Introducing Helios

A small, practical microkernel

Drew DeVault
SourceHut

January 9, 2023

Introduction

- Hi
- I'm Drew DeVault
- Project lead for Hare
- Other stuff too
- Moving swiftly along

`https://ares-os.org`
`https://harelang.org`
`https://drewdevault.com`

What is Helios?

Helios is a microkernel, largely inspired by seL4. It is written in Hare and runs on x86_64 and aarch64; RISC-V is planned.

- $\approx$ 8,500 lines of portable code
- $\approx$ 3,000 lines non-portable per architecture
- GPL 3.0

Note: Line counts do not include the bootloaders

What is Helios, cotd

What works?

- Capability-based security
- IPC (similar to seL4)
- Preemptive scheduling (single core, no SMP)
- Hardware I/O (ports or mmio), IRQs
- EFI (aarch64) or multiboot (x86_64)

Why is Helios?

- Kernel hacking is really fun
- Prove if Hare is useful for this purpose
- Can we do better than seL4?
- Can we do better than, dare I suggest, Linux?

A brief introduction to Hare

Hare is a systems programming language designed to be simple, stable, and robust. Hare uses a static type system, manual memory management, and a minimal runtime. It is well-suited to writing operating systems, system tools, compilers, networking software, and other low-level, high performance tasks.

A brief introduction to Hare

- General purpose systems programming language
- 3 years in development
- 18,000 line compiler (C11)
- 12,000 line backend (C99)
- x86_64, aarch64, riscv64

What does Hare look like?

```
export @noreturn fn kmain(ctx: arch::bootctx) void = {
        log::printfln("Booting Helios kernel");

        const pages = init::pages(&ctx);
        let heap = init::heap_init(&ctx, pages);
        let task = init::task_init(&heap, ctx.argv);
        init::load(&task, &ctx.mods[0]);
        init::heap_finalize(&task, &heap, &ctx);
        init::devmem_init(&task);
        init::finalize(&task);

        log::printfln("Entering userspace");
        sched::init();
        sched::enteruser(task.task);
};
```

The design of Helios

Let's go over the main talking points about its design:

- Capabilities
- Memory management
- Address spaces & tasks
- IPC

And implementation:

- Bootloader
- System initialization
- Runtime API

Capabilities

	CSpace
0x00	Memory
0x01	Device memory
0x02	Task
0x03	I/O port
0x04	IRQ
0x05	null (link 0x06)
0x06	null (link 0x07)
0x07	null (link ...)
	...
0xFF	(null)

- Each task (process) has a CSpace
- Configurable number of capabilities
- Uses MMU to enforce capabilities

Similar to seL4, but:

- **Not** a guarded page table
- Free list for O(1) capalloc in kernel

Capabilities

- Null
- Memory
- Device memory
- Capability space
- Virtual address space
- Task
- Endpoint
- Notification
- Reply
- ASID control
- ASID pool

Plus, on x86_64:

- PDPT
- PD
- PT
- Page
- I/O control
- I/O port
- IRQ control
- IRQ handler

Memory management

- Different from seL4: Free list instead of high watermark
- It's not that interesting
- This slide only exists to mention the seL4 thing
- But I feel the need to fill in some of this whitespace
- You just lost the game

Address spaces

VSpace capabilities (like seL4) are capabilities that manage address spaces. Pages can be shared, but not page tables, at least until we know why seL4 doesn't let you share page tables so we can stop cargo-culting that constraint.

Tasks

Tasks have a CSpace (optional) and VSpace and receive CPU time when configured appropriately. Simple round-robin scheduler for now; more sophisticated later.

IPC: Notifications

- Asynchronous: signal never blocks, wait blocks
- Same as seL4

IPC: Endpoints

- Synchronous: send and recv both block, wait for rendesvouz
- Sends registers and/or capabilities
- Supports capalloc
- seL4-style call/reply supported
- Code generation!

IPC: Poll

```
let poll: [_]pollcap = [
        pollcap { cap = IRQ, events = pollflags::RECV },
        pollcap { cap = EP, events = pollflags::RECV },
];
for (true) {
        helios::poll(poll)!;
        if (poll[0].events & pollflags::RECV != 0) {
                poll_irq();
        };
        if (poll[1].events & pollflags::RECV != 0) {
                poll_endpoint();
        };
};
```

Booting Helios

System initialization

Runtime kernel API

Does it work?

I am using Helios to present this slide deck.

- Ported system to aarch64 over the last eight weeks
- Simple GPU driver in userspace
- Serial port to switch slides/etc (USB in eight weeks? hah!)
- Slide deck is PNG files in a tarball functioning as the initramfs

Userspace ambitions

Plans:

- Helios: kernel
- Also: Vulcan: test userspace
- Mercury: driver framework
- Gaia: userspace interface
- Luna: POSIX compatibility layer
- Ares: complete operating system package

What's next?

Acknowledgements

yyp, ecs, hare community

Closing thoughts

Mention IRC channel here