

# Robigalia

## Rust, seL4, and Persistent Caps

Corey Richardson

Clarkson University

Slides URL - <https://robigalia.org/seL4-ws-2016-12-15.pdf>

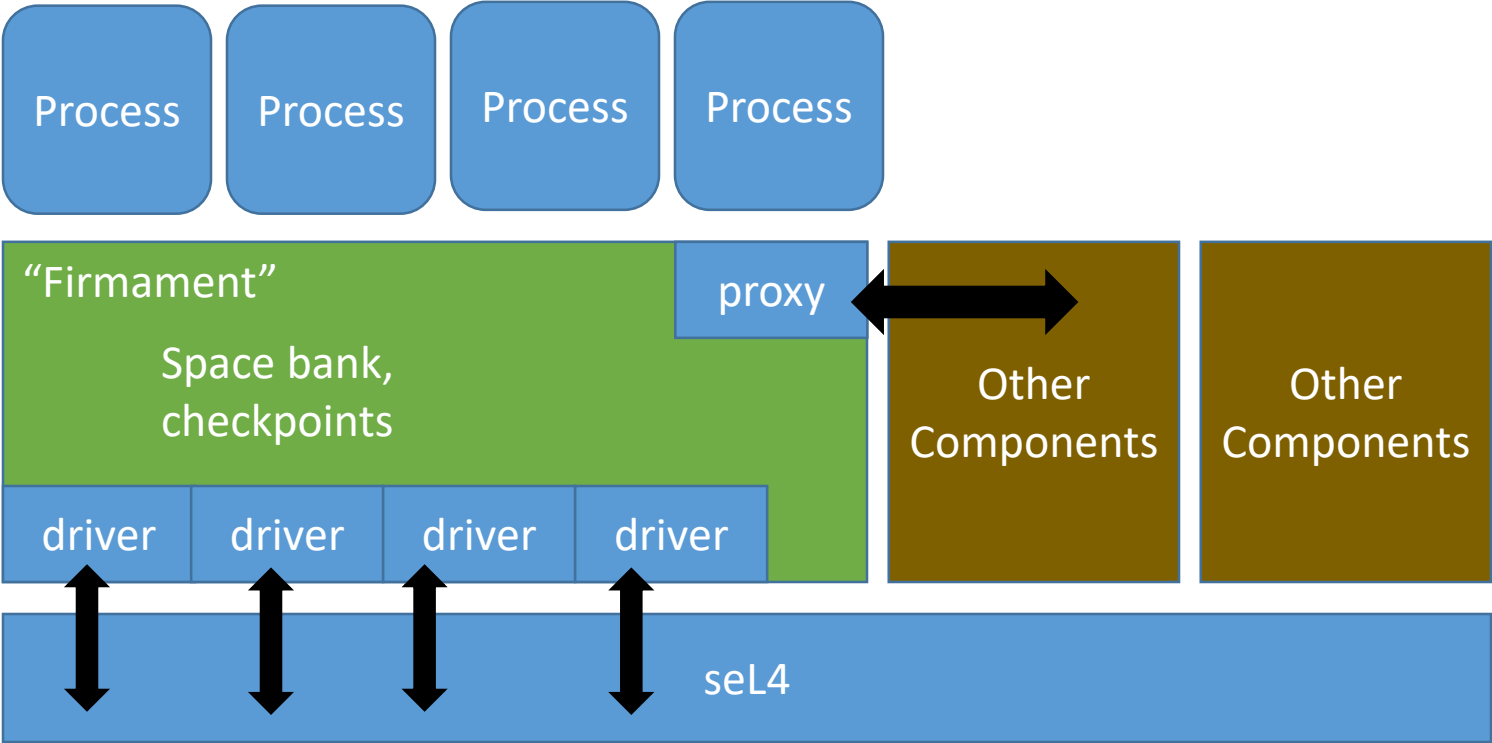
To the extent possible under law, Corey Richardson has waived all copyright and related or neighboring rights to Robigalia: Rust, seL4, and Persistent Caps. This work is published from: United States.



# What?

- Create a reliable dynamic operating system
- Using Rust (some components will be implemented in C and formally verified)
- With seL4
- Initially targeting x86\_64 workstation/server class HW.
- ... but also aiming at RISC-V and some ARM boards, in the future.
- Primary target is eventual verified RISC-V implementations.
- Vaguely like Genode in scope, but more specific to seL4 for now.

# System Structure



# Why?

- I enjoy working on systems, and seL4 is a neat kernel
- Try to continue the legacy of KeyKOS, EROS, and Coyotos
- See how far Rust can be pushed as an OS implementation language

# Who?

- Me, Corey Richardson
- Alex Elsayed
- <https://robigalia.org/>

# Rust

- Rust is a recently developed language out of Mozilla Research
- “Safe, concurrent, fast: pick three”
- Implementation language for their new parallel web browser engine
- Advanced type system for managing concurrent access to data, while forbidding data races and memory unsafety
- ... but for this talk, not terribly relevant!

# Challenges

- Resource Accounting
- Extending seL4's capability model
- System persistence
- Drivers

# Resource Accounting

- Finite amount of storage capacity on a local system
  - What processes get to use it?
  - How does that come to pass?
- 
- Finite amount of processing power on a local system
  - What processes get to use it?
  - How does that come to pass?



# Space Banks for Resource Accounting

- Name ripped from EROS, but inspired by seL4's Untyped objects
- Intuitive, visceral understanding: the system manages all storage in a central bank. Can't use storage unless you pay for it.
- Can invoke a space bank to purchase address space, capability space, other seL4 objects.
- Space bank is primary trusted OS service.

# Space Banks

- A space bank at runtime is an endpoint.
- Important piece of state: current quota
- Three management operations: revoke, create\_child, and verify.
- Revoking a space bank revokes all seL4 objects created from the space bank, returning that storage to your quota.
- Creating a child space bank gives you a new space bank with its own (larger or smaller) quota.
- Verify lets clients ask their (trusted) space bank if it recognizes another capability (received from a potentially untrusted client) as a valid space bank.

# Space Banks

- Can also do “auctions”. Allows processes to collaborate to exchange storage capacity.
- Can create a bid object from a space bank, and store objects in that bid. Objects stored in a bid are inaccessible until bid is destroyed.
- Can offer a bid into an auction, and send the auction cap (badged endpoint) to another thread which can offer its own bid.
- At any point, a thread can destroy its bid, which cancels the auction. **Important** to prevent malicious threads holding resources “hostage”.
- Once both threads agree to the auction, the space bank implementation swaps ownership of the objects between the source space banks, including properly adjusting their quotas to reflect this.

# Space Banks

- Possibilities with auctions:
- Give another space bank more quota, from your own.
- Give another space bank a region of address space that you no longer need/use.
- ... many more possibilities, not all useful. Very general mechanism.
- **Important:** once you auction off some quota, you can't get it back by revocation!
- But once the space bank that bought your quota is revoked, the storage returns to you.

# Space Banks

- **Some important properties:**

- All bytes are always accounted for
- All owned storage can always be revoked, returning quota to the caller promptly
- Storage cannot be manufactured except for by the firmament (which is hopefully only in response to things like new disk inserted or memory hotplug)

# ??? for Time Accounting

- Still kicking around ideas for how to do this. Some initial ideas, nothing ready yet.
- Need to think hard about the new RT extensions and what properties we want.

# Safe Revocation

- Revoking an untyped object which has some page tables mapped into it will cause gnashing of teeth for the process using those page tables.
- Need to *very carefully* design servers exposed to untrusted clients to both avoid resource DoS, resilience to revocation.
- How do you do it?

# Bushels

- A **bushel** is the minimum granularity of isolation in a Robigalia system. Not a specific thing, but a design pattern for robust cooperation.
- Corresponds directly to a badged endpoint.
- Server manages a map from badge -> bushel.
- Few different ways to do this, but a nice one possible where state per client is fixed size is to use it as index into huge array of address space.
- Some more complex mappings possible that are robust to revocation.



# Bushels

- Each bushel is associated with a single space bank, which is usually given by the client when creating the bushel.
- When receiving a message on an endpoint, server first inspects the badge and determines the bushel ID.
- Stores the bushel ID while servicing the request.
- **Only access memory that was allocated from that bushel's space bank**, or the server's own space bank (but be careful to avoid DoS!)
- If there is a fault while processing a request for a bushel, the fault handler will reset the server, which will then deallocate any other state corresponding to that particular bushel.

# Bushels

- This is revocation-safe. If a thread non-cooperatively revokes its space bank that it gave to a server to service its requests, it can only hurt itself.
- Server still able to service requests by other clients.
- Does this scale? **Not necessarily.** Hard for mutually untrusting processes to collaborate to pool their resources for the server to use.
- Our hope is that this will be rare enough to allow per-case design of protocols for those cases.
- If not, some ideas of how to extend.

# Extending seL4's IPC

- Desired features: large messages ( $>$  IPC buffer size)
- “Extended Virtual Message Registers”
- Two threads can agree to use some shared memory to act as extended storage for messages.
- Write first bit of message into IPC buffer, rest into shared memory, then send seL4 IPC. Minor isolation hole.
  
- Also a protocol for sending multiple different capabilities (very simple), but not atomic.
- Possibility: central trusted implementing bulk transfer of caps and data which does copying across address spaces. Closes isolation hole.

# Persistence and Drivers

- Still working on a design for this.
- Eventual goal is to have ~all userspace processes able to be transparently persisted to secondary storage, with checkpointing and restore.
- Driver story is unfortunate. Currently working on things like ACPI and PCIe. virtio drivers soon.
- Hope to use rump kernel drivers in the short run.
- Long run: verified drivers for some devices.

Questions?