Making io_uring pervasive in QEMU

Stefan Hajnoczi
stefanha@redhat.com

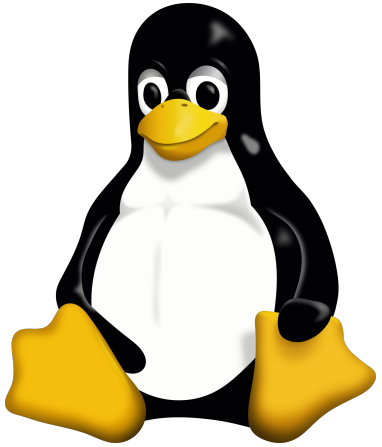
Early io_uring

New syscall API for asynchronous I/O

Added in Linux v5.1 (2019)

Supported 8 operations:

- File I/O (read, write, fsync)
- File descriptor monitoring



Operations
NOP
READV
WRITEV
FSYNC
READ_FIXED
WRITE_FIXED
POLL_ADD
POLL_REMOVE

QEMU's initial io_uring support

New –blockdev aio=io_uring option in QEMU v5.0 (2020)

Uses io_uring for file I/O to disk images

Alternative to aio=threads and aio=native

Performance on par or better than aio=native

"io_uring in QEMU: high-performance disk IO for Linux", Julia Suvorova at FOSDEM 2020
https://archive.fosdem.org/2020/schedule/event/vai_io_uring_in_qemu/attachments/slides/4145/export/events/attachments/vai_io_uring_in_qemu/slides/4145/io_uring_fosdem.pdf

io_uring in 2025

There are 63 operations as of Linux v6.16:

- File I/O
- File descriptor monitoring
- Network I/O
- Timeouts and cancellation
- File system
- uring_cmd (like an async ioctl)
- Synchronization (futex and process wait)
- ...and more

io_uring is not just for disk I/O anymore

Pervasive io_uring in QEMU

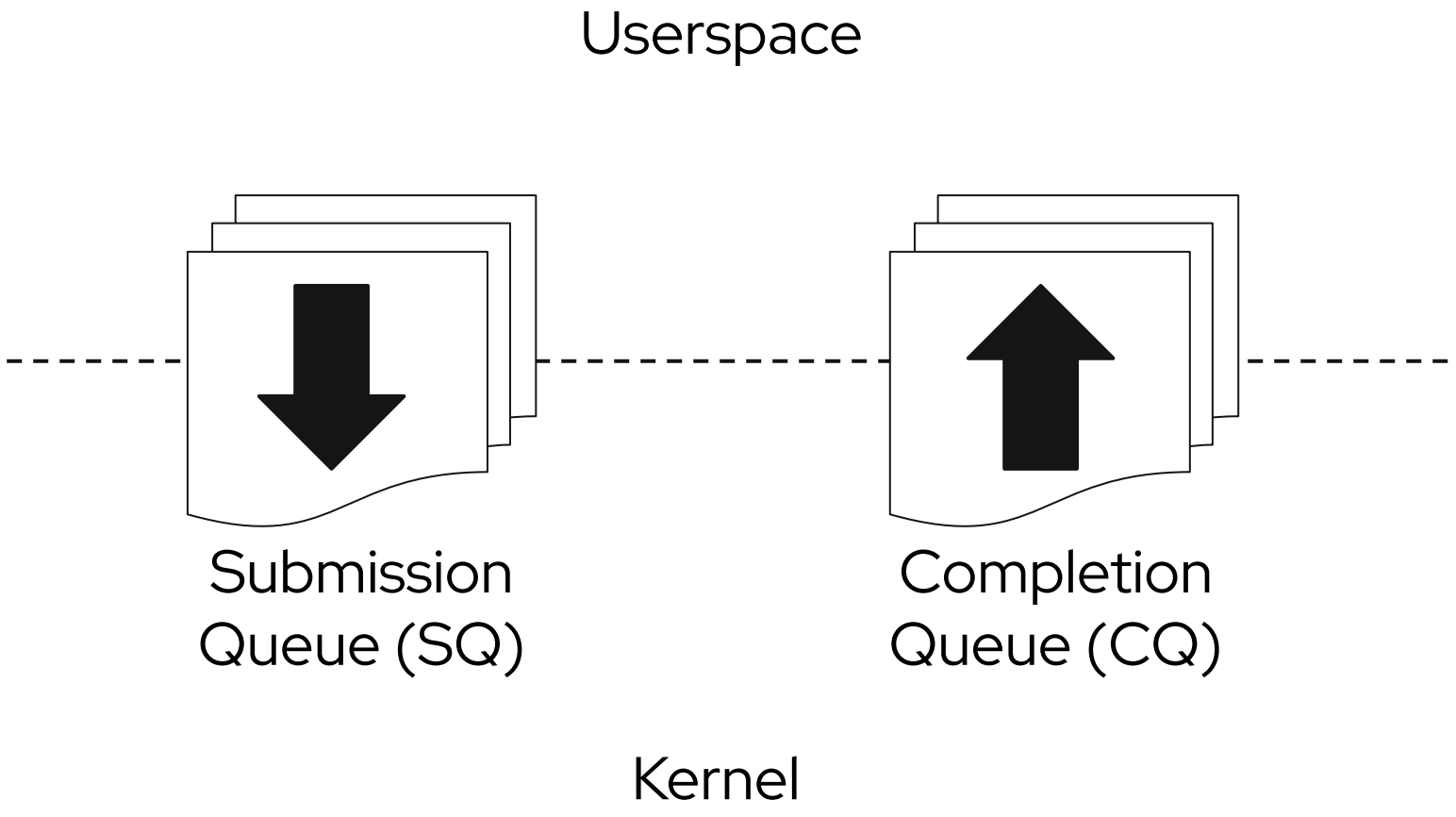
Non-disk I/O code should be able to use io_uring too

Requirements:

- API with minimal setup
- Integration with QEMU's event loop
- Support for custom uring_cmds
- Clean way to detect when is io_uring supported

The io_uring system call interface

io_uring overview



Asynchronous:

- Submission and completion are separate steps

Many requests can be pending at a time

Basically an async system call interface

Batched submission and completion

One system call submits and completes multiple requests:

```
io_uring_enter(fd, unsigned to_submit,
               unsigned min_complete, …)
```

Without io_uring

1. **read(fd_a)**
2. **read(fd_b)**
3. **read(fd_c)**

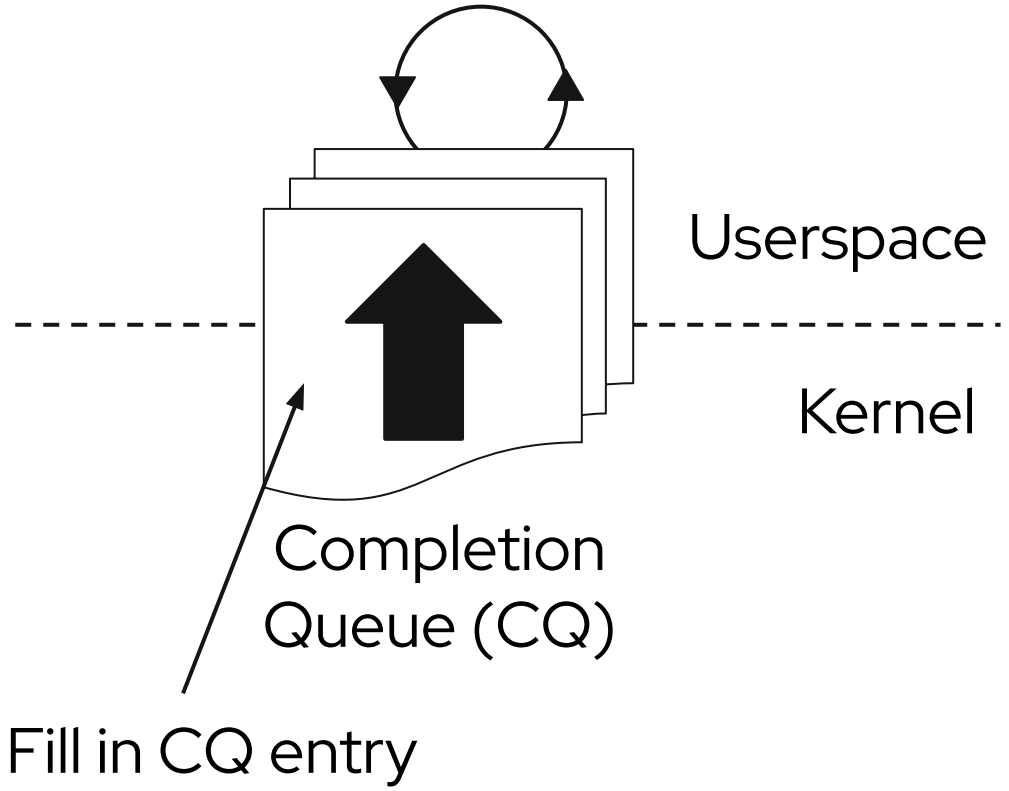
With io_uring

1. Prepare READ fd_a, READ fd_b, and READ fd_c SQ entries.
2. **io_uring_enter()**
3. Check CQ entries.

Fewer syscalls -> less overhead

Userspace completion queue polling

Busy wait for completions



No syscalls required to detect new CQ entries

Good fit for QEMU's adaptive polling in event loop

"Applying polling techniques to QEMU", Stefan Hajnoczi at KVM Forum 2017, https://vmsplice.net/~stefan/stefanha-kvm-forum-2017.pdf

Other modes of operation

QEMU does not use the following:

- SQPOLL - kernel thread busy waits on submission queue
- IOPOLL - busy wait on I/O requests

They don't fit QEMU's architecture and/or have limitations.

How to make all this available in QEMU code?

aio_add_sqe() API

```
typedef struct {
    CqeHandler cqe_handler;
    …per-request data…
} MySqe;
```

```
void prep_my_sqe(
    struct io_uring_sqe *sqe,
    void *opaque)
{
    …fill in sqe fields…
}
```

```
void my_cqe_handler(CqeHandler *h)
{
    …process h->cqe…
}
```

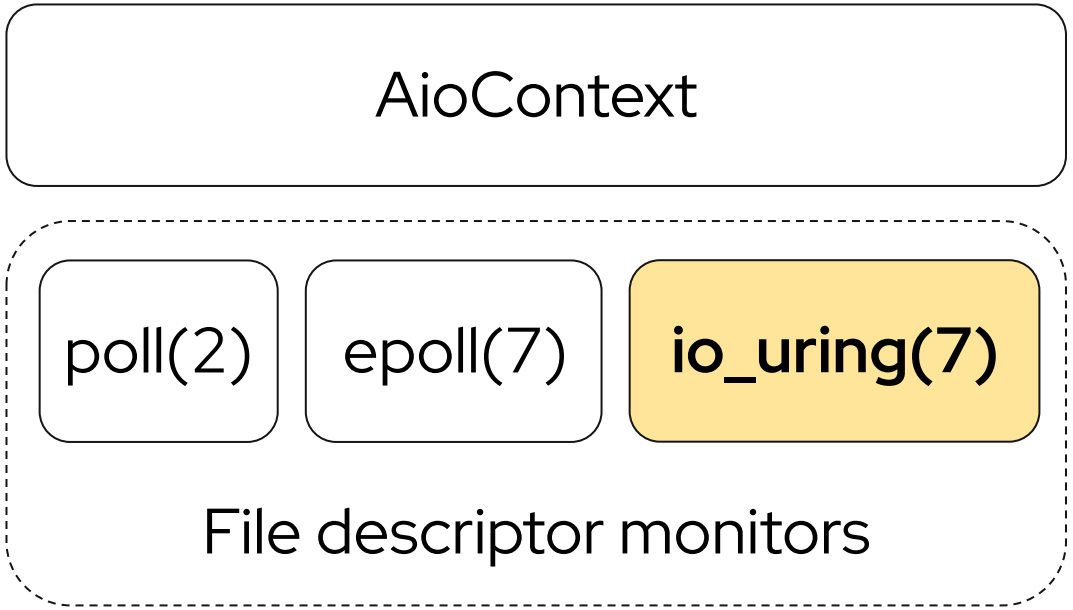
```
my_sqe->sqe_handler.cb =
    my_cqe_handler;
aio_add_sqe(prep_my_sqe, my_sqe,
            &my_sqe->cqe_handler);
```

Per-request struct

SQE builder function

CQE handler function

Implementation



New fdmon implementation for io_uring

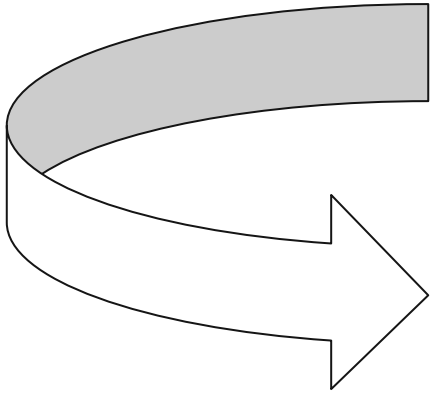
- Monitors fds for classic aio_set_fd_handler() APIs
- Submits custom SQEs for aio_add_sqe().

Source code: util/fdmon-io_uring.c

Gives io_uring powers to QEMU's event loop!

How adaptive polling works

Check CQ ring alongside other polling sources:



```
while (busy_waiting) {
    …poll…
    if (io_uring_cq_ready(io_uring)) {
        break; /* go process CQEs */
    }
}
```

Pretty nifty: adaptive polling now works on all file descriptors!

Portability

1. Only compile io_uring code on Linux hosts:

   #ifdef CONFIG_LINUX_IO_URING

2. Detect availability at runtime:

   bool aio_has_io_uring(void)

   io_uring may be disabled on host or inside container runtime.

Request cancellation

Submit ASYNC_CANCEL using aio_add_sqe()

Future work: an aio_cancel_sqe() API to avoid code duplication?

Use cases

Rewriting aio=io_uring on top of aio_add_sqe()

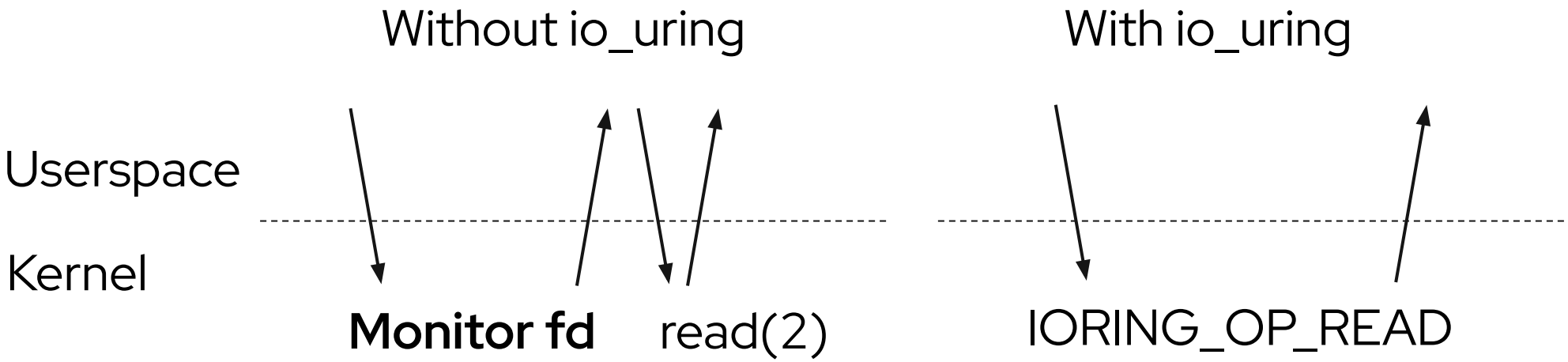
Stop open coding io_uring in block/io_uring.c

Submit READ/READV, WRITE/WRITEV, FLUSH SQEs

10 files changed, **140 insertions(+), 493 deletions(-)**

-> aio_add_sqe() reduces code duplication

Replacing eventfd monitoring with io_uring ops



Eliminate high-frequency syscall using io_uring

System call pattern with io_uring

(0.052 ms): iothread1 io_uring_enter(fd: 11 io_uring, to_submit: 73, argsz: 8) = 73

(0.004 ms): iothread1 write(fd: 41 irqfd, buf: 0x1, count: 8) = 8

(0.003 ms): iothread1 write(fd: 42 irqfd, buf: 0x1, count: 8) = 8

(0.003 ms): iothread1 write(fd: 43 irqfd, buf: 0x1, count: 8) = 8

(0.002 ms): iothread1 write(fd: 40 irqfd, buf: 0x1, count: 8) = 8

(0.051 ms): iothread1 io_uring_enter(fd: 11 io_uring, to_submit: 71, argsz: 8) = 71

...

irqfd write, latency matters, maybe don't batch it

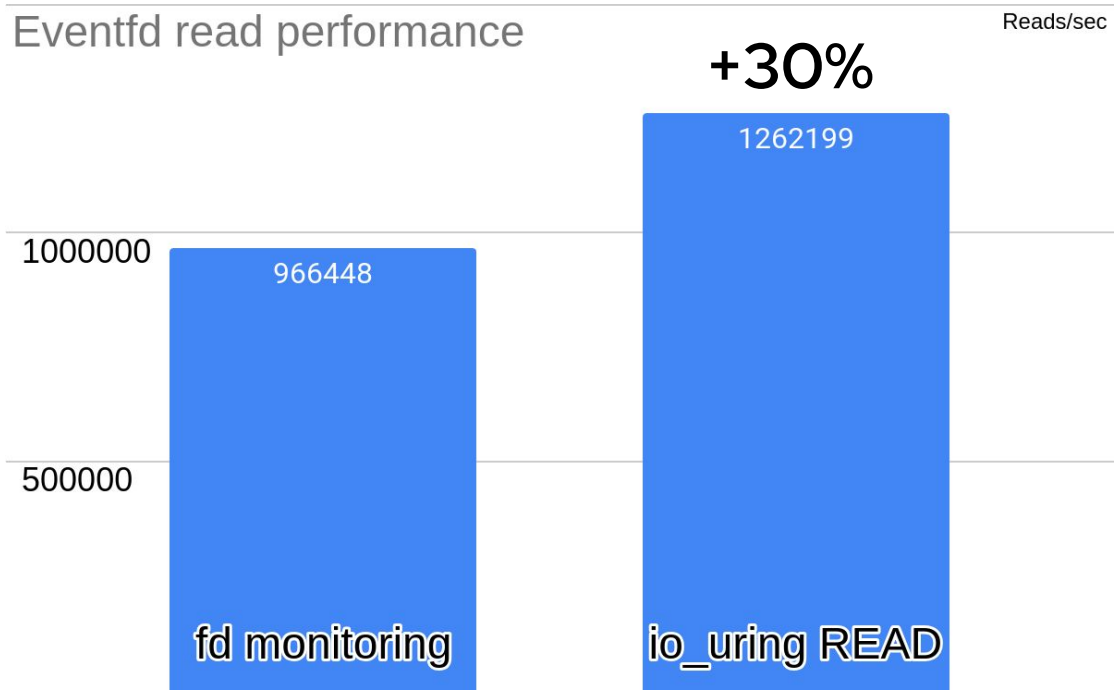
No more read(2) syscalls

Microbenchmark

Measure eventfd reads per second:

- QEMU IOThread increments counter each time eventfd is readable
- Other thread writes to eventfd as fast as possible

Macro virtio-blk results inconclusive so far.



Benchmark code: https://gitlab.com/stefanha/qemu/-/commit/7552bae27d5953ac08e90816172dd5a55dbd3378

Conclusion

Status

aio_add_sqe

- Patches on qemu-devel mailing list
- **Expected in QEMU 10.2**

EventNotifier using io_uring READ

- Still investigating macro performance impact

Use aio_add_sqe()

You can now try io_uring in your code without big QEMU changes

Current work:

- FUSE-over-io_uring (Brian Song via Google Summer of Code)

Future work:

- Accelerate live migration with io_uring networking?
- Your ideas!