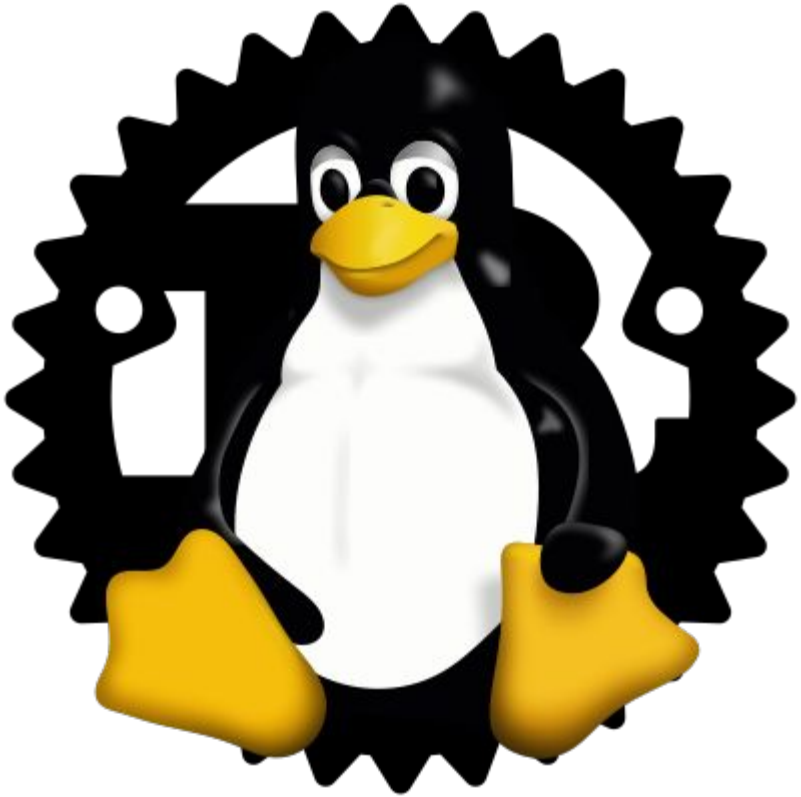
LF *live*

MENTORSHIP SERIES



Rust for Linux: Writing Safe Abstractions & Drivers

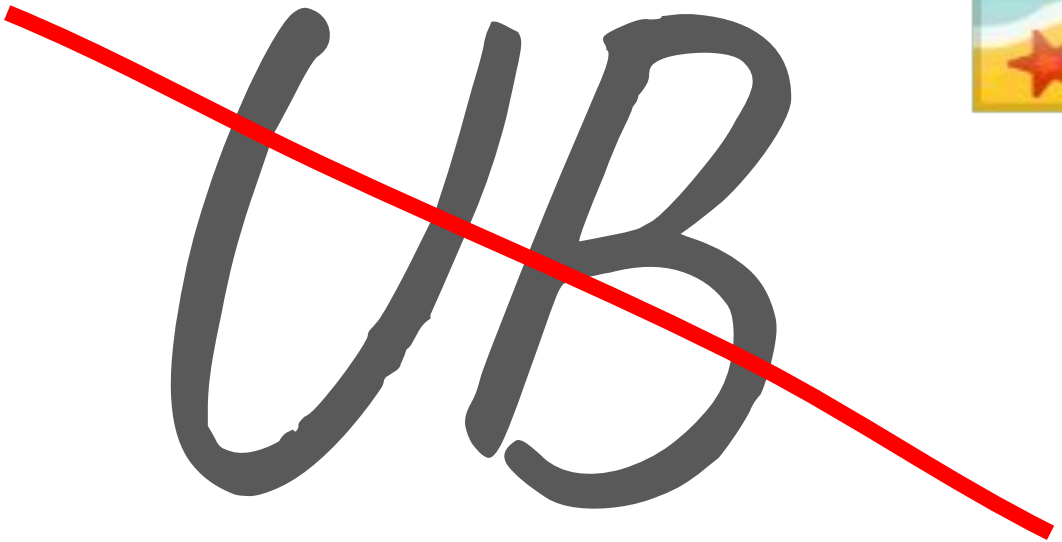
Miguel Ojeda

ojeda@kernel.org

Key concepts

So, what does Rust offer?

So, what does Rust offer?



So, what does Rust offer?

~~UB~~*

*Conditions apply

What is Undefined Behavior?

3.4.3

1 **undefined behavior**

behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this document imposes no requirements

2 **Note 1 to entry:** Possible undefined behavior ranges from ignoring the situation completely with unpredictable results, to behaving during translation or program execution in a documented manner characteristic of the environment (with or without the issuance of a diagnostic message), to terminating a translation or execution (with the issuance of a diagnostic message).

3 **Note 2 to entry:** J.2 gives an overview over properties of C programs that lead to undefined behavior.

4 **EXAMPLE** An example of undefined behavior is the behavior on dereferencing a null pointer.

— N2596 C2x Working Draft

Key concepts

Key concepts

Safe function: a function that does not trigger undefined behavior in any context and/or for any possible inputs.

Key concepts

Safe function: a function that does not trigger undefined behavior in any context and/or for any possible inputs.

Unsafe function: a function that is not safe, prefixed with the `unsafe` keyword.

Key concepts

Safe function: a function that does not trigger undefined behavior in any context and/or for any possible inputs.

That is, it does not have any precondition (regarding undefined behavior).

Unsafe function: a function that is not safe, prefixed with the `unsafe` keyword.

Key concepts

Safe function: a function that does not trigger undefined behavior in any context and/or for any possible inputs.

That is, it does not have any precondition (regarding undefined behavior).

Unsafe function: a function that is not safe, prefixed with the `unsafe` keyword.

This means it has safety preconditions.

Key concepts

Safe function: a function that does not trigger undefined behavior in any context and/or for any possible inputs.

> That is, it does not have any precondition (regarding undefined behavior).
>
> In other words, whatever a caller does, it does not produce undefined behavior.

Unsafe function: a function that is not safe, prefixed with the `unsafe` keyword.

> This means it has safety preconditions.

Key concepts

Safe function: a function that does not trigger undefined behavior in any context and/or for any possible inputs.

That is, it does not have any precondition (regarding undefined behavior).

In other words, whatever a caller does, it does not produce undefined behavior.

Unsafe function: a function that is not safe, prefixed with the `unsafe` keyword.

This means it has safety preconditions.

Callers have to declare they are upholding the contract.

Unsafe function

```
int f(int a, int b) {
    return a / b;
}
```

Unsafe function

```
int f(int a, int b) {
    return a / b;
}
```

UB ∀x f(x, 0);

Unsafe function

```
int f(int a, int b) {
    return a / b;
}
```

UB ∀x `f(x, 0);`

UB `f(INT_MIN, -1);`

Safe function

```
int f(int a, int b) {
    if (b == 0)
        abort();

    if (a == INT_MIN && b == -1)
        abort();

    return a / b;
}
```

Key concepts

Unsafe code: code inside an unsafe block.

Key concepts

Unsafe code: code inside an unsafe block.

Safe code: code that is outside an unsafe block (i.e. the default).

Key concepts

Unsafe code: code inside an unsafe block.

Safe code: code that is outside an unsafe block (i.e. the default).

Unsafe block: a block of code prefixed with the `unsafe` keyword.

Key concepts

Unsafe code: code inside an unsafe block.

It has access to all operations.

Safe code: code that is outside an unsafe block (i.e. the default).

Unsafe block: a block of code prefixed with the `unsafe` keyword.

Key concepts

Unsafe code: code inside an unsafe block.

It has access to all operations.

Safe code: code that is outside an unsafe block (i.e. the default).

It cannot perform a few operations (e.g. calling unsafe functions or dereferencing raw pointers).

Unsafe block: a block of code prefixed with the `unsafe` keyword.

(Un)safe functions vs. (un)safe code

Safe functions *may or may not* contain unsafe blocks.

Unsafe functions *may or may not* contain unsafe blocks.

(Un)safe functions vs. (un)safe code

Safe function with only safe code	Unsafe function with only safe code
Safe function with unsafe code	Unsafe function with unsafe code

Safe function with only safe code

```
fn f(x: i32) -> i32 {
    x + 1
}
```

Unsafe function with unsafe code

```
unsafe fn f(p: *const i32) -> i32 {
    unsafe { *p }
}
```

Safe function with unsafe code

```
unsafe fn g(p: *const i32) -> i32 {
    unsafe { *p }
}
```

```
fn f() -> i32 {
    unsafe { g(&42) }
}
```

Unsafe function with only safe code

```
mod m {
    pub struct S {
        p: *const i32,
    }

    impl S {
        pub fn new() -> Self {
            Self { p: &42 }
        }

        pub unsafe fn set(&mut self, p: *const i32) {
            self.p = p;
        }

        pub fn dereference(&self) -> i32 {
            unsafe { *self.p }
        }
    }
}
```

What happens if we make a mistake?

If a safe function is not actually safe, then it is called **unsound**.

- This is considered a bug.
- In the standard library, a CVE is assigned.

What happens if we make a mistake?

If a safe function is not actually safe, then it is called **unsound**.

This is considered a bug.

In the standard library, a CVE is assigned.

```
fn f(p: *const i32) -> i32 {
    unsafe { *p }
}
```

Common misconceptions

“Safe functions cannot trigger UB”

Common misconceptions

~~*“Safe functions cannot trigger UB”*~~

They should not, but if they are unsound, then they can.

This is considered a bug.

Common misconceptions

~~*“Safe functions cannot trigger UB”*~~

They should not, but if they are unsound, then they can.

This is considered a bug.

“Unsafe block means UB will necessarily be produced”

Common misconceptions

~~*“Safe functions cannot trigger UB”*~~

They should not, but if they are unsound, then they can.

This is considered a bug.

~~*“Unsafe block means UB will necessarily be produced”*~~

UB should never be produced.

An unsafe block only means the developer is the one responsible to avoid UB, instead of the compiler.

The safe/unsafe split in the kernel

The goal is:

- To write leaf modules / drivers in safe Rust (ideally 100%).
- To keep unsafe code in the abstractions / subsystems.

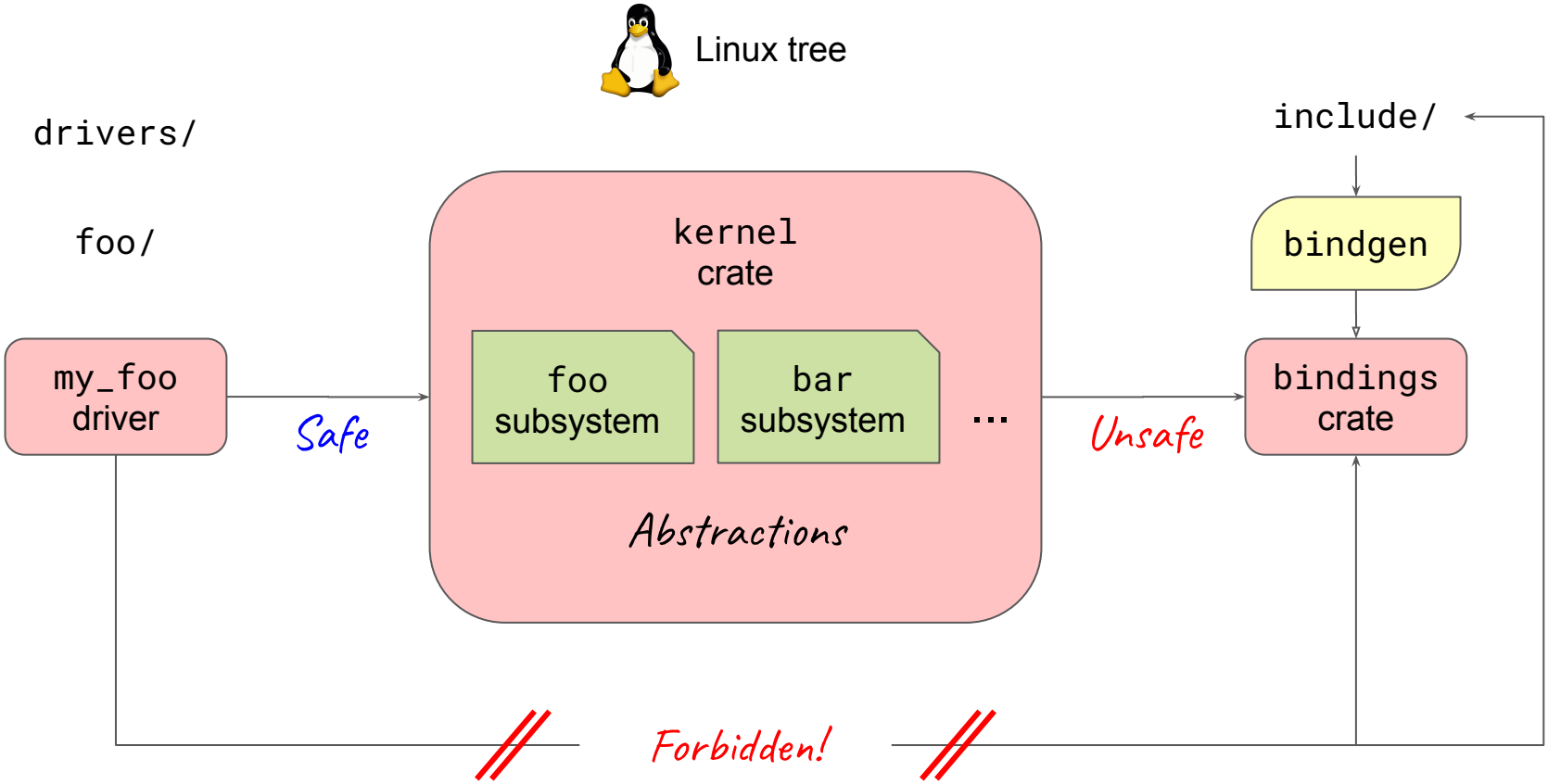
The safe/unsafe split in the kernel

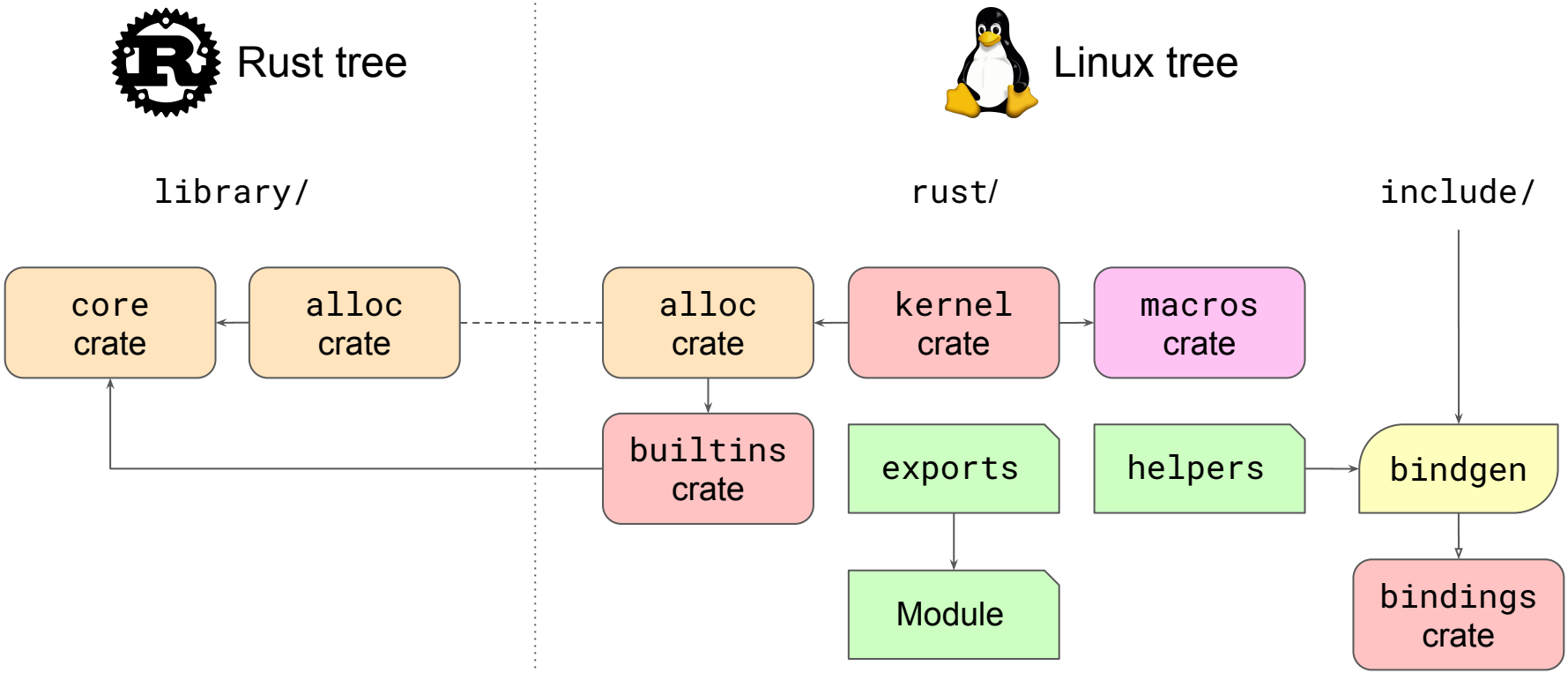
The goal is:

- To write leaf modules / drivers in safe Rust (ideally 100%).
- To keep unsafe code in the abstractions / subsystems.

Calling the C side of the kernel requires an unsafe block.

- This is the reason we aim to forbid calling the C side directly.


Linux tree
drivers/
foo/
my_foo
driver
Safe
kernel
crate
foo
subsystem
bar
subsystem
...
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
Module
bindings crate

Setup

Note

These instructions are meant as an example.

For the latest instructions and details, see the guide at:

`Documentation/rust/quick-start.rst`

Kernel tree

Checkout the **`rust`** branch from:

```
https://github.com/Rust-for-Linux/linux.git
```

To follow these examples on your own, checkout the **`mentor`** branch from:

```
https://github.com/ojeda/linux.git
```

LLVM toolchain

Prefer **`LLVM=1`** builds.

- **`CC=clang`** should also work.
- GCC builds may or may not work.

In all cases, **`libclang`** is needed for **`bindgen`**.

Rust toolchain

There are several ways to install Rust – here the **`rustup`** approach is shown.

```
$ curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
info: downloading installer

Welcome to Rust!

...

Current installation options:

   default host triple: x86_64-unknown-linux-gnu
     default toolchain: stable (default)
               profile: default
  modify PATH variable: yes
```

Rust toolchain

```
info: profile set to 'default'
info: default host triple is x86_64-unknown-linux-gnu
info: syncing channel updates for 'stable-x86_64-unknown-linux-gnu'
info: latest update on 2021-11-01, rust version 1.56.1 (59eed8a2a 2021-11-01)
info: downloading component 'cargo'
...
info: installing component 'cargo'
...
info: default toolchain set to 'stable-x86_64-unknown-linux-gnu'

  stable-x86_64-unknown-linux-gnu installed - rustc 1.56.1 (59eed8a2a 2021-11-01)


Rust is installed now. Great!

...
```

Rust standard library sources

We also need to download the **`core`** library sources on top.

```
$ rustup component add rust-src
info: downloading component 'rust-src'
info: installing component 'rust-src'
```

Bindgen

This is the tool that generates Rust code from C headers.

```
$ cargo install --locked --version 0.56.0 bindgen
    Updating crates.io index
  Downloaded bindgen v0.56.0
  Downloaded 1 crate (198.3 KB) in 0.17s
  Installing bindgen v0.56.0
  Downloaded proc-macro2 v1.0.24
         ...
  Downloaded cfg-if v0.1.10
  Downloaded 35 crates (2.0 MB) in 0.32s
   Compiling libc v0.2.80
         ...
   Compiling cexpr v0.4.0
    Finished release [optimized] target(s) in 1m 06s
  Installing .../.cargo/bin/bindgen
   Installed package `bindgen v0.56.0` (executable `bindgen`)
```

The example subsystem

drivers/mentor/

Extremely simple “subsystem”, a key-value store.

Provides a few “addresses” to read and write from.

Some of them cannot be read from, some cannot be written to.

A particular address can be used to fetch the number of total writes so far.

We will ignore data races here — the subsystem takes care of locking.

Enable it in Device Drivers -> Mentor Support.

Header

```c
/* SPDX-License-Identifier: GPL-2.0 */
/*
 * The example mentor subsystem: a key-value "database".
 *
 * Valid addresses go from 0x00 to 0x05. Accessing others is UB.
 *
 * Reading address 0x05 gives the total number of writes.
 * Writing to it is UB.
 */
#ifndef __LINUX_MENTOR_H
#define __LINUX_MENTOR_H

#include <linux/compiler.h>

#define MENTOR_TOTAL_WRITES_ADDR 0x05

/* Public interface */
#define mentor_read(addr) \
        __mentor_read(addr)
void mentor_write(u8 addr, u32 value);

/* Do not use! */
u32 __mentor_read(u8 addr);

#endif /* __LINUX_MENTOR_H */
```

Bindings

To use this C header from Rust, we need to generate the **bindings** to it.

`bindgen` will read the header and generate Rust code from it.

It is invoked automatically by Kbuild; but we need to add the header to a list.

In the future, how this is specified will likely change.

```
diff --git a/rust/kernel/bindings_helper.h
b/rust/kernel/bindings_helper.h
index b01169f7609f..d1cc999679bc 100644
--- a/rust/kernel/bindings_helper.h
+++ b/rust/kernel/bindings_helper.h
@@ -19,6 +19,7 @@
 #include <linux/of_platform.h>
 #include <linux/security.h>
 #include <asm/io.h>
+#include <linux/mentor.h>
```

Helpers

For C macros that are not trivial **`#define`**'s, we need to create a **helper**.

```
diff --git a/rust/helpers.c b/rust/helpers.c
index b42ce8405d68..e65fefd221f3 100644
--- a/rust/helpers.c
+++ b/rust/helpers.c
@@ -12,6 +12,7 @@
 #include <linux/platform_device.h>
 #include <linux/security.h>
 #include <asm/io.h>
+#include <linux/mentor.h>

 __noreturn void rust_helper_BUG(void)
 {
@@ -323,6 +324,12 @@ void rust_helper_write_seqcount_end(seqcount_t *s)
 }
 EXPORT_SYMBOL_GPL(rust_helper_write_seqcount_end);

+u32 rust_helper_mentor_read(u8 addr)
+{
+       return mentor_read(addr);
+}
+EXPORT_SYMBOL_GPL(rust_helper_mentor_read);
+
```

Writing a Rust module

Boilerplate

```rust
// SPDX-License-Identifier: GPL-2.0

//! Mentor test

#![no_std]
#![feature(allocator_api, global_asm)]

use kernel::{mentor, prelude::*, str::CStr, ThisModule};

module! {
    type: MentorTest,
    name: b"mentor_test",
    author: b"Rust for Linux Contributors",
    description: b"Mentor Test",
    license: b"GPL v2",
    params: {
        write_addr: u8 {
            default: 0,
            permissions: 0,
            description: b"Address to write",
        },
        write_value: u32 {
            default: 42,
            permissions: 0,
            description: b"Value to write",
        },
    },
}
```

Implementation (no safe abstraction!)

```
struct MentorTest;

impl KernelModule for MentorTest {
    fn init(_name: &'static CStr, _module: &'static ThisModule) -> Result<Self> {
        let addr = *write_addr.read();
        let value = *write_value.read();

        pr_info!("Writing value {} to address {}\n", value, addr);
        unsafe { bindings::mentor_write(addr, value) };

        pr_info!("Reading from address {}\n", addr);
        let value = unsafe { bindings::mentor_read(addr) };
        pr_info!("Read value = {}\n", value);

        let total_writes = unsafe { bindings::mentor_read(bindings::MENTOR_TOTAL_WRITES_ADDR as u8) };
        pr_info!("Total writes = {}\n", total_writes);

        // We can produce undefined behavior, just like in C.
        let bad_addr = 0x42;
        pr_info!("Reading from address {}\n", bad_addr);
        let _ = unsafe { bindings::mentor_read(bad_addr) };

        Ok(MentorTest)
    }
}
```

Writing a Safe Abstraction

Boilerplate

The abstractions are currently in `rust/kernel/`

This may also change in the future.

```rust
// SPDX-License-Identifier: GPL-2.0

//! Mentor subsystem.
//!
//! C headers: [`include/linux/mentor.h`](../../../../include/linux/mentor.h)

use crate::{bindings, error::Error, Result};

const TOTAL_WRITES_ADDR: u8 = bindings::MENTOR_TOTAL_WRITES_ADDR as u8;

fn is_valid(addr: u8) -> bool {
    addr < TOTAL_WRITES_ADDR
}
```

Read

```rust
/// Reads from an address.
///
/// To read the total number of writes, use [`read_total_writes`] instead.
///
/// Returns an error if the address is invalid.
///
/// # Examples
///
/// ```
/// # use kernel::prelude::*;
/// # use kernel::mentor;
/// # fn test() -> Result {
/// let result = mentor::read(0x01)?;
/// # Ok(())
/// # }
/// ```
pub fn read(addr: u8) -> Result<u32> {
    if !is_valid(addr) {
        return Err(Error::EINVAL);
    }

    // SAFETY: FFI call, we have verified the address is valid.
    Ok(unsafe { bindings::mentor_read(addr) })
}
```

Write

```rust
/// Writes a value to an address.
///
/// Returns an error if the address is invalid.
///
/// # Examples
///
/// ```
/// # use kernel::prelude::*;
/// # use kernel::mentor;
/// # fn test() -> Result {
/// mentor::write(0x01, 42)?;
/// # Ok(())
/// # }
/// ```
pub fn write(addr: u8, value: u32) -> Result {
    if !is_valid(addr) {
        return Err(Error::EINVAL);
    }

    // SAFETY: FFI call, we have verified the address is valid.
    unsafe { bindings::mentor_write(addr, value) }

    Ok(())
}
```

Read total number of writes

```rust
/// Reads the total number of writes (from the special Mentor address).
///
/// # Examples
///
/// ```
/// # use kernel::prelude::*;
/// # use kernel::mentor;
/// # fn test() {
/// let total_writes = mentor::read_total_writes();
/// # }
/// ```
pub fn read_total_writes() -> u32 {
    // SAFETY: FFI call, this address is always valid.
    unsafe { bindings::mentor_read(TOTAL_WRITES_ADDR) }
}
```

Creating the documentation

It is quite fast and does not require Sphinx.

```
$ make LLVM=1 -j3 rustdoc
  CALL    scripts/atomic/check-atomics.sh
  CALL    scripts/checksyscalls.sh
  RUSTC L rust/kernel.o
  EXPORTS rust/exports_kernel_generated.h
  RUSTDOC   .../rustlib/src/rust/library/core/src/lib.rs
  RUSTDOC H rust/macros/lib.rs
  RUSTDOC   rust/compiler_builtins.rs
  RUSTDOC   rust/alloc/lib.rs
  RUSTDOC   rust/kernel/lib.rs
```

Module kernel::mentor [-][src]

[–] Mentor subsystem.

C headers: `include/linux/mentor.h`

Functions

read	Reads from an address.
read_total_writes	Reads the total number of writes (from the special Mentor address).
read_unchecked⚠	Reads from an address (unchecked version).
write	Writes a value to an address.
write_unchecked⚠	Writes a value to an address (unchecked version).

Module mentor

Functions

Function kernel::mentor::read [-][src]

```
pub fn read(addr: u8) -> Result<u32>
```

[–] Reads from an address.

To read the total number of writes, use `read_total_writes` instead.

Returns an error if the address is invalid.

Examples

```
let result = mentor::read(0x01)?;
```

Other items in kernel::mentor

Functions

read
read_total_writes
read_unchecked
write
write_unchecked

Tests

```rust
#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn test_is_valid() {
        assert!(is_valid(0x00));
        assert!(is_valid(0x04));
        assert!(!is_valid(0x05));
    }
}
```

Running the tests

```
$ make LLVM=1 -j3 rusttest
  CALL    scripts/atomic/check-atomics.sh
  CALL    scripts/checksyscalls.sh
  RUSTSYSROOT
   Compiling compiler_builtins v0.1.49
   ...
    Finished test [unoptimized + debuginfo] target(s) in 46.80s
     Running unittests (rust/test/dummy/target/x86_64-unknown-linux-gnu/debug/deps/dummy-ecc6d3fa6cf7d238)

...

running 5 tests
.....
test result: ok. 5 passed; 0 failed; 0 ignored; 0 measured; 900 filtered out; finished in 0.01s

  RUSTC TL rust/kernel/lib.rs
  RUSTDOC T rust/kernel/lib.rs

running 52 tests
.ii.....ii.........................................
test result: ok. 48 passed; 0 failed; 4 ignored; 0 measured; 0 filtered out; finished in 17.34s
```

Formatting the code

```
$ make LLVM=1 -j3 rustfmt
```

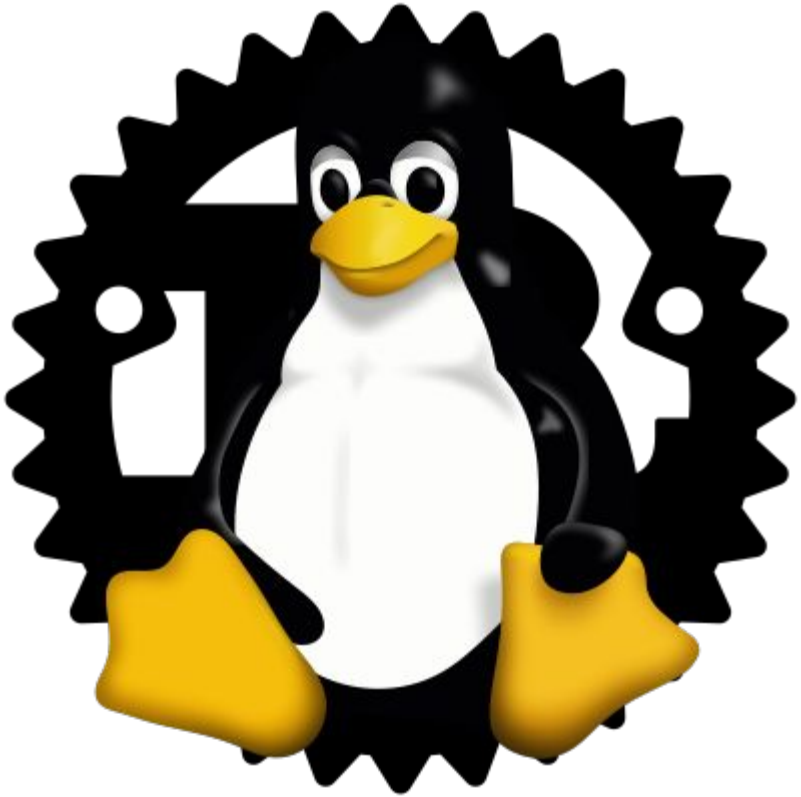
Back to the Rust module

Implementation (no safe abstraction!)

```
struct MentorTest;

impl KernelModule for MentorTest {
    fn init(_name: &'static CStr, _module: &'static ThisModule) -> Result<Self> {
        let addr = *write_addr.read();
        let value = *write_value.read();

        pr_info!("Writing value {} to address {}\n", value, addr);
        unsafe { bindings::mentor_write(addr, value) };

        pr_info!("Reading from address {}\n", addr);
        let value = unsafe { bindings::mentor_read(addr) };
        pr_info!("Read value = {}\n", value);

        let total_writes = unsafe { bindings::mentor_read(bindings::MENTOR_TOTAL_WRITES_ADDR as u8) };
        pr_info!("Total writes = {}\n", total_writes);

        // We can produce undefined behavior, just like in C.
        let bad_addr = 0x42;
        pr_info!("Reading from address {}\n", bad_addr);
        let _ = unsafe { bindings::mentor_read(bad_addr) };

        Ok(MentorTest)
    }
}
```

Implementation (using the safe abstraction)

```
struct MentorTest;

impl KernelModule for MentorTest {
    fn init(_name: &'static CStr, _module: &'static ThisModule) -> Result<Self> {
        let addr = *write_addr.read();
        let value = *write_value.read();

        pr_info!("Writing value {} to address {}\n", value, addr);
        mentor::write(addr, value)?;

        pr_info!("Reading from address {}\n", addr);
        let value = mentor::read(addr)?;
        pr_info!("Read value = {}\n", value);

        let total_writes = mentor::read_total_writes();
        pr_info!("Total writes = {}\n", total_writes);

        // Whatever we try to do here, as long as it is safe code,
        // we cannot produce UB.
        let bad_addr = 0x42;
        pr_info!("Reading from address {}\n", bad_addr);
        if mentor::read(bad_addr).is_err() {
            pr_info!("Expected failure\n");
        }

        Ok(MentorTest)
    }
}
```



Rust for Linux: Writing Safe Abstractions & Drivers

Miguel Ojeda

ojeda@kernel.org

LF live MENTORSHIP SERIES

Thank you for joining us today!

We hope it will be helpful in your journey to learning more about effective and productive participation in open source projects. We will leave you with a few additional resources for your continued learning:

- The LF Mentoring Program is designed to help new developers with necessary skills and resources to experiment, learn and contribute effectively to open source communities.
- Outreachy remote internships program supports diversity in open source and free software
- Linux Foundation Training offers a wide range of free courses, webinars, tutorials and publications to help you explore the open source technology landscape.
- Linux Foundation Events also provide educational content across a range of skill levels and topics, as well as the chance to meet others in the community, to collaborate, exchange ideas, expand job opportunities and more. You can find all events at events.linuxfoundation.org.

Backup slides

Common misconceptions

~~*“Rust should only be used if there is no unsafe code in your program”*~~

~~*“Safe Rust is a compiler mode”*~~

~~*“Safe Rust is a subset like MISRA C”*~~

Safe and unsafe Rust are intended to be mixed, even within the same function.

Virtually all Rust programs contain unsafe code when taking into account dependencies (e.g. the standard library).

Unchecked read

```rust
/// Reads from an address (unchecked version).
///
/// To read the total number of writes, use [`read_total_writes`] instead.
///
/// # Safety
///
/// The address must be valid.
///
/// # Examples
///
/// ```
/// # use kernel::prelude::*;
/// # use kernel::mentor;
/// # fn test() {
/// let result = unsafe { mentor::read_unchecked(0x01) };
/// # }
/// ```
pub unsafe fn read_unchecked(addr: u8) -> u32 {
    // SAFETY: FFI call, the caller guarantees the address is valid.
    unsafe { bindings::mentor_read(addr) }
}
```

Unchecked write

```rust
/// Writes a value to an address (unchecked version).
///
/// # Safety
///
/// The address must be valid.
///
/// # Examples
///
/// ```
/// # use kernel::prelude::*;
/// # use kernel::mentor;
/// # fn test() {
/// unsafe { mentor::write_unchecked(0x01, 42); }
/// # }
/// ```
pub unsafe fn write_unchecked(addr: u8, value: u32) {
    // SAFETY: FFI call, the caller guarantees the address is valid.
    unsafe { bindings::mentor_write(addr, value) }
}
```

Other items in **kernel**::mentor

Functions

read
read_total_writes
read_unchecked
write
write_unchecked

Function kernel::mentor::read_unchecked [-][src]

```
pub unsafe fn read_unchecked(addr: u8) -> u32
```

[–] Reads from an address (unchecked version).

To read the total number of writes, use `read_total_writes` instead.

Safety

The address must be valid.

Examples

```
let result = unsafe { mentor::read_unchecked(0x01) };
```

Running the `mentor_test` module

```
$ insmod mentor_test.ko write_addr=0x03 write_value=42

[    0.952950] mentor_test: --- Without an abstraction (do not use!)
[    0.953950] mentor_test: Writing value 42 to address 3
[    0.954950] mentor_test: Reading from address 3
[    0.954950] mentor_test: Read value = 42
[    0.955950] mentor_test: Total writes = 1
[    0.955950] mentor_test: Reading from address 66
[    0.955950] mentor: undefined behavior!

[    0.955950] mentor_test: --- With a safe abstraction
[    0.956950] mentor_test: Writing value 42 to address 3
[    0.956950] mentor_test: Reading from address 3
[    0.956950] mentor_test: Read value = 42
[    0.956950] mentor_test: Total writes = 2
[    0.957949] mentor_test: Reading from address 66
[    0.957949] mentor_test: Expected failure
```

Safety

Safety in Rust

=

No undefined behavior

Safety

Safety in Rust

≠

Safety in “safety-critical”

as in functional safety (DO-178B/C, ISO 26262, EN 50128…)

Is avoiding UB that important?

~70%

of vulnerabilities in C/C++ projects come from UB

See more at https://www.memorysafety.org/docs/memory-safety/

Some examples where Rust helps

COMPILER EXPLORER

Add... More

Sponsors intel. PC-lint Share Policies Other

Rust source #2

Rust

```rust
pub fn main() {
    let a = Box::new(42);
    drop(a);
    drop(a);
}
```

rustc 1.55.0 (Rust, Editor #2, Compiler #1)

rustc 1.55.0 --edition=2018

```
<Compilation failed>

# For more information see the output
# To open the output window, click or
```

Output (0/13) rustc 1.55.0 - cached

Executor rustc 1.55.0 (Rust, Editor #2)

Wrap lines Libraries Compilation Arguments Stdin Compiler output

rustc 1.55.0 --edition=2018 -O -Cpanic=abort

```
Could not execute the program
Compiler returned: 1
Compiler stderr
error[E0382]: use of moved value: `a`
 --> <source>:4:10
  |
2 |     let a = Box::new(42);
  |         - move occurs because `a` has type `Box<i32>`, which does not implement the `Copy` trait
3 |     drop(a);
  |          - value moved here
4 |     drop(a);
  |          ^ value used here after move

error: aborting due to previous error

For more information about this error, try `rustc --explain E0382`.
```

rustc 1.55.0 - 1050ms

C source #1

Save/Load Add new... Vim C

```c
#include <stdlib.h>

int main(void)
{
    int * const a = malloc(sizeof(int));
    if (a == NULL)
        abort();
    *a = 42;
    free(a);
    free(a);
}
```

x86-64 gcc (trunk) (C, Editor #1, Compiler #2)

x86-64 gcc (trunk) -std=gnu99 -Wall -Wextra -Wpedantic -O2

Output... Filter... Libraries Add new... Add tool...

```
main:
        push    rbp
        mov     edi, 4
        call    malloc
        test    rax, rax
        je      .L3
        mov     rbp, rax
        mov     rdi, rax
        call    free
        mov     rdi, rbp
        call    free
        xor     eax, eax
        pop     rbp
        ret
main.cold:
```

Output (0/0) x86-64 gcc (trunk) - cached (8745B) ~564 lines filtered

Executor x86-64 gcc (trunk) (C, Editor #1)

Wrap lines

x86-64 gcc (trunk) all -Wextra -Wpedantic -O2

```
Program returned: 139
Program stderr
free(): double free detected in tcache 2
```

x86-64 gcc (trunk) - cached

Rust source #2

```rust
pub fn main() {
    let a = Box::new(42);
    drop(a);
    println!("{}", *a);
}
```

rustc 1.55.0 (Rust, Editor #2, Compiler #1)

rustc 1.55.0 --edition=2018

```
<Compilation failed>

# For more information see the output
# To open the output window, click or
```

Output (0/13) rustc 1.55.0 - 301ms

Executor rustc 1.55.0 (Rust, Editor #2)

Wrap lines Libraries Compilation Arguments Stdin Compiler output

rustc 1.55.0 --edition=2018 -O -Cpanic=abort

```
Could not execute the program
Compiler returned: 1
Compiler stderr
error[E0382]: borrow of moved value: `a`
 --> <source>:4:20
  |
2 |     let a = Box::new(42);
  |         - move occurs because `a` has type `Box<i32>`, which does not implement the `Copy` trait
3 |     drop(a);
  |          - value moved here
4 |     println!("{}", *a);
  |                    ^^ value borrowed here after move

error: aborting due to previous error

For more information about this error, try `rustc --explain E0382`.
```

rustc 1.55.0 - 802ms

C source #1

Save/Load Add new... Vim C

```c
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    int * const a = malloc(sizeof(int));
    if (a == NULL)
        abort();
    *a = 42;
    free(a);
    printf("%i\n", *a);
}
```

x86-64 gcc (trunk) (C, Editor #1, Compiler #2)

x86-64 gcc (trunk) std=gnu99 -Wall -Wextra -Wpedantic -O

Output... Filter... Libraries Add new... Add tool...

```
.LC0:
        .string "%i\n"
main:
        push    rbx
        mov     edi, 4
        call    malloc
        test    rax, rax
        je      .L3
        mov     rbx, rax
        mov     rdi, rax
        call    free
        mov     esi, DWORD PTR [rbx]
        mov     edi, OFFSET FLAT:.LC0
        xor     eax, eax
        call    printf
```

Output (0/0) x86-64 gcc (trunk) - cached (9175B) ~588 lines filtered

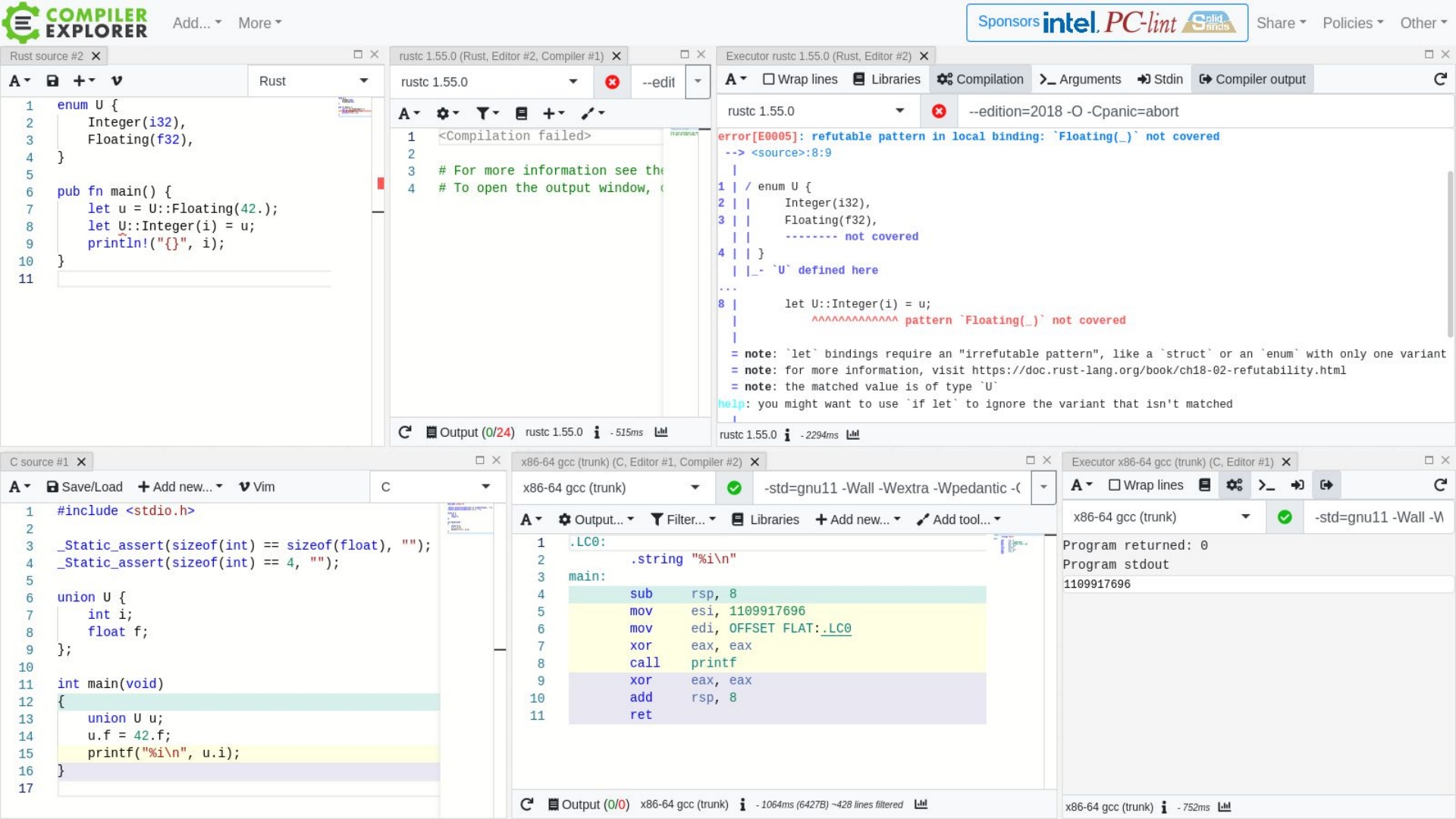
Executor x86-64 gcc (trunk) (C, Editor #1)

Wrap lines

x86-64 gcc (trunk) -std=gnu99 -Wall -W

```
Program returned: 0
Program stdout
0
```

x86-64 gcc (trunk) - cached


COMPILER EXPLORER
Add...
More
Sponsors intel PC-lint
Share
Policies
Other
Rust source #2
Rust
enum U {
Integer(i32),
Floating(f32),
}
pub fn main() {
let u = U::Floating(42.);
let U::Integer(i) = u;
println!("{}", i);
}
rustc 1.55.0 (Rust, Editor #2, Compiler #1)
rustc 1.55.0
--edit
<Compilation failed>
For more information see the
To open the output window,
Output (0/24)
rustc 1.55.0
- 515ms
Executor rustc 1.55.0 (Rust, Editor #2)
Wrap lines
Libraries
Compilation
Arguments
Stdin
Compiler output
rustc 1.55.0
--edition=2018 -O -Cpanic=abort
error[E0005]: refutable pattern in local binding: `Floating(_)` not covered
--> <source>:8:9
1 | / enum U {
2 | | Integer(i32),
3 | | Floating(f32),
| | -------- not covered
4 | | }
| |_- `U` defined here
...
8 | let U::Integer(i) = u;
| ^^^^^^^^^^^^ pattern `Floating(_)` not covered
= note: `let` bindings require an "irrefutable pattern", like a `struct` or an `enum` with only one variant
= note: for more information, visit https://doc.rust-lang.org/book/ch18-02-refutability.html
= note: the matched value is of type `U`
help: you might want to use `if let` to ignore the variant that isn't matched
rustc 1.55.0
- 2294ms
C source #1
Save/Load
Add new...
Vim
C
#include <stdio.h>
_Static_assert(sizeof(int) == sizeof(float), "");
_Static_assert(sizeof(int) == 4, "");
union U {
int i;
float f;
};
int main(void)
{
union U u;
u.f = 42.f;
printf("%i\n", u.i);
}
x86-64 gcc (trunk) (C, Editor #1, Compiler #2)
x86-64 gcc (trunk)
-std=gnu11 -Wall -Wextra -Wpedantic -(
Output...
Filter...
Libraries
Add new...
Add tool...
.LC0:
.string "%i\n"
main:
sub rsp, 8
mov esi, 1109917696
mov edi, OFFSET FLAT:.LC0
xor eax, eax
call printf
xor eax, eax
add rsp, 8
ret
Output (0/0)
x86-64 gcc (trunk)
- 1064ms (6427B) ~428 lines filtered
Executor x86-64 gcc (trunk) (C, Editor #1)
Wrap lines
x86-64 gcc (trunk)
-std=gnu11 -Wall -W
Program returned: 0
Program stdout
1109917696
x86-64 gcc (trunk)
- 752ms

C source #1

Save/Load Add new... Vim C

```c
#include <stdlib.h>

// On success, `result` is written and the return value is 0.
// On error, the return value is < 0.
int get_some_data(int * result);
void do_something(int foo);

void f(void) {
    int data;
    if (get_some_data(&data) < 0)
        abort();

    do_something(data);
}

```

x86-64 gcc 11.2 (C, Editor #1, Compiler #1)

x86-64 gcc 11.2 -std=c11 -Wall -Wextra -Wpedantic -O2

Output... Filter... Libraries Add new... Add tool...

```asm
f:
        sub     rsp, 24
        lea     rdi, [rsp+12]
        call    get_some_data
        test    eax, eax
        js      .L3
        mov     edi, DWORD PTR [rsp+12]
        call    do_something
        add     rsp, 24
        ret
f.cold:
.L3:
        call    abort
```

Output (0/0) x86-64 gcc 11.2 - 838ms (6182B) ~402 lines filtered

Rust source #2

Save/Load Add new... Vim Rust

```rust
extern "Rust" {
    fn get_some_data_() -> Result<i32, ()>;
    fn do_something_(foo: i32);
}

pub fn f() {
    let data = get_some_data().unwrap();
    do_something(data);
}

```

rustc 1.55.0 (Rust, Editor #2, Compiler #2)

rustc 1.55.0 --edition=2018 -O -Cpanic=abort

Output... Filter... Libraries Add new... Add tool...

```asm
example::f:
        push    rax
        call    qword ptr [rip + get_some_data_@GOTPCREL]
        test    eax, eax
        jne     .LBB2_1
        mov     edi, edx
        pop     rax
        jmp     qword ptr [rip + do_something_@GOTPCREL]
.LBB2_1:
        lea     rdi, [rip + .L__unnamed_2]
        lea     rcx, [rip + .L__unnamed_3]
        lea     r8, [rip + .L__unnamed_4]
        mov     rdx, rsp
        mov     esi, 43
        call    qword ptr [rip + core::result::unwrap_failed@GOTPCREL]
        ud2
```

Output (0/0) rustc 1.55.0 - cached (9350B) ~450 lines filtered

Building an abstraction

Rust source #2

Save/Load Add new... Vim Rust

```
1  // Bindings
2  extern {
3      fn get_pointer() -> *mut i32;
4      fn use_pointer(ptr: *mut i32);
5  }
6
```

rustc 1.55.0 (Rust, Editor #2, Compiler #2)

rustc 1.55.0 --edition=2018 -O -Cpanic=abort

Output... Filter... Libraries Add new... Add tool...

```
1  <No assembly to display (~5 lines filtered)>
```

Output (0/16) rustc 1.55.0 - 2977ms (142B) ~5 lines filtered

Rust source #2

Save/Load Add new... Vim Rust

```rust
// Bindings
extern {
    fn get_pointer() -> *mut i32;
    fn use_pointer(ptr: *mut i32);
}


// Abstractions code
mod foo {
    /// # Invariants
    ///
    /// The pointer is valid.
    pub struct Foo {
        ptr: *mut i32,
    }

    impl Foo {
        pub fn new() -> Self {
            Foo {
                // SAFETY: `get_pointer()` is always safe to call.
                ptr: unsafe { crate::get_pointer() }
            }
        }

        pub fn do_something(&mut self) {
            // SAFETY: `use_pointer()` requires that the pointer
            // is valid, which holds due to the type invariant.
            unsafe { crate::use_pointer(self.ptr); }
        }
    }
}

```

rustc 1.55.0 (Rust, Editor #2, Compiler #2)

rustc 1.55.0 --edition=2018 -O -Cpanic=abort

Output... Filter... Libraries Add new... Add tool...

```
<No assembly to display (~5 lines filtered)>
```

Output (0/34) rustc 1.55.0 - 3826ms (142B) ~5 lines filtered

Rust source #2

Save/Load Add new... Vim

Rust

```rust
// Bindings
extern {
    fn get_pointer() -> *mut i32;
    fn use_pointer(ptr: *mut i32);
}


// Abstractions code
mod foo {
    /// # Invariants
    ///
    /// The pointer is valid.
    pub struct Foo {
        ptr: *mut i32,
    }

    impl Foo {
        pub fn new() -> Self {
            Foo {
                // SAFETY: `get_pointer()` is always safe to call.
                ptr: unsafe { crate::get_pointer() }
            }
        }

        pub fn do_something(&mut self) {
            // SAFETY: `use_pointer()` requires that the pointer
            // is valid, which holds due to the type invariant.
            unsafe { crate::use_pointer(self.ptr); }
        }
    }
}


// User code
use foo::*;

pub fn f() {
    let mut my_foo = Foo::new();
    my_foo.do_something();
}

```

rustc 1.55.0 (Rust, Editor #2, Compiler #2)

rustc 1.55.0

--edition=2018 -O -Cpanic=abort

Output... Filter... Libraries Add new... Add tool...

```asm
example::f:
        push    rax
        call    qword ptr [rip + get_pointer@GOTPCREL]
        mov     rdi, rax
        pop     rax
        jmp     qword ptr [rip + use_pointer@GOTPCREL]
```

Output (0/0) rustc 1.55.0 - 532ms (3945B) ~238 lines filtered

Output of rustc 1.55.0 (Compiler #2)

Wrap lines

```
Compiler returned: 0
```

Rust source #2

Save/Load Add new... Vim Rust

```rust
// Bindings
extern {
    fn get_pointer() -> *mut i32;
    fn use_pointer(ptr: *mut i32);
}


// Abstractions code
mod foo {
    /// # Invariants
    ///
    /// The pointer is valid.
    pub struct Foo {
        ptr: *mut i32,
    }

    impl Foo {
        pub fn new() -> Self {
            Foo {
                // SAFETY: `get_pointer()` is always safe to call.
                ptr: unsafe { crate::get_pointer() }
            }
        }

        pub fn do_something(&mut self) {
            // SAFETY: `use_pointer()` requires that the pointer
            // is valid, which holds due to the type invariant.
            unsafe { crate::use_pointer(self.ptr); }
        }
    }
}


// User code
use foo::*;

pub fn f() {
    let mut my_foo = Foo::new();
    my_foo.ptr = 42 as *mut i32;
    my_foo.do_something();
}
```

rustc 1.55.0 (Rust, Editor #2, Compiler #2)

rustc 1.55.0 --edition=2018 -O -Cpanic=abort

Output... Filter... Libraries Add new... Add tool...

```
<Compilation failed>

# For more information see the output window
```

Output (0/9) rustc 1.55.0 - 1569ms

Output of rustc 1.55.0 (Compiler #2)

Wrap lines

```
error[E0616]: field `ptr` of struct `foo::Foo` is private
  --> <source>:39:12
   |
39 |     my_foo.ptr = 42 as *mut i32;
   |            ^^^ private field

error: aborting due to previous error

For more information about this error, try `rustc --explain E0616`.
Compiler returned: 1
```

Supported architectures

`arm`	(`armv6` only)
`arm64`	
`powerpc`	(`ppc64le` only)
`riscv`	(`riscv64` only)
`x86`	(`x86_64` only)

...so far!

32-bit and other restrictions should be easy to remove

Kernel LLVM builds work for `mips` and `s390`

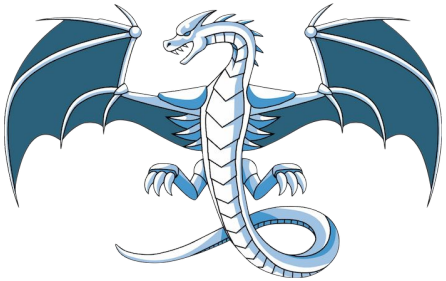
GCC codegen paths should open up more

See `Documentation/rust/arch-support.rst`

Rust codegen paths for the kernel

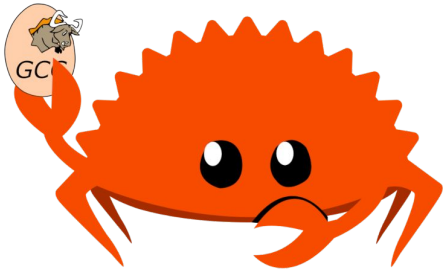
`rustc_codegen_gcc`

Already passes most rustc tests



`rustc_codegen_llvm`

Main one



Rust GCC

Expected in 1-2 years (rough estimate)

Documentation

Crate kernel [-][src]

[-] The `kernel` crate.

This crate contains the kernel APIs that have been ported or wrapped for usage by Rust code in the kernel and is shared by all of them.

In other words, all the rest of the Rust code in the kernel (e.g. kernel modules written in Rust) depends on `core`, `alloc` and this crate.

If you need a kernel C API that is not ported or wrapped yet here, then do so first instead of bypassing this crate.

Modules

buffer	Struct for writing to a pre-allocated buffer with the `write!` macro.
c_types	C types for the bindings.
chrdev	Character devices.
file	Files and file descriptors.
file_operations	File operations.
io_buffer	Buffers used in IO.
io_mem	Memory-mapped IO.
iov_iter	IO vector iterators.
linked_list	Linked lists.
mentor	Mentor subsystem.
miscdev	Miscellaneous devices.
of	Devicetree and Open Firmware abstractions.
pages	Kernel page allocation and management.
platdev	Platform devices.
power	Power management interfaces.
prelude	The `kernel` prelude.
print	Printing facilities.
random	Random numbers.
rbtree	Red-black trees.
security	Linux Security Modules (LSM).
str	String representations.
sync	Synchronisation primitives.
sysctl `CONFIG_SYSCTL`	System control.

Crate kernel
See all kernel's items
Modules
Macros
Structs
Constants
Traits
Functions
Type Definitions
Crates
alloc
compiler_builtins
core
kernel
macros

Struct kernel::sync::Mutex [-][src]

```
pub struct Mutex<T: ?Sized> { /* fields omitted */ }
```

[−] Exposes the kernel's `struct mutex`. When multiple threads attempt to lock the same mutex, only one at a time is allowed to progress, the others will block (sleep) until the mutex is unlocked, at which point another thread will be allowed to wake up and make progress.

A `Mutex` must first be initialised with a call to `Mutex::init_lock` before it can be used. The `mutex_init` macro is provided to automatically assign a new lock class to a mutex instance.

Since it may block, `Mutex` needs to be used with care in atomic contexts.

Implementations

[−] `impl<T> Mutex<T>` [src]

[−] `pub unsafe fn new(t: T) -> Self` [src]

Constructs a new mutex.

Safety

The caller must call `Mutex::init_lock` before using the mutex.

[−] `impl<T: ?Sized> Mutex<T>` [src]

[−] `pub fn lock(&self) -> GuardMut<'_, Self>` [src]

Locks the mutex and gives the caller access to the data protected by it. Only one thread at a time is allowed to access the protected data.

Trait Implementations

[−] `impl<T> CreatableLock for Mutex<T>` [src]

[−] `unsafe fn new_lock(data: Self::Inner) -> Self` [src]

```rust
/// A string that is guaranteed to have exactly one `NUL` byte, which is at the
/// end.
///
/// Used for interoperability with kernel APIs that take C strings.
#[repr(transparent)]
pub struct CStr([u8]);

impl CStr {
    /// Returns the length of this string excluding `NUL`.
    #[inline]
    pub const fn len(&self) -> usize {
        self.len_with_nul() - 1
    }

    /// Returns the length of this string with `NUL`.
    #[inline]
    pub const fn len_with_nul(&self) -> usize {
        // SAFETY: This is one of the invariant of `CStr`.
        // We add a `unreachable_unchecked` here to hint the optimizer that
        // the value returned from this function is non-zero.
        if self.0.is_empty() {
            unsafe { core::hint::unreachable_unchecked() };
        }
        self.0.len()
    }

    /// Returns `true` if the string only includes `NUL`.
    #[inline]
    pub const fn is_empty(&self) -> bool {
        self.len() == 0
    }

    /// Wraps a raw C string pointer.
    ///
    /// # Safety
    ///
    /// `ptr` must be a valid pointer to a `NUL`-terminated C string, and it must
    /// last at least `'a`. When `CStr` is alive, the memory pointed by `ptr`
    /// must not be mutated.
    #[inline]
```

Crate kernel

See all kernel's items

Modules
Macros
Structs
Constants
Traits
Functions
Type Definitions

Crates

alloc
compiler_builtins
core
kernel
macros

Results for pr

In Names (176) | In Parameters (44) | In Return Types (119)

kernel::print	Printing facilities.
kernel::platdev::PlatformDriver::probe	Platform driver probe.
kernel::prelude::Error::EPROTO	Protocol error.
kernel::pr_err	Prints an error-level message (level 3).
kernel::pr_cont	Continues a previous log message in the same line.
kernel::pr_crit	Prints a critical-level message (level 2).
kernel::pr_info	Prints an info-level message (level 6).
kernel::pr_warn	Prints a warning-level message (level 4).
kernel::prelude	The `kernel` prelude.
kernel::pr_alert	Prints an alert-level message (level 1).
kernel::pr_debug	Prints a debug-level message (level 7).
kernel::pr_emerg	Prints an emergency-level message (level 0).
kernel::linked_list::CursorMut::peek_prev	Returns the element immediately before the one the cursor ...
kernel::pr_notice	Prints a notice-level message (level 5).
kernel::prelude::Error::EPROTOTYPE	Protocol wrong type for socket.
kernel::prelude::Error::EINPROGRESS	Operation now in progress.
kernel::prelude::Error::ENOPROTOOPT	Protocol not available.
kernel::prelude::Vec::swap_remove	Removes an element from the vector and returns it.
kernel::prelude::Box::is_prefix_of	
kernel::prelude::Error::EPROTONOSUPPORT	Protocol not supported.
kernel::prelude::Box::strip_prefix_of	
alloc::prelude	The alloc Prelude
core::prelude	The libcore prelude
core::iter::Product	Trait to represent types that can be created by ...
core::iter::Product::product	Method which takes an iterator and generates `Self` from the ...
core::iter::Iterator::product	Iterates over the entire iterator, multiplying all the ...
core::option::Option::product	Takes each element in the `Iterator`: if it is a `None`, no ...
core::result::Result::product	Takes each element in the `Iterator`: if it is an `Err`, no ...
core::num::Wrapping::product	
core::arch::nvptx::vprintf	Print formatted output from a kernel to a host-side output ...

Documentation code

```rust
/// Wraps the kernel's `struct task_struct`.
///
/// # Invariants
///
/// The pointer `Task::ptr` is non-null and valid. Its reference count is also non-zero.
///
/// # Examples
///
/// The following is an example of getting the PID of the current thread with
/// zero additional cost when compared to the C version:
///
/// ```
/// # use kernel::prelude::*;
/// use kernel::task::Task;
///
/// # fn test() {
/// Task::current().pid();
/// # }
/// ```
pub struct Task {
    pub(crate) ptr: *mut bindings::task_struct,
}
```

Conditional compilation

Rust code has access to conditional compilation based on the kernel config

```
#[cfg(CONFIG_X)]       // `CONFIG_X` is enabled (`y` or `m`)
#[cfg(CONFIG_X="y")]   // `CONFIG_X` is enabled as a built-in (`y`)
#[cfg(CONFIG_X="m")]   // `CONFIG_X` is enabled as a module   (`m`)
#[cfg(not(CONFIG_X))] // `CONFIG_X` is disabled
```

Coding guidelines

No direct access to C bindings

No undocumented public APIs

No implicit `unsafe` block

Docs follows Rust standard library style

`// SAFETY` proofs for all `unsafe` blocks

Clippy linting enabled

Automatic formatting enforced

Rust 2018 edition & idioms

No unneeded panics

No infallible allocations

...

Aiming to be as strict as possible

Abstractions code

```rust
/// Wraps the kernel's `struct file`.
///
/// # Invariants
///
/// The pointer `File::ptr` is non-null and valid.
/// Its reference count is also non-zero.
pub struct File {
   pub(crate) ptr: *mut bindings::file,
}
```

```rust
impl File {
    /// Constructs a new [`struct file`] wrapper from a file descriptor.
    ///
    /// The file descriptor belongs to the current process.
    pub fn from_fd(fd: u32) -> Result<Self> {
        // SAFETY: FFI call, there are no requirements on `fd`.
        let ptr = unsafe { bindings::fget(fd) };
        if ptr.is_null() {
            return Err(Error::EBADF);
        }

        // INVARIANTS: We checked that `ptr` is non-null, so it is valid.
        // `fget` increments the ref count before returning.
        Ok(Self { ptr })
    }

    // ...
}
```

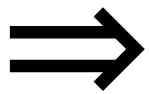
Driver code

```c
static int pl061_resume(struct device *dev)
{
    int offset;

    struct pl061 *pl061 = dev_get_drvdata(dev);

    for (offset = 0; offset < PL061_GPIO_NR; offset++) {
        if (pl061->csave_regs.gpio_dir & (BIT(offset)))
            pl061_direction_output(&pl061->gc, offset,
                    pl061->csave_regs.gpio_data &
                    (BIT(offset)));
        else
            pl061_direction_input(&pl061->gc, offset);


    }

    writeb(pl061->csave_regs.gpio_is, pl061->base + GPIOIS);
    writeb(pl061->csave_regs.gpio_ibe, pl061->base + GPIOIBE);
    writeb(pl061->csave_regs.gpio_iev, pl061->base + GPIOIEV);
    writeb(pl061->csave_regs.gpio_ie, pl061->base + GPIOIE);

    return 0;
}
```

```rust
fn resume(data: &Ref<DeviceData>) -> Result {


    let inner = data.lock();
    let pl061 = data.resources().ok_or(Error::ENXIO)?;

    for offset in 0..PL061_GPIO_NR {
        if inner.csave_regs.gpio_dir & bit(offset) != 0 {
            let v = inner.csave_regs.gpio_data & bit(offset) != 0;
            let _ = <Self as gpio::Chip>::direction_output(
                data, offset.into(), v);
        } else {
            let _ = <Self as gpio::Chip>::direction_input(
                data, offset.into());
        }
    }

    pl061.base.writeb(inner.csave_regs.gpio_is, GPIOIS);
    pl061.base.writeb(inner.csave_regs.gpio_ibe, GPIOIBE);
    pl061.base.writeb(inner.csave_regs.gpio_iev, GPIOIEV);
    pl061.base.writeb(inner.csave_regs.gpio_ie, GPIOIE);

    Ok(())
}
```

— https://lwn.net/Articles/863459/

Safety examples

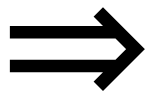


`abort()`s in C

are

Rust-safe

Safety examples



`abort()`s in C

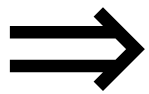
are

Rust-safe

Even if your company goes bankrupt.

Safety examples

`abort()`s in C



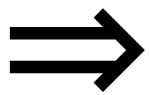
are

Rust-safe

Even if your company goes bankrupt.

Even if somebody is injured.

Safety examples

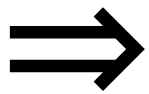


Rust panics

are

Rust-safe

Safety examples

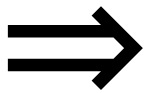


Kernel panics

are

Rust-safe

Safety examples



Uses after free, null derefs, double frees,

OOB accesses, uninitialized memory reads,

invalid inhabitants, data races...

are **not**

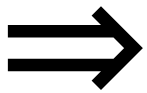
Rust-safe

Safety examples

Uses after free, null derefs, double frees,

OOB accesses, uninitialized memory reads,

invalid inhabitants, data races...

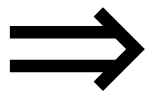


are **not**

Rust-safe

Even if your system still works.

Safety examples

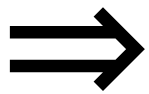


Race conditions

are

Rust-safe

Safety examples

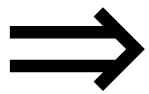


Memory leaks

are

Rust-safe

Safety examples

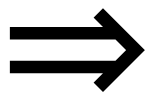


Deadlocks

are

Rust-safe

Safety examples



Integer overflows

are

Rust-safe