Rust for Linux

User guidance

Kangrejos 2024

Dirk Behme <dirk.behme@gmail.com>

Abstract:

The Rust for Linux (RFL) project is gaining momentum. More and more abstractions are developed and discussed. First basic abstractions are even merged in mainline. The change rate increases. While this is really good, it increases the effort to keep an overview, as well. And this overview is even needed for *users* of the RFL abstractions. Typical user questions are for example: How do I use abstraction X? Or: I use abstraction Y from x month ago. Now it has changed. What is changed? How do I adapt to that change? Or: What is new in latest mainline and rust-next/devel branch? Discuss these topics and talk about options to support users for easy RFL usage.

(skip this page at presentation)

Introduction

- For beginners, Rust is said to have a steep learning curve

- Rust for Linux (RFL) even goes ‘on top’ of Rust in the sense of
  - it is slightly different to ‘normal’ Rust (e.g. no_std etc)
  - users need to use abstractions
  - heavy development implying frequent changes

How can we make life easier for RFL users?

Introduction (contd)

At the moment, typically the developer of an abstraction and the user of the abstraction (driver, file system etc) are the *same*. I.e. the user of the abstraction develops the required abstraction, as well. With this, the developer ‘knows all about the abstraction’.

However, there will be more and more *users* of RFL. Without much or deep knowledge of the RFL details and the abstractions.

This will result in questions like:

- How do I use abstraction X?
- I use abstraction Y from x month ago. Now it has changed. What is changed?
  How do I adapt to that change?
- What is new in latest mainline and rust-next/devel branch?
- ….

Example #1: How to write a module?

Help > "Mentorship Session: Writing Linux Kernel Modules in Rust" AUG 3

Nikolai Shv 10:31

I'd like to know are many things changed since this video was made?
https://youtu.be/-l-8WrGHEGI?si=s9fbIISVwa8xNpV5

- Should I use origin/rust branch to build kernel?
- Are there any "more up to date" tutorials which I can refer to?

My main goal is to create a linux kernel module in Rust with some C code. Basically it's just a kernel module which was written in C and I want to rewrite it in Rust (openvpn-dco)[https://github.com/OpenVPN/ovpn-dco].
Does this version of linux has enough Rust support to make this kernel module work with Rust?

I'm completely new in systems programming.

Alice Ryhl 11:02

You don't want the rust branch anymore. See the website: https://rust-for-linux.com/

Example #2: How to use the RFL changes from one kernel version to the next one?

General > ✓ How to use new FileShareMode from struct file abstraction? MAY 30

Dirk Behme EDITED 14:02

Switching to recent v6.10-rc1 based rust-dev it looks like the rust: file: add Rust abstraction for struct file has more or less recently added a new FileShareMode.

So far I have used the v6.9-rc1 version of rust-dev which didn't had that.

I did the adaption for my `File` users by adding `<NoFdgetPos>` everywhere. For example:

```
-   _file: &File,
+   _file: &File<NoFdgetPos>,
```

While this seem to make the compiler happy, I'm not sure if this is the way to go?

Do we have any usage examples of this? I.e. how to convert from the v6.9-rc1 version of the struct file abstraction to the v6.10-rc1 one?

Benno Lossin EDITED 14:12

I think @Alice Ryhl is still developing the file API and there will be additional changes (IIRC `NoFdgetPos` will get removed). I haven't yet looked at the newer versions, so I might be wrong, but what generic parameter you pass to `File` depends on how that file is shared.

Alice Ryhl EDITED 14:13

Most likely the next version will rename `File<NoFdgetPos>` to just `File` and `File<MaybeFdgetPos>` to `LocalFile`.

EDITED Just using `File<NoFdgetPos>` everywhere should be fine. 14:16

You

Example #3: How to use the RFL changes from one kernel version to the next one?

General > Abstraction for struct device in driver-core git JUN 19

Dirk Behme 06:49

For everybody not following the mailing list that closely just wanted to mention here that rust: add abstraction for struct device and rust: add firmware abstractions made it into Greg's driver-core-testing. If I understood correctly this is planned to be merged into mainline for upcoming 6.11-rc1.

Many **thanks** to everybody working on this! 👍 😀

🎉 12

Fabien Parent 19:32

For people using some of my branches for development, I'm currently working on rebasing them on top of it. Though it taking a bit more time given the changes made to device::Data.

Example question: What is this change about and what needs to be done to adapt to it?

Example #4

It took me some significant time to figure out how this abstraction works and how it's intended to be used.

The driver just wants to *use* this to parse the device tree …

AsahiLinux / linux

Type / to search

Code | Issues 63 | Pull requests 7 | Actions | Projects | Security | Insights

Commit

rust: of: Add OF node abstraction

```
This abstraction enables Rust drivers to walk Device Tree nodes and
query their properties.

Signed-off-by: Asahi Lina <lina@asahilina.net>
```

gpu/rust-wip-6.3

asahilina authored and **marcan** committed on May 30, 2023

Showing **4 changed files** with **485 additions** and **2 deletions**.

Filter changed files

- rust
  - bindings
    - bindings_helper.h
    - helpers.c
  - kernel
    - device.rs
    - of.rs

3 rust/bindings/bindings_helper.h

```
@@ -10,6 +10,9 @@
 10  10  #include <linux/device.h>
 11  11  #include <linux/io-pgtable.h>
 12  12  #include <linux/ktime.h>
     13 + #include <linux/of.h>
     14 + #include <linux/of_address.h>
     15 + #include <linux/of_device.h>
 13  16  #include <linux/platform_device.h>
 14  17  #include <linux/refcount.h>
 15  18  #include <linux/wait.h>
```

14 rust/helpers.c

```
@@ -337,6 +337,20 @@ const struct of_device_id *rust_helper_of_match_device(
 337  337  }
 338  338  EXPORT_SYMBOL_GPL(rust_helper_of_match_device);
 339  339
      340 + bool rust_helper_of_node_is_root(const struct device_node *np)
      341 + {
      342 +         return of_node_is_root(np);
      343 + }
      344 + EXPORT_SYMBOL_GPL(rust_helper_of_node_is_root);
```

https://github.com/AsahiLinux/linux/commit/9e496b356ee8e25f9bee9258491aa6ae3a4f1ddf

Example #5

rust-for-linux.vger.kernel.org archive mirror

search help / color / mirror / Atom feed

From: Greg KH <gregkh@linuxfoundation.org>
To: Lyude Paul <lyude@redhat.com>
Cc: rust-for-linux@vger.kernel.org,
"Danilo Krummrich" <dakr@redhat.com>,
airlied@redhat.com, "Ingo Molnar" <mingo@redhat.com>,
"Will Deacon" <will@kernel.org>,
"Waiman Long" <longman@redhat.com>,
"Peter Zijlstra" <peterz@infradead.org>,
"Miguel Ojeda" <ojeda@kernel.org>,
"Alex Gaynor" <alex.gaynor@gmail.com>,
"Wedson Almeida Filho" <wedsonaf@gmail.com>,
"Boqun Feng" <boqun.feng@gmail.com>,
"Gary Guo" <gary@garyguo.net>,
"Björn Roy Baron" <bjorn3_gh@protonmail.com>,
"Benno Lossin" <benno.lossin@proton.me>,
"Andreas Hindborg" <a.hindborg@samsung.com>,
"Alice Ryhl" <aliceryhl@google.com>
Subject: Re: [PATCH 0/3] rust: Add irq abstraction, IrqSpinLock
Date: Fri, 26 Jul 2024 07:39:59 +0200 [thread overview]
Message-ID: <2024072626-bluff-remark-3827@gregkh> (raw)
In-Reply-To: <20240725222822.1784931-1-lyude@redhat.com>

On Thu, Jul 25, 2024 at 06:27:49PM -0400, Lyude Paul wrote:
> This adds a simple interface for disabling and enabling CPUs, along with
> the ability to mark a function as expecting interrupts be disabled -
> along with adding bindings for spin_lock_irqsave/spin_lock_irqrestore().

Do you have some example code that actually uses this? Without that, it's hard, if not impossible, to review it to see how it works and if it works properly.

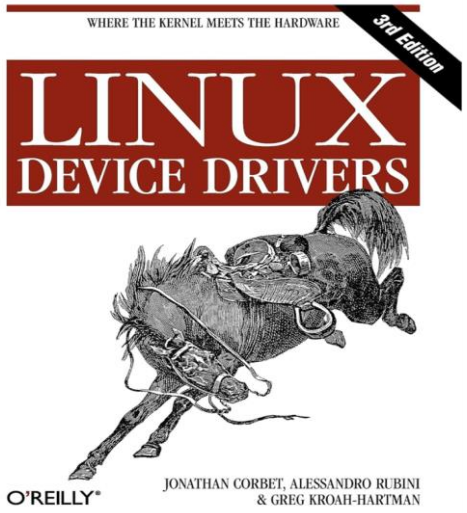
thanks,

greg k-h

https://lore.kernel.org/rust-for-linux/2024072626-bluff-remark-3827@gregkh/

Whats about the C world?

If you are about to write a Linux kernel driver in C, i.e. you want to *use* the various device driver interfaces (in kernel APIs), we usually have various similar existing, tested and mainlined drivers serving as usage example. This existing code gives even beginners and not that experienced developers a path how the in kernel APIs are supposed to be used. Due to the young history of RFL this doesn’t exist for RFL, yet.



Discussion

What options do we see to ease the usage of RFL?

- What's about usage examples in samples/rust/ for each abstraction?
- What's about ensuring that the inline examples / Kunit tests are verbose?
- What's about more documentation about dos and don'ts?
- What's about regular "RFL news"?
- What's about some description what's new in each –rc1, how to use it?
  And what's new in rust-next/devel?
- Do we consider the functionality the abstractions expose as new "API"?
  Do we need a "RFL in kernel (abstraction) API" documentation?

Do we need to "translate" this to RFL (abstractions)?