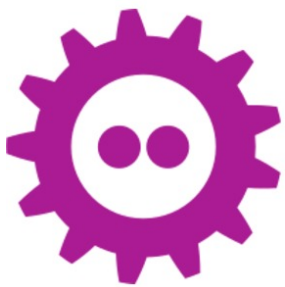
Cancelling POSIX syscalls in Managarm - an asynchronous microkernel-based OS

Geert Custers
geert@managarm.org

The Managarm Project



FOSDEM

Bio

Education (Technical University of Delft)

- MSc. Computer Engineering (2022-current)
- Bsc. Electrical Engineering (graduated cum laude 2022)

Experience

- Software Engineer at Hadrian Security since 2022
- Part of the Managarm project since 2019.

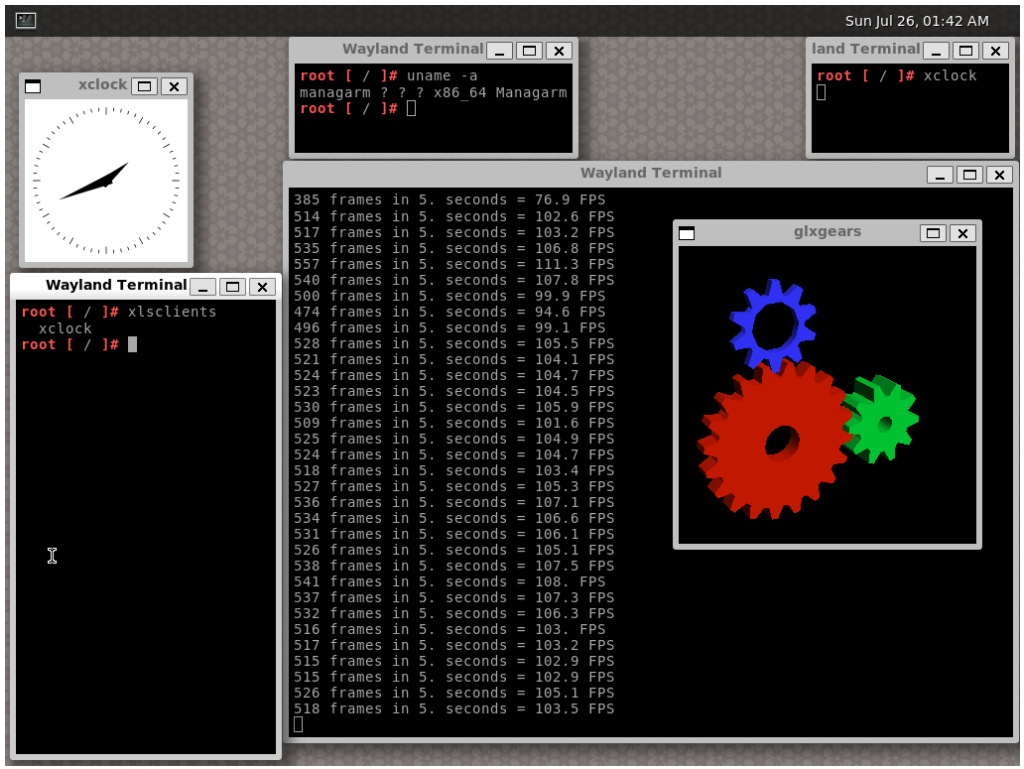
Github: `@Geertiebear`

Agenda

- Basics of POSIX signals
- The kernel side of POSIX signals
  - Monolithic kernels
  - The challenges in microkernels
  - Managarm's solution
- Lessons learned

What is Managarm?

- Microkernel based, fully asynchronous hobbyist operating system



POSIX signals

- Asynchronous interrupts for userspace.
- Blocking syscalls can be interrupted by signals.

Interrupting syscalls

- Suppose we enter `read()`
- Return value depends on when syscall is interrupted

```c
int ret = read(fd, buf, sizeof(buf));
if (ret == -1 && errno == EINTR) {
    // read() was interrupted before the kernel could complete
    // any work, we should try again.
} else {
    // read() finished successfully, or was interrupted but completed
    // partial work
}
```

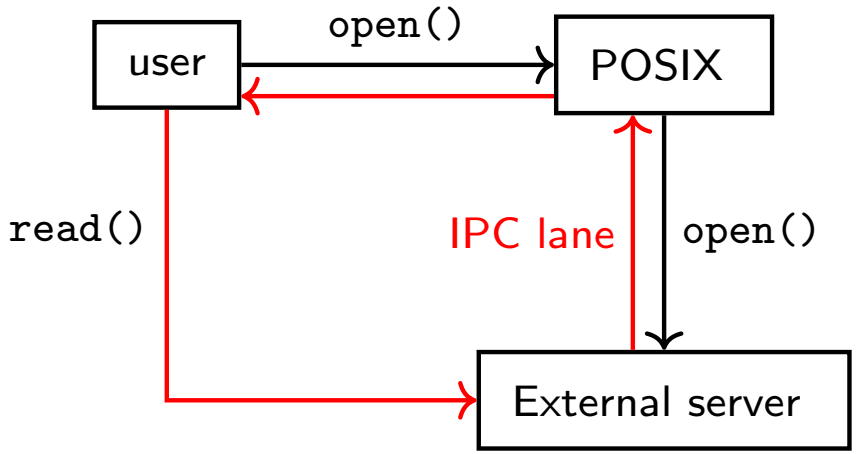
Fun fact: `close`'s behaviour on interruption is undefined.

Monolithic implementation

- All bookkeeping is in the same address space
- Identify cancellation points in each syscall
- Check for signal delivery at each point

What about Managarm?

- Managarm has single POSIX server.
- Can delegate to external servers.
- Passthrough lanes are especially difficult to handle.



Managarm is asynchronous

- User submits request, this is added to a working queue.
- Coroutine lifetime is independent of process lifetime.

```
auto requestData = co_await helix::exchangeMessages(
                                        conversation, ...);
// Calling process can be interrupted here, how do we know?
auto readResult = co_await file->read(...);

// Same here :/
co_await helix::exchangeMessages(conversation, readResult)
```

Leveraging Managarm events and IPC

- Kernel provides `oneshotEvent` primitive.
- Can be easily passed across process boundaries.

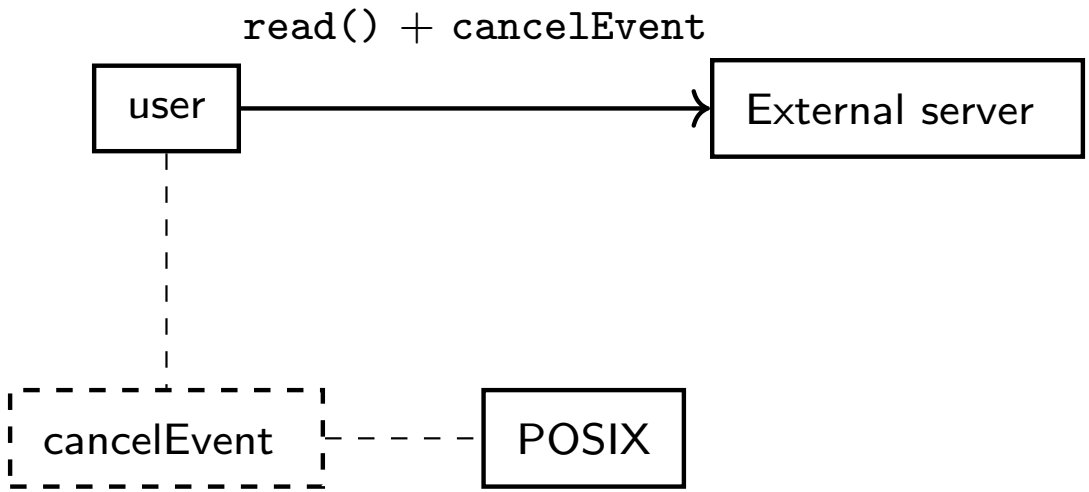
```
auto [...] = exchangeMsgsSync(handle,
    helix_ng::offer(
        // ...
        helix_ng::pushDescriptor(cancel_event),
        // ...
    ))
```

- We can take advantage of consolidated POSIX server.

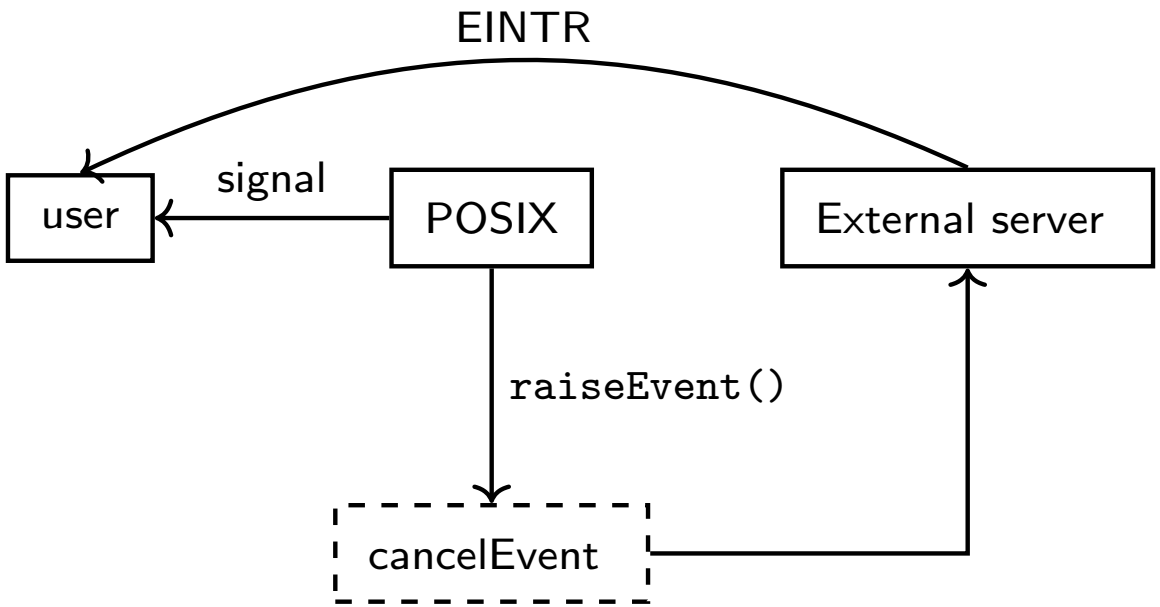
Current syscall model

- POSIX loads event handle into a shared memory page.
- Userspace attaches this event to every EINTR syscall.
- If POSIX delivers signal it also raises the event.

Current syscall model



Current syscall model



Lessons learned

- Asynchronous programming across process boundaries makes for confusing lifetimes.
- Abstracting cancellation using cancel tokens and events makes for readable code.
- Pragmatic approach of POSIX server makes burden on userspace smaller.

Acknowledgements

Check out the project: `github.com/managarm/managarm`

Blog: `managarm.org` Youtube: `@TheManagarmProject`

Twitch: `twitch.tv/geertiebear` Discord: `discord.gg/7WB6Ur3`

Thanks to all contributors!