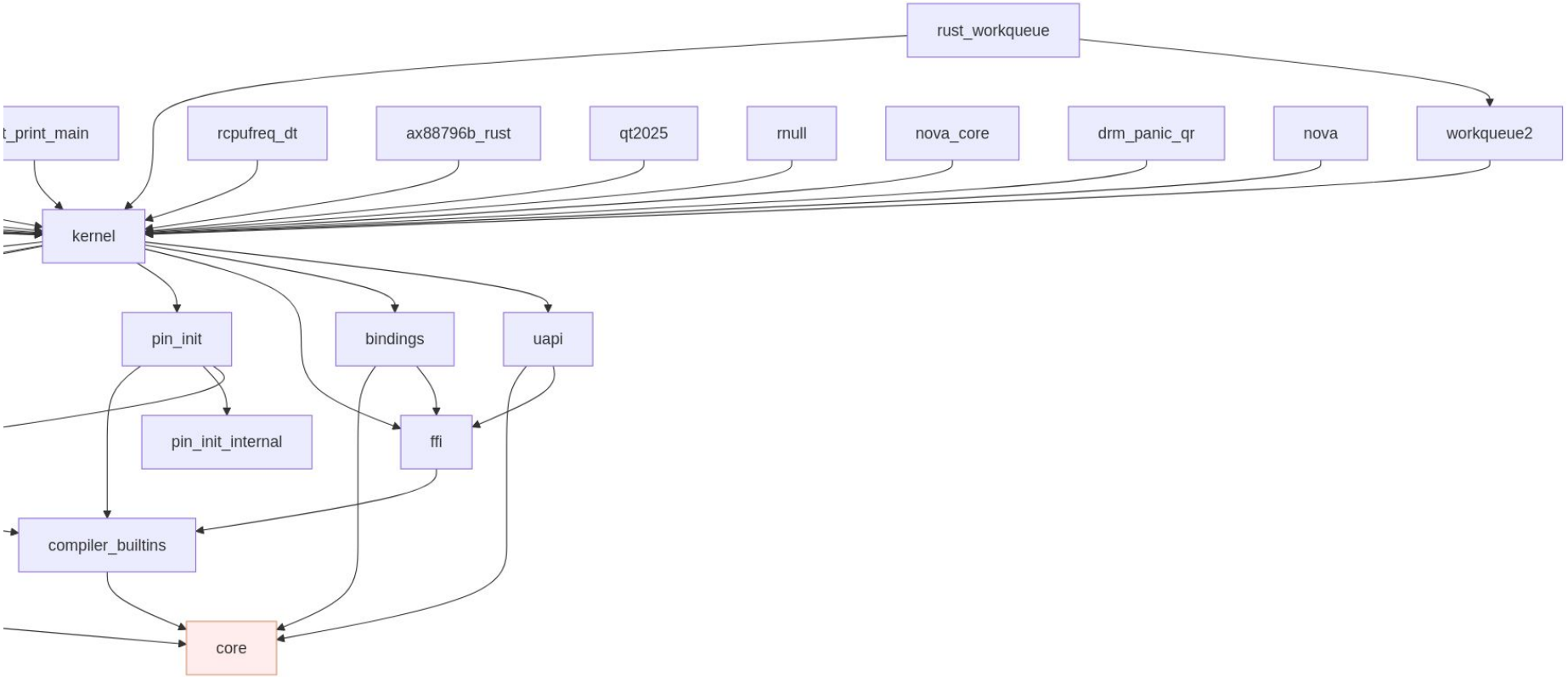
   This generates `ssdt.aml`

3. **Run QEMU** with the compiled AML file:

```
qemu-system-x86_64 -m 512M \
    -enable-kvm \
    -kernel path/to/bzImage \
    -append "root=/dev/sda console=ttyS0" \
    -hda rootfs.img \
```


rust_workqueue
rust_configfs
rust_dma
rust_driver_auxiliary
rust_driver_faux
rust_driver_pci
rust_driver_platform
rust_minimal
rust_misc_device
rust_print_main
rcpufreq_dt
ax88796b_rust
qt2025
rnull
nova_core
drm_panic_qr
nova
workqueue2
kernel
pin_init
bindings
uapi
build_error
macros
pin_init_internal
ffi
syn
compiler_builtins
quote
core
proc_macro2


latform
rust_minimal
rust_misc_device
rust_print_main
rcpufreq_dt
ax88796b_rust
qt2025
kernel
pin_init
bindings
uapi
build_error
macros
pin_init_internal
ffi
syn
compiler_builtins
quote
core
proc_macro2


rust_workqueue
t_print_main
rcpufreq_dt
ax88796b_rust
qt2025
rnull
nova_core
drm_panic_qr
nova
workqueue2
kernel
pin_init
bindings
uapi
pin_init_internal
ffi
compiler_builtins
core

Sponsors & Industry support

— https://rust-for-linux.com/sponsors
— https://rust-for-linux.com/industry-and-academia-support
— https://fosdem.org/2025/schedule/event/fosdem-2025-6507-rust-for-linux/

Related upcoming LPC sessions

Moving kernel swapping infrastructure to Rust (already presented).

https://lpc.events/event/19/contributions/2265/

Tyr: a new Rust GPU driver (following this one).

https://lpc.events/event/19/contributions/2304/

The Rust MC (tomorrow).

https://lpc.events/event/19/sessions/223/

Rust for Linux Office Hours (Saturday).

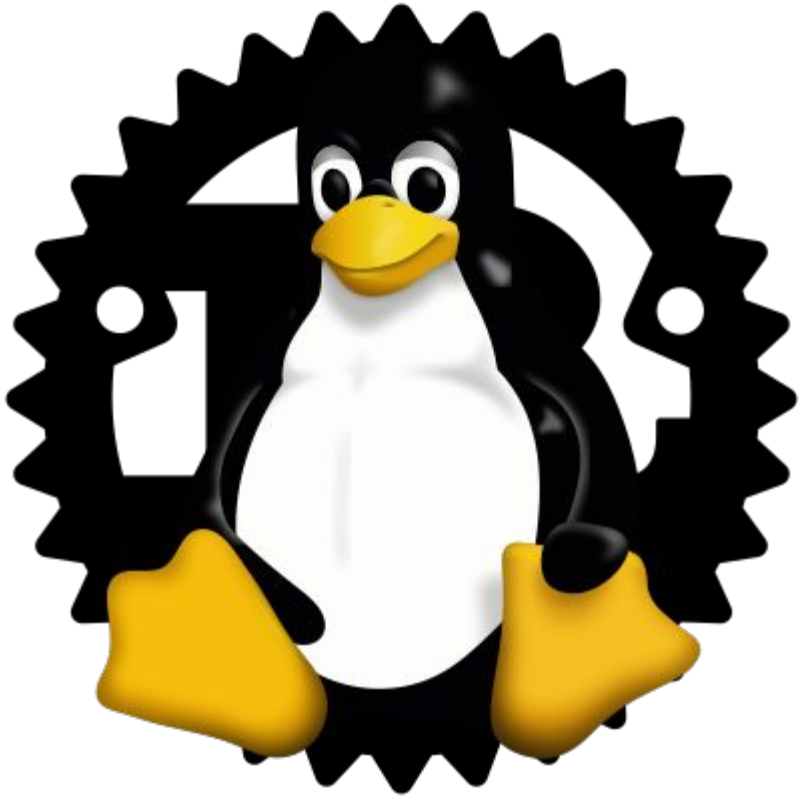
https://lpc.events/event/19/contributions/2291/

Danilo’s talk possibly recorded later on.

How to contribute?

https://rust-for-linux.com/contributing

https://rust-for-linux.com/contact



Rust for Linux

Miguel Ojeda

ojeda@kernel.org

Backup slides

syn

How do we integrate it?

Originally we felt adding syn was a bit **too early**.

Too big relatively speaking to the Rust code we had.

Especially since Rust was still an experiment.

Another concern was the **build time** w.r.t. the total.

We didn’t want to encourage macro-based solutions.

We wanted to push ourselves to look for alternatives.

syn

One possibility for integration is the “**take it from the distribution**” approach.

After all, it is just a host library (a compiler “plugin”).

We “just” need to build it.

The idea is conceptually simple.

Just point the build system to the right files.

syn

But it is **never that simple**:

- One needs to consider how to pick the right version.
  - Folders may be versioned.
  - There may be several versions.
  - The latest version of each dependency may not be the right one.
- One needs to allow local checkouts for other distributions and custom setups.
- The files may not be the right ones.

syn

Solving versioning with a **local registry could be done**, but:

- Requires adding Cargo as a dependency just for that.
- Getting the right information out of Cargo may not be too simple on its own.
- At least Debian and Fedora agree on the registry location.

Upstream could change dependencies:

- In principle, even to the number of crates!
- Cargo would be needed to properly build the graph.

One **cannot “customize” the sources**, if needed:

- Like dropping one of the dependencies as we did.

syn

Vendoring is less fragile and much simpler:

- The files are always the expected ones.
- A few less degrees of freedom (one per dependency).
  - Since otherwise we would likely need to support different versions.
- No user configuration mistakes.
- Even if an update is needed, the diff should generally not be huge.
  - In the last year, it would have been **+7k -3k lines across the 3 crates** needed for syn.

Build system

The original merge into the kernel was very minimal.

- Including the build system support.

Having a single “`kernel`” crate that contains abstractions is simple, in several ways:

- Conceptually (e.g. my usual safe/unsafe split slide).
- In terms of avoiding changes to Kbuild (i.e. I could reuse most of it).
- Bypasses extra complexities in the languages (e.g. orphan rule).

Build system

The current system has actually worked quite well so far.

- It allowed us to bootstrap everything easily.
  - Less maintenance effort, too.
- It would have allowed to remove the Rust support without much pain, if that would have happened.
  - We were an experiment, after all.
- There is so far no completed user that needed more.

Now those non-trivial users are getting closer to completed, and thus it is the time to make a move here.

Build system

Nevertheless, why do we need something else?

- We want to split the `kernel` crate.
- We want to move the files to their own directories.
- We want to have Rust modules that depend on Rust modules.
- We want to have exports, doctests, `#[test]`s etc. in other crates.
- ...and so on and so forth.

The root issue is that Rust is compiled as a graph of crates.

- Unlike C, which one usually can build in parallel independently.
- Essentially, Rust builds the “headers” that C provides by hand.

Build system

So I thought about what approach would be best.

There are a few, with a fairly wide range of scale/scope.

The approach I decided on was to split the changes in steps:

First, introduce the simplest change we can to support what we need.

A generalization of what we have, with a limitation.

Not being able to tweak compiler flags on later `Makefiles`.

Then, later on, remove the limitation, if needed.

Which would be an implementation-side only change, unlike the first step.

Crate syn

Source

Syn is a parsing library for parsing a stream of Rust tokens into a syntax tree of Rust source code.

Currently this library is geared toward use in Rust procedural macros, but contains some APIs that may be useful more generally.

- **Data structures** — Syn provides a complete syntax tree that can represent any valid Rust source code. The syntax tree is rooted at `syn::File` which represents a full source file, but there are other entry points that may be useful to procedural macros including `syn::Item`, `syn::Expr` and `syn::Type`.
- **Derives** — Of particular interest to derive macros is `syn::DeriveInput` which is any of the three legal input items to a derive macro. An example below shows using this type in a library that can derive implementations of a user-defined trait.
- **Parsing** — Parsing in Syn is built around parser functions with the signature `fn(ParseStream) -> Result<T>`. Every syntax tree node defined by Syn is individually parsable and may be used as a building block for custom syntaxes, or you may dream up your own brand new syntax without involving any of our syntax tree types.
- **Location information** — Every token parsed by Syn is associated with a `Span` that tracks line and column information back to the source of that token. These spans allow a procedural macro to display detailed error messages pointing to all the right places in the user's code. There is an example of this below.
- **Feature flags** — Functionality is aggressively feature gated so your procedural macros enable only what they need, and do not pay in compile time for all the rest.

Example of a derive macro

The canonical derive macro using Syn looks like this. We write an ordinary Rust function tagged with a `proc_macro_derive` attribute and the name of the trait we are deriving. Any time that derive appears in the user's code, the Rust compiler passes their data structure as tokens into our macro. We get to execute arbitrary Rust code to figure out what to do with those tokens, then hand

```
CLIPPY  rust/doctests_ffi_generated.o
CC      rust/doctests_ffi_generated_kunit.o
CLIPPY  rust/doctests_bindings_generated.o
CC      rust/doctests_bindings_generated_kunit.o
CLIPPY  rust/doctests_uapi_generated.o
CC      rust/doctests_uapi_generated_kunit.o
CLIPPY  rust/doctests_kernel_generated.o
CC      rust/doctests_kernel_generated_kunit.o
AR      rust/built-in.a
LD [R]  drivers/gpu/drm/drm_panic_qr.o
AR      drivers/gpu/drm/built-in.a
AR      drivers/gpu/built-in.a
CC      drivers/base/firmware_loader/main.o
AR      drivers/base/firmware_loader/built-in.a
AR      drivers/base/built-in.a
AR      drivers/built-in.a
AR      samples/rust/built-in.a
LD [R]  samples/rust/rust_misc_device.o
LD [R]  samples/rust/rust_print_main.o
CC [M]  samples/rust/rust_print_events.o
LD [M]  samples/rust/rust_print.o
LD [R]  samples/rust/rust_dma.o
LD [R]  samples/rust/rust_driver_pci.o
LD [R]  samples/rust/rust_driver_platform.o
LD [R]  samples/rust/rust_driver_faux.o
LD [R]  samples/rust/rust_driver_auxiliary.o
LD [R]  samples/rust/rust_configfs.o
LD [R]  samples/rust/rust_workqueue.o
```

```
CLIPPY [M]  kernel/workqueue.o
EXPORTS H   kernel/workqueue.o.exports.h
EXPORTS C   kernel/workqueue.o.exports.c
CLIPPY [M]  samples/rust/rust_workqueue.o
EXPORTS H   samples/rust/rust_workqueue.o.exports.h
EXPORTS C   samples/rust/rust_workqueue.o.exports.c
CLIPPY [M]  drivers/gpu/nova-core/nova_core.o
EXPORTS H   drivers/gpu/nova-core/nova_core.o.exports.h
EXPORTS C   drivers/gpu/nova-core/nova_core.o.exports.c
CLIPPY [M]  drivers/gpu/drm/nova/nova.o
EXPORTS H   drivers/gpu/drm/nova/nova.o.exports.h
EXPORTS C   drivers/gpu/drm/nova/nova.o.exports.c
CC [M]      samples/rust/rust_configfs.o.exports.o
CC [M]      samples/rust/rust_dma.o.exports.o
CC [M]      samples/rust/rust_driver_auxiliary.o.exports.o
CC [M]      samples/rust/rust_driver_faux.o.exports.o
CC [M]      samples/rust/rust_driver_pci.o.exports.o
CC [M]      samples/rust/rust_driver_platform.o.exports.o
CC [M]      samples/rust/rust_misc_device.o.exports.o
CC [M]      samples/rust/rust_print_main.o.exports.o
CC [M]      kernel/workqueue.o.exports.o
CC [M]      samples/rust/rust_workqueue.o.exports.o
CC [M]      drivers/gpu/nova-core/nova_core.o.exports.o
CC [M]      drivers/gpu/drm/nova/nova.o.exports.o
```

```
Kernel: arch/x86/boot/bzImage is ready  (#29)
  CC [M]  kernel/workqueue.mod.o
  CC [M]  .module-common.o
  LD [M]  kernel/workqueue.ko
  LD [M]  fs/efivarfs/efivarfs.ko
  LD [M]  drivers/thermal/intel/x86_pkg_temp_thermal.ko
  CC [M]  samples/rust/rust_misc_device.mod.o
  LD [M]  samples/rust/rust_misc_device.ko
  CC [M]  samples/rust/rust_print.mod.o
  LD [M]  samples/rust/rust_print.ko
  CC [M]  samples/rust/rust_dma.mod.o
  LD [M]  samples/rust/rust_dma.ko
  CC [M]  samples/rust/rust_driver_pci.mod.o
  LD [M]  samples/rust/rust_driver_pci.ko
  CC [M]  samples/rust/rust_driver_platform.mod.o
  LD [M]  samples/rust/rust_driver_platform.ko
  CC [M]  samples/rust/rust_driver_faux.mod.o
  LD [M]  samples/rust/rust_driver_faux.ko
  CC [M]  samples/rust/rust_driver_auxiliary.mod.o
  LD [M]  samples/rust/rust_driver_auxiliary.ko
  CC [M]  samples/rust/rust_configfs.mod.o
  LD [M]  samples/rust/rust_configfs.ko
  CC [M]  samples/rust/rust_workqueue.mod.o
  LD [M]  samples/rust/rust_workqueue.ko
```

```
RUSTDOC   sysroot/.../lib/rustlib/src/rust/library/core/src/lib.rs
RUSTDOC H rust/proc-macro2/lib.rs
RUSTDOC H rust/quote/lib.rs
RUSTDOC H rust/syn/lib.rs
RUSTDOC   rust/compiler_builtins.rs
RUSTDOC H rust/macros/lib.rs
RUSTDOC   rust/build_error.rs
RUSTDOC H rust/pin-init/internal/src/lib.rs
RUSTDOC   rust/pin-init/src/lib.rs
RUSTDOC   rust/ffi.rs
RUSTDOC   rust/bindings/lib.rs
RUSTDOC   rust/uapi/lib.rs
RUSTDOC   rust/kernel/lib.rs
RUSTDOC   samples/rust/rust_configfs.rs
RUSTDOC   samples/rust/rust_dma.rs
RUSTDOC   samples/rust/rust_driver_auxiliary.rs
RUSTDOC   samples/rust/rust_driver_faux.rs
RUSTDOC   samples/rust/rust_driver_pci.rs
RUSTDOC   samples/rust/rust_driver_platform.rs
RUSTDOC   samples/rust/rust_minimal.rs
RUSTDOC   samples/rust/rust_misc_device.rs
RUSTDOC   samples/rust/rust_print_main.rs
RUSTDOC   kernel/workqueue.rs
RUSTDOC   samples/rust/rust_workqueue.rs
RUSTDOC   drivers/cpufreq/rcpufreq_dt.rs
RUSTDOC   drivers/net/phy/ax88796b_rust.rs
RUSTDOC   drivers/net/phy/qt2025.rs
RUSTDOC   drivers/block/rnull.rs
RUSTDOC   drivers/gpu/nova-core/nova_core.rs
RUSTDOC   drivers/gpu/drm/drm_panic_qr.rs
RUSTDOC   drivers/gpu/drm/nova/nova.rs
```

```
[    1.987422] KTAP version 1
[    1.987634] 1..9
[    1.988966]     KTAP version 1
[    1.989515]     # Subtest: rust_doctests_compiler_builtins
[    1.989823]     # module:
[    1.989945]     1..0
[    1.990263] ok 1 rust_doctests_compiler_builtins # SKIP
[    1.990368]     KTAP version 1
[    1.990416]     # Subtest: rust_doctests_build_error
[    1.990478]     # module:
[    1.990491]     1..0
[    1.990559] ok 2 rust_doctests_build_error # SKIP
[    1.990646]     KTAP version 1
[    1.990752]     # Subtest: rust_doctests_ffi
[    1.990914]     # module: doctests_ffi_generated_kunit
[    1.990952]     1..1
[    1.992558]     # rust_doctest_ffi_rs_0.location: rust/ffi.rs:13
[    1.993742]     # rust_doctest_ffi_rs_0: ASSERTION FAILED at rust/ffi.rs:14
[    1.993742]     Expected size_of::<c_int>() == 1 + size_of::<i32>() to be true, but is false
[    1.995701]     not ok 1 rust_doctest_ffi_rs_0
[    1.995829] not ok 3 rust_doctests_ffi
[    1.996015]     KTAP version 1
[    1.996210]     # Subtest: rust_doctests_bindings
[    1.996308]     # module:
[    1.996322]     1..0
```

```
[    3.549976] KTAP version 1
[    3.550176] 1..1
[    3.551166]     KTAP version 1
[    3.551285]     # Subtest: rust_workqueue_example
[    3.551354]     # speed: normal
[    3.551375]     1..1
[    3.553201]     # test_example.speed: normal
[    3.553243]     ok 1 test_example
[    3.553333] ok 1 rust_workqueue_example
[    3.572343] workqueue: test 42 43
[    3.572648] rust_workqueue: result = 85
[    3.572833] rust_workqueue: The value is: 42
```

Rust 1.90

Just a few changes, like:

`rustdoc` fix so that it can use dependencies with target modifiers.

Workaround landed in stable kernels.

Clippy fix for `undocumented_unsafe_blocks` false negatives around attributes (`accept-comment-above-attributes`).

Fix for long standing issue affecting Klint.

Rust 1.91

Added `-Zindirect-branch-cs-prefix` for Retpoline mitigation.

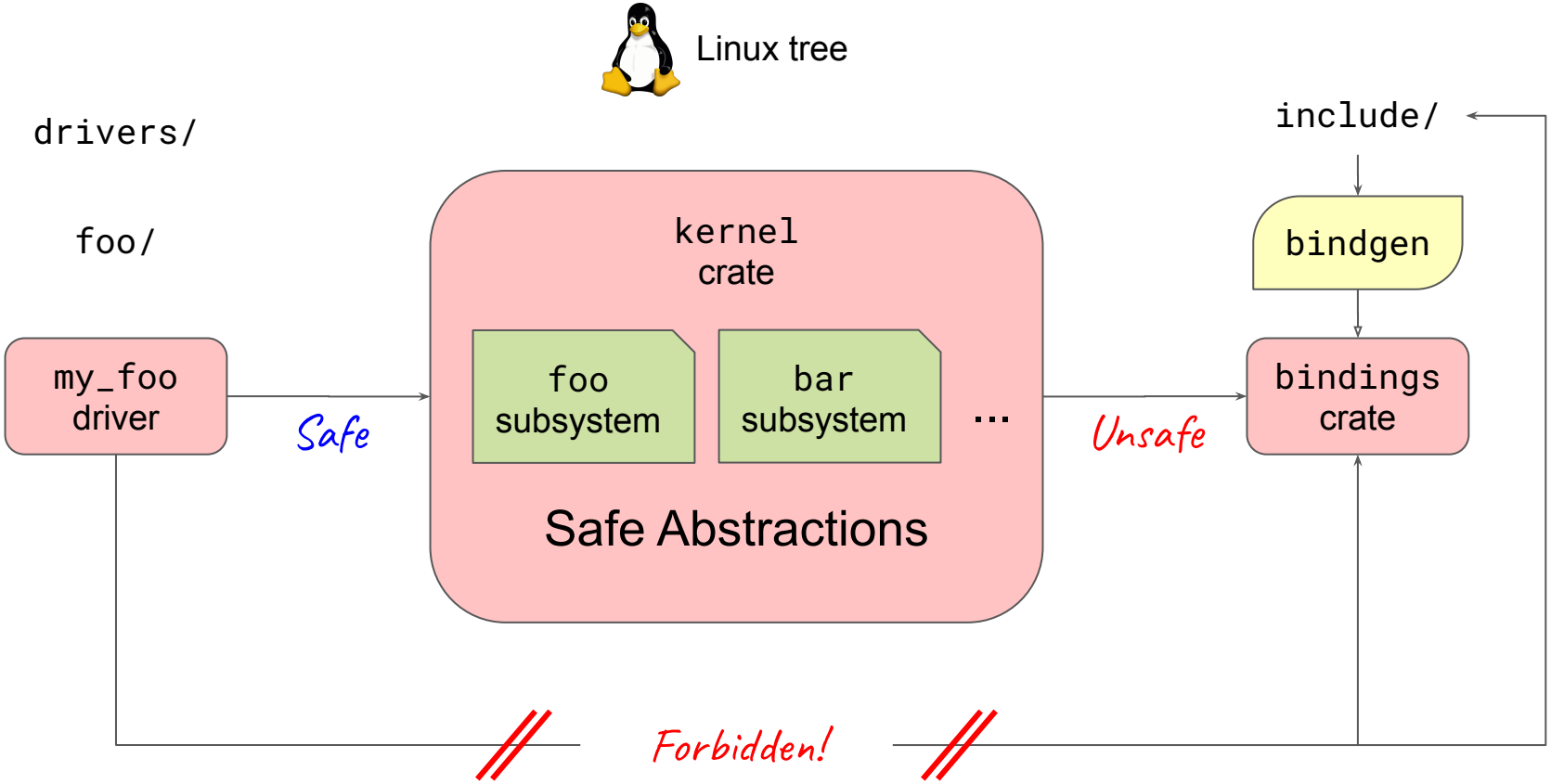
Progress to stabilize `file_as_c_str` (rename).

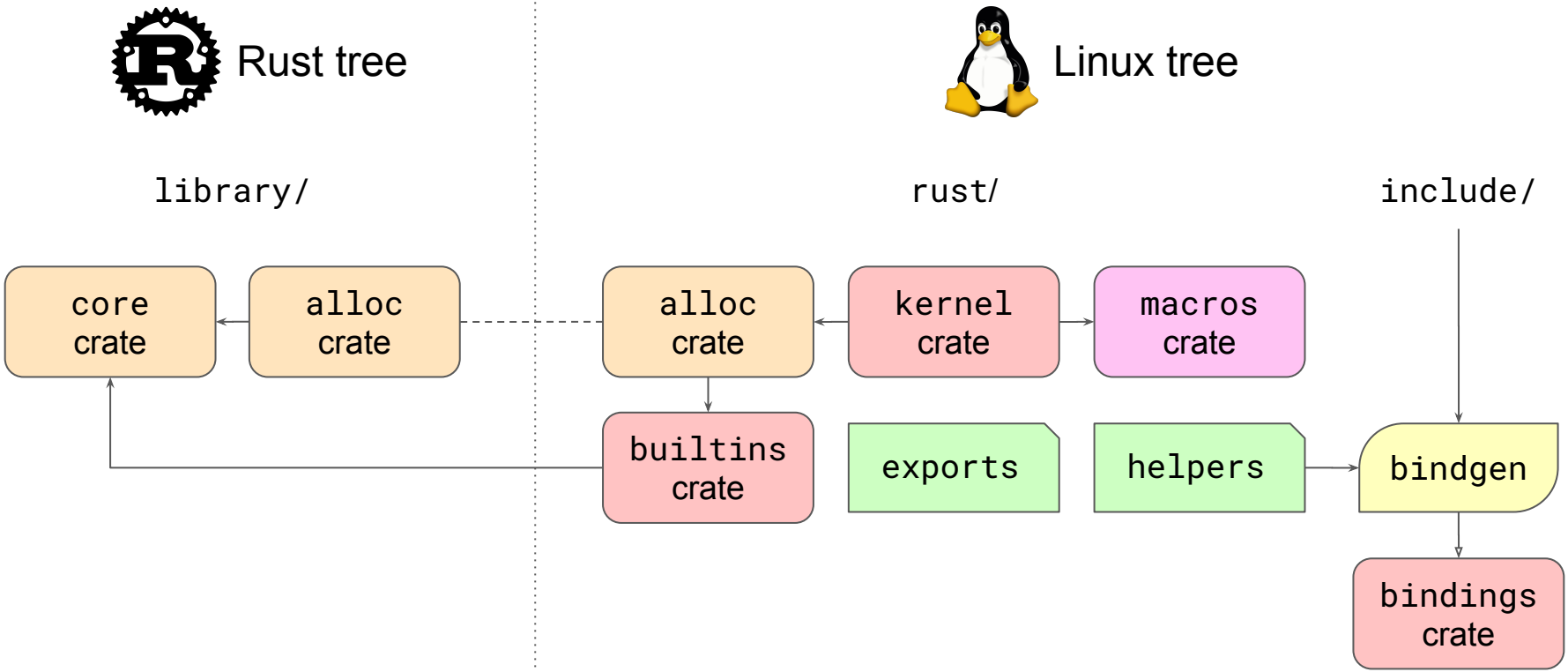
A fix for `-Zregparm` for 32-bit x86.

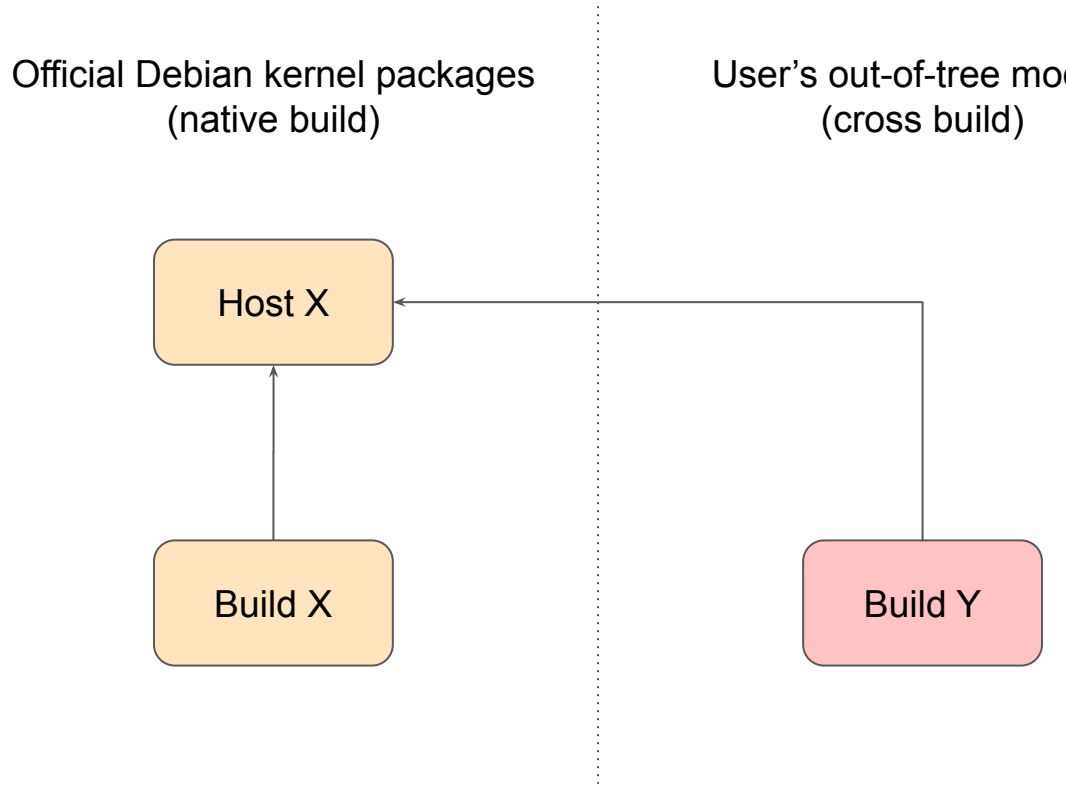
- Tests for `-Zregparm` and `-Zreg-struct-return`.

Make some sanitizers be target modifiers, which could prevent misuse.

A couple Clippy fixes for `unnecessary_safety_comment`.


Linux tree
drivers/
foo/
my_foo
driver
Safe
kernel
crate
foo
subsystem
bar
subsystem
...
Safe Abstractions
Unsafe
include/
bindgen
bindings
crate
Forbidden!


Rust tree
Linux tree
library/
rust/
include/
core crate
alloc crate
alloc crate
kernel crate
macros crate
builtins crate
exports
helpers
bindgen
bindings crate


Official Debian kernel packages
(native build)
User’s out-of-tree module
(cross build)
Host X
Build X
Build Y