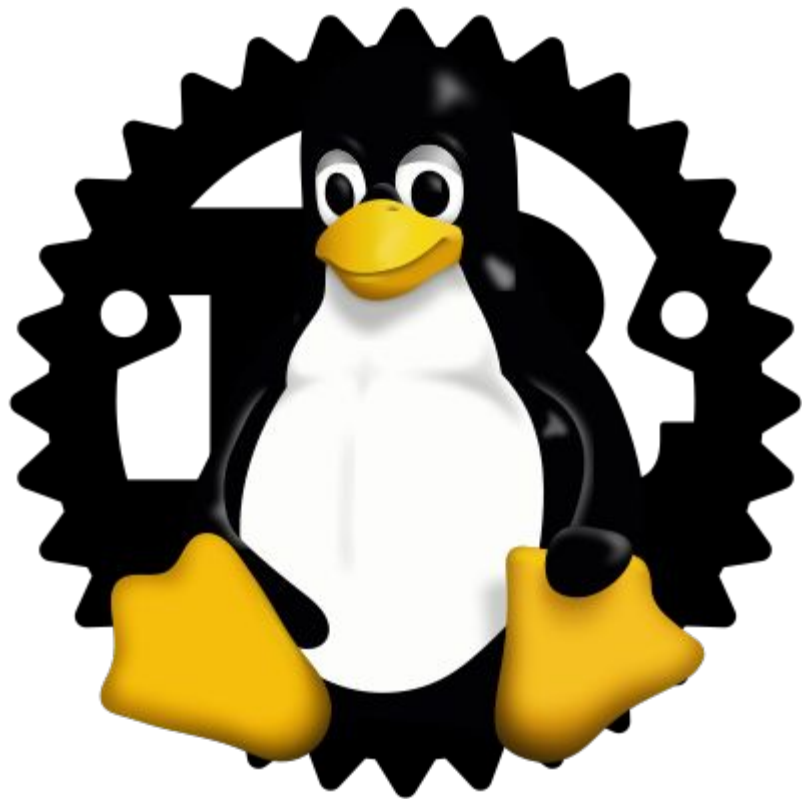




Rust for Linux

Miguel Ojeda

ojeda@kernel.org

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

~4,400 patch series (*up from ~2,500 last year*).

~500 submitters (*up from ~300 last year*).

Plus ~700 PRs early on in GitHub.

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

~170 PRs to Linus Torvalds (*up from ~100 last year*).

~32 Rust-related entries in MAINTAINERS (*up from ~20 last year*).

~40 maintainers/reviewers (*up from ~30 last year*).

~11 active core team members (*up from ~7 last year*).

Growing Team

Daniel Almeida

Tamir Duberstein

Alexandre Courbot

Onur Özkan

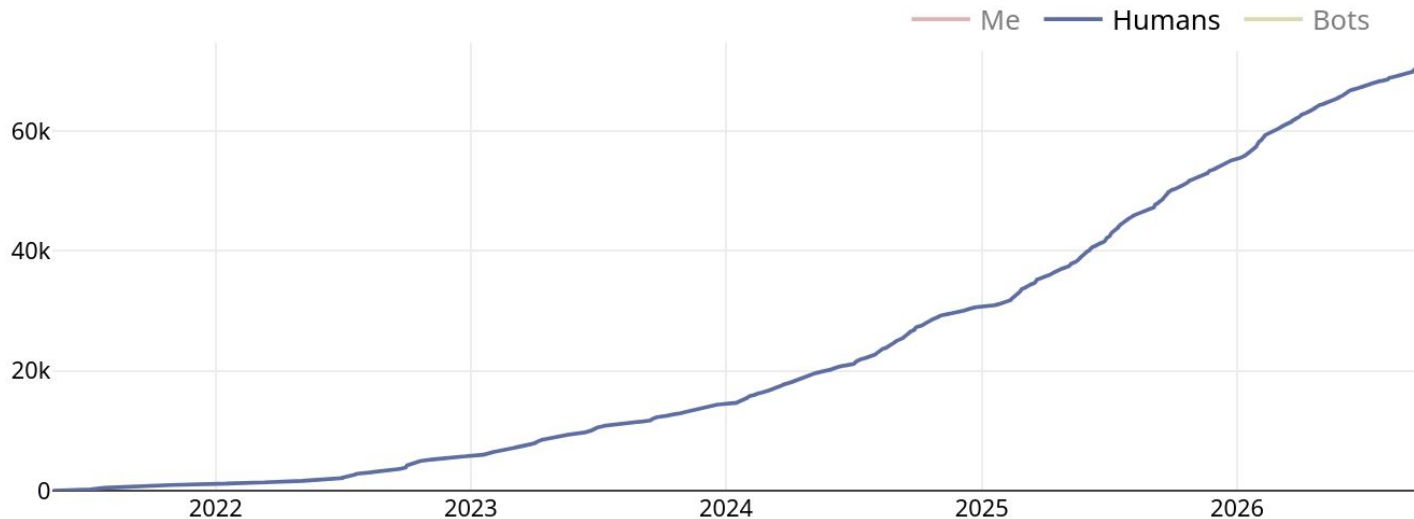
RUST

M: Miguel Ojeda <ojeda@kernel.org>
R: Boqun Feng <boqun@kernel.org>
R: Gary Guo <gary@garyguo.net>
R: Björn Roy Baron <bjorn3_gh@protonmail.com>
R: Benno Lossin <lossin@kernel.org>
R: Andreas Hindborg <a.hindborg@kernel.org>
R: Alice Ryhl <aliceryhl@google.com>
R: Trevor Gross <tmgross@umich.edu>
R: Danilo Krummrich <dakr@kernel.org>
R: Daniel Almeida <daniel.almeida@collabora.com>
R: Tamir Duberstein <tamird@kernel.org>
R: Alexandre Courbot <acourbot@nvidia.com>
R: Onur Özkan <work@onurozkan.dev>

Growing Community

~70k messages sent in the Zulip instance (i.e. chat).

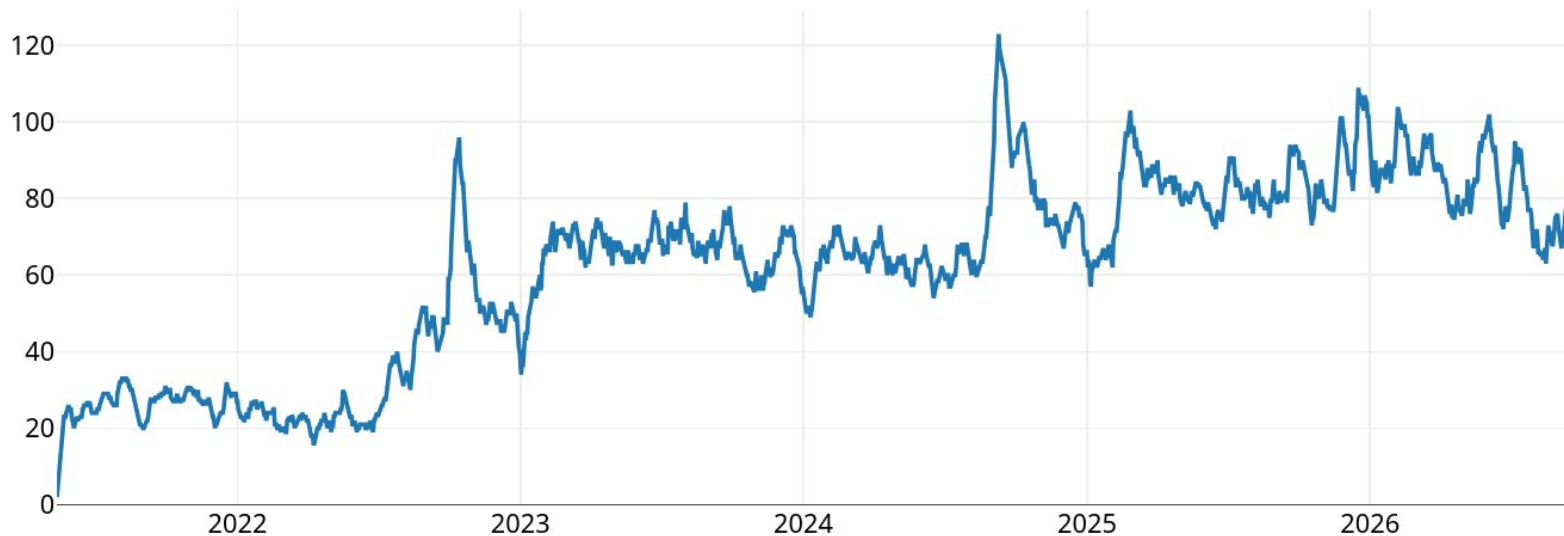
Up from ~54k a year ago.



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

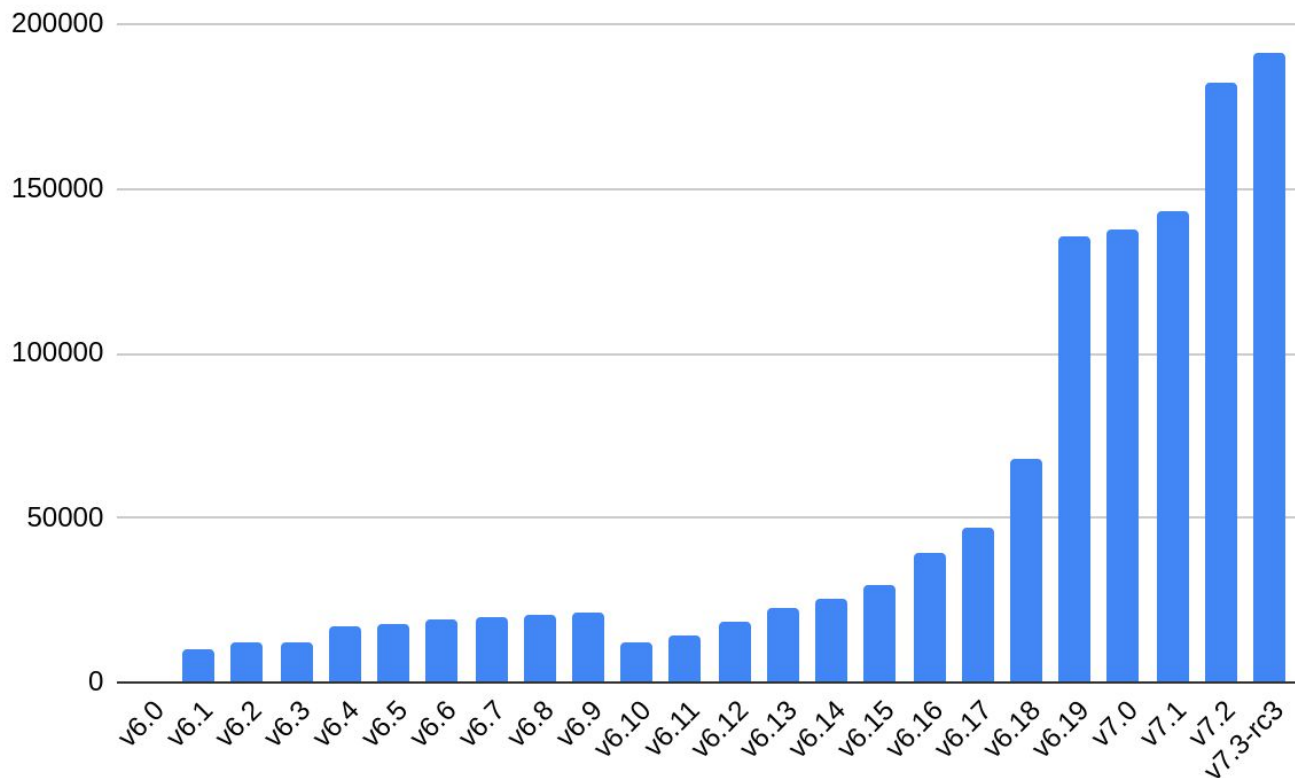
Growing Community

Active users — spikes are due to news/events.

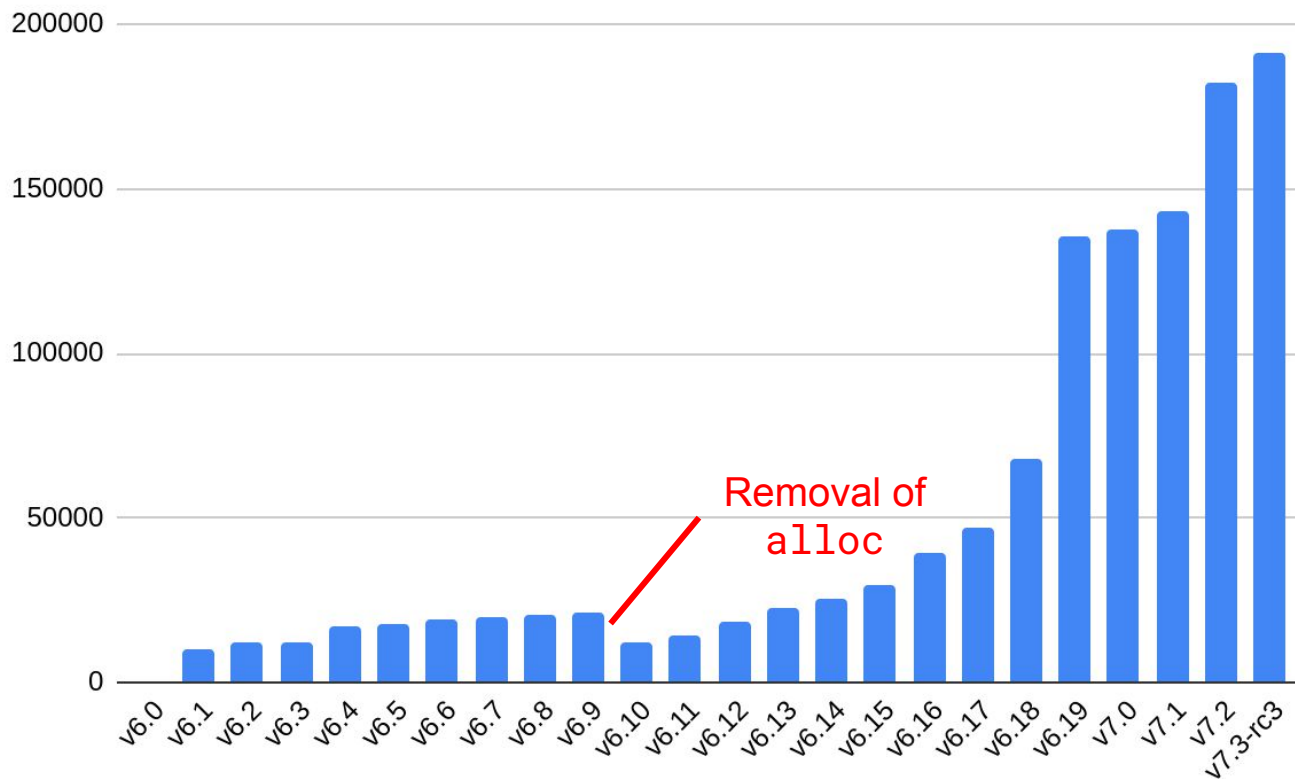


— <https://rust-for-linux.zulipchat.com/stats>

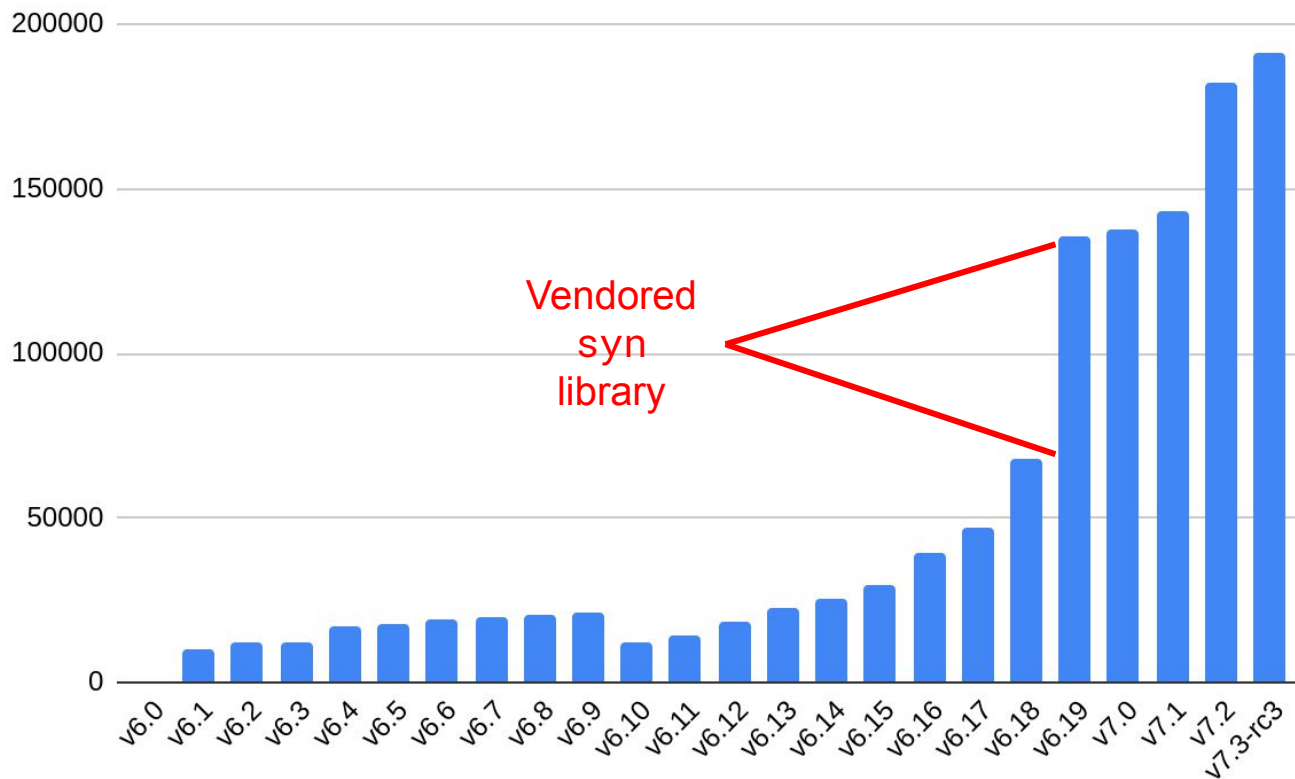
Lines of Rust code in Linux



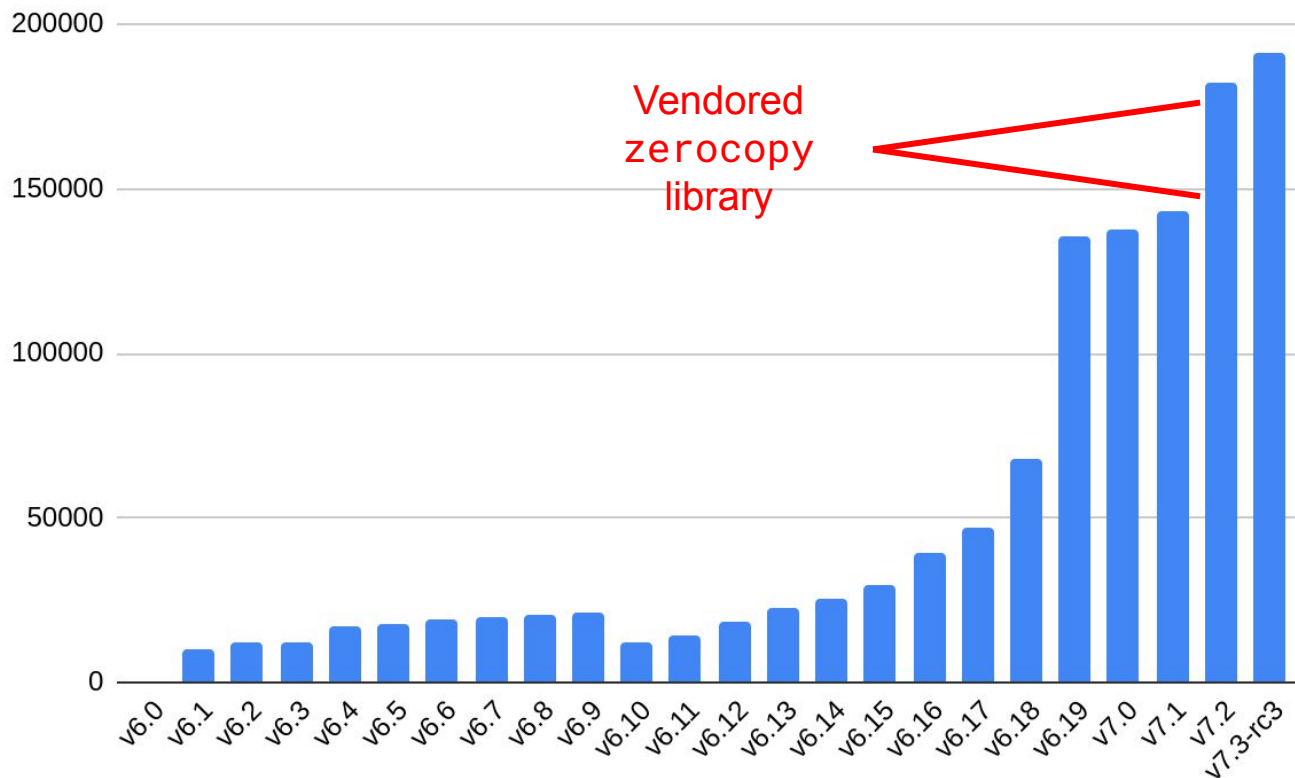
Lines of Rust code in Linux



Lines of Rust code in Linux



Lines of Rust code in Linux



What is zerocopy?

*“Fast, safe, **compile error**. Pick two.*

Zerocopy makes zero-cost memory manipulation effortless. We write `unsafe` so you don't have to.”

Among the **most downloaded Rust crates** (top #50 recent, top #100 all-time).

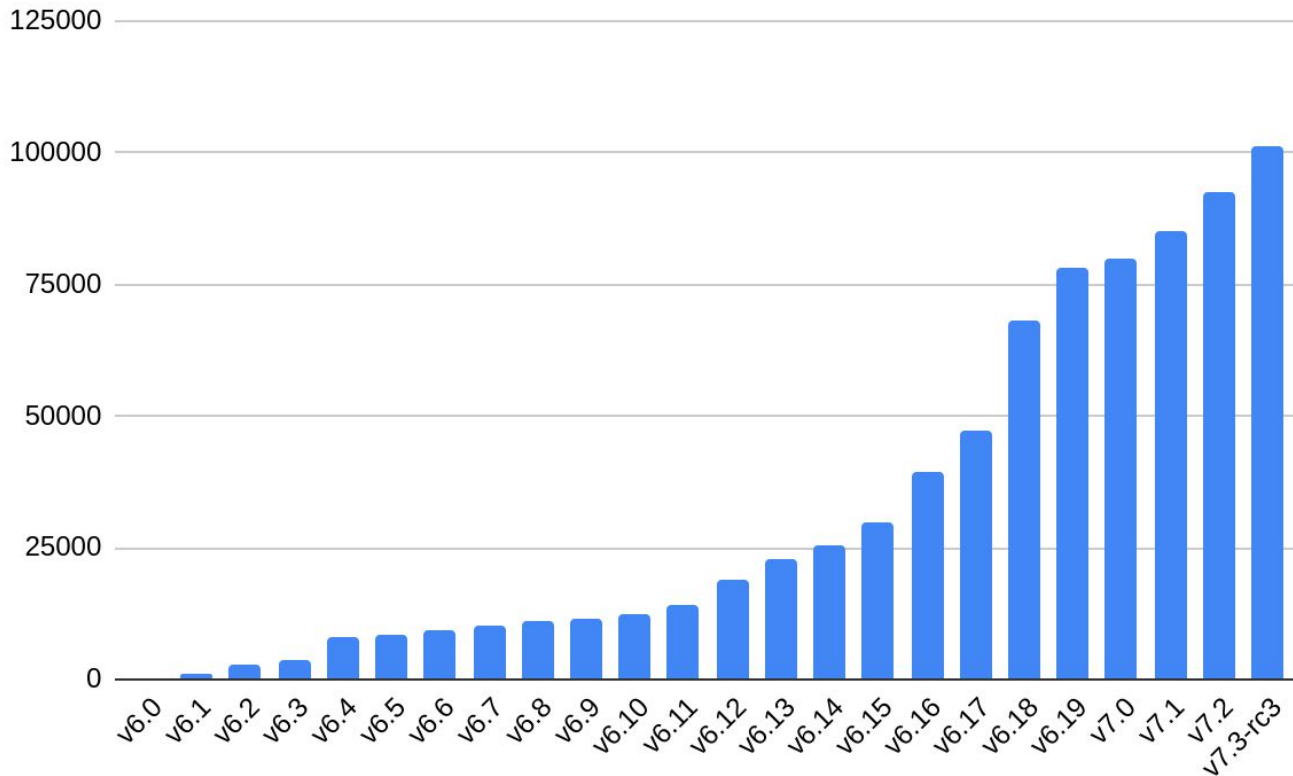
Used by the Rust compiler itself as well, like `syn`.

Allows us to **safely convert between byte sequences and other types**.

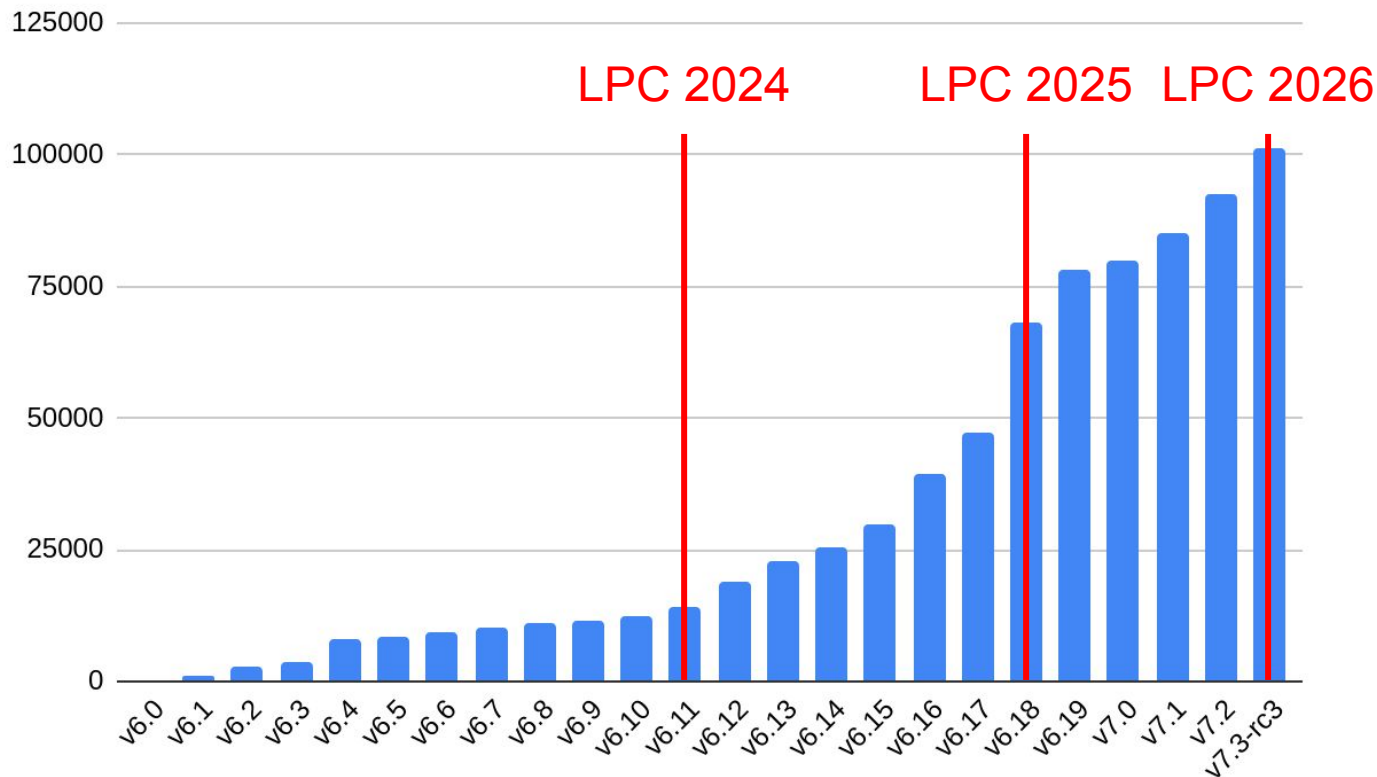
Thus allows us to **remove some `unsafe` code**.

- <https://crates.io/crates/zerocopy>
- <https://crates.io/crates?sort=downloads>
- <https://rust-for-linux.com/third-party-crates>

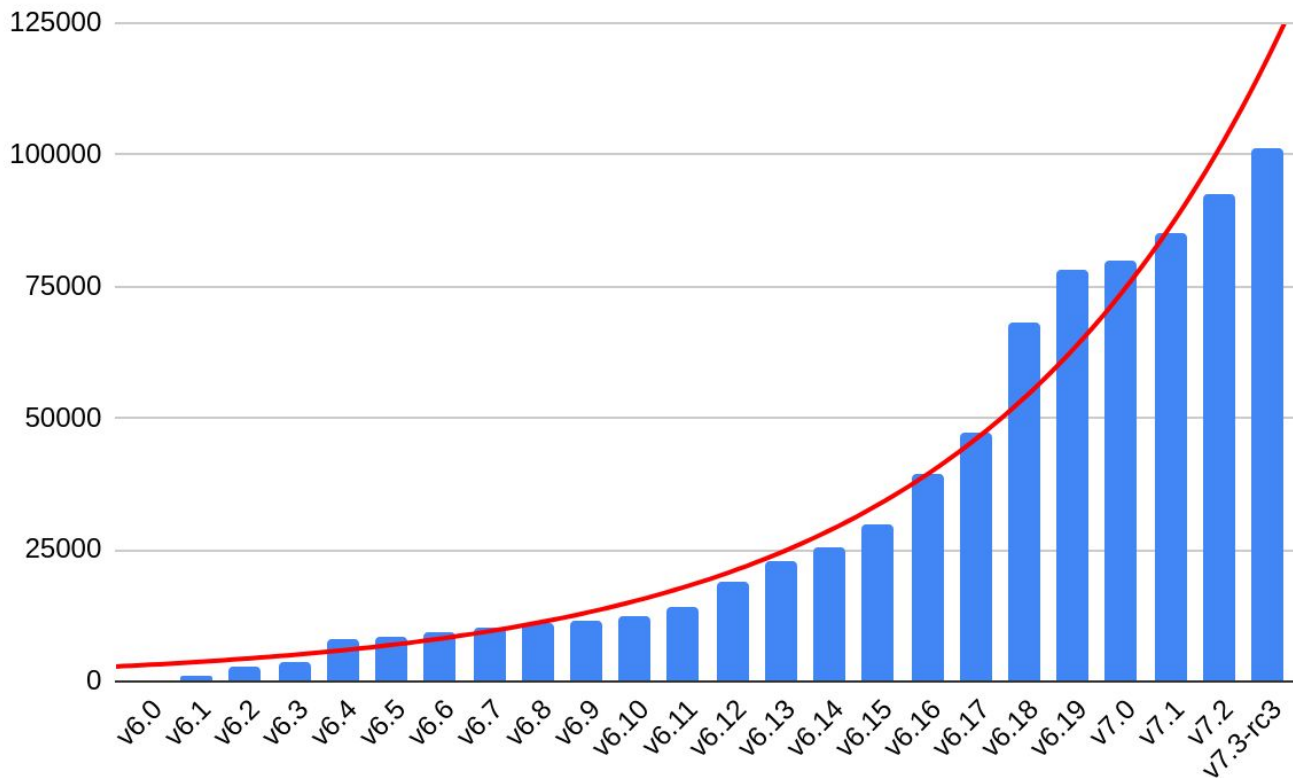
Lines of Rust code in Linux (no vendored code)



Lines of Rust code in Linux (no vendored code)



Lines of Rust code in Linux (no vendored code)



Major milestones

Major milestones

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.

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>

Major milestones

**“Android’s 6.12 Linux kernel is
our first kernel with Rust support enabled and
our first production Rust driver.”**

— <https://rust-for-linux.com/android-%60ashmem%60>

— <https://security.googleblog.com/2025/11/rust-in-android-move-fast-fix-things.html>

Major milestones

**“Android’s 6.12 Linux kernel is
our first kernel with Rust support enabled and
our first production Rust driver.”**

**⇒ ~Hundreds of millions of devices
already running Rust for Linux.**

Major milestones

**“Android’s 6.12 Linux kernel is
our first kernel with Rust support enabled and
our first production Rust driver.”**

**⇒ ~Hundreds of millions of devices
already running Rust for Linux.**

Eventually, billions.

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

Finally!

```
Author: Carlos Linares López <llinares@kernel.org>  
Date: Sat Sep 12 21:41:10 2020 +0000  
  
binder: re-o-f binder.c  
  
The day has finally come. We are dropping the legacy C implementation  
of the Binder IPC driver in favor of its Rust version.  
  
For 11+ years, the C driver has grown increasingly complex, making it  
increasingly painful to maintain and land new features without tripping  
over vulnerabilities. The Rust implementation alleviates most of  
these issues, so it's time for a change.  
  
Alice Ryhl has worked hard on the Rust binder for the past couple of  
years, not only achieving full feature parity but also proving that it  
can match and often beat the performance of the C counterpart, having  
run successfully on Android devices for some time now, we can no longer  
call this an "experimental".  
  
It's time to move on, yank the C code, and focus all new features and  
optimizations on the Rust driver.  
  
Long live the new Rust Binder king!  
  
Cc: rust-for-linux@kernel.org  
Signed-off-by: Alice Ryhl <alicer@ryhl.net>  
Signed-off-by: Carlos Linares <llinares@kernel.org>  
Link: https://patch-work.kernel.org/r=cc-binder@people.com  
Signed-off-by: Greg Kroah-Hartman <gregkh@linuxfoundation.org>  
  
drivers/android/binder.c 7100  
drivers/android/binder_alloc.c 1499  
drivers/android/binder_alloc.h 169  
drivers/android/binder_deferred.h 507  
drivers/android/binder_deferred.c 22  
drivers/android/binder_deferred.h 266  
drivers/android/binder_deferred.c 766  
drivers/android/binder_deferred.c 189  
@ files changed, 10537 deletions(-)
```

Finally!

```
commit fd2bc059ed466bdd95347a4f7891d150d94e7de2
```

```
Author: Carlos Llamas <cmlamas@google.com>
```

```
Date: Sat Sep 12 21:41:10 2026 +0000
```

```
binder: rm -f binder.c
```

The day has finally come. We are dropping the legacy C implementation of the Binder IPC driver in favor of its Rust version.

For 15+ years, the C driver has grown increasingly complex, making it incredibly painful to maintain and land new features without tripping over vulnerabilities. The Rust implementation alleviates most of these issues, so it's time for a change.

Alice Ryhl has worked hard on the Rust binder for the past couple of years, not only achieving full feature parity but also proving that it can match and often beat the performance of the C counterpart. Having run successfully on Android devices for some time now, we can no longer call this an "experiment".

It's time to move on, yank the C code, and focus all new features and optimizations on the Rust driver.

Long live the new Rust Binder king!

Cc: rust-for-linux@vger.kernel.org

Acked-by: Alice Ryhl <aliceryhl@google.com>

Signed-off-by: Carlos Llamas <cmlamas@google.com>

Link: <https://patch.msgid.link/rm-c-binder@google.com>

Signed-off-by: Greg Kroah-Hartman <gregkh@linuxfoundation.org>

```
drivers/android/binder.c  
drivers/android/binder_alloc.c  
drivers/android/binder_alloc.h  
drivers/android/binder_internal.h  
drivers/android/binder_netlink.c  
drivers/android/binder_netlink.h  
drivers/android/binder_trace.h  
drivers/android/binderfs.c  
drivers/android/dbitmap.h  
9 files changed, 10837 deletions(-)
```

```
7186  
1409  
189  
597  
32  
21  
448  
786  
169
```

Major milestones

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

Major milestones

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

Available in **Forky** (next Stable).

And in **Trixie** (current Stable) via `trixie-backports` (opt-in).

Toolchain and infrastructure

Collaboration with upstream 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”.

2025H2:

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

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

Plus **Field Projections**, **In-place initialization** and `build-std`.

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>
- <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 upstream Rust

For **2026**, an ongoing **roadmap**:

A “zoomed out” view of the Rust project direction.

It includes **13 project goals** with a 2026-2027 timespan.

Plus **18 other topics** of importance to the Linux kernel.

5.7. Rust for Linux

- Summary
- Motivation
 - The status quo
 - Design axioms
 - What we are shooting for
 - How we get there
- Funding
- Frequently asked questions
 - When is this done?
 - How do the goals relate to each other?
- Other topics

5.8. Safety-Critical Rust

5.9. The Borrow Checker Within

6. Goals

7. Frequently asked questions

🔧 2025H2 goal process

8. Overview

9. Goals

10. Goals not accepted

11. Reports

📖 Learn about

12. Goal format

13. Goal motivations

14. Roadmaps

15. Invited goals

16. Contact

17. Team asks

18. Task owners

19. Design axioms

20. RFC

Rust Project Goals		
What we are shooting for Build Linux kernel releases using the stable Rust language (no feature gates), support all the targets that Linux supports, and address the long tail of features Rust for Linux needs.		
How we get there		
Goal	Timespan	What and why
Arbitrary self types	2026	Let user-defined smart pointers work as method receivers, stabilizing the <code>arbitrary_self_types</code> feature.
Deref / Receiver split chain experiment	2026	Experiment with letting <code>Receiver::Target</code> and <code>Deref::Target</code> diverge, collecting data on utility and use cases.
<code>derive(CoercePointee)</code>	2026	Support <code>dyn Trait</code> coercion for user-defined smart pointers.
build-std	2026	Let Cargo rebuild the standard library from source for custom targets and configurations
Stabilize const traits MVP	2026	Finalize the RFC, complete the compiler implementation, and stabilize so <code>const fn</code> can call trait methods.
Map the field projection design space	2026-2028	Explore the virtual places approach, document it in the beyond-refs wiki, formalize borrow checker integration, and build a compiler experiment.
ADT const params	2026	Support structs, tuples, arrays in const generics.
Min generic const arguments	2026	Support associated constants and generic parameters embedded in other expressions.
Move trait	2026-2027	Let types opt out of being relocated in memory, encoding immovability as a type property rather than a place property.
Guaranteed destructors	2026-2027	Explore letting types opt out of <code>mem::forget</code> , enabling patterns like safe scoped spawn for

Collaboration with upstream 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.

`-Zprofile-sample-use` for AutoFDO stabilized for Rust 1.100.

`-Zsanitize=kernel-hwaddress` for arm64 added as unstable in Rust 1.96.

`rustdoc's --remap-path-prefix` stabilized in Rust 1.97.

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

— <https://github.com/rust-lang/goals/issues/407>

Collaboration with upstream Rust

Even **library** ones: `{highest, lowest}_one (ffs, fls)` stabilized in Rust 1.97.

A **future** compiler and language one: `-Coverflow-checks=report`.

Rust supports two overflow behaviors: panicking or wrapping (EB).

Reporting would be a good “middle ground” between those, for Linux.

Thus, we would like to customize what happens when overflow happens.

Ells (Externally Implementable Items) allow to have “forward declarations” in Rust: a crate can declare e.g. a function that then a downstream crate can implement (customize).

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

— <https://github.com/rust-lang/rust/issues/125418>

Linux in Rust's and bindgen's CI

It has been 2 years that **every Rust PR now build-tests the Linux kernel**.

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

From **experience**, in general, apart from intentional changes, the upcoming Rust compiler versions generally work (“nightlies”).

We try our best to **proactively fix nightlies** as soon as possible, so that we are ahead of the curve.

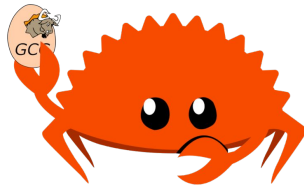
Generally, fixes hit the LTS kernels before compilers are even released!

bindgen also build-tests Linux in its CI.

It has worked well so far throughout these 2 years.

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

gccrs



`core` compiles until lowering.

Now **entirely name resolved**, which has been a major milestone.

When `core` works well, building the rest of the kernel is not far away.

"**Compatibility mode**" for the Linux kernel to support features defined in later language versions even when using an older `core` version.

Already added modern features that Linux uses such as `offset_of!`.

Performance work: “not as fast as `rustc`, but **manageable now**”.

More active members: 6 core contributors, 5 non-core contributors.

CI building the kernel weekly.

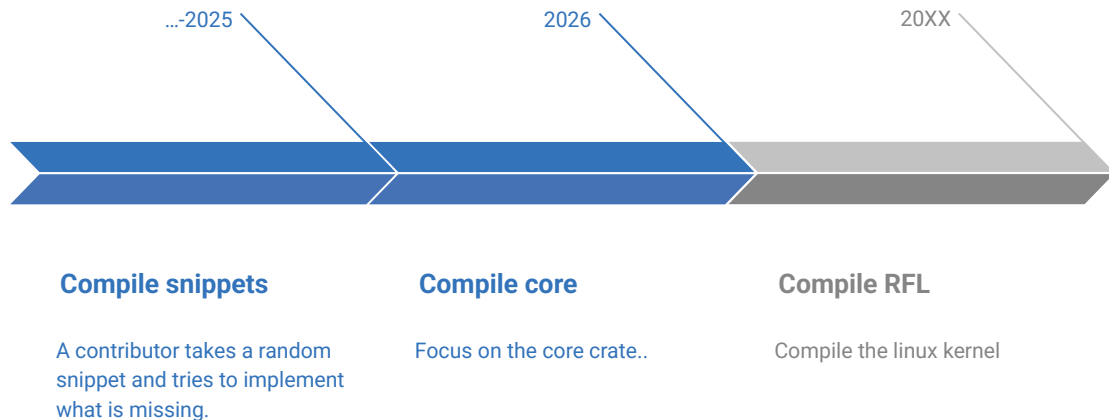
More details in yesterday’s LWN article: <https://lwn.net/Articles/1095553/>

— Pierre-Emmanuel Patry and Arthur Cohen

— <https://rust-gcc.github.io>



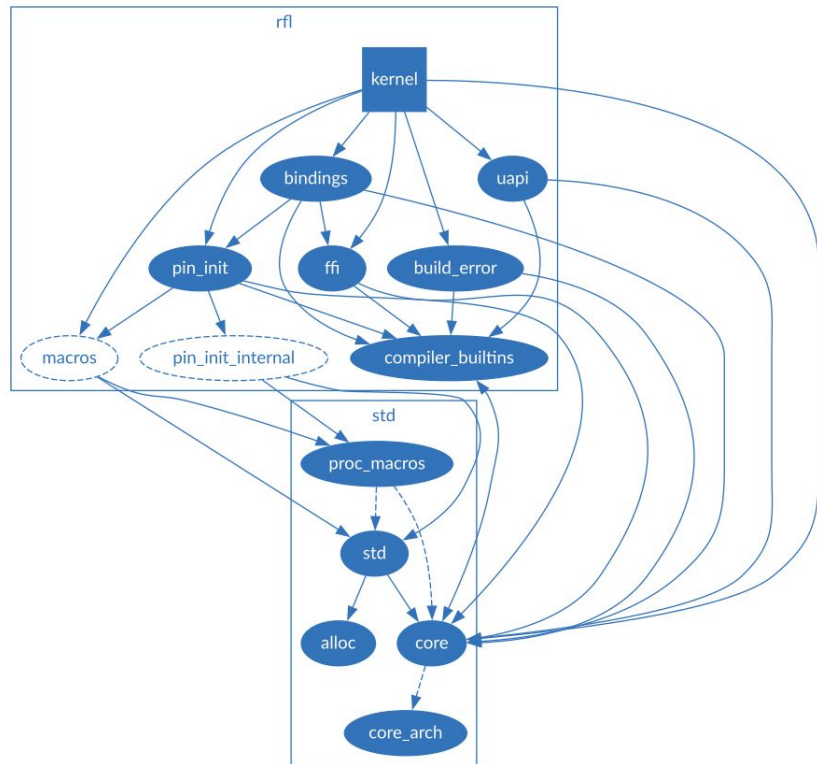
Development focus





Progress summary

- `core` compiles until lowering
 - We thought we would be able to compile `core` for `rustconf`
- `ffi` compiles properly
- `build_error` compiles properly
- `alloc` compiles properly
 - Requires a mocked `core`
 - Only compiles with edition 2015

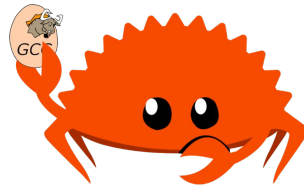




Next steps

- Catching up to stable Rust instead of having a chimera compiler
 - Most of the changes are features being stabilized, APIs being added
 - The language itself does not change *that much*
 - Trait upcasting
 - `use<...>` syntax
 - RPITIT
- Ensure the kernel gets compiled correctly
- Ensure that the behavior of binaries produced by both compilers matches

gccrs



More details as well in Philip Herron's GCC Rust talk yesterday.

Upcoming: Compiler Explorer support for **GCC Rust targeting BPF**.

It is **another good initial use case** of GCC Rust, apart from the kernel.

rustc_codegen_gcc

Implemented:

Fixed issues with **LTO**.

Cross-language LTO.

Run most UI tests for `rustc_codegen_gcc` **in Rust CI**.

Run the libcore's tests in the CI of the Rust repo.

`va_start` and `va_arg`.

Guaranteed tail calls.

Stack probes.

Stack protector.

Unwinding is now working well even in release mode.

Many bug fixes.

rustc_codegen_gcc

Milestones:

`rustc_codegen_gcc` can now build a non-working `rustc` for m68k.

`rustc_codegen_gcc` is now **distributed via rustup for Linux x86-64**.

The number of failing UI tests from the Rust test suite is down to 20.

rustc_codegen_gcc

Plans for the future:

Short term:

Finish implementing function and variable attributes (most of them are supported, but some of them are currently not supported).

ABI parity with `rustc_codegen_llvm`.

Building `rustc` for other targets: next are **Aarch64**, m68k, DEC Alpha.

Running a crater run.

Medium term:

Improve the support for debug info.

SIMD for other architectures than x86-64.

CoccinelleForRust



- Automated code refactoring tool for large-scale rust repositories
- Uses **SeMantic Patch Language**(SMPL) to specify semantic changes in a rust codebase, which can be used for identifying as well as transforming parts of the repository
- Use cases: collateral evolution of codebases to follow new standards and APIs, fixing known bugs, perform analysis using inbuilt python scripting, etc.

Updates (2026)

- Improved engine for faster matching and transformation
- Better integration for control flow and disjunctions
- Miscellaneous bug fixes

— Tathagata Roy and Julia Lawall

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

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

CoccinelleForRust



Project page: <https://rust-for-linux.com/coccinelle-for-rust>

Repository: <https://gitlab.inria.fr/coccinelle/coccinelleforrust>

Contact:

tathagata.roy1278@gmail.com

julia.lawall@inria.fr

Please feel free to reach out to us if you have any queries!

— Tathagata Roy and Julia Lawall

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

Rust minimum version upgrade

Last LPC we proposed **to follow Debian Stable's Rust version.**

i.e. **upgrade** the minimum **every ~2 years** (a few months after Debian gets released).

For instance, we upgraded the minimum to **Trixie's Rust 1.85** (2025-02-20) from the previous 1.78 (2024-05-02).

In about a year from now, we will upgrade again to **Forky's expected Rust ~1.104.0** (2027-04-29).

It is still a smaller window than C.

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

A possible next step would be to increase the window to **“2 Debians”** later on.

The issue of Rust Editions and backporting

Rust Editions are used to introduce **backwards-incompatible changes** into the Rust language.

The kernel currently uses **Rust 2021**.

Using **newer editions**, such as Rust 2024, **would improve our code**.

e.g. new features like let chains and less surprises with drop order.

The issue of Rust Editions and backporting

Rust Editions are used to introduce **backwards-incompatible changes** into the Rust language.

The kernel currently uses **Rust 2021**.

Using **newer editions**, such as Rust 2024, **would improve our code**.

e.g. new features like let chains and less surprises with drop order.

However, there is an issue: what do we do about **backports**?

The issue of Rust Editions and backporting

Rust 2024 changes the meaning of existing code.

i.e. it may still successfully build, but with its semantics changed... subtly so!

Moving mainline **forward is fine — tooling can assist.**

We can review the places that need attention, and it is not too much code yet.

The issue of Rust Editions and backporting

Rust 2024 changes the meaning of existing code.

i.e. it may still successfully build, but with its semantics changed... subtly so!

Moving mainline **forward is fine** — **tooling can assist**.

We can review the places that need attention, and it is not too much code yet.

However, patches with `Fixes:` and/or `Cc:` stable@vger.kernel.org may be “automatically” applied if there are no conflicts.

So what happens if stable takes a fix written for Rust 2024 into a kernel using an older Edition?

The issue of Rust Editions and backporting

Rust 2024 changes the meaning of existing code.

i.e. it may still successfully build, but with its semantics changed... subtly so!

Moving mainline **forward is fine — tooling can assist.**

We can review the places that need attention, and it is not too much code yet.

However, patches with `Fixes:` and/or `Cc:` stable@vger.kernel.org may be “automatically” applied if there are no conflicts.

So what happens if stable takes a fix written for Rust 2024 into a kernel using an older Edition?

⇒ **The fix may actually break the stable kernel.**

The issue of Rust Editions and backporting

There are several “easy” solutions, e.g.

- Keep the kernel forever on Edition 2021.

- Moving all stable kernels to the new edition (!).

The issue of Rust Editions and backporting

There are several “easy” solutions, e.g.

- Keep the kernel forever on Edition 2021.

- Moving all stable kernels to the new edition (!).

The one I would like:

- An automated **tool** that **detects** if a patch has **different semantics** with **zero false negatives**.

- Then the stable kernel team can email if an “edition conflict” happens.

The issue of Rust Editions and backporting

There are several “easy” solutions, e.g.

- Keep the kernel forever on Edition 2021.

- Moving all stable kernels to the new edition (!).

The one I would like:

- An automated **tool** that **detects** if a patch has **different semantics** with **zero false negatives**.

- Then the stable kernel team can email if an “edition conflict” happens.

If you or your company are interested in finding or funding a solution to this issue (perhaps because you maintain stable branches in other projects), please ping me!

bindgen and GCC

`rustc_codegen_gcc` and/or GCC Rust are coming.

However, there is still a piece missing to have a **kernel fully built with GCC**.



Linux tree

drivers/

foo/

my_foo
driver

Safe

kernel
crate

foo
subsystem

bar
subsystem

...

Safe Abstractions

Unsafe

include/

bindgen

bindings
crate

Forbidden!

bindgen and GCC

`rustc_codegen_gcc` and/or GCC Rust are coming.

However, there is still a piece missing to have a **kernel fully built with GCC**.

bindgen uses libclang to parse C headers and generate Rust bindings.

Discussed in GNU Cauldron on Sunday and in LPC's Toolchain track as well.

Two major approaches available:

Teach `bindgen` to query/use GCC.

Support for a GCC-based backend #1949

New issue

Open

ojeda opened on Dec 20, 2020 Last edited by ojeda Member

Currently, `bindgen` uses `libclang` to understand C/C++ code. #1740 considers moving to a lower-level interface for LLVM, with discussion of whether keeping support for several interfaces is feasible.

It would be great to have support for non-LLVM backends as well. In particular, a GCC-based one. Clang is very compatible with GCC nowadays (in both the Clang driver and the support for GNU C language extensions), but there are still differences here and there, which means one needs to be careful when applying `bindgen` in complex projects compiled with GCC.

It would require compiling a GCC plugin and using callbacks like `PLUGIN_PRE_GENERICIZE` ("Allows to see low level AST in C and C++ frontends.") and/or others like `PLUGIN_FINISH_PARSE_FUNCTION`, `PLUGIN_FINISH_TYPE`, `PLUGIN_FINISH_DECL`, etc.; or perhaps another way to use GCC's parser/tooling/etc.; or possibly using other approaches that fetch the information in another way (e.g. querying the compiler via generated C files or extracting it from debug information).

Of course, tackling this would be non-trivial, and it might not be worth the maintenance cost, but it might be an interesting project to expand `bindgen`'s capabilities, as well as relaxing the dependency on LLVM.

Related: #2962.

Create sub-issue 24

emilio added enhancement on Dec 20, 2020

ojeda mentioned this on Apr 5, 2021

GCC support Rust-for-Linux/linux#167

fengguang added a commit that references this issue on Apr 14, 2021

Assignees

No one - Assign yourself

Labels

enhancement rust-for-linux

Type

No type

Fields

Priority Choose an option

Projects

No projects

Milestone

No milestone

Relationships

None yet

Development

Create a branch for this issue or link a pull request.

bindgen and GCC

`rustc_codegen_gcc` and/or GCC Rust are coming.

However, there is still a piece missing to have a **kernel fully built with GCC**.

bindgen uses libclang to parse C headers and generate Rust bindings.

Discussed in GNU Cauldron on Sunday and in LPC's Toolchain track as well.

Two major approaches available:

- Teach `bindgen` to query/use GCC.

- New, equivalent tool in GCC itself (several possibilities here).

 - It would have to output ideally the exact same code as `bindgen`.

 - Shared infrastructure to benefit several frontends?

Events

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 2026
Las Palmas, Canary Islands



Related LPC sessions

16+ Rust for Linux related sessions across a few tracks.

Detailed Schedule: <https://lpc.events/event/20/timetable/?view=lpc>

Rust for Linux Office Hours (Monday): <https://lpc.events/event/20/contributions/2382/>

The Nova GPU & DRM Rust Workshop (Monday): <https://lpc.events/event/20/sessions/277/>

The **Rust MC** (tomorrow): <https://lpc.events/event/20/sessions/257/>

Benno's **Field Projections** talk (right after this one): <https://lpc.events/event/20/contributions/2376/>

Plus nearby events:

GNU Cauldron's Toolchain and Linux Kernel BoF (Sunday):
<https://conf.gnu-tools-cauldron.org/prg26/talk/TRZN7Z/>

Rust for Linux Training Day (Sunday): <https://lore.kernel.org/rust-for-linux/rust-at-lpc2026@google.com/>

Sponsors & Industry support



NETFLIX



SAMSUNG



arm



— <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/>

The future

Rust growing outside the kernel world

A dark blue horizontal banner with faint, stylized gear patterns in a lighter blue shade on the left and right sides. The text is centered in white.

Guest Post: Rust Is Tier-1 Language at Microsoft

September 10, 2026 / Victor Ciura

— <https://rustfoundation.org/media/guest-post-rust-is-tier-1-language-at-microsoft/>

Rust growing outside the kernel world

“...and one of the things that I realized a long time ago is that no matter how much we really want to make C and C++ better, we can't make it as good as what Rust starts with.”

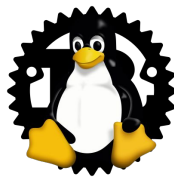
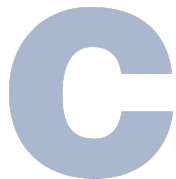
— Mark Russinovich, CTO of Microsoft Azure

— <https://www.youtube.com/watch?v=uDtMuS7BExE>

Device Resources — DMA Coherent

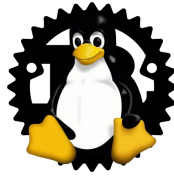
```
1 pub struct Coherent<'a, T: KnownSize + ?Sized> {
2     dev: &'a device::Device<Bound>,
3     dma_addr: DmaAddress,
4     cpu_addr: NonNull<T>,
5     dma_attrs: Attrs,
6 }
7
8 impl<'a, T: AsBytes + FromBytes> Coherent<'a, T> {
9     /// Allocates a region of `T` of coherent memory.
10    fn alloc_with_attrs(
11        dev: &'a device::Device<Bound>,
12        gfp_flags: kernel::alloc::Flags,
13        dma_attrs: Attrs,
14    ) -> Result<Self>;
15 }
```

The future



The future

C

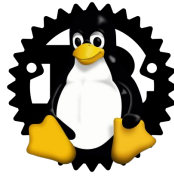
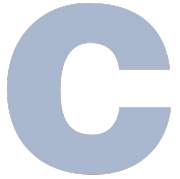


?

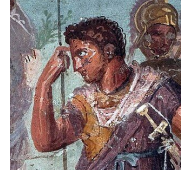


It is not about C or Rust. It is about making kernel and systems programming in general, better.

The future

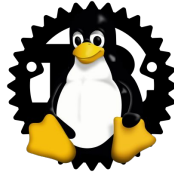
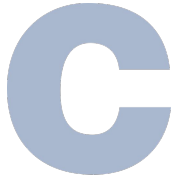


Formal
Verification

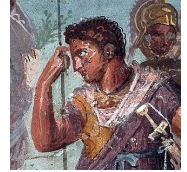


LEAN
THEOREM PROVER

The future

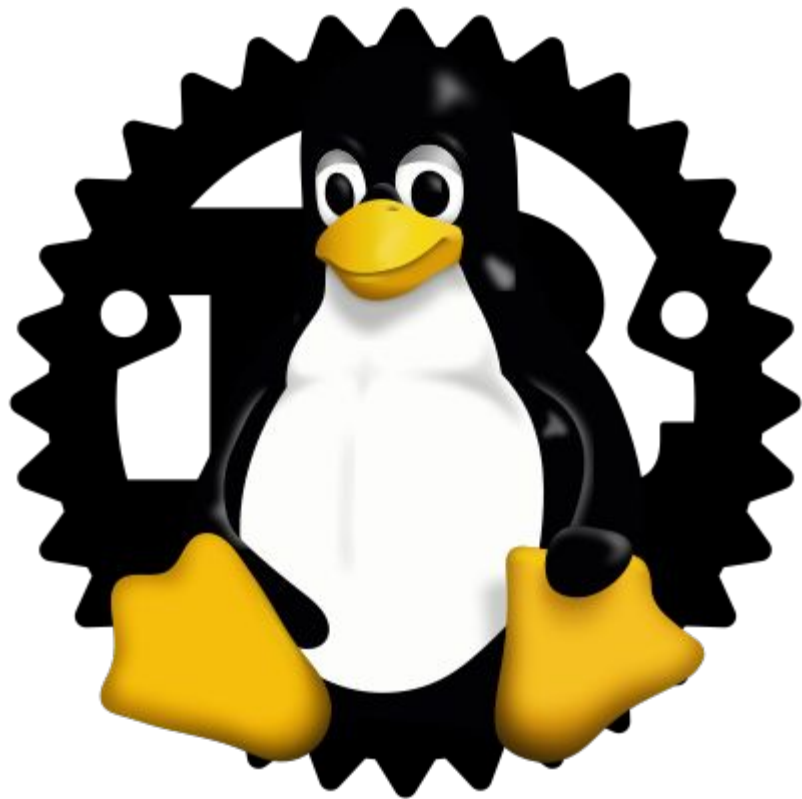


AI-assisted
Formal
Verification



LEAN
THEOREM PROVER

Thank you!



Rust for Linux

Miguel Ojeda

ojeda@kernel.org