

Rust support in seL4 userspace

Overview and update

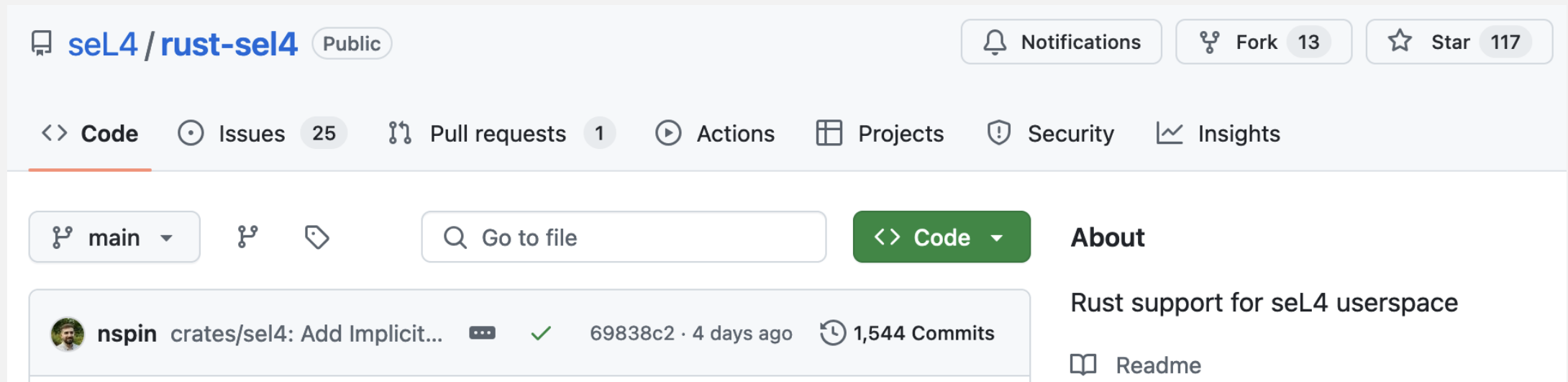
Nick Spinale <nick@nickspinale.com>
seL4 Summit
October 16th, 2024



Official Rust support for seL4 userspace

Rust support for seL4 userspace has been an official seL4 Foundation project since November 2023

<https://github.com/seL4/rust-sel4>

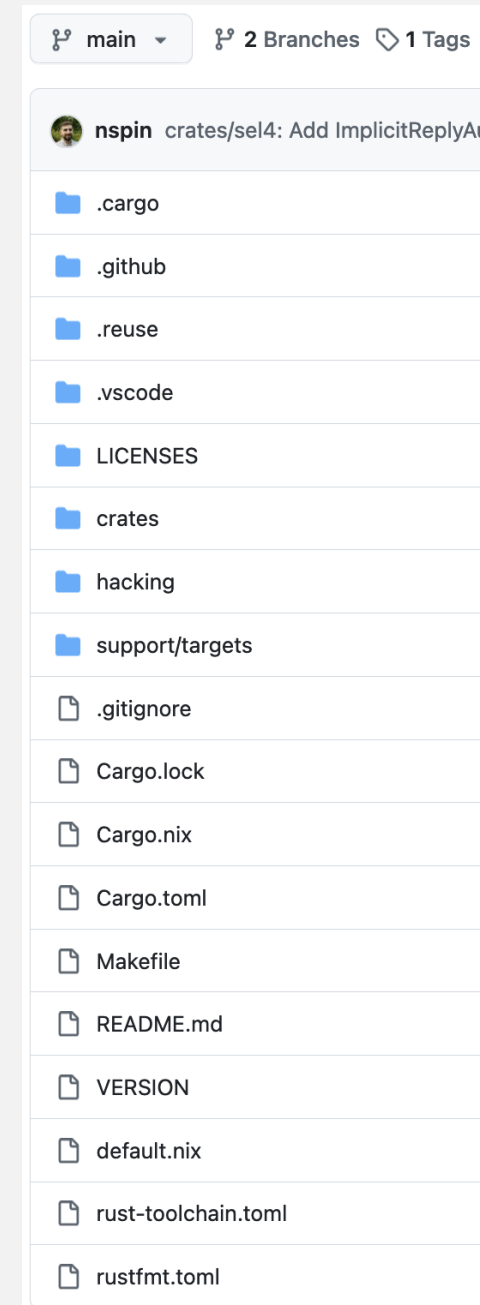


The screenshot shows the GitHub repository page for `seL4/rust-sel4`. The repository is public and has 117 stars, 13 forks, and 25 issues. The main branch is selected. The repository description is "Rust support for seL4 userspace". The repository was last updated 4 days ago with commit `69838c2` and has 1,544 commits. The repository is owned by `nspin` and is part of the `crates/sel4` project. The repository has a README file.

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
- General-purpose kernel loader
- CapDL-based system initializer
- Rustc target specs
- Examples
- Tests



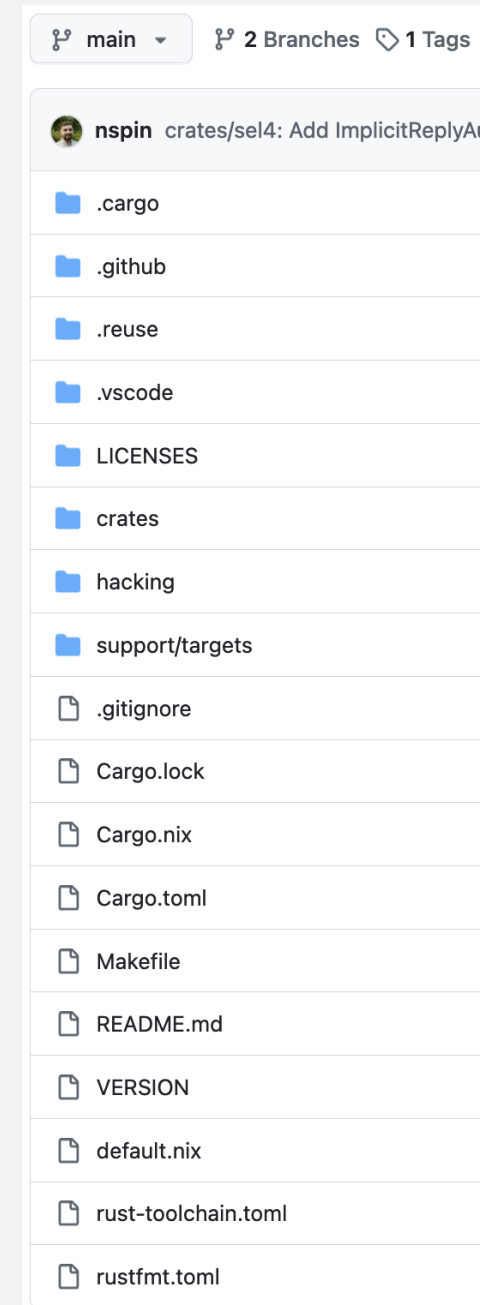
Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
- General-purpose kernel loader
- CapDL-based system initializer
- Rustc target specs
- Examples
- Tests

Last year's talk:

<https://sel4.systems/Foundation/Summit/2023/abstracts2023#a-rust-support>



Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
 - se14: Rust bindings for the seL4 API

Repository contents

<https://github.com/seL4/rust-seL4>

- Rust libraries
 - seL4: Rust bindings for the seL4 API

Implemented in two layers, both pure Rust

seL4-sys Generated from libseL4 headers (including .bf and .xml)

seL4 The “real” Rust libseL4: wraps seL4-sys, leveraging the Rust type system and idioms to present a cleaner and more ergonomic API

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
 - se14: Rust bindings for the seL4 API

Implemented in two layers, both pure Rust

Minimal dependencies

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
 - seL4: Rust bindings for the seL4 API

Implemented in two layers, both pure Rust

Minimal dependencies

Easy to build: just supply libsel4 headers via `$SEL4_INCLUDE_DIRS`

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
 - seL4: Rust bindings for the seL4 API

Implemented in two layers, both pure Rust

Minimal dependencies

Easy to build: just supply libsel4 headers via `$SEL4_INCLUDE_DIRS`

Flexible: thread-local storage optional

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
 - seL4: Rust bindings for the seL4 API

Implemented in two layers, both pure Rust

Minimal dependencies

Easy to build: just supply libsel4 headers via `$SEL4_INCLUDE_DIRS`

Flexible: thread-local storage optional

Plays nicely with C libsel4

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
 - se14: Rust bindings for the seL4 API

Ergonomic: LoC for a minimal cross-platform root task with no dependencies beyond the se14 crate that...

...spawns a thread: <300 LoC

...spawns a task: <400 LoC

...maps and drives a serial device: <300 LoC

<https://github.com/seL4/rust-sel4/tree/main/crates/examples/root-task>

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
 - se14: Rust bindings for the seL4 API

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
 - `sel4`: Rust bindings for the seL4 API
 - Modular runtime building blocks

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
 - `seL4`: Rust bindings for the seL4 API
 - Modular runtime building blocks

Language runtime:

- Entrypoint: `_start` (*required*)
- Stack (*required*)
- Thread local storage (*optional*)
- Heap allocator: `#[global_allocator]` (*optional*)
- Panic handler: `#[panic_handler]` (*required*)
- Exception handling (*optional*)

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
 - `sel4`: Rust bindings for the seL4 API
 - Modular runtime building blocks
 - `sel4-dlmalloc`
 - `sel4-elf-header`
 - `sel4-initialize-tls`
 - `sel4-stack`
 - `sel4-panicking`
 - `sel4-backtrace`
 - `sel4-reset`
 - ...and more

Language runtime:

- Entrypoint: `_start` (*required*)
- Stack (*required*)
- Thread local storage (*optional*)
- Heap allocator: `#[global_allocator]` (*optional*)
- Panic handler: `#[panic_handler]` (*required*)
- Exception handling (*optional*)

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
 - `sel4`: Rust bindings for the seL4 API
 - Modular runtime building blocks
 - `sel4-dlmalloc`
 - `sel4-elf-header`
 - `sel4-initialize-tls`
 - `sel4-stack`
 - `sel4-panicking`
 - `sel4-backtrace`
 - `sel4-reset`
 - ...and more

Language runtime:

- Entrypoint: `_start` (*required*)
- Stack (*required*)
- Thread local storage (*optional*)
- Heap allocator: `#[global_allocator]` (*optional*)
- Panic handler: `#[panic_handler]` (*required*)
- Exception handling (*optional*)

Recent highlight: Support for resettable runtimes

At build time:

Pack all initialization data for writeable segments into read-only segments
Result has no writeable segments with `filesz > 0`

At runtime:

```
_reset:  
    // Using temporary stack, reset writeable segments  
    // from data in read-only segments  
_start:  
    // ...
```

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
 - `sel4`: Rust bindings for the seL4 API
 - Modular runtime building blocks
 - `sel4-dlmalloc`
 - `sel4-elf-header`
 - `sel4-initialize-tls`
 - `sel4-stack`
 - `sel4-panicking`
 - `sel4-backtrace`
 - `sel4-reset`
 - ...and more

Language runtime:

- Entrypoint: `_start` (*required*)
- Stack (*required*)
- Thread local storage (*optional*)
- Heap allocator: `#[global_allocator]` (*optional*)
- Panic handler: `#[panic_handler]` (*required*)
- Exception handling (*optional*)

Repository contents

<https://github.com/seL4/rust-seL4>

- Rust libraries
 - `seL4`: Rust bindings for the seL4 API
 - Modular runtime building blocks
 - `seL4-root-task`: Runtime for root tasks
 - `seL4-microkit`: Runtime for root tasks

Crate: sel4-root-task

Language runtime #1

Configurable (\pm TLS, \pm heap, \pm unwinding)

Glues together:

- sel4
- sel4-initialize-tls
- sel4-panicking
- sel4-dlmalloc
- ...and more

Crate: sel4-root-task

Language runtime #1

```
#![no_std]
#![no_main]
#![feature(never_type)]

use sel4_root_task::root_task;

#[root_task]
fn main(_bootinfo: &sel4::BootInfo) -> ! {
    sel4::debug_println!("Hello, World!");

    sel4::BootInfo::init_thread_tcb().tcb_suspend().unwrap();

    unreachable!()
}
```

Crate: sel4-microkit

Language runtime #2

Configurable (\pm TLS, \pm heap, \pm unwinding)

Crate: sel4-microkit

Language runtime #2

```
#![no_std]
#![no_main]

use sel4_microkit::{debug_println, protection_domain, Channel, Handler, MessageInfo};

#[protection_domain(stack_size = 4096 * 4, heap_size = 4096 * 12)]
fn init() -> HandlerImpl {
    debug_println!("Hello, World!");
    HandlerImpl {}
}

struct HandlerImpl {}

impl Handler for HandlerImpl {
    fn notified(&mut self, channel: Channel) -> Result<(), Self::Error> {
        todo!()
    }

    fn protected(
        &mut self,
        channel: Channel,
        msg_info: MessageInfo,
    ) -> Result<MessageInfo, Self::Error> {
        todo!()
    }
}
```


Repository contents

<https://github.com/seL4/rust-seL4>

- Rust libraries
 - `seL4`: Rust bindings for the seL4 API
 - Modular runtime building blocks
 - `seL4-root-task`: Runtime for root tasks
 - `seL4-microkit`: Runtime for root tasks

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
 - `sel4`: Rust bindings for the seL4 API
 - Modular runtime building blocks
 - `sel4-root-task`: Runtime for root tasks
 - `sel4-microkit`: Runtime for root tasks
 - Higher-level crates

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
 - `sel4`: Rust bindings for the seL4 API
 - Modular runtime building blocks
 - `sel4-root-task`: Runtime for root tasks
 - `sel4-microkit`: Runtime for root tasks
 - Higher-level crates
 - `sel4-sync`
 - `sel4-logging`
 - `sel4-externally-shared`
 - `sel4-bounce-buffer-allocator`
 - `sel4-shared-ring-buffer`
 - `sel4-driver-interfaces`
 - `sel4-microkit-driver-adapters`
 - `sel4-*-driver`
 - `sel4-async-*`

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
 - `sel4`: Rust bindings for the seL4 API
 - Modular runtime building blocks
 - `sel4-root-task`: Runtime for root tasks
 - `sel4-microkit`: Runtime for root tasks
 - Higher-level crates
 - `sel4-sync`
 - `sel4-logging`
 - `sel4-externally-shared`
 - `sel4-bounce-buffer-allocator`
 - `sel4-shared-ring-buffer`
 - `sel4-driver-interfaces`
 - `sel4-microkit-driver-adapters`
 - `sel4-*-driver`
 - `sel4-async-*`

(recent highlight) Library layers to support defensive clients and servers

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
 - `sel4`: Rust bindings for the seL4 API
 - Modular runtime building blocks
 - `sel4-root-task`: Runtime for root tasks
 - `sel4-microkit`: Runtime for root tasks
 - Higher-level crates
 - `sel4-sync`
 - `sel4-logging`
 - `sel4-externally-shared`
 - `sel4-bounce-buffer-allocator`
 - `sel4-shared-ring-buffer`
 - `sel4-driver-interfaces`
 - `sel4-microkit-driver-adapters`
 - `sel4-*-driver`
 - `sel4-async-*`

(recent highlight) Builds on work presented by Ben from Galois at last year's summit

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
 - `sel4`: Rust bindings for the seL4 API
 - Modular runtime building blocks
 - `sel4-root-task`: Runtime for root tasks
 - `sel4-microkit`: Runtime for root tasks
 - Higher-level crates
 - `sel4-sync`
 - `sel4-logging`
 - `sel4-externally-shared`
 - `sel4-bounce-buffer-allocator`
 - `sel4-shared-ring-buffer`
 - `sel4-driver-interfaces`
 - `sel4-microkit-driver-adapters`
 - `sel4-*-driver`
 - `sel4-async-*`

Repository contents

<https://github.com/seL4/rust-seL4>

- Rust libraries
 - `seL4`: Rust bindings for the seL4 API
 - Modular runtime building blocks
 - `seL4-root-task`: Runtime for root tasks
 - `seL4-microkit`: Runtime for root tasks
- Higher-level crates
 - `seL4-sync`
 - `seL4-logging`
 - `seL4-externally-shared`
 - `seL4-bounce-buffer-allocator`
 - `seL4-shared-ring-buffer`
 - `seL4-driver-interfaces`
 - `seL4-microkit-driver-adapters`
 - `seL4-*-driver`
 - `seL4-async-*`

Last year:

Example: HTTP server using seL4 [Microkit](#)

<https://github.com/seL4/rust-microkit-http-server-demo>

The screenshot shows two windows side-by-side. The left window is a terminal displaying boot logs for the seL4 microkernel. The right window is a web browser showing the seL4 website.

```
LDR|INFO: jumping to kernel
Bootstrapping kernel
Warning: Could not infer GIC interrupt target ID, assuming 0.
available phys memory regions: 1
[40000000..80000000]
reserved virt address space regions: 3
[ffffff8040000000..ffffff8040243000]
[ffffff8040243000..ffffff8041575000]
[ffffff8041575000..ffffff804157c000]
Booting all finished, dropped to user space
MON|INFO: Microkit Bootstrap
MON|INFO: bootinfo untyped list matches expected list
MON|INFO: Number of bootstrap invocations: 0x0000000e
MON|INFO: Number of system invocations: 0x00001373
MON|INFO: completed bootstrap invocations
MON|INFO: completed system invocations
INFO [seL4_async_network] DHCP config lost
INFO [seL4_async_network] DHCP config acquired
INFO [seL4_async_network] IP address: 10.0.2.15/24
INFO [seL4_async_network] Default gateway: 10.0.2.2
INFO [seL4_async_network] DNS server 0: 10.0.2.3
```

The browser window shows the seL4 website with the following content:

Home | seL4

What is seL4? | seL4 Foundation | Stay in Touch | Contribute | Use | Learn

More Info | News

The seL4[®] Microkernel

Security is no excuse for bad performance

The benchmark for performance.
The world's most highly assured OS kernel.
Open source & community-supported under the seL4 Foundation.

News

- 23 Jun 2023: The seL4 summit Program is available
- 19 Jun 2023: Keynotes for seL4 summit 2023 announced
- 29 May 2023: Galois now part of the seL4 Foundation more...

47 © 2023 Colias Group, LLC

Colias
Group

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
 - `sel4`: Rust bindings for the seL4 API
 - Modular runtime building blocks
 - `sel4-root-task`: Runtime for root tasks
 - `sel4-microkit`: Runtime for root tasks
 - Higher-level crates
 - `sel4-sync`
 - `sel4-logging`
 - `sel4-externally-shared`
 - `sel4-bounce-buffer-allocator`
 - `sel4-shared-ring-buffer`
 - `sel4-driver-interfaces`
 - `sel4-microkit-driver-adapters`
 - `sel4-*-driver`
 - `sel4-async-*`

(next) Integration with LionsOS

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
 - `sel4`: Rust bindings for the seL4 API
 - Modular runtime building blocks
 - `sel4-root-task`: Runtime for root tasks
 - `sel4-microkit`: Runtime for root tasks
 - Higher-level crates

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
 - `sel4`: Rust bindings for the seL4 API
 - Modular runtime building blocks
 - `sel4-root-task`: Runtime for root tasks
 - `sel4-microkit`: Runtime for root tasks
 - Higher-level crates
 - ...and more

Repository contents

<https://github.com/seL4/rust-seL4>

- Rust libraries
 - `seL4`: Rust bindings for the seL4 API
 - Modular runtime building blocks
 - `seL4-root-task`: Runtime for root tasks
 - `seL4-microkit`: Runtime for root tasks
 - Higher-level crates
 - ...and more

(recent highlight) Test harness for running first- or third-party unit `#[test]`s in a seL4 root task

Repository contents

<https://github.com/seL4/rust-sel4>

- Rust libraries
 - `sel4`: Rust bindings for the seL4 API
 - Modular runtime building blocks
 - `sel4-root-task`: Runtime for root tasks
 - `sel4-microkit`: Runtime for root tasks
 - Higher-level crates
 - ...and more

Assurance



Ferrocene

<https://ferrocene.dev>

A Rust compiler toolchain called Ferrocene is ISO 26262 and IEC 61508 qualified¹



It's official: Ferrocene is ISO 26262 and IEC 61508 qualified!

You can even find the [certificate in TÜV SÜD's certificate database](#).

This means we achieved qualification for the open source Ferrocene toolchain. Ferrocene 23.06.0, based on Rust 1.68, is now fully usable in safety critical environments.

¹<https://ferrous-systems.com/blog/officially-qualified-ferrocene/>

Kani

<https://model-checking.github.io/kani>

Symbolic execution

```
use my_crate::{function_under_test, meets_specification, precondition};

#[kani::proof]
fn check_my_property() {
    // Create a nondeterministic input
    let input = kani::any();

    // Constrain it according to the function's precondition
    kani::assume(precondition(input));

    // Call the function under verification
    let output = function_under_test(input);

    // Check that it meets the specification
    assert!(meets_specification(input, output));
}
```

Kani

<https://model-checking.github.io/kani>

Symbolic execution

We use it to verify capability invocation message marshaling in `se14-sys`

Verus

<https://github.com/verus-lang/verus>

Annotate Rust code with pre/post-conditions, etc.

More suitable for downstream crates

```
use vstd::prelude::*;

verus! {
    pub fn max(a: u64, b: u64) -> (ret: u64)
        ensures
            ret == a || ret == b,
            ret >= a && ret >= b,
        {
            if a >= b {
                a
            } else {
                b
            }
        }
}
```

Dafny

<https://dafny.org>

Separate verification-aware programming language

```
method Max(a: int, b:int) returns (c: int)
  ensures a < b ==> c == b
  ensures b <= a ==> c == a
{
  if (a < b) {
    return b;
  } else {
    return a;
  }
}
```

Dafny

<https://dafny.org>

Separate verification-aware programming language

Compiles to Rust, so we can easily run it in seL4 userspace!

```
method Max(a: int, b:int) returns (c: int)
  ensures a < b ==> c == b
  ensures b <= a ==> c == a
{
  if (a < b) {
    return b;
  } else {
    return a;
  }
}
```


Low-level Rust supporting other languages

Leverage type-safety and modular runtime code from this project for:

- Dafny
- OCaml/MirageOS (<https://github.com/coliasgroup/seL4-MirageOS-PoC>)
- C (via newlib or musl)
- ...Rust itself?

The Rust Standard Library

Layer	Provides	Requires	
libstd	<code>std::fs</code> <code>std::net</code> <code>std::thread</code> <code>std::process</code> Language runtime	OS services	<i>depends on</i>
liballoc	<code>alloc::vec</code> <code>alloc::collections</code> <code>alloc::string</code>	heap allocator	
libcore	<code>core::mem</code> <code>core::num</code> <code>core::iter</code> <code>core::ffi</code>	nothing! (except panic handler)	



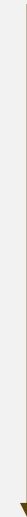
The Rust Standard Library

	Layer	Provides	Requires	
<code>#![no_std]</code>	libstd	<code>std::fs</code> <code>std::net</code> <code>std::thread</code> <code>std::process</code> Language runtime	OS services	<i>depends on</i>
	liballoc	<code>alloc::vec</code> <code>alloc::collections</code> <code>alloc::string</code>	heap allocator	
	libcore	<code>core::mem</code> <code>core::num</code> <code>core::iter</code> <code>core::ffi</code>	nothing! (except panic handler)	



The Rust Standard Library

	Layer	Provides	Requires	
via musl via WASI?	libstd	<code>std::fs</code> <code>std::net</code> <code>std::thread</code> <code>std::process</code> Language runtime	OS services	<i>depends on</i>
	liballoc	<code>alloc::vec</code> <code>alloc::collections</code> <code>alloc::string</code>	heap allocator	
	libcore	<code>core::mem</code> <code>core::num</code> <code>core::iter</code> <code>core::ffi</code>	nothing! (except panic handler)	



Next: Stabilization

Collect more feedback

Minimize interfaces

Eventually host on crates.io

Discussion

<https://github.com/seL4/rust-sel4>

<mailto:nick@nickspinale.com>

