Linux (PCI) NVMe driver in Rust

Andreas Hindborg

Kangrejos '22

September 7 2022

Outline

- Rationale
- Plumbing
- Benchmarks
- Future work

`$> whois` Andreas

- Bare metal / embedded RTOS / embedded **Linux** since 2013
- C, C++, HDL (for FPGA)
- Low power medical/IoT to high performance industrial systems
- Deployed Rust for embedded system firmware and Linux user space ~2020
- Worked with **Rust** in the Linux kernel since March

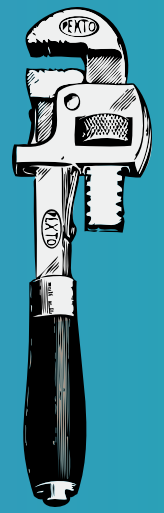
Why Rust

- Not the scope of this talk
- ... but **memory safety**
- And other nice things (the **Rust Experience™**)

Why NVMe

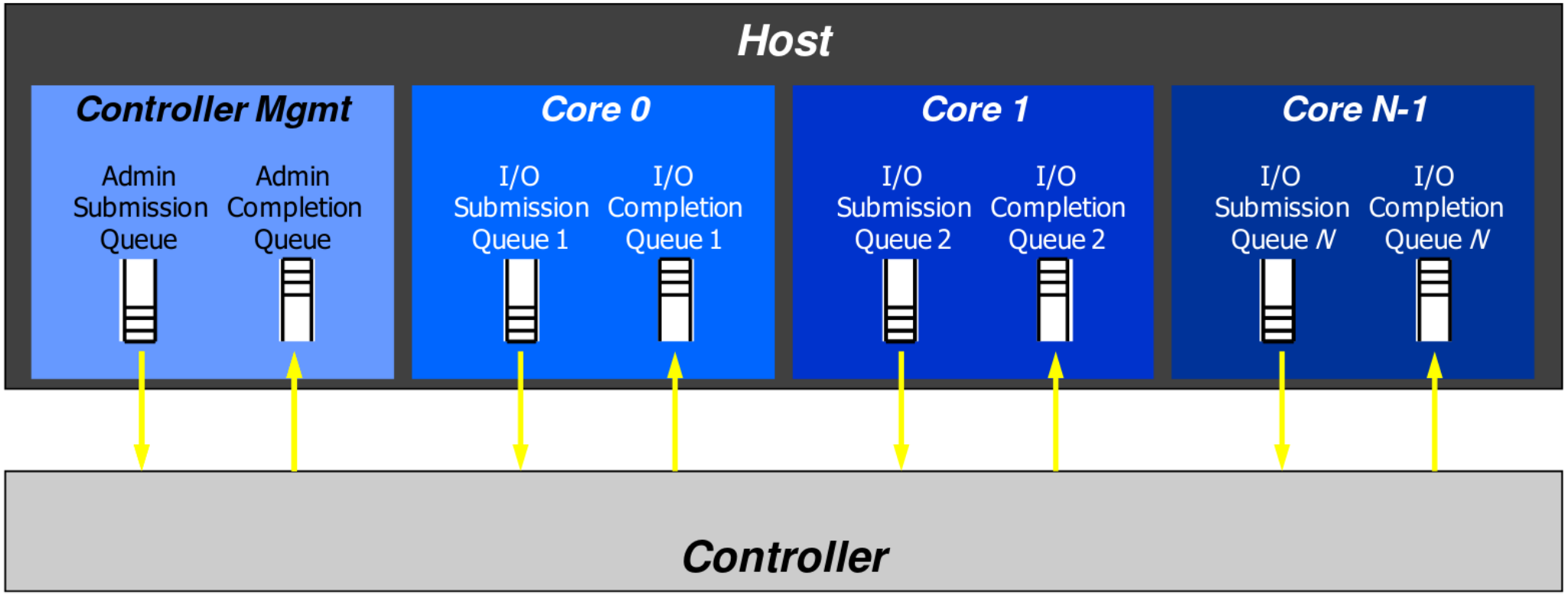
- The current NVMe driver is **fine** - no need to replace it
- NVMe is an interesting target for **evaluating** Rust as a driver implementation language
  - Simple
  - High performance requirements
  - Widely deployed
  - Mature reference implementation available
  - Diverse set of interfaces (dev, pci, dma, **blk-mq**, gendisk, sysfs)

The Plumbing

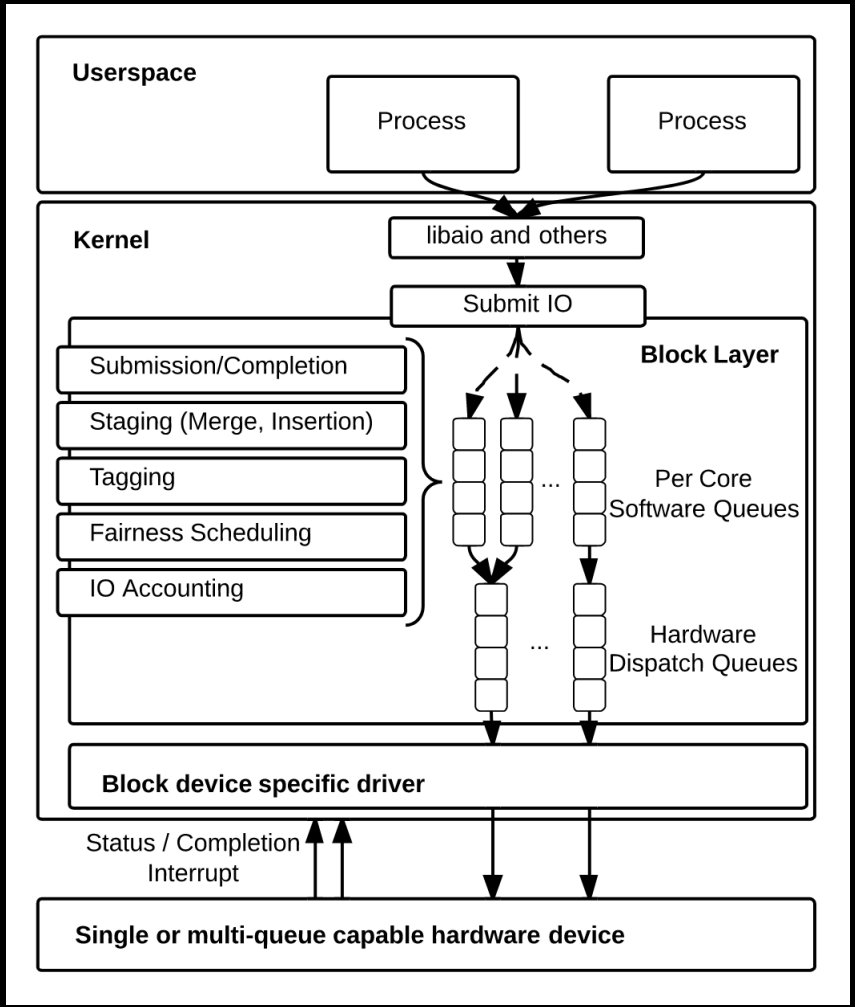


NVMe

Figure 6: Queue Pair Example, 1:1 Mapping

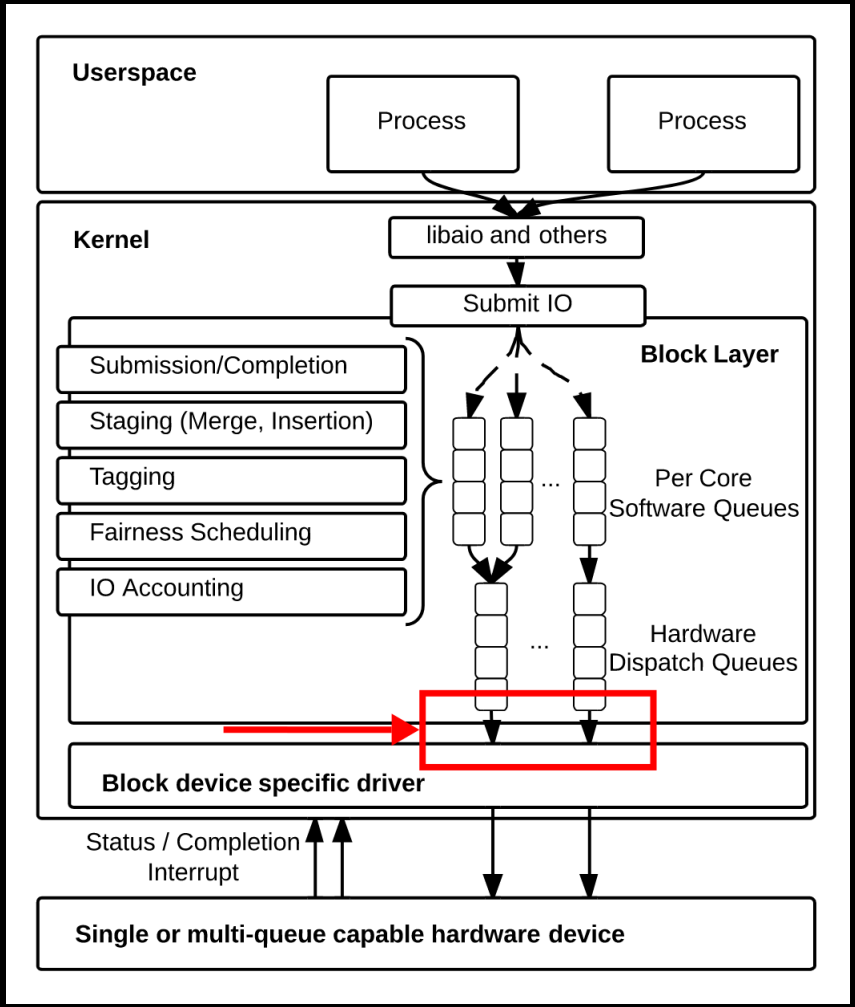


blk-mq



Matias Bjørling, Jens Axboe, David Nellans, and Philippe Bonnet. 2013. Linux block IO: introducing multi-queue SSD access on multi-core systems. In Proceedings of the 6th International Systems and Storage Conference (SYSTOR '13). Association for Computing Machinery, New York, NY, USA, Article 22, 1–10. https://doi.org/10.1145/2485732.2485740

blk-mq



Matias Bjørling, Jens Axboe, David Nellans, and Philippe Bonnet. 2013. Linux block IO: introducing multi-queue SSD access on multi-core systems. In Proceedings of the 6th International Systems and Storage Conference (SYSTOR '13). Association for Computing Machinery, New York, NY, USA, Article 22, 1–10. https://doi.org/10.1145/2485732.2485740

blk-mq Interface

```rust
#[macros::vtable]
pub trait Operations: Sized {
    type RequestData;
    type QueueData: PointerWrapper;
    type HwData: PointerWrapper;
    type TagSetData: PointerWrapper;

    fn new_request_data(
        _tagset_data: <Self::TagSetData as PointerWrapper>::Borrowed<'_>,
    ) -> Result<Self::RequestData>;

    fn init_request_data(
        _tagset_data: <Self::TagSetData as PointerWrapper>::Borrowed<'_>,
        _data: Pin<&mut Self::RequestData>,
    ) -> Result {
        Ok(())
    }

    fn queue_rq(
        hw_data: <Self::HwData as PointerWrapper>::Borrowed<'_>,
        queue_data: <Self::QueueData as PointerWrapper>::Borrowed<'_>,
        rq: &Request<Self>,
        is_last: bool,
    ) -> Result;

    fn commit_rqs(
        hw_data: <Self::HwData as PointerWrapper>::Borrowed<'_>,
        queue_data: <Self::QueueData as PointerWrapper>::Borrowed<'_>,
    );

    fn complete(_rq: &Request<Self>);

    fn init_hctx(
        tagset_data: <Self::TagSetData as PointerWrapper>::Borrowed<'_>,
        hctx_idx: u32,
    ) -> Result<Self::HwData>;


    fn poll(hw_data: <Self::HwData as PointerWrapper>::Borrowed<'_>) -> i32 {
        unreachable!()
    }

    fn map_queues(tag_set: &TagSetRef) -> Result {
        unreachable!()
    }
}
```

```rust
struct blk_mq_ops {

        blk_status_t (*queue_rq)(struct blk_mq_hw_ctx *,
                                 const struct blk_mq_queue_data *);

        void (*commit_rqs)(struct blk_mq_hw_ctx *);

        int (*poll)(struct blk_mq_hw_ctx *, struct io_comp_batch *);

        void (*complete)(struct request *);

        int (*init_hctx)(struct blk_mq_hw_ctx *, void *, unsigned int);

        void (*exit_hctx)(struct blk_mq_hw_ctx *, unsigned int);

        int (*init_request)(struct blk_mq_tag_set *set, struct request *,
                            unsigned int, unsigned int);

        void (*exit_request)(struct blk_mq_tag_set *set, struct request *,
                             unsigned int);

        int (*map_queues)(struct blk_mq_tag_set *set);

};
```

gendisk Interface

```
let disk = mq::GenDisk::try_new(tagset, ns)?;
disk.set_name(format_args!("nvme{}n{}", instance, nsid))?;
disk.set_capacity(id.nsze.into() << (lba_shift - bindings::SECTOR_SHIFT))
disk.set_queue_logical_block_size(1 << lba_shift);
disk.set_queue_max_hw_sectors(max_sectors);
disk.set_queue_max_segments(nvme_driver_defs::NVME_MAX_SEGS as _);
disk.set_queue_virt_boundary(nvme_driver_defs::NVME_CTRL_PAGE_SIZE - 1);
disk.add()?
```

- `gendisk.fops` not currently supported, default ops only

Implementing a blk-mq Device

```
// Wrappers
pub trait Operations: Sized {/*...*/}
pub struct TagSet<T: Operations> {/*...*/}

// ----------------------------------------------------------

// Implement these
pub(crate) struct MyMqOperations {/*...*/}

#[kernel::macros::vtable]
impl mq::Operations for MyMqOperations {/*...*/}

struct MyHwCtx<T: mq::Operations> {/*...*/} // NvmeQueue
impl<T: mq::Operations> MyHwCtx<T> {/*...*/}

// ----------------------------------------------------------

// Initialize like this
fn init {
    let tagset = TagSet::try_new(/*...*/);
    let my_hw_queue = MyHwCtx::try_new(tagset.clone(),/*...*/);
    let disk = GenDisk::try_new(tagset.clone(),/*...*/);
    disk.add();
}
```

Registering a PCI driver

Kernel provides:

- `kernel::driver::Registration<T: kernel::driver::DriverOps>`
- `impl kernel::driver::DriverOps for kernel::pci::Adapter<T: kernel::pci::Driver>`

PCI driver implements:

- `impl kernel::pci::Driver for nvme::NvmeDevice`

PCI driver abstractions by Fujita Tomonori https://github.com/Rust-for-Linux/linux/pull/856

```rust
struct NvmeModule {
    _registration: Pin<Box<driver::Registration<pci::Adapter<NvmeDevice>>>>,
}

impl kernel::Module for NvmeModule {
    fn init(_name: &'static CStr, module: &'static ThisModule) -> Result<Self> {
        // ...
        let registration = driver::Registration::new_pinned(c_str!("nvme"), module)?;
        // ...
    }
}

impl pci::Driver for NvmeDevice {
    type Data = Ref<DeviceData>;

    define_pci_id_table! {
        (),
        [ (pci::DeviceId::with_class(bindings::PCI_CLASS_STORAGE_EXPRESS, 0xffffff), None) ]
    }

    fn probe(dev: &mut pci::Device, _id: Option<&Self::IdInfo>) -> Result<Ref<DeviceData>> {
        // ...
    }

    fn remove(_data: &Self::Data) {
        // ...
    }
}
```

PCI driver abstractions by Fujita Tomonori https://github.com/Rust-for-Linux/linux/pull/856

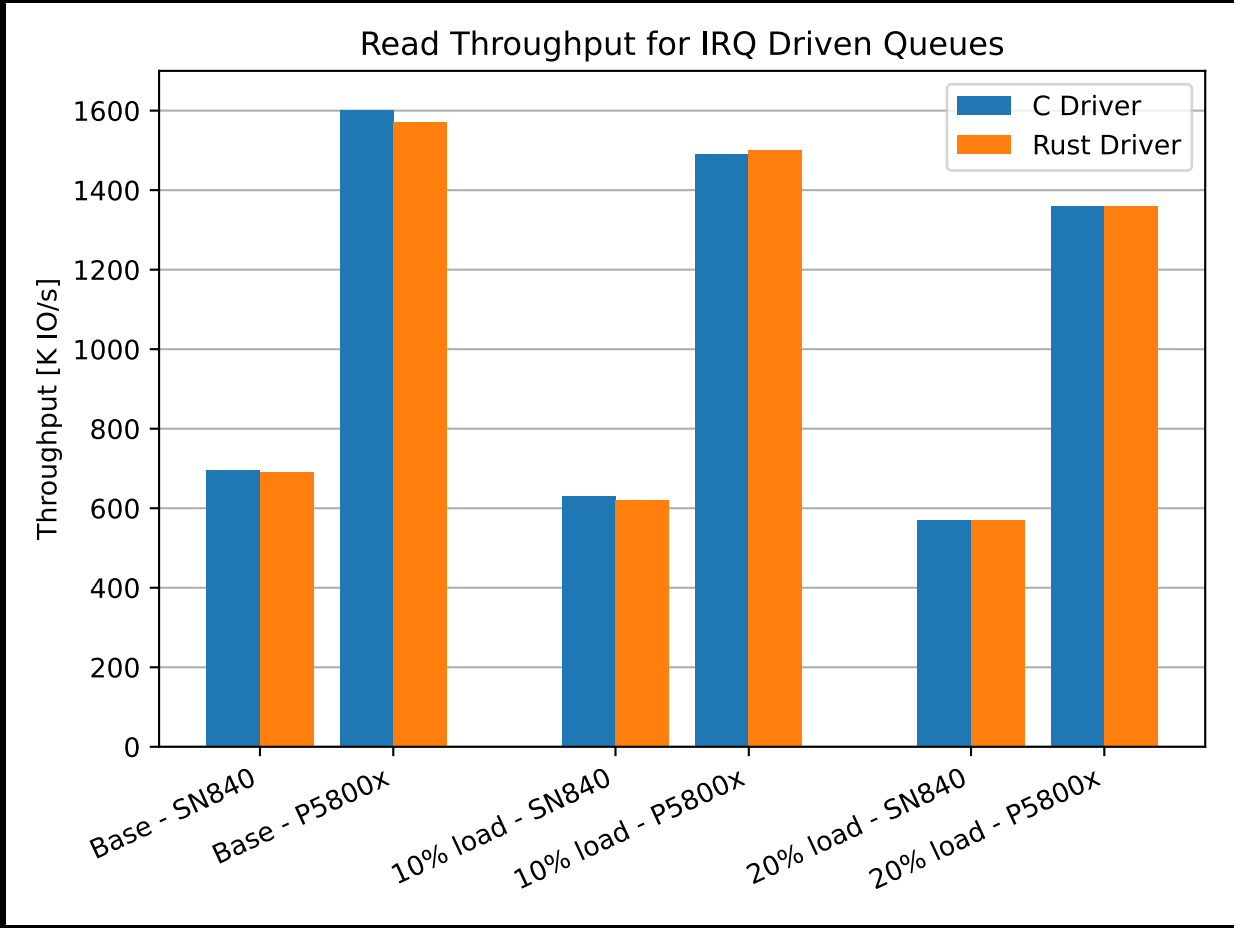
The Benchmarks

Benchmark Setup

- Dell PowerEdge R6525
- 1 CPU socket populated - EPYC 7313, 16 cores
- 128 GB DRAM
- 1x SN840 8GT/s x4 3.94 GB/s (PCIe 3)
- 3x P5800x 16GT/s x4 7.88 GB/s (PCIe 4)
- Debian bullseye (linux 5.10.0-15)
- QEMU 5.2.0 (Debian 1:5.2+dfsg-11+deb11u2)
  - `--enable-kvm`, `-m 32G`, `-cpu host`, `--smp 2`
  - PCI pass-through (vfio-pci)

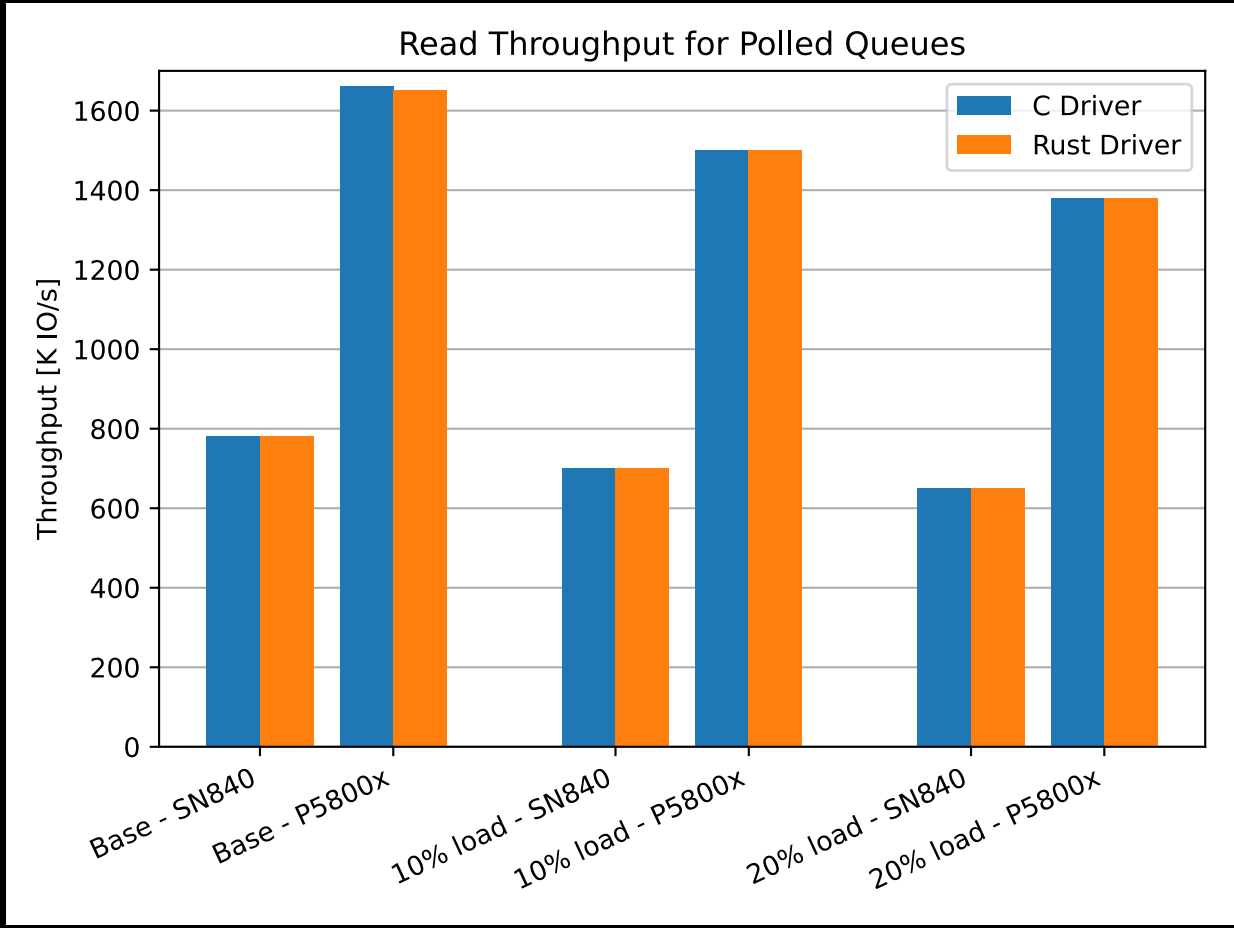
IRQ Driven

```
echo 0 > /sys/block/nvme0n1/queue/iostats
stress-ng --cpu 1 --cpu-load ${LOAD} --taskset 0 &
taskset -c 0 fio/t/io_uring -n1 -R1 -p0 -d128 -s32 -c1  -b4096 -O1 -X1 -r0 -- /dev/nvme0n1
```

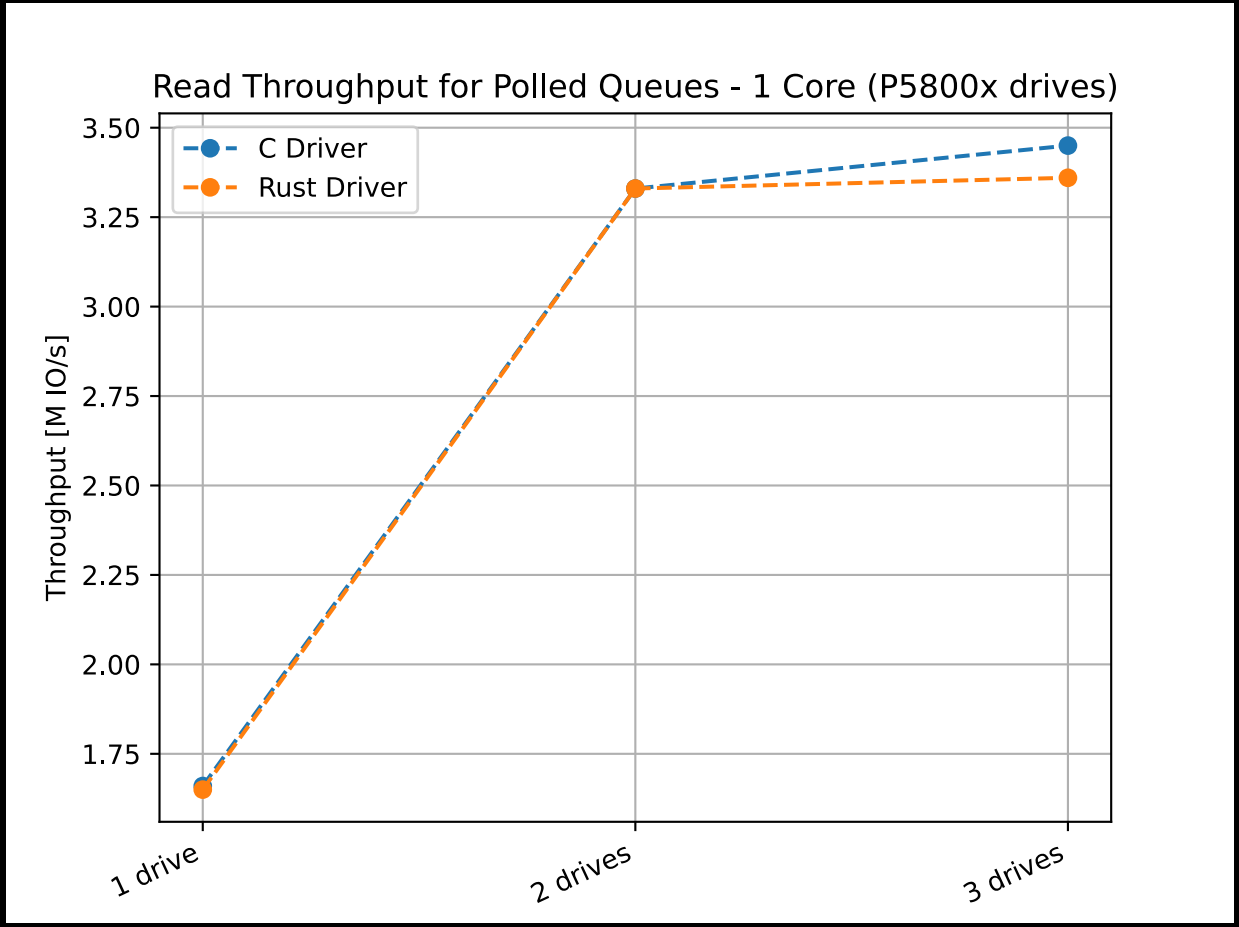


Polled

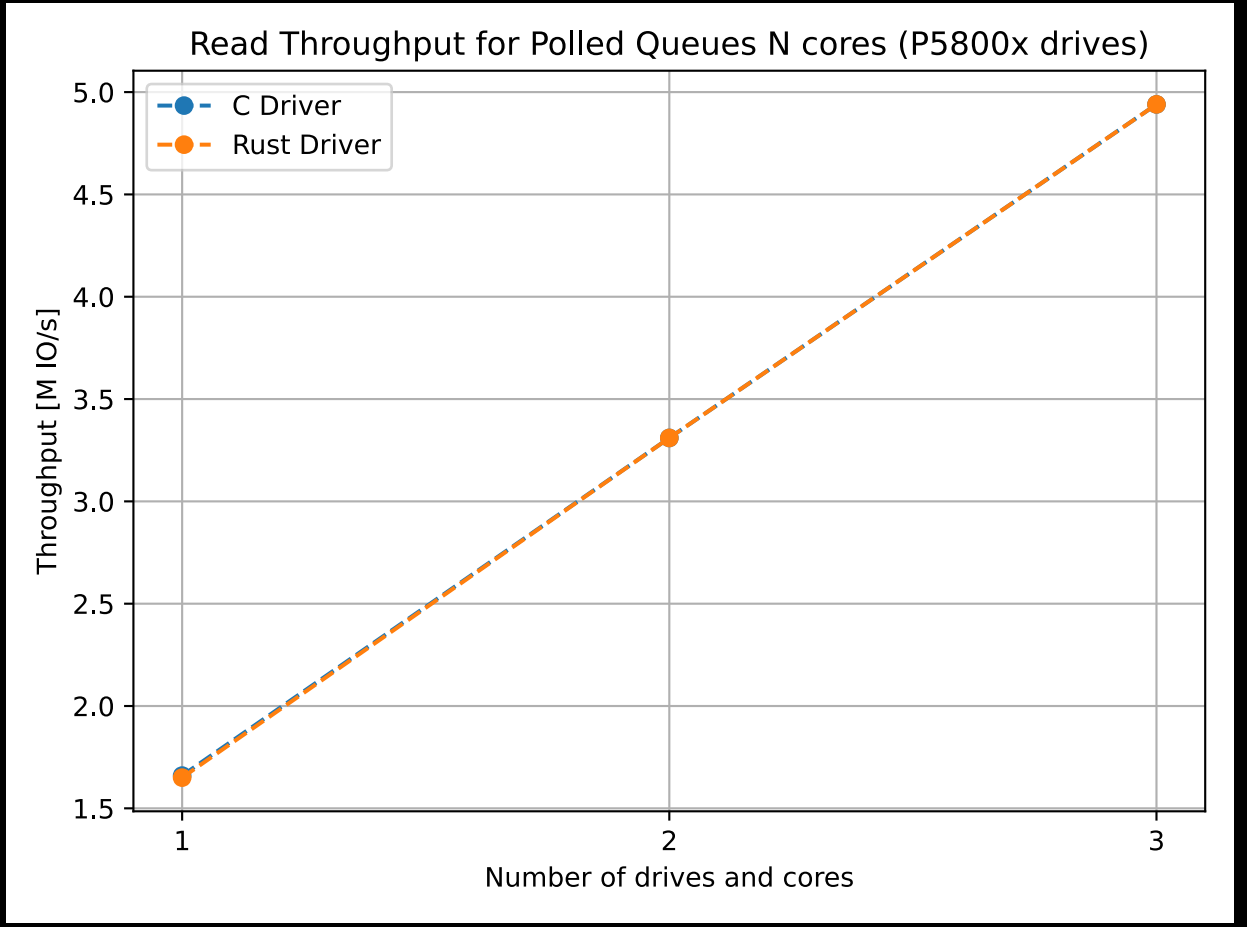
```
echo 0 > /sys/block/nvme0n1/queue/iostats
stress-ng --cpu 1 --cpu-load ${LOAD} --taskset 0 &
taskset -c 0 fio/t/io_uring -n1 -R1 -p1 -d128 -s32 -c1  -b4096 -O1 -X1 -r0 -- /dev/nvme0n1
```



Polled - Multiple Drives 1 Core



Polled - Horizontal Scaling



Credits

- Wedson Almeida Filho: **Rust NVMe driver**
- Andreas Hindborg:
  - Support for physical drives
  - Polling support
  - Multiple device support
  - Refactoring and maintenance
- Contributions welcome!

```
git clone -b nvme https://github.com/metaspace/rust-linux
```

```
git clone -b nvme https://github.com/wedsonaf/linux
```

Future Work

- Get rid of `unsafe` blocks in driver code (implement missing abstractions)
- Support device and driver removal
- Add sysfs nodes (for `nvme-cli`)
- Support delayed initialization - init on task queue
- Build minimal example `blk-mq` driver
- Investigate Rust `async` programming model for `queue_rq` and `complete` 🚀🥳

Future Work - Async

```
fn queue_rq(request: mq::Request) -> Result {
    dma::map_data(request)?;
    queue.submit_cmd(nvme::Command::new(request))?
}
```

```
fn queue_handle_irq(completion: nvme::Completion) -> Result {
    let request = tagset.find_request(completion.command_id)?;
    request.complete()?
}
```

```
fn complete(request: Request) -> Result {
    dma::unmap_data(request)?;
    request.end_ok()?
}
```

```
async fn queue_rq(request: mq::Request) -> Result {
    let dma_guard = dma::map_data(request)?;
    queue.submit_cmd(nvme::Command::new(request)).await?
    request.end_ok()?
}
```

```
fn queue_handle_irq(completion: nvme::Completion) -> Result {
    let task = find_continuation(completion.command_id)?;
    task.mark_ready();
    executor.run_now(task)?
}
```

Questions

Completion Queue Entry Read Ordering

```rust
#[repr(C, packed)]
pub(crate) struct NvmeCompletion {
    pub(crate) result: le<u32>,
    reserved: u32,
    pub(crate) sq_head: le<u16>,
    pub(crate) sq_id: le<u16>,
    pub(crate) command_id: u16,
    pub(crate) status: le<u16>,
}

fn process_completions_bugged(&self) -> i32 {
  loop {
    let cqe = self.cq.read_volatile(head.into()).unwrap();

    if cqe.status.into() & 1 != phase {
        break;
    }

    // Process entry - BUG because read order of CQE fields
  }
}
```

A Quick Fix

```
fn process_completions_fixed(&self) -> i32 {
  loop {
    let cqe = self.cq.read_volatile(head.into()).unwrap();

    if cqe.status.into() & 1 != phase {
        break;
    }

    let cqe = self.cq.read_volatile(head.into()).unwrap();

    // Process entry
  }
}
```