FerrOS

Experience Report

Zack Pierce

Auxon Corporation

Zack Pierce

- Reliable, distributed systems

Auxon Corporation

- Tools that solve problems rather than defer them
- Focus on cyber-physical

SERIOUS

pragmatic

SERIOUS

- Strong memory isolation for operational units
- Identify and handle all sources of fallibility in software
- Software unit isolation for fault localization
- Integrate with industrial software tooling
- Operate on low-powered devices
- Formally verified for all the things

SERIOUS

- Strong memory isolation for operational units
- Identify and handle all sources of fallibility in software
- Software unit isolation for fault localization
- Integrate with industrial software tooling
- Operate on low-powered devices
- Formally verified for all the things

  - Work alongside black-box software from third party vendors
  - Handle custom device and virtual memory mapping schemes
  - Guarantee no runtime memory allocation failures
  - Deployable to moderately esoteric platforms

SERIOUS

- Work alongside black-box software from third party vendors
- Handle custom device and virtual memory mapping schemes
- Guarantee no runtime memory allocation failures
- Deployable to moderately esoteric platforms

- Strong memory isolation for operational units
- Identify and handle all sources of fallibility in software
- Software unit isolation for fault localization
- Integrate with industrial software tooling
- Operate on low-powered devices
- Formally verified for all the things

- Rapid startup from cold beginning
- Component requirements made explicit in contracts for development unit coordination
- Detect or prevent distribution of inaccessible resources to unprivileged components
- Auditable internal communication graph

pragmatic

- Cheap

pragmatic

- Cheap developers
  - Formal verification skills not required
  - Divine C skills not required
- Effective development process
  - Doesn’t bog down in unnecessary work

FerrOS Wins!

- Align seL4 capabilities with functionality
- Never run out of [your resource here]
- Compose isolated, interacting processes
- Integrate with dev-friendly tooling

Agenda

- Foundation
- How did FerrOS get those wins?
- Tradeoffs

Foundation

- Rust
- selfe
  - selfe-sys
  - selfe-config (and selfe executable)
  - selfe-arc
- Open Source

FerrOS Wins!

Align seL4 capabilities with functionality

Align seL4 capabilities with functionality

- Autocomplete support for capability functions
- Compile-time overwrite checks for capabilities
- Hide capability pointer address math

Align seL4 capabilities with functionality

```
let x: PageTable = …;

x.signal();
```

Compile time error!

```
let x: Notification = …;

x.signal();
```

Compilation success.

Align seL4 capabilities with functionality

```
seL4_Untyped_Retype(
    service_cptr,
    sel4_type_id,
    size_bits,
    dest_cptr,
    index,
    depth,
    dest_offset,
    1,
)
```

```
let x: ThreadControlBlock =
    untyped.retype(c_slot);
```

Align seL4 capabilities with functionality

```
struct Cap<CapType, CNodeRole> {
    cptr: usize,
    cap_data: CapType,
    _role: PhantomData<CNodeRole>
}
```

Never run out of [your resource here]*

* at runtime

Never run out of [memory]

```
let ut: Cap<Untyped<11, _>, >> = …;

let (ut_a, ut_b) = ut.split();
// They’re both Untyped<10, _>

let tcb = ut_a.retype(cslot_a);

let other_thing = ut_b.retype(cslot_b);
```

Never run out of [capability slots]

```
let slots: CNodeSlots<22, Local> = ...;

let (slot_a, leftover_slots) = slots.alloc();
// Now leftover_slots = CNodeSlots<21, >

let useful = untyped.retype(slot_a);
```

Never run out of [ASID Pool space]

```
let (unassigned_asid, remaining_pool) =
  asid_pool.alloc();

// unassigned_asid holds type UnassignedASID

let assigned_asid: Cap<AssignedASID> =
  unassigned_asid.assign(&paging_root);
```

Compose isolated, interacting processes

Compose isolated, interacting processes

- IPC made easy and safe
- Thread and process startup
- Process-embedding and loading

Compose isolated, interacting processes

```
let (tx_maker, rx) =
  call_channel(ut, root_cnode,
               slots, rx_slot )?;

let caller = tx_maker.create_caller(slot);

let response: YourResponse =
  caller.blocking_call(&MyFancyStruct {…})?;
```

Compose isolated, interacting processes

- Tree-like thread and process creation
- Start processes with strongly typed parameters
- Organize sending capabilities accessible from the child process

Compose isolated, interacting processes

```
pub struct ProcParams<Role> {
    pub uart: UART1,

    pub int_consumer: InterruptConsumer<uart1::Irq, Role>,

    pub storage_caller: Caller<
        persistent_storage::Request,
        Result<persistent_storage::Response, persistent_storage::ErrorCode>,
        Role>,

    pub udp_producer: Producer<Role, IpcUdpTransmitBuffer>,
}


// In the process binary (main.rs)
pub fn _start(params: ProcParams<role::Local>) -> ! {
    // Do the work of the process here, using the provided params
}
```

Integrate with dev-friendly tooling

Integrate with dev-friendly tooling

- Lean on extant Cargo ecosystem and idioms
- ferros-build utility library
  - Delegate to selfe-arc for embedding
  - ELF FTW
-
- No separate specification languages or markup required

Agenda

- ~~Foundation~~
- ~~How did FerrOS get those wins?~~
- Tradeoffs

Trade-offs

```
let (c, a) = a.do(b);
let (d, a) = a.go(q);
        ...
```

Macros!

Trade-offs

Typenum math slows down compilation

Patience

Smaller code units

Hope for core support for more `const` math

Trade-offs

Seriously strong mode can be rigid

Weak-mode utilities
Could be more consistently symmetrical

FerrOS Wins!

- Align seL4 capabilities with functionality
- Never run out of [your resource here]
- Compose isolated, interacting processes
- Integrate with dev-friendly tooling

FerrOS

Experience Report

Jon Lamb
Russ Mull
Zack Pierce
Dan Pittman

@ Auxon Corporation