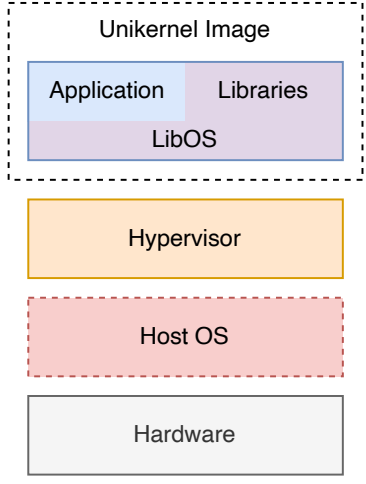
Towards a Safe and Sound Operating System

Martin Kröning, Jonathan Klimt, Stefan Lankes

E.ON Energy Research Center | RWTH AACHEN UNIVERSITY

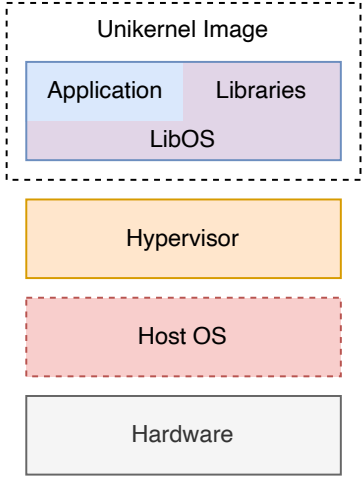
Unikernels

Unikernels


Unikernel Image
Application
Libraries
LibOS
Hypervisor
Host OS
Hardware

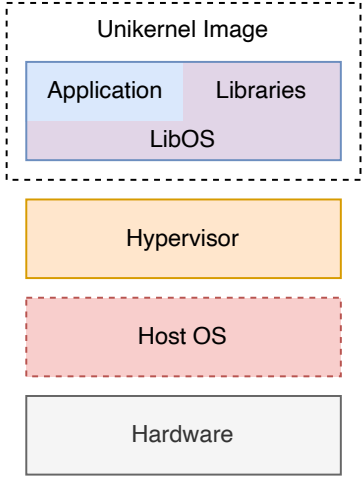
Unikernels

- Specialized for use cause
  - Tiny images



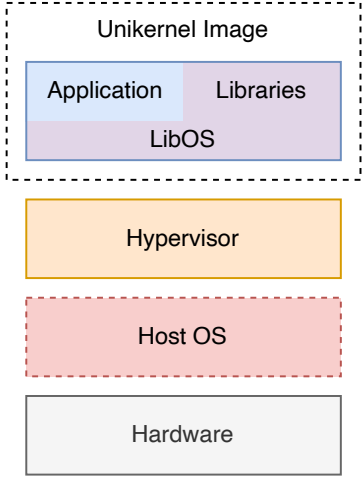
Unikernels

- Specialized for use cause
  - Tiny images
- *One* process per image
  - No isolation necessary



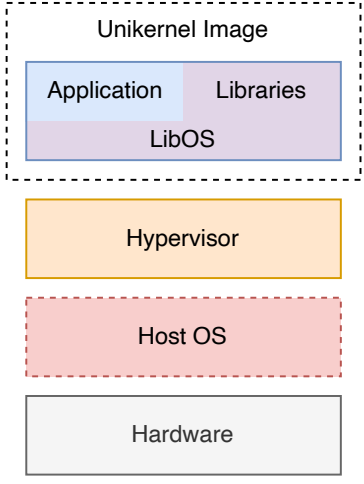
Unikernels

- Specialized for use cause
  - Tiny images
- *One* process per image
  - No isolation necessary
- Single address space operating system
  - No address space context switch



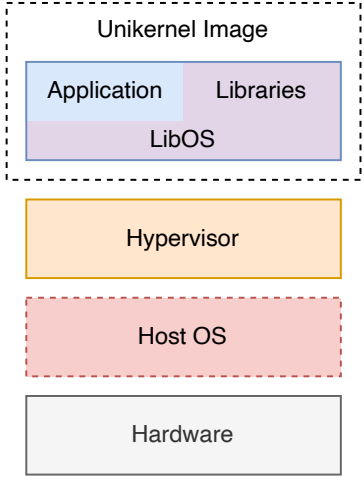
Unikernels

- Specialized for use cause
  - Tiny images
- *One* process per image
  - No isolation necessary
- Single address space operating system
  - No address space context switch
- Single privilege level
  - No privilege context switch



Unikernels

- Specialized for use cause
  - Tiny images
- *One* process per image
  - No isolation necessary
- Single address space operating system
  - No address space context switch
- Single privilege level
  - No privilege context switch
- System calls are just function calls



The Hermit Operating System

The Hermit Operating System

- Unikernel Project

The Hermit Operating System

- Unikernel Project
- Written in Rust

The Hermit Operating System

- Unikernel Project
- Written in Rust
- Official tier 3 Rust target for Rust application

The Hermit Operating System

- Unikernel Project
- Written in Rust
- Official tier 3 Rust target for Rust application
- GCC fork for C applications

The Hermit Operating System

- Unikernel Project
- Written in Rust
- Official tier 3 Rust target for Rust application
- GCC fork for C applications

Features

- Network (virtio, RTL8139)
- Multi-core support
- Easily configurable

The Hermit Operating System

- Unikernel Project
- Written in Rust
- Official tier 3 Rust target for Rust application
- GCC fork for C applications

Features

- Network (virtio, RTL8139)
- Multi-core support
- Easily configurable

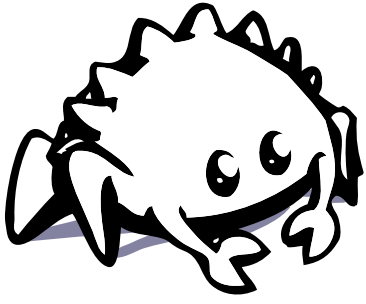
Architectures

- x86-64 (primary)
- AArch64 (emerging)
- 64-bit RISC-V (upcoming)

Rust's raison d'être

Rust's raison d'être

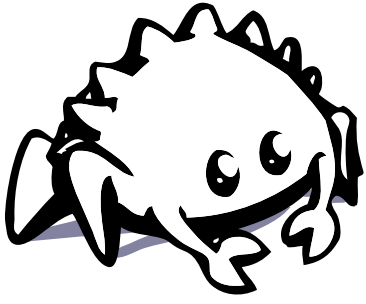
Memory Safety



Rust's raison d'être

Memory Safety

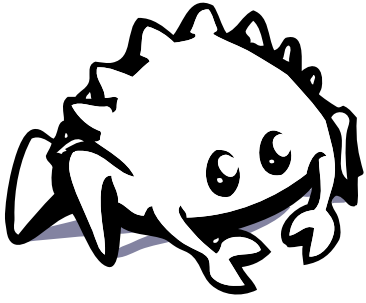
- No more out-of-bounds accesses



Rust's raison d'être

Memory Safety

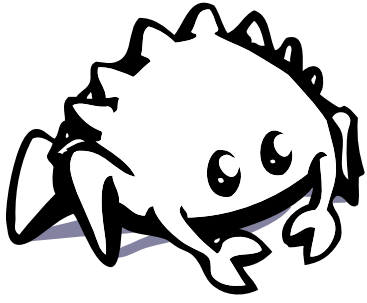
- No more out-of-bounds accesses
- No more use-after-free errors



Rust's raison d'être

Memory Safety

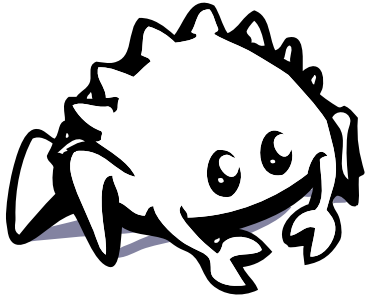
- No more out-of-bounds accesses
- No more use-after-free errors
- No more data races



Rust's raison d'être

Memory Safety

- No more out-of-bounds accesses
- No more use-after-free errors
- No more data races
- ...



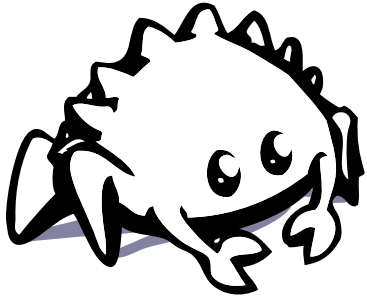
Rust's raison d'être

Memory Safety

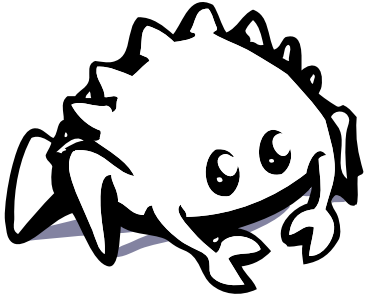
- No more out-of-bounds accesses
- No more use-after-free errors
- No more data races
- ...

Could prevent 65 % of security vulnerabilities

(https://alexgaynor.net/2020/may/27/science-on-memory-unsafety-and-security/)

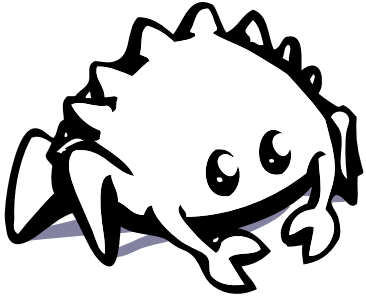


How Rust Guarantees Memory Safety



How Rust Guarantees Memory Safety

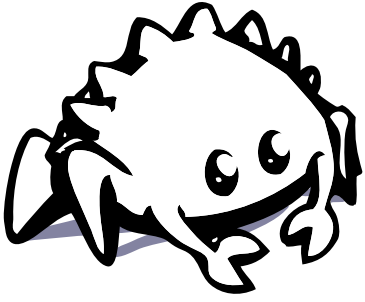
Safe Rust



How Rust Guarantees Memory Safety

Safe Rust

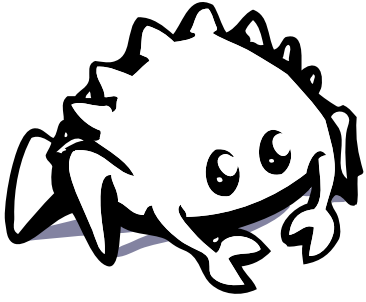
- Specifically constructed to be statically checked



How Rust Guarantees Memory Safety

Safe Rust

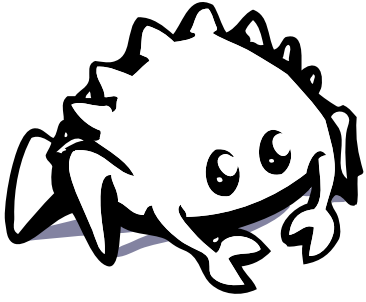
- Specifically constructed to be statically checked
- References are always valid



How Rust Guarantees Memory Safety

Safe Rust

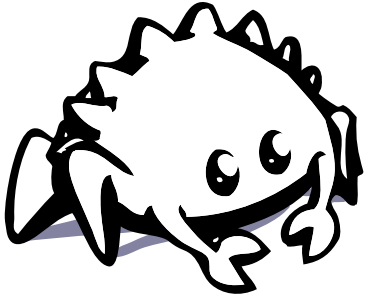
- Specifically constructed to be statically checked
- References are always valid
- Memory is always initialized



How Rust Guarantees Memory Safety

Safe Rust

- Specifically constructed to be statically checked
- References are always valid
- Memory is always initialized
- Aliasing `XOR` Mutation



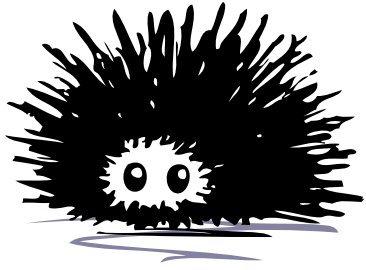
How Rust Guarantees Memory Safety

Safe Rust

- Specifically constructed to be statically checked
- References are always valid
- Memory is always initialized
- Aliasing `XOR` Mutation

Unsafe Rust

Removes some of safe Rust's restrictions:



How Rust Guarantees Memory Safety

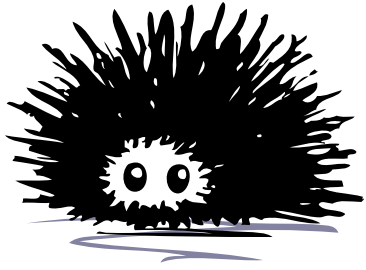
Safe Rust

- Specifically constructed to be statically checked
- References are always valid
- Memory is always initialized
- Aliasing `XOR` Mutation

Unsafe Rust

Removes some of safe Rust's restrictions:

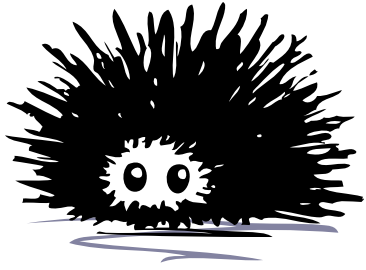
- Raw pointers



How Rust Guarantees Memory Safety

Safe Rust

- Specifically constructed to be statically checked
- References are always valid
- Memory is always initialized
- Aliasing `XOR` Mutation



Unsafe Rust

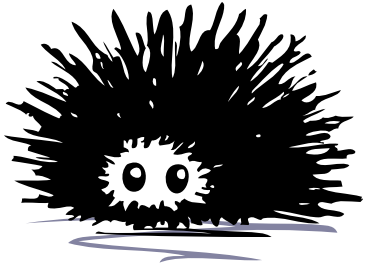
Removes some of safe Rust's restrictions:

- Raw pointers
- Uninitialized memory

How Rust Guarantees Memory Safety

Safe Rust

- Specifically constructed to be statically checked
- References are always valid
- Memory is always initialized
- Aliasing `XOR` Mutation



Unsafe Rust

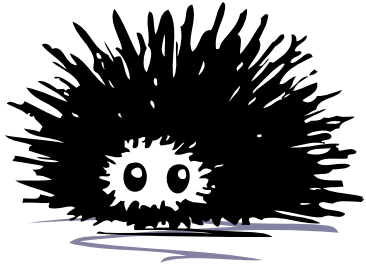
Removes some of safe Rust's restrictions:

- Raw pointers
- Uninitialized memory
- Inline assembly

How Rust Guarantees Memory Safety

Safe Rust

- Specifically constructed to be statically checked
- References are always valid
- Memory is always initialized
- Aliasing `XOR` Mutation

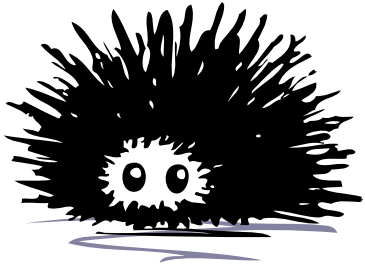


Unsafe Rust

Removes some of safe Rust's restrictions:

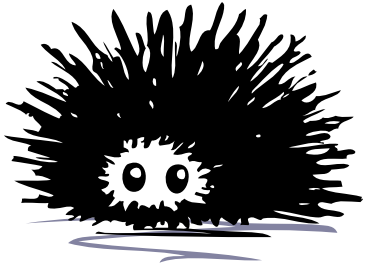
- Raw pointers
- Uninitialized memory
- Inline assembly
- Unsafe functions

How Rust Guarantees Memory Safety (cont.)



How Rust Guarantees Memory Safety (cont.)

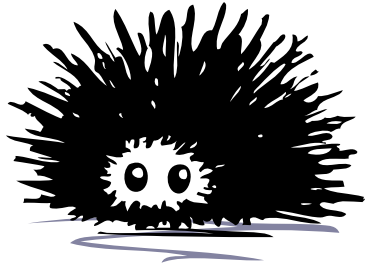
Undefined Behavior



How Rust Guarantees Memory Safety (cont.)

Undefined Behavior

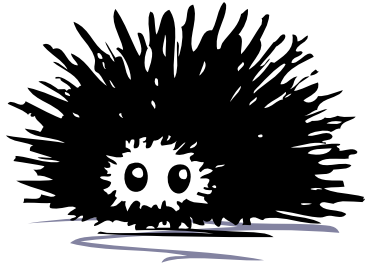
- Program is optimized assuming the absence of UB



How Rust Guarantees Memory Safety (cont.)

Undefined Behavior

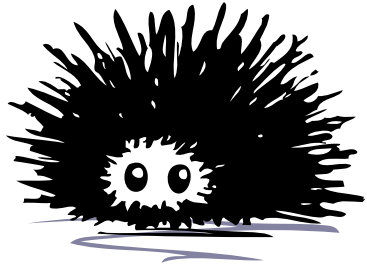
- Program is optimized assuming the absence of UB
- Safe Rust is fine



How Rust Guarantees Memory Safety (cont.)

Undefined Behavior

- Program is optimized assuming the absence of UB
- Safe Rust is fine
- Unsafe Rust may cause issues

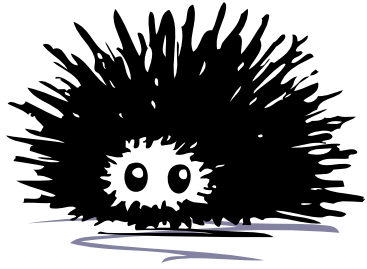


How Rust Guarantees Memory Safety (cont.)

Undefined Behavior

- Program is optimized assuming the absence of UB
- Safe Rust is fine
- Unsafe Rust may cause issues

Soundness



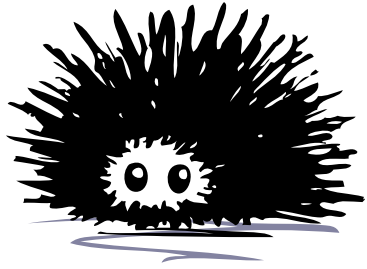
How Rust Guarantees Memory Safety (cont.)

Undefined Behavior

- Program is optimized assuming the absence of UB
- Safe Rust is fine
- Unsafe Rust may cause issues

Soundness

- Unsafe Rust may never break safe Rust's invariants



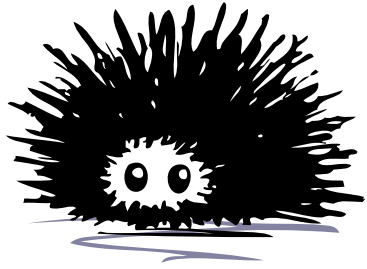
How Rust Guarantees Memory Safety (cont.)

Undefined Behavior

- Program is optimized assuming the absence of UB
- Safe Rust is fine
- Unsafe Rust may cause issues

Soundness

- Unsafe Rust may never break safe Rust's invariants
- A *safe* Rust function must be *sound*



What's the Problem?

What's the Problem?

- Hermit has been ported from C to Rust

What's the Problem?

- Hermit has been ported from C to Rust
- Hermit was sticking to C-isms

What's the Problem?

- Hermit has been ported from C to Rust
- Hermit was sticking to C-isms
- Unsynchronized statics

What's the Problem?

- Hermit has been ported from C to Rust
- Hermit was sticking to C-isms
- Unsynchronized statics
- Aliasing `AND` Mutation (data races)

What's the Approach?

What's the Approach?

- Audit unsafe code from the bottom up

What's the Approach?

- Audit unsafe code from the bottom up
- Rework fundamentally unsound code

What's the Approach?

- Audit unsafe code from the bottom up
- Rework fundamentally unsound code
- Document safety invariants

What's the Approach?

- Audit unsafe code from the bottom up
- Rework fundamentally unsound code
- Document safety invariants
- Publish modularized solutions

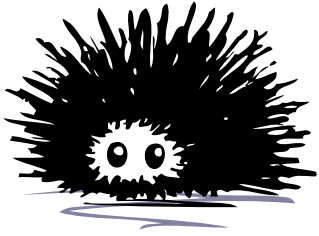
What's the Approach?

- Audit unsafe code from the bottom up
- Rework fundamentally unsound code
- Document safety invariants
- Publish modularized solutions
- Try to find more UB with tools

Unsynchronized Statics

```
static mut IDT: Idt = Idt::new();

fn init() {
    let idt = unsafe { &mut IDT };
    // Populate IDT entries
    // Load IDT
}
```



take-static

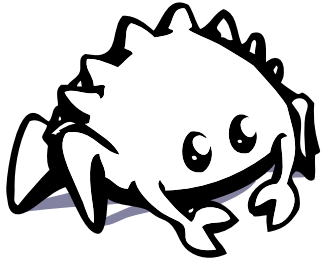
```rust
pub struct TakeStatic<T> {
    taken: AtomicBool,
    data: UnsafeCell<T>,
}

impl<T> TakeStatic<T> {
    pub fn take(&self) -> Option<&mut T> {
        if self
            .taken
            .compare_exchange(false, true, Ordering::Relaxed, Ordering::Relaxed)
            .is_ok()
        {
            Some(unsafe { &mut *self.data.get() })
        } else {
            None
        }
    }
}
```

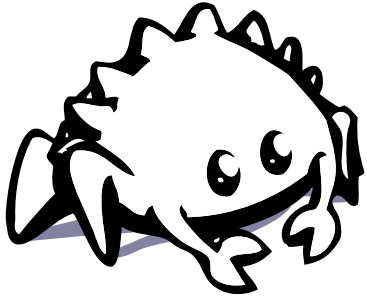
Synchronized Statics

```
take_static! {
    static IDT: Idt = Idt::new();
}

fn init() {
    let idt: &mut Idt = IDT.take().unwrap();
    assert!(IDT.take() == None);
    // Populate IDT entries
    // Load IDT
}
```



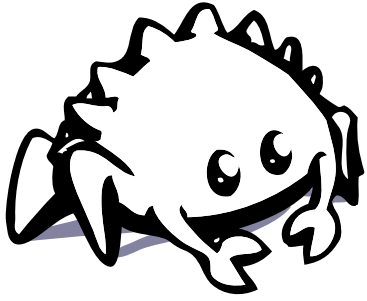
What's the Outcome?



What's the Outcome?

hermit-sync (github.com/hermit-os/hermit-sync)

Sound Bare-metal Rust-style synchronization primitives

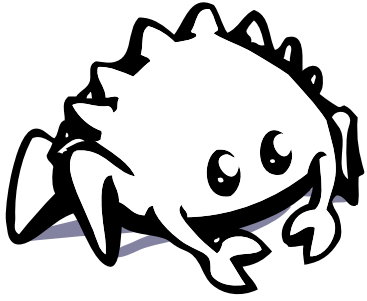


What's the Outcome?

hermit-sync (github.com/hermit-os/hermit-sync)

Sound Bare-metal Rust-style synchronization primitives

- `SpinMutex`

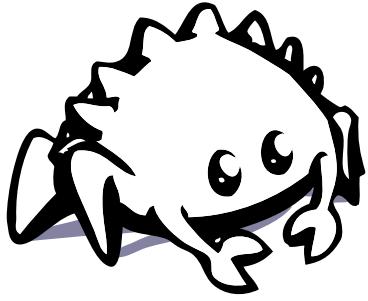


What's the Outcome?

hermit-sync (github.com/hermit-os/hermit-sync)

Sound Bare-metal Rust-style synchronization primitives

- `SpinMutex`
- `OnceCell`

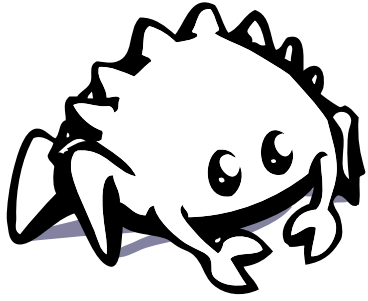


What's the Outcome?

hermit-sync (github.com/hermit-os/hermit-sync)

Sound Bare-metal Rust-style synchronization primitives

- `SpinMutex`
- `OnceCell`
- `Lazy`

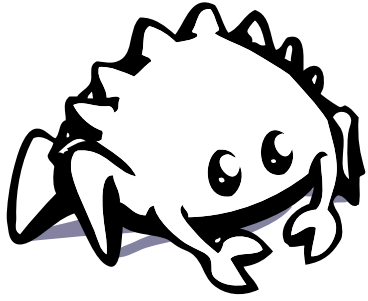


What's the Outcome?

hermit-sync (github.com/hermit-os/hermit-sync)

Sound Bare-metal Rust-style synchronization primitives

- `SpinMutex`
- `OnceCell`
- `Lazy`
- `TakeStatic`

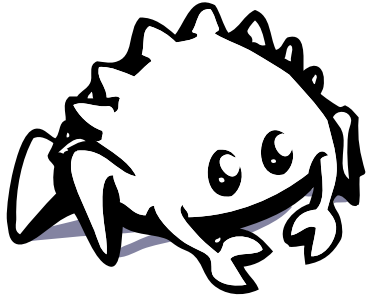


What's the Outcome?

hermit-sync (github.com/hermit-os/hermit-sync)

Sound Bare-metal Rust-style synchronization primitives

- `SpinMutex`
- `OnceCell`
- `Lazy`
- `TakeStatic`
- `InterruptMutex`

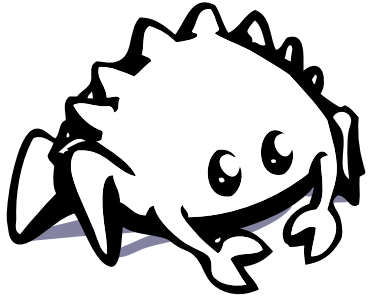


What's the Outcome?

hermit-sync (github.com/hermit-os/hermit-sync)

Sound Bare-metal Rust-style synchronization primitives

- `SpinMutex`
- `OnceCell`
- `Lazy`
- `TakeStatic`
- `InterruptMutex`
- `InterruptRefCell`

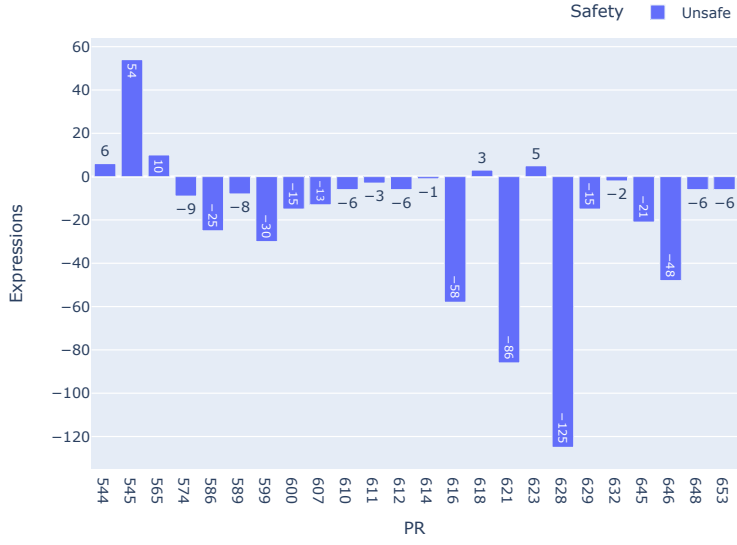


Results

- Created, used, and published the new *count-unsafe* tool.

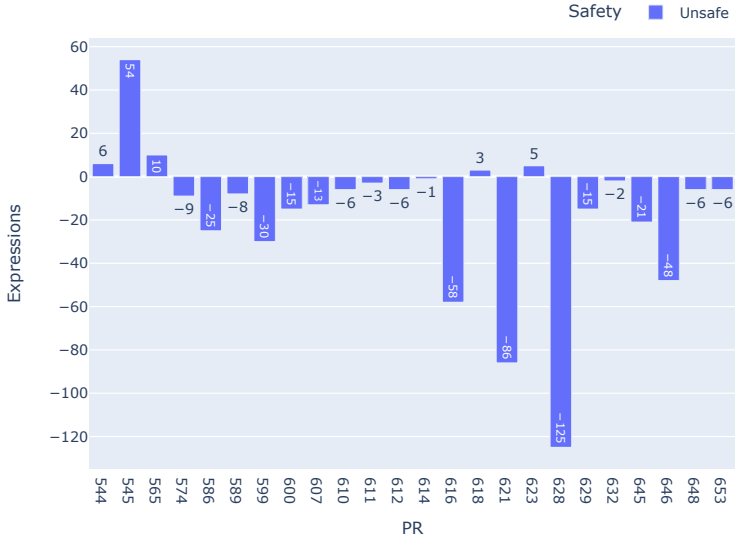
Results

- Created, used, and published the new *count-unsafe* tool.



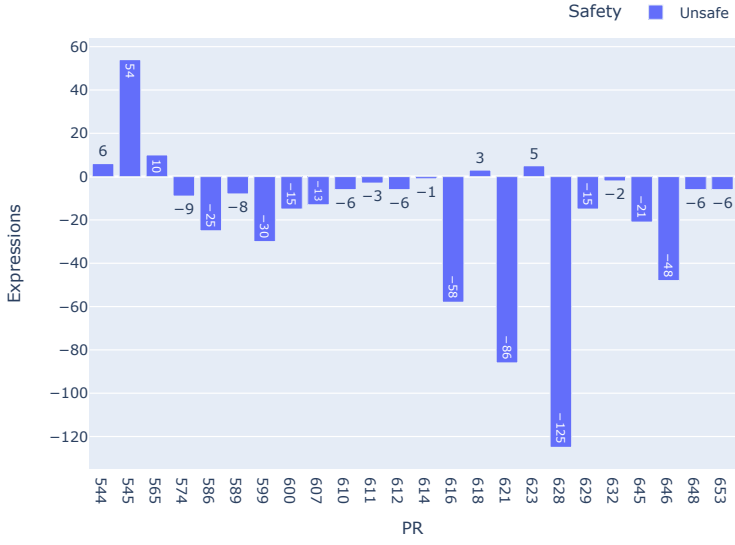
Results

- Created, used, and published the new *count-unsafe* tool.
- Removed more than 400 `unsafe` expressions.



Results

- Created, used, and published the new *count-unsafe* tool.
- Removed more than 400 `unsafe` expressions.
- Solved *real* memory safety issues.



What's next?

What's next?

- Rework Drivers, Filesystems, Network.

What's next?

- Rework Drivers, Filesystems, Network.
- Continue discussing libraries with the Rust OS-dev community.

What's next?

- Rework Drivers, Filesystems, Network.
- Continue discussing libraries with the Rust OS-dev community.
- Continue discussing fundamental soundness issues with Rust's operational semantics team (T-opsem).

What's next?

- Rework Drivers, Filesystems, Network.
- Continue discussing libraries with the Rust OS-dev community.
- Continue discussing fundamental soundness issues with Rust's operational semantics team (T-opsem).
- Research running Hermit on *Miri*.

Acknowledgements

This work was supported by the European Union's Horizon Europe research and innovation programme under Grant Agreement No. 101070118 (NEMO).

Thank you for your kind attention!

Martin Kröning, Jonathan Klimt, Stefan Lankes – martin.kroening@eonerc.rwth-aachen.de

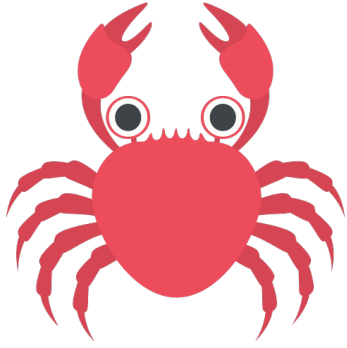
Institute for Automation of Complex Power Systems
E.ON Energy Research Center, RWTH Aachen University
Mathieustraße 10
52074 Aachen

http://hermit-os.org

History of Hermit

HermitCore (2015)

- Unikernel *and* Multikernel
- Runs side by side with Linux in HPC clusters
- HermitCore occupies some cores
  - Like a hermit crab ;)
- Lightweight HermitCore computes
- "Heavyweight" Linux drives hardware



History of Hermit (cont.)

RustyHermit (2018)

- Rust is modern and exciting
  - Rewrite it in Rust (RIIR)
- One true toolchain
  - Easy cross-compilation
- Vibrant community
  - Easy dependencies through central registry
- Recently rebranded as *Hermit*