LF *live*

MENTORSHIP SERIES

LF Live MENTORSHIP SERIES

Writing Linux Kernel Modules In Rust

Wedson Almeida Filho, SW Engineer, Google

LF live MENTORSHIP SERIES

Agenda

Getting ready

Background information

Writing the module

Conclusion

Getting ready

Boot VM

Log in as `guest` (password is also `guest`)

Getting ready (cont'd)

Make `vim` the `git` editor

```
git config --global core.editor vim
```

Fetch latest version, configure and build it

```
cd linux
git fetch --depth=1 origin
git checkout origin/rust
rustup override set $(scripts/min-tool-version.sh rustc)
rustup component add rust-src
make allnoconfig qemu-busybox-min.config rust.config
make
```

LF Live MENTORSHIP SERIES

Background

Rust for Linux

Source code available at https://github.com/rust-for-linux/linux

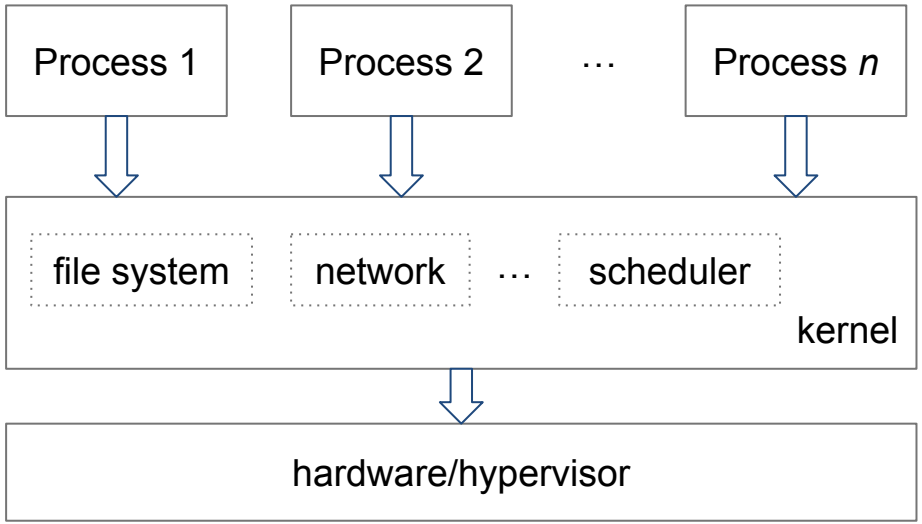
Goal: make Rust a first-class language for Linux kernel development

Rust for Linux (cont'd)

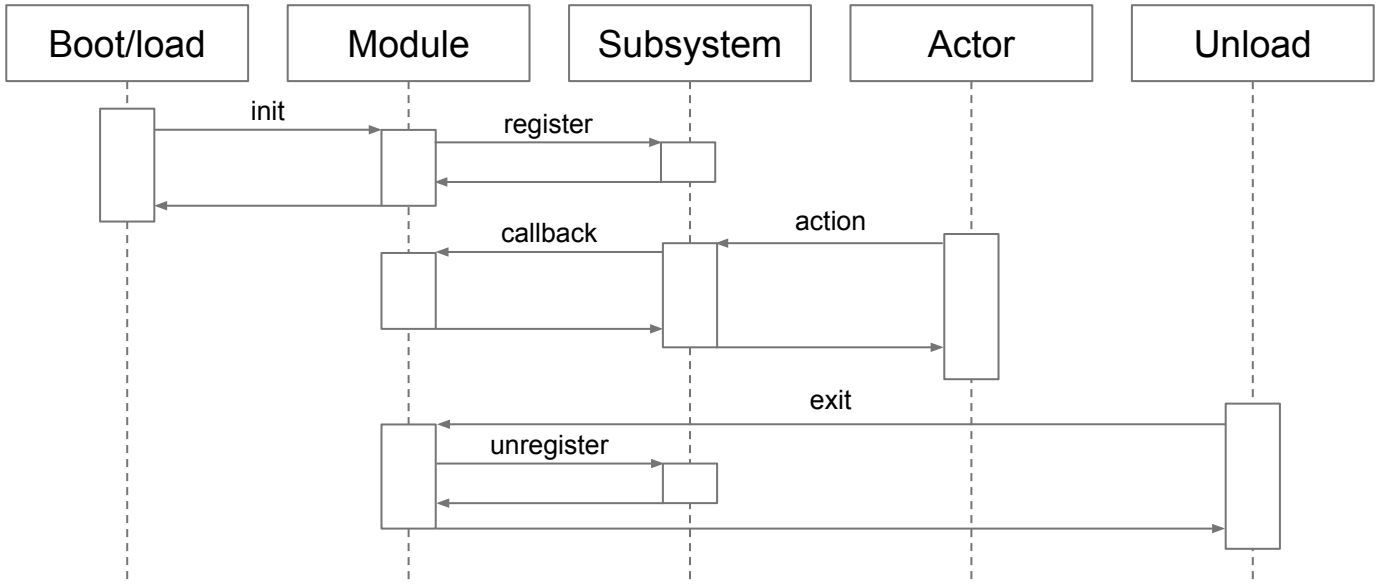
Why?

- Memory safety
  - Reduce the number of memory vulnerabilities in new code
- Productivity
  - Rich type system catches more errors at compile time
- Performance comparable to C

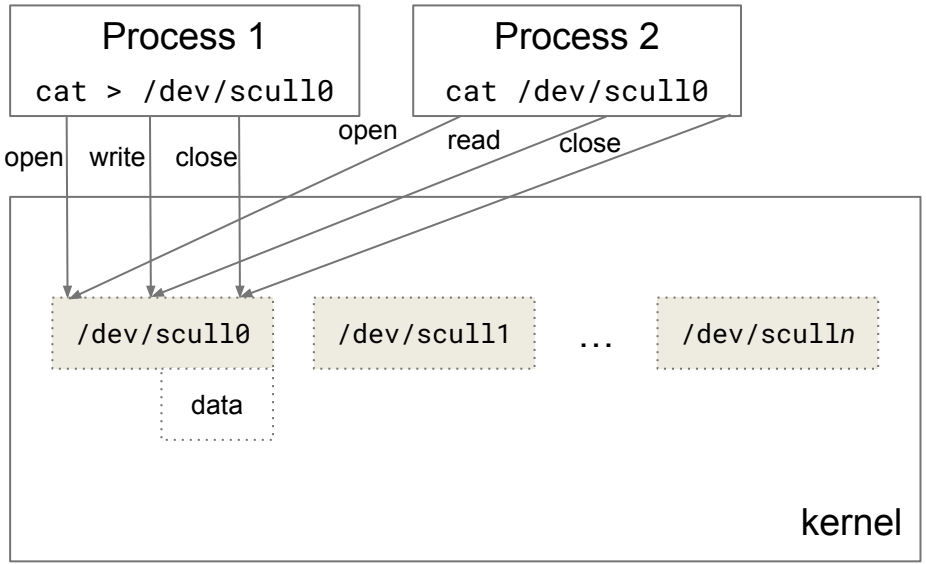
User vs kernel space



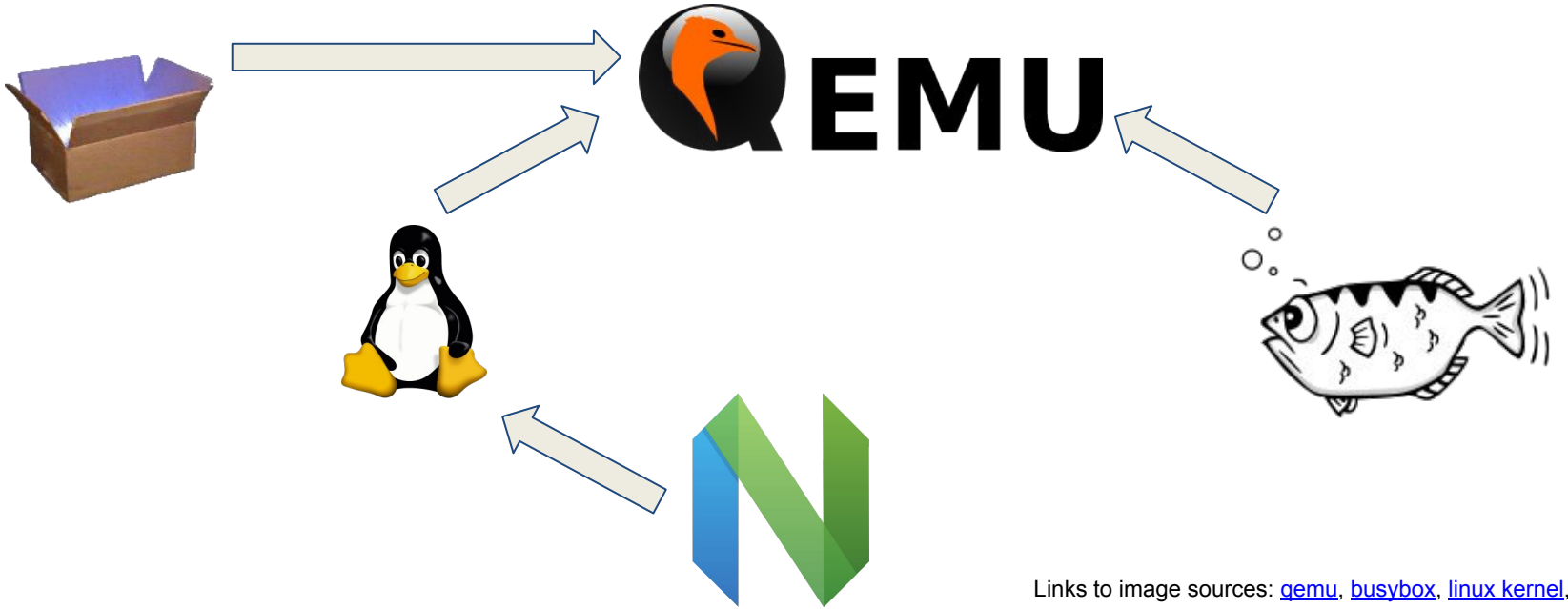
The life of a kernel module



What we are going to build



Development workflow



Links to image sources: qemu, busybox, linux kernel, gdb, neovim

Kernel configuration

The kernel has hundreds of configuration options

To minimise build and boot times, we only enable the necessary ones:

```
allnoconfig qemu-busybox-min.config rust.config
```

We'll add a new one for our module

More on the setup

tmux is a terminal multiplexer

- Sessions run on the background, can be reattached later
- Prefix key is C-q (different from the default C-b)

`neovim` is a `vim` clone

- With LSP support
- `rust-analyzer` is a Rust LSP

LF Live MENTORSHIP SERIES

Writing the module

Step 1: Add config and empty file

Modify `Kconfig` and `Makefile`

```
make menuconfig
touch samples/rust/rust_scull.rs
```

The kernel builds (with a new warning)

Very similar to C

Step 2: Module declaration and init

Add minimal code to `samples/rust/rust_scull.rs`

Fixes warning

No visible changes yet

Step 3: Hello world!

Add a print message to module init

Message visible when kernel boots

Step 4: Add minimal file operations implementation

Add open function that prints a message

No noticeable change in behaviour

Step 5: Register a misc device

Add registration to module init

Device now reachable via `/dev/scull`

Also appears in `/proc/misc`

Reading and writing to it fails

Step 6: Add minimal `read` implementation

Add `read` that prints a message to kernel log

Now `cat /dev/misc` doesn't fail anymore

Write still fails: `echo test > /dev/scull`

Step 7: Add minimal `write` implementation

Add `write` that prints a message to kernel log

Now `echo test > /dev/scull` doesn't fail anymore

Step 8: Add device state

When opening a file, it logs the device number

Different instances of the device may hold different state

Currently we only have one instance though

Step 9: Add file state

Update open to store device pointer

There is no way to forget to increment refcount

Now reading/writing also prints the device number

Step 10: Save data written

Update `write` implementation

It doesn't compile. Why?

We can't use copied data after assigning. Why?

Step 11: Add mutual exclusion

Update state type and `write` implementation

Now written data is stored in device

No data races

Step 12: Return saved data on read

Update `read` implementation

Now previously written data is read back

What happens if more data is written?

Step 13: Improved buffering of data

Update `open` to clear the buffer if opened for write

Update `write` to use the offset

`cat > /dev/scull` now stores all lines

Step 14: Add module parameters

Update module definition to include parameter

Parameters can now be specified at boot, e.g., `scull.nr_devs=10`

Step 15: Creating the number of specified devices

Update module init

Several independent devices now: `/dev/scull0`, `/dev/scull1`, etc.

Step 16: Compiling as a separate module

Enable `CONFIG_MODULES` and `CONFIG_MODULE_UNLOAD`

Switch `CONFIG_RUST_SCULL` to `m`

Rebuild image: `find . | cpio -o -H newc | gzip > ../../linux/initrd.img`

insmod/rmmod to insert and remove module

Parameter can also be specified at insertion time

LF Live MENTORSHIP SERIES

Conclusion

What we have done

We have written a kernel module

- Can be compiled into the kernel or as separate module
- Takes the number of device instances as an argument
- Registers with the miscdev subsystem
- Implements file operations

Code available here: `https://github.com/wedsonaf/linux/commits/lf-session`

LF live MENTORSHIP SERIES

Coming up

Sessions on

- Setting up an environment
- Writing Rust async code in the kernel

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.