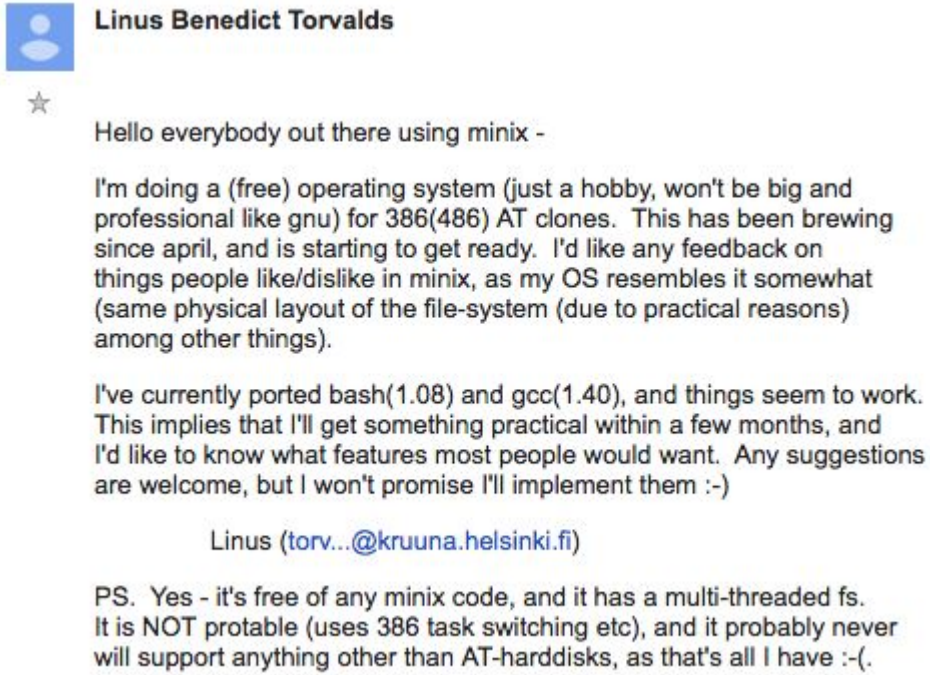
KU LEUVEN GENT VUB VRIJE UNIVERSITEIT BRUSSEL VLAIO

Evolution of Rust for Linux and Lessons Learned

Once Upon a Time



Linus Benedict Torvalds

Hello everybody out there using minix -

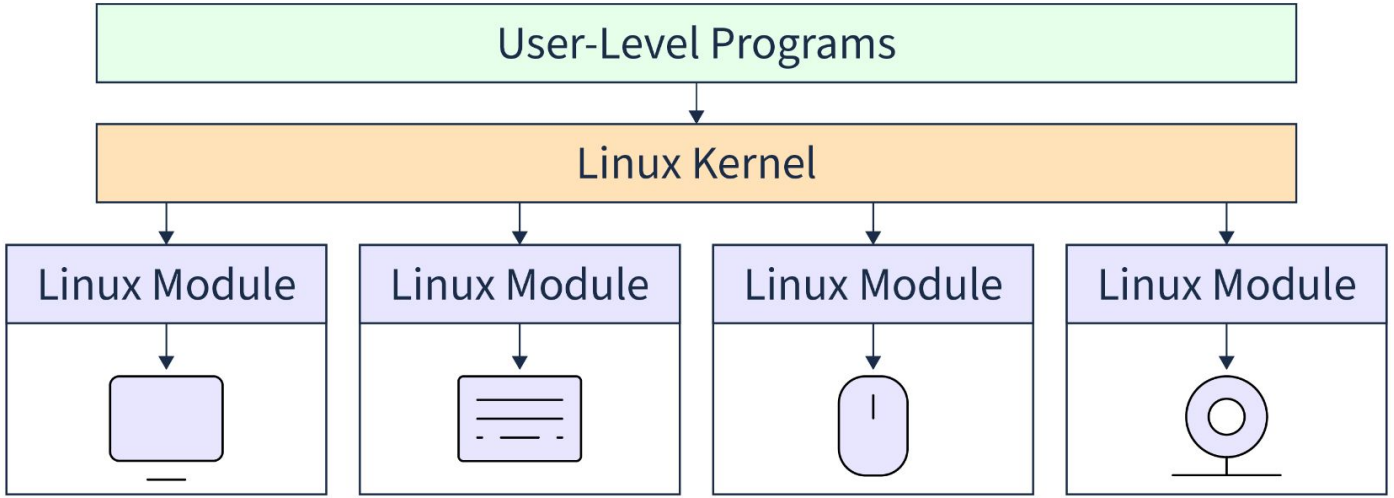
I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torv...@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT protable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.

Structure Kernel



https://www.scaler.com/topics/linux-kernel-module/

Kinds of Device Drivers

Network device: capable of exchanging data with other hosts

Block device: random access in blocks

eg: hard drives, cd roms

Character device: sequentially accessed as a stream of bytes

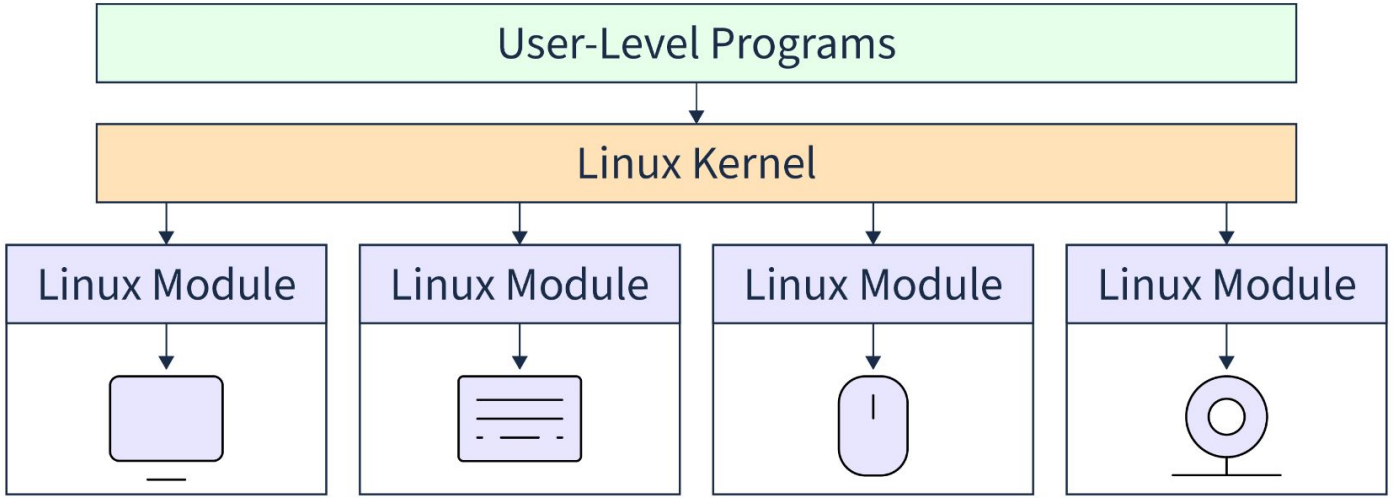
eg: the text console, the serial ports

Miscellaneous device:

- Small character device, eg: watchdog
- Strange device, eg: compass

https://static.lwn.net/images/pdf/LDD3/ch01.pdf

Structure Kernel



https://www.scaler.com/topics/linux-kernel-module/

Why Only C?

Fast and predictable code

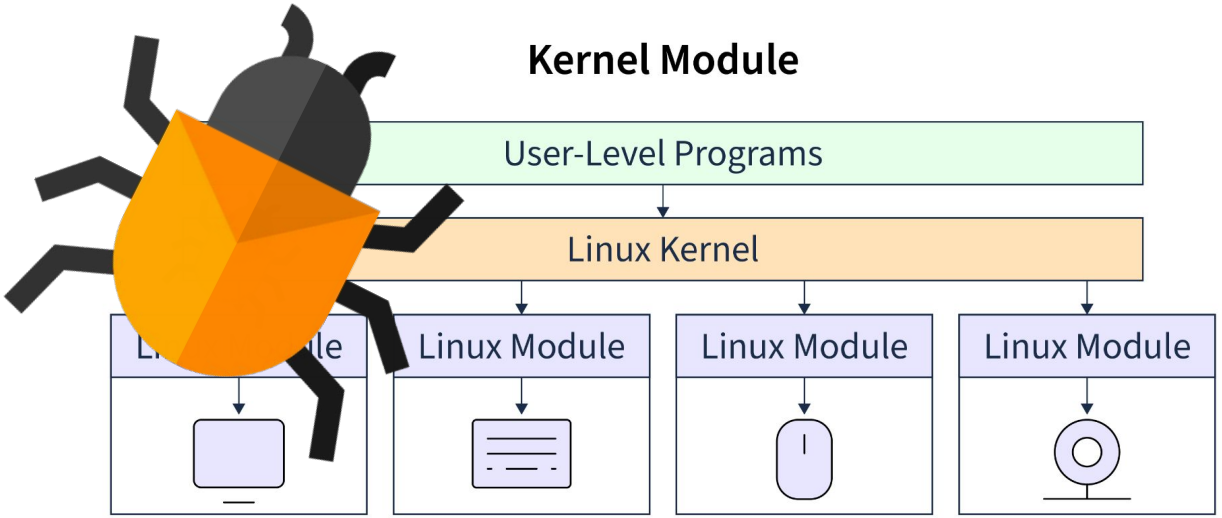
- ~~Java~~
- ~~Go~~
- ~~C#~~
- …

Why Not C++?

- Has all the same problems as C
- Adds problems:
  - Difficult code review
  - Inefficient abstracted programming models
  - Boost and STL not portable nor stable

https://web.archive.org/web/20080216000358/http://article.gmane.org/gmane.comp.version-control.git/57918

Proposing Rust



Is Memory Safety a Problem?

- 40/65 are memory errors: Cody Cutler et. al. (Usenix ‘18)
- 39/40 found in drivers: Paul Emmerich et. al. (Arxiv ‘19)
- v6.1: 15% are first time contributors

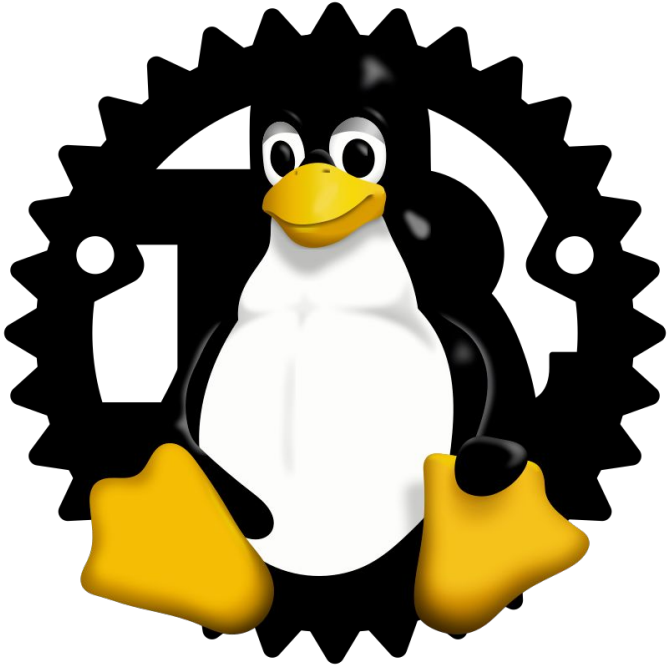
https://lwn.net/Articles/915435/

Is Introducing Rust Practically Possible?

Rust for Linux

Goal: create convincing implementation

- Mostly support for drivers



Rust for Linux Progress

- Rust versions of core kernel code
- Support for:
  - Miscellaneous device driver
  - Character device driver

Use Case: Parallel Port Driver

- Rust and C Mutexes are incompatible

```c
static long pp_ioctl(struct file *file,
unsigned int cmd, unsigned long arg)
{
    mutex_lock(&pp_do_mutex);
    ret = pp_do_ioctl(file, cmd, arg);
    mutex_unlock(&pp_do_mutex);
    return ret;
}
```

```rust
struct PPStruct{
    pdev: Mutex<Option<*mut
bindings::pardevice>>,
    ...
}

let pdev_lock = *data.pdev.lock();
let pdev = match pdev_lock  {
    Some(p) => p,
    None => return Err(EINVAL),
};
```

Use Case: Parallel Port Driver

- Rust and C Mutexes are incompatible
- Interior mutability: Look out for race conditions

```
struct PPStruct{
   pdev: AtomicPtr<Option<*mut bindings::pardevice>>,
   flags: AtomicU32,
}

fn read(
   data: RefBorrow<'_, PPStruct>
) -> Result<usize>{
   let pdev = *data.pdev;
   bindings::parport_set_timeout(pdev.load(), bindings::PARPORT_INACTIVITY_O_NONBLOCK as i64);
   *data.flags.store(TIMEOUT);
}
```

Use Case: Parallel Port Driver

```
struct PPStruct{
   pdev: Option<*mut bindings::pardevice>,
   flags: u32,
}

struct PPStructWrapper{
   interior_mutable: Mutex<PPStruct>
}

fn read(
   data: RefBorrow<'_, PPStructWrapper>
) -> Result<usize>{
   let locked_data = *data.lock();
   let pdev = locked_data.pdev;
   bindings::parport_set_timeout(pdev, bindings::PARPORT_INACTIVITY_O_NONBLOCK as i64);
   locked_data.flags = TIMEOUT;
}
```

Use Case: Parallel Port Driver

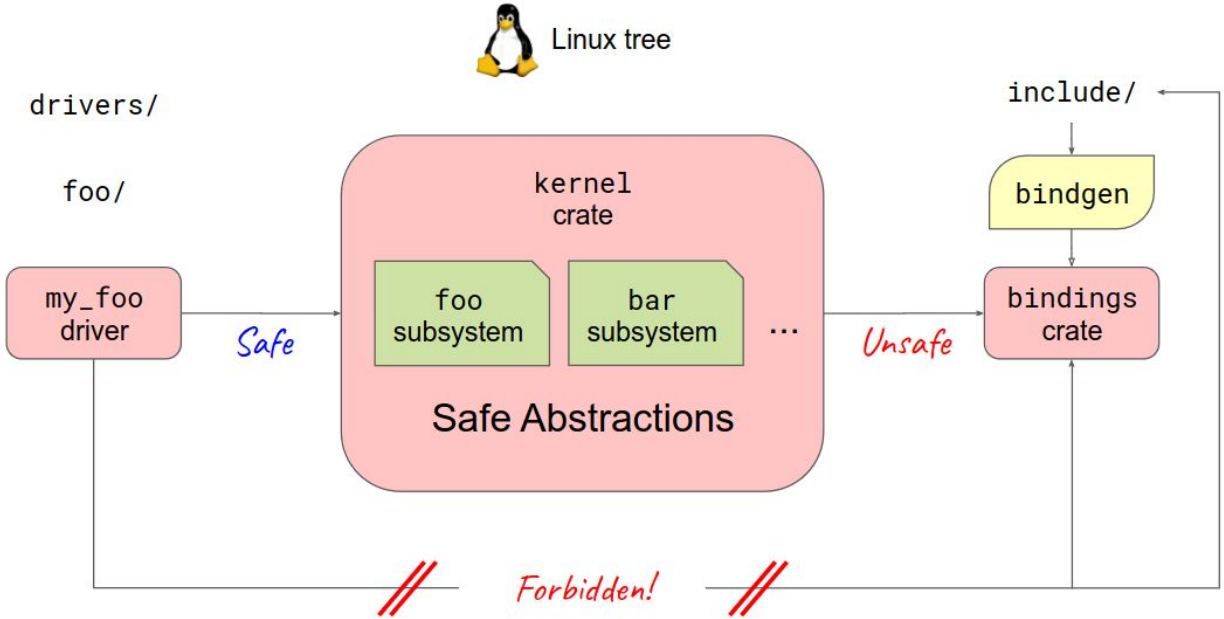
- Rust and C Mutexes are incompatible
- Interior mutability: Look out for race conditions
- Need a lot of C functionality

Use Case: Parallel Port Driver

- Need a lot of C functionality

```
/* UNSAFE:
* this transmute is horrible but as long as all changes that happen to ppstruct->private are protected
* by either a mutex or are atomic it should remain threadsafe even though you are now casting non mutable
* to mutable
* It will later also be cast to non mutable
*/
ppdev_cb.private = unsafe{core::mem::transmute::<*const Device, *mut core::ffi::c_void>(self as *const Device)};
```

Lessons Learned: Linux Rust Rules



```rust
fn read_status(dev: &mut phy::Device) -> Result<u16> {
    dev.genphy_update_link()?;
    if !dev.is_link_up() {
        return Ok(0);
    }
    ...
}
```

ax88796b_rust.rs

```rust
#[repr(transparent)]
pub struct Device(Opaque<bindings::phy_device>);

impl Device {
    /// Gets the current link state.
    ///
    /// It returns true if the link is up.
    pub fn is_link_up(&self) -> bool {
        const LINK_IS_UP: u64 = 1;
        // TODO: the code to access to the bit field will be replaced with automatically
        // generated code by bindgen when it becomes possible.
        // SAFETY: The struct invariant ensures that we may access
        // this field without additional synchronization.
        let bit_field = unsafe { &(*self.0.get())._bitfield_1 };
        bit_field.get(14, 1) == LINK_IS_UP
    }
}
```

phy.rs

Kernel v6.1

Linux 6.1: Rust to hit mainline kernel

New language will be official, probably within a couple of months

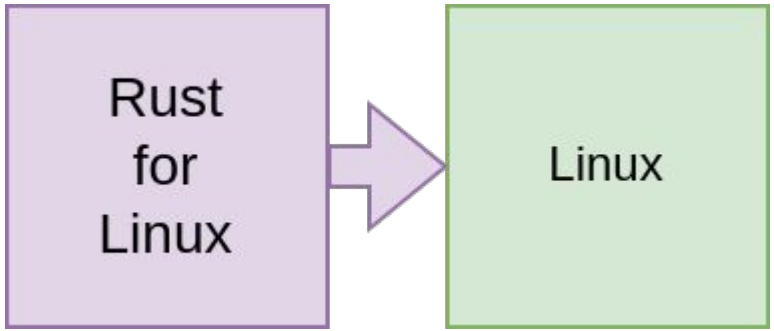
Liam Proven

Wed 5 Oct 2022 // 15:02 UTC

Rust for Linux Restructure

Before Mainline Rust

After Mainline Rust



Current Status

Rust for Linux

- Network device driver support
- PCI driver
- Block device driver
- Device drivers

Mainline Linux

- Network driver support

Support Mainline Kernel

Linux Rust Documentation

Currently, the **Rust support is primarily intended for kernel developers and maintainers** interested in the Rust support, so that they can start working on abstractions and drivers, as well as helping the development of infrastructure and tools.

If you are an end user, please note that there are currently **no in-tree drivers/modules suitable or intended for production use**, and that the Rust support is still in development/experimental, especially for certain kernel configurations.

Existing Rust Drivers

1. NVMe Driver
2. Null Block Driver
3. Android Binder Driver
4. PuzzleFS filesystem driver
5. Apple AGX GPU driver
6. Nova GPU Driver

How to Create a Driver

- Do not use deprecated tutorials!

  eg: Linux Foundation mentorship session
- A lot of fundamental work

More info: Rust Android binder driver talk