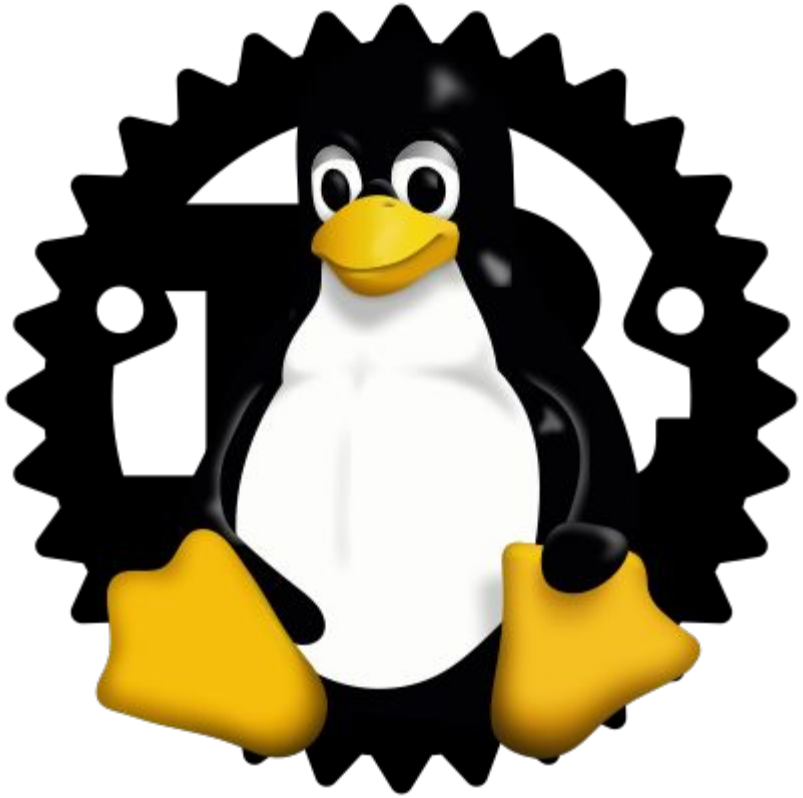


Rust for Linux

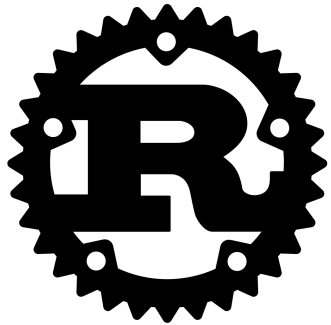
Miguel Ojeda

ojeda@kernel.org

Credits & Acknowledgments

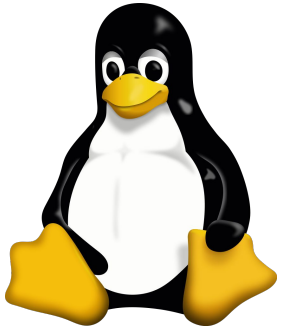
Rust

...for being a breath of fresh air



Kernel maintainers

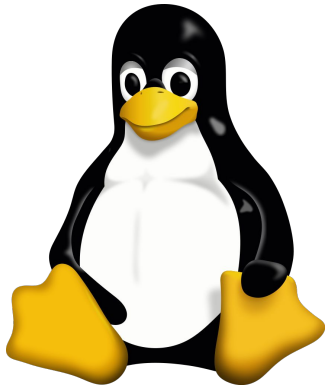
...for being open-minded



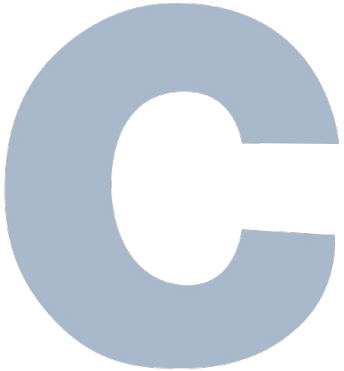
Everyone that has helped Rust for Linux

(see credits in the RFC & patch series)

History

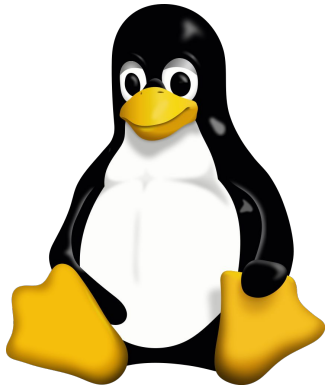


30 years of Linux



30 years of ISO C

Love story*



❤️*

C

30 years of Linux

30 years of ISO C

** Terms and Conditions Apply.*

An easy task?

An easy task?

“Do you see any language except C which is suitable for development of operating systems?”

An easy task?

“Do you see any language except C which is suitable for development of operating systems?”

“I like interacting with hardware from a software perspective. And I have yet to see a language that comes even close to C.”

— Linus Torvalds 2012

Why is C a good language for the kernel?

"You can use C to generate good code for hardware."

Fast

"When I read C, I know what the assembly language will look like."

Low-level

"The people that designed C ... designed it at a time when compilers had to be simple."

Simple

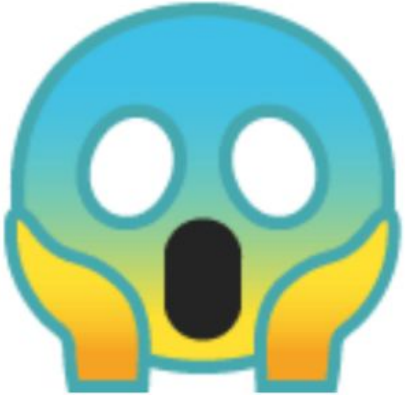
"If you think like a computer, writing C actually makes sense."

Fits the domain

But...

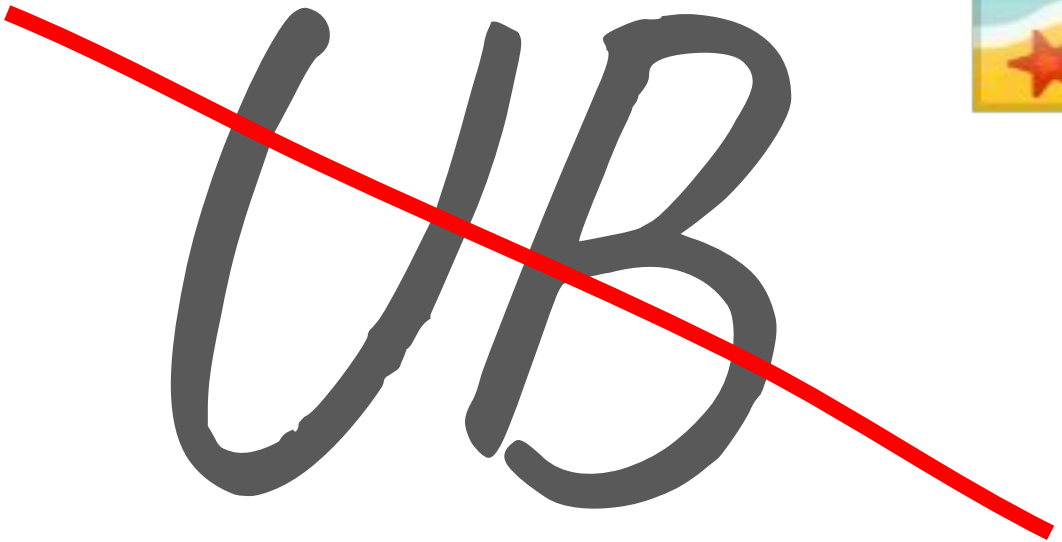
But...

UB



So, what does Rust offer?

So, what does Rust offer?



Safety

Safety in Rust

=

No undefined behavior

Safety

Safety in Rust

≠

Safety in “safety-critical”

as in functional safety (DO-178B/C, ISO 26262, EN 50128…)

Is avoiding UB that important?

Is avoiding UB that important?

~70%

of vulnerabilities in C/C++ projects come from UB

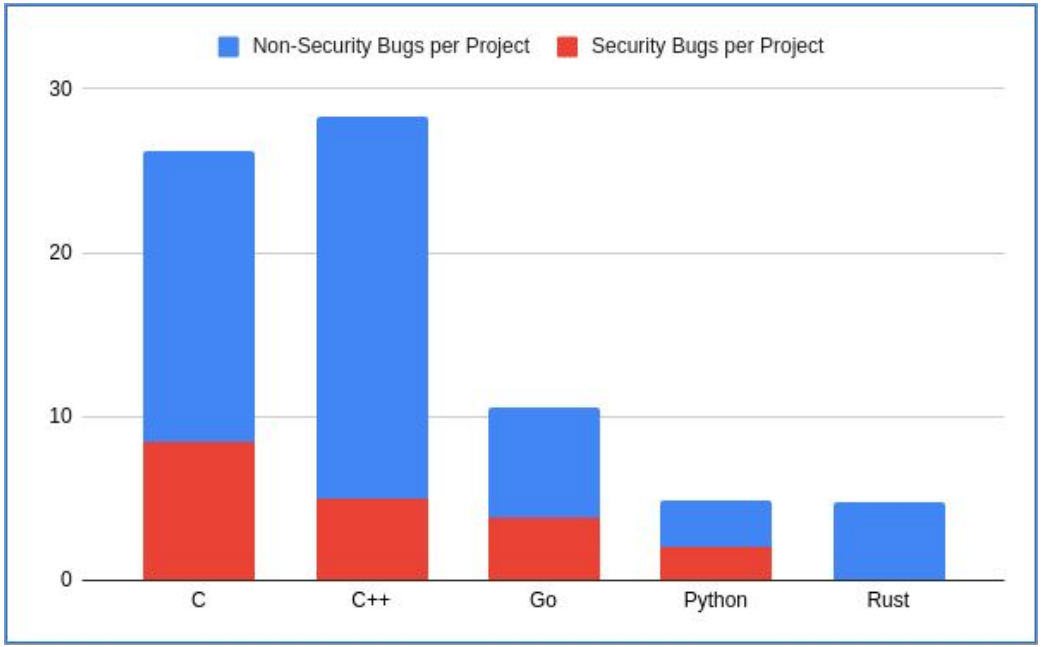
See more at https://www.memorysafety.org/docs/memory-safety/

Sure, UB is an issue and safe Rust does not have it…

Sure, UB is an issue and safe Rust does not have it…

...does Rust really help, though?

Does Rust help?



Derived using data from https://adalogics.com/blog/fuzzing-100-open-source-projects-with-oss-fuzz

What else does Rust offer?

Language

What else does Rust offer?

Shared & exclusive references

Modules & visibility | Generics | Lifetimes

Stricter type system | Language | Pattern matching

Safe/unsafe split | RAII | Sum types

Powerful hygienic and procedural macros

What else does Rust offer?

Freestanding standard library

What else does Rust offer?

Vocabulary types like `Result` and `Option`

Pinning

Formatting

Freestanding standard library

Checked, saturating & wrapping integer arithmetic primitives

Iterators

What else does Rust offer?

Tooling

What else does Rust offer?

Documentation generator

Unit & integration tests

Static analyzer

C ↔ Rust bindings generators

Linter

Tooling

Macro debugging

Formatter

IDE tooling

Great compiler error messages

UBSAN-like interpreter

What else does Rust offer?

Documentation generator

Unit & integration tests

Static analyzer

C ↔ Rust bindings generators

Linter

Tooling

Macro debugging

Formatter

IDE tooling

Great compiler error messages

UBSAN-like interpreter

plus the usual friends: `gdb`, `lldb`, `perf`, `valgrind`...

What is the catch?

What is the catch?

Cannot model everything ⇒ Unsafe code required

What is the catch?

Cannot model everything ⇒ Unsafe code required

More information to provide ⇒ More complex language

What is the catch?

Cannot model everything ⇒ Unsafe code required

More information to provide ⇒ More complex language

Extra runtime checks ⇒ Potentially expensive

What is the catch?

Cannot model everything	⇒	Unsafe code required
More information to provide	⇒	More complex language
Extra runtime checks	⇒	Potentially expensive
An extra language to learn	⇒	Logistics & maintenance burden

Why is C a good language for the kernel?

"You can use C to generate good code for hardware."

Fast

"When I read C, I know what the assembly language will look like."

Low-level

"The people that designed C ... designed it at a time when compilers had to be simple."

Simple

"If you think like a computer, writing C actually makes sense."

Fits the domain

Why is C (Rust) a good language for the kernel?

"You can use C to generate good code for hardware."

Fast — Yes

"When I read C, I know what the assembly language will look like."

Low-level — Sometimes

"The people that designed C ... designed it at a time when compilers had to be simple."

Simple — Not really

"If you think like a computer, writing C actually makes sense."

Fits the domain — ...

An easy task?

“Do you see any language except C which is suitable for development of operating systems?”

“I like interacting with hardware from a software perspective. And I have yet to see a language that comes even close to C.”

— Linus Torvalds 2012

An easy task? *maybe?*

"Do you see any language except C which is suitable for development of operating systems?"

"I like interacting with hardware from a software perspective. And I have yet to see a language that comes even close to C."

— Linus Torvalds 2012

Some examples where Rust helps

COMPILER EXPLORER

Add... More

Sponsors intel. PC-lint Share Policies Other

Rust source #2

Rust

```rust
pub fn main() {
    let a = Box::new(42);
    drop(a);
    drop(a);
}
```

rustc 1.55.0 (Rust, Editor #2, Compiler #1)

rustc 1.55.0 --edition=2018

```
<Compilation failed>

# For more information see the output
# To open the output window, click or
```

Output (0/13) rustc 1.55.0 - cached

Executor rustc 1.55.0 (Rust, Editor #2)

Wrap lines Libraries Compilation Arguments Stdin Compiler output

rustc 1.55.0 --edition=2018 -O -Cpanic=abort

```
Could not execute the program
Compiler returned: 1
Compiler stderr
error[E0382]: use of moved value: `a`
 --> <source>:4:10
  |
2 |     let a = Box::new(42);
  |         - move occurs because `a` has type `Box<i32>`, which does not implement the `Copy` trait
3 |     drop(a);
  |          - value moved here
4 |     drop(a);
  |          ^ value used here after move

error: aborting due to previous error

For more information about this error, try `rustc --explain E0382`.
```

rustc 1.55.0 - 1050ms

C source #1

Save/Load Add new... Vim C

```c
#include <stdlib.h>

int main(void)
{
    int * const a = malloc(sizeof(int));
    if (a == NULL)
        abort();
    *a = 42;
    free(a);
    free(a);
}
```

x86-64 gcc (trunk) (C, Editor #1, Compiler #2)

x86-64 gcc (trunk) -std=gnu99 -Wall -Wextra -Wpedantic -O2

Output... Filter... Libraries Add new... Add tool...

```
main:
        push    rbp
        mov     edi, 4
        call    malloc
        test    rax, rax
        je      .L3
        mov     rbp, rax
        mov     rdi, rax
        call    free
        mov     rdi, rbp
        call    free
        xor     eax, eax
        pop     rbp
        ret
main.cold:
```

Output (0/0) x86-64 gcc (trunk) - cached (8745B) ~564 lines filtered

Executor x86-64 gcc (trunk) (C, Editor #1)

Wrap lines

x86-64 gcc (trunk) all -Wextra -Wpedantic -O2

```
Program returned: 139
Program stderr
free(): double free detected in tcache 2
```

x86-64 gcc (trunk) - cached

Rust source #2

```rust
pub fn main() {
    let a = Box::new(42);
    drop(a);
    println!("{}", *a);
}
```

rustc 1.55.0 (Rust, Editor #2, Compiler #1)

rustc 1.55.0 --edition=2018

```
<Compilation failed>

# For more information see the output
# To open the output window, click or
```

Output (0/13) rustc 1.55.0 - 301ms

Executor rustc 1.55.0 (Rust, Editor #2)

Wrap lines Libraries Compilation Arguments Stdin Compiler output

rustc 1.55.0 --edition=2018 -O -Cpanic=abort

```
Could not execute the program
Compiler returned: 1
Compiler stderr
error[E0382]: borrow of moved value: `a`
 --> <source>:4:20
  |
2 |     let a = Box::new(42);
  |         - move occurs because `a` has type `Box<i32>`, which does not implement the `Copy` trait
3 |     drop(a);
  |          - value moved here
4 |     println!("{}", *a);
  |                    ^^ value borrowed here after move

error: aborting due to previous error

For more information about this error, try `rustc --explain E0382`.
```

rustc 1.55.0 - 802ms

C source #1

Save/Load Add new... Vim C

```c
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    int * const a = malloc(sizeof(int));
    if (a == NULL)
        abort();
    *a = 42;
    free(a);
    printf("%i\n", *a);
}
```

x86-64 gcc (trunk) (C, Editor #1, Compiler #2)

x86-64 gcc (trunk) std=gnu99 -Wall -Wextra -Wpedantic -O

Output... Filter... Libraries Add new... Add tool...

```
.LC0:
        .string "%i\n"
main:
        push    rbx
        mov     edi, 4
        call    malloc
        test    rax, rax
        je      .L3
        mov     rbx, rax
        mov     rdi, rax
        call    free
        mov     esi, DWORD PTR [rbx]
        mov     edi, OFFSET FLAT:.LC0
        xor     eax, eax
        call    printf
```

Output (0/0) x86-64 gcc (trunk) - cached (9175B) ~588 lines filtered

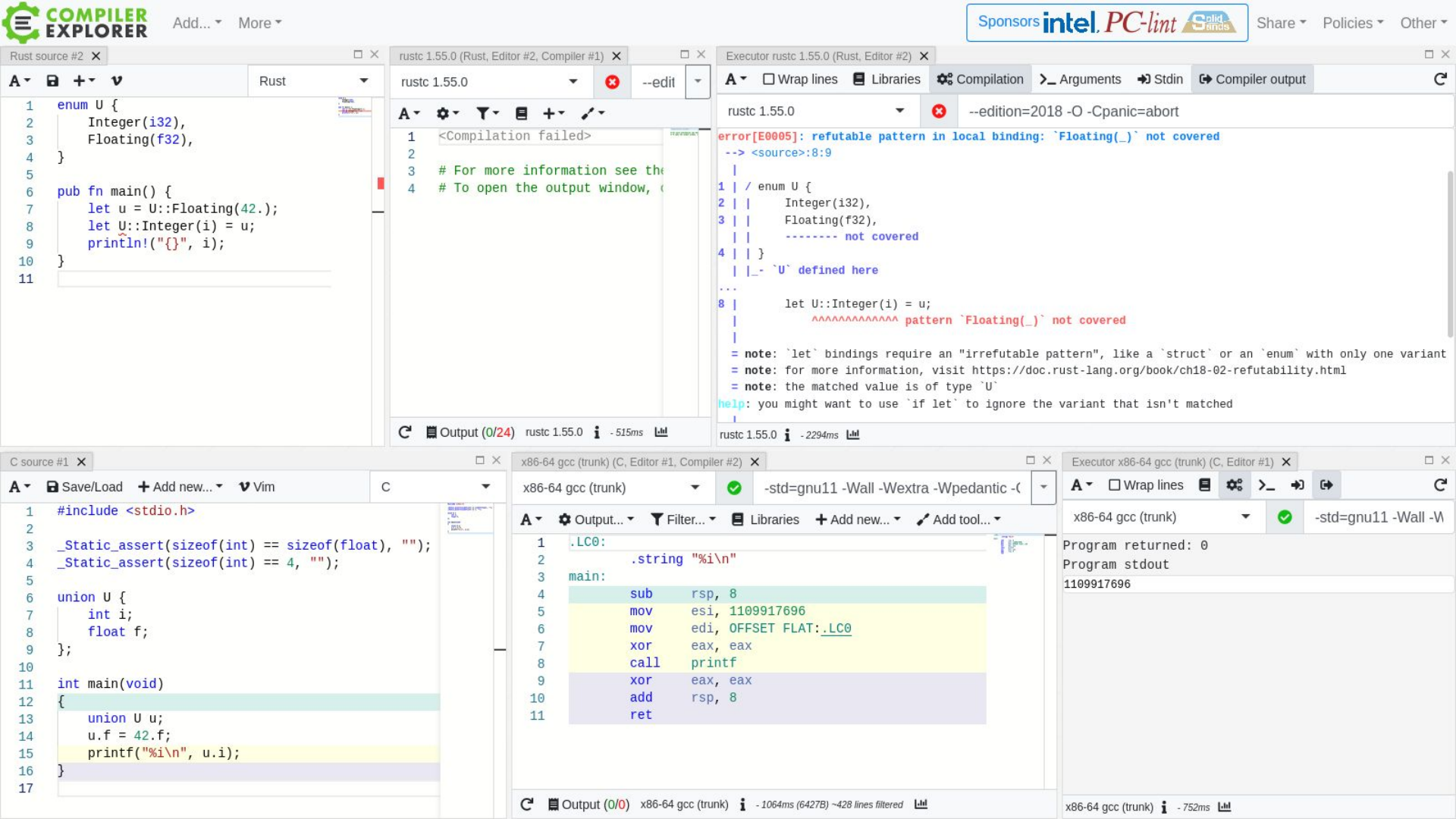
Executor x86-64 gcc (trunk) (C, Editor #1)

Wrap lines

x86-64 gcc (trunk) -std=gnu99 -Wall -W

```
Program returned: 0
Program stdout
0
```

x86-64 gcc (trunk) - cached


COMPILER EXPLORER
Add...
More
Sponsors intel PC-lint
Share
Policies
Other
Rust source #2
Rust
enum U {
Integer(i32),
Floating(f32),
}
pub fn main() {
let u = U::Floating(42.);
let U::Integer(i) = u;
println!("{}", i);
}
rustc 1.55.0 (Rust, Editor #2, Compiler #1)
rustc 1.55.0
--edit
<Compilation failed>
For more information see the
To open the output window,
Output (0/24)
rustc 1.55.0
- 515ms
Executor rustc 1.55.0 (Rust, Editor #2)
Wrap lines
Libraries
Compilation
Arguments
Stdin
Compiler output
rustc 1.55.0
--edition=2018 -O -Cpanic=abort
error[E0005]: refutable pattern in local binding: `Floating(_)` not covered
--> <source>:8:9
1 | / enum U {
2 | | Integer(i32),
3 | | Floating(f32),
| | -------- not covered
4 | | }
| |_- `U` defined here
...
8 | let U::Integer(i) = u;
| ^^^^^^^^^^^^ pattern `Floating(_)` not covered
= note: `let` bindings require an "irrefutable pattern", like a `struct` or an `enum` with only one variant
= note: for more information, visit https://doc.rust-lang.org/book/ch18-02-refutability.html
= note: the matched value is of type `U`
help: you might want to use `if let` to ignore the variant that isn't matched
rustc 1.55.0
- 2294ms
C source #1
Save/Load
Add new...
Vim
C
#include <stdio.h>
_Static_assert(sizeof(int) == sizeof(float), "");
_Static_assert(sizeof(int) == 4, "");
union U {
int i;
float f;
};
int main(void)
{
union U u;
u.f = 42.f;
printf("%i\n", u.i);
}
x86-64 gcc (trunk) (C, Editor #1, Compiler #2)
x86-64 gcc (trunk)
-std=gnu11 -Wall -Wextra -Wpedantic -(
Output...
Filter...
Libraries
Add new...
Add tool...
.LC0:
.string "%i\n"
main:
sub rsp, 8
mov esi, 1109917696
mov edi, OFFSET FLAT:.LC0
xor eax, eax
call printf
xor eax, eax
add rsp, 8
ret
Output (0/0)
x86-64 gcc (trunk)
- 1064ms (6427B) ~428 lines filtered
Executor x86-64 gcc (trunk) (C, Editor #1)
Wrap lines
x86-64 gcc (trunk)
-std=gnu11 -Wall -W
Program returned: 0
Program stdout
1109917696
x86-64 gcc (trunk)
- 752ms

C source #1

Save/Load Add new... Vim C

```c
#include <stdlib.h>

// On success, `result` is written and the return value is 0.
// On error, the return value is < 0.
int get_some_data(int * result);
void do_something(int foo);

void f(void) {
    int data;
    if (get_some_data(&data) < 0)
        abort();

    do_something(data);
}

```

x86-64 gcc 11.2 (C, Editor #1, Compiler #1)

x86-64 gcc 11.2 -std=c11 -Wall -Wextra -Wpedantic -O2

Output... Filter... Libraries Add new... Add tool...

```asm
f:
        sub     rsp, 24
        lea     rdi, [rsp+12]
        call    get_some_data
        test    eax, eax
        js      .L3
        mov     edi, DWORD PTR [rsp+12]
        call    do_something
        add     rsp, 24
        ret
f.cold:
.L3:
        call    abort
```

Output (0/0) x86-64 gcc 11.2 - 838ms (6182B) ~402 lines filtered

Rust source #2

Save/Load Add new... Vim Rust

```rust
extern "Rust" {
    fn get_some_data_() -> Result<i32, ()>;
    fn do_something_(foo: i32);
}

pub fn f() {
    let data = get_some_data().unwrap();
    do_something(data);
}

```

rustc 1.55.0 (Rust, Editor #2, Compiler #2)

rustc 1.55.0 --edition=2018 -O -Cpanic=abort

Output... Filter... Libraries Add new... Add tool...

```asm
example::f:
        push    rax
        call    qword ptr [rip + get_some_data_@GOTPCREL]
        test    eax, eax
        jne     .LBB2_1
        mov     edi, edx
        pop     rax
        jmp     qword ptr [rip + do_something_@GOTPCREL]
.LBB2_1:
        lea     rdi, [rip + .L__unnamed_2]
        lea     rcx, [rip + .L__unnamed_3]
        lea     r8, [rip + .L__unnamed_4]
        mov     rdx, rsp
        mov     esi, 43
        call    qword ptr [rip + core::result::unwrap_failed@GOTPCREL]
        ud2
```

Output (0/0) rustc 1.55.0 - cached (9350B) ~450 lines filtered

Building an abstraction

Rust source #2

Save/Load Add new... Vim Rust

```
1  // Bindings
2  extern {
3      fn get_pointer() -> *mut i32;
4      fn use_pointer(ptr: *mut i32);
5  }
6
```

rustc 1.55.0 (Rust, Editor #2, Compiler #2)

rustc 1.55.0 --edition=2018 -O -Cpanic=abort

Output... Filter... Libraries Add new... Add tool...

```
1  <No assembly to display (~5 lines filtered)>
```

Output (0/16) rustc 1.55.0 - 2977ms (142B) ~5 lines filtered

Rust source #2

Save/Load Add new... Vim Rust

```rust
// Bindings
extern {
    fn get_pointer() -> *mut i32;
    fn use_pointer(ptr: *mut i32);
}


// Abstractions code
mod foo {
    /// # Invariants
    ///
    /// The pointer is valid.
    pub struct Foo {
        ptr: *mut i32,
    }

    impl Foo {
        pub fn new() -> Self {
            Foo {
                // SAFETY: `get_pointer()` is always safe to call.
                ptr: unsafe { crate::get_pointer() }
            }
        }

        pub fn do_something(&mut self) {
            // SAFETY: `use_pointer()` requires that the pointer
            // is valid, which holds due to the type invariant.
            unsafe { crate::use_pointer(self.ptr); }
        }
    }
}

```

rustc 1.55.0 (Rust, Editor #2, Compiler #2)

rustc 1.55.0 --edition=2018 -O -Cpanic=abort

Output... Filter... Libraries Add new... Add tool...

```
<No assembly to display (~5 lines filtered)>
```

Output (0/34) rustc 1.55.0 - 3826ms (142B) ~5 lines filtered

Rust source #2

Save/Load Add new... Vim

Rust

```rust
// Bindings
extern {
    fn get_pointer() -> *mut i32;
    fn use_pointer(ptr: *mut i32);
}


// Abstractions code
mod foo {
    /// # Invariants
    ///
    /// The pointer is valid.
    pub struct Foo {
        ptr: *mut i32,
    }

    impl Foo {
        pub fn new() -> Self {
            Foo {
                // SAFETY: `get_pointer()` is always safe to call.
                ptr: unsafe { crate::get_pointer() }
            }
        }

        pub fn do_something(&mut self) {
            // SAFETY: `use_pointer()` requires that the pointer
            // is valid, which holds due to the type invariant.
            unsafe { crate::use_pointer(self.ptr); }
        }
    }
}


// User code
use foo::*;

pub fn f() {
    let mut my_foo = Foo::new();
    my_foo.do_something();
}

```

rustc 1.55.0 (Rust, Editor #2, Compiler #2)

rustc 1.55.0

--edition=2018 -O -Cpanic=abort

Output... Filter... Libraries Add new... Add tool...

```asm
example::f:
        push    rax
        call    qword ptr [rip + get_pointer@GOTPCREL]
        mov     rdi, rax
        pop     rax
        jmp     qword ptr [rip + use_pointer@GOTPCREL]
```

Output (0/0) rustc 1.55.0 - 532ms (3945B) ~238 lines filtered

Output of rustc 1.55.0 (Compiler #2)

Wrap lines

```
Compiler returned: 0
```

Rust source #2

Save/Load Add new... Vim Rust

```rust
// Bindings
extern {
    fn get_pointer() -> *mut i32;
    fn use_pointer(ptr: *mut i32);
}


// Abstractions code
mod foo {
    /// # Invariants
    ///
    /// The pointer is valid.
    pub struct Foo {
        ptr: *mut i32,
    }

    impl Foo {
        pub fn new() -> Self {
            Foo {
                // SAFETY: `get_pointer()` is always safe to call.
                ptr: unsafe { crate::get_pointer() }
            }
        }

        pub fn do_something(&mut self) {
            // SAFETY: `use_pointer()` requires that the pointer
            // is valid, which holds due to the type invariant.
            unsafe { crate::use_pointer(self.ptr); }
        }
    }
}


// User code
use foo::*;

pub fn f() {
    let mut my_foo = Foo::new();
    my_foo.ptr = 42 as *mut i32;
    my_foo.do_something();
}
```

rustc 1.55.0 (Rust, Editor #2, Compiler #2)

rustc 1.55.0 --edition=2018 -O -Cpanic=abort

Output... Filter... Libraries Add new... Add tool...

```
<Compilation failed>

# For more information see the output window
```

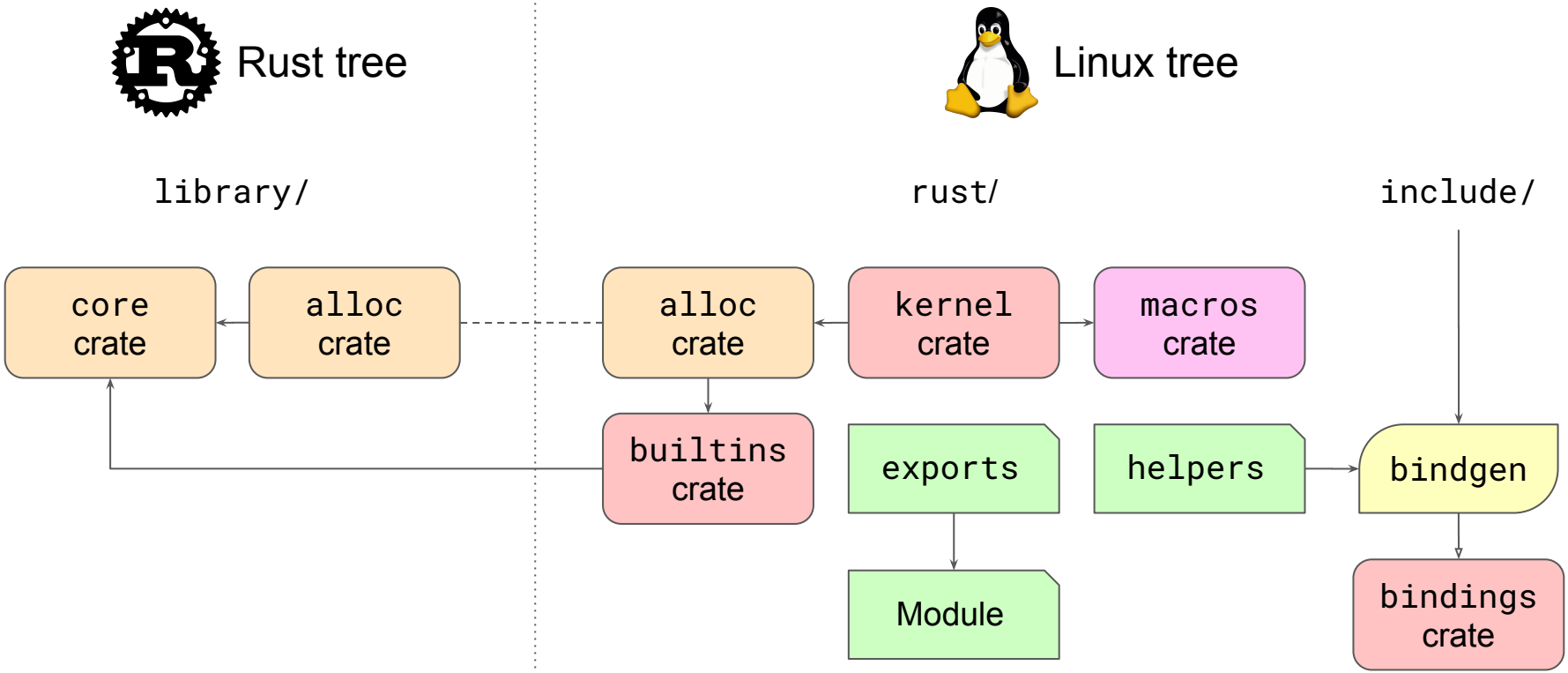
Output (0/9) rustc 1.55.0 - 1569ms

Output of rustc 1.55.0 (Compiler #2)

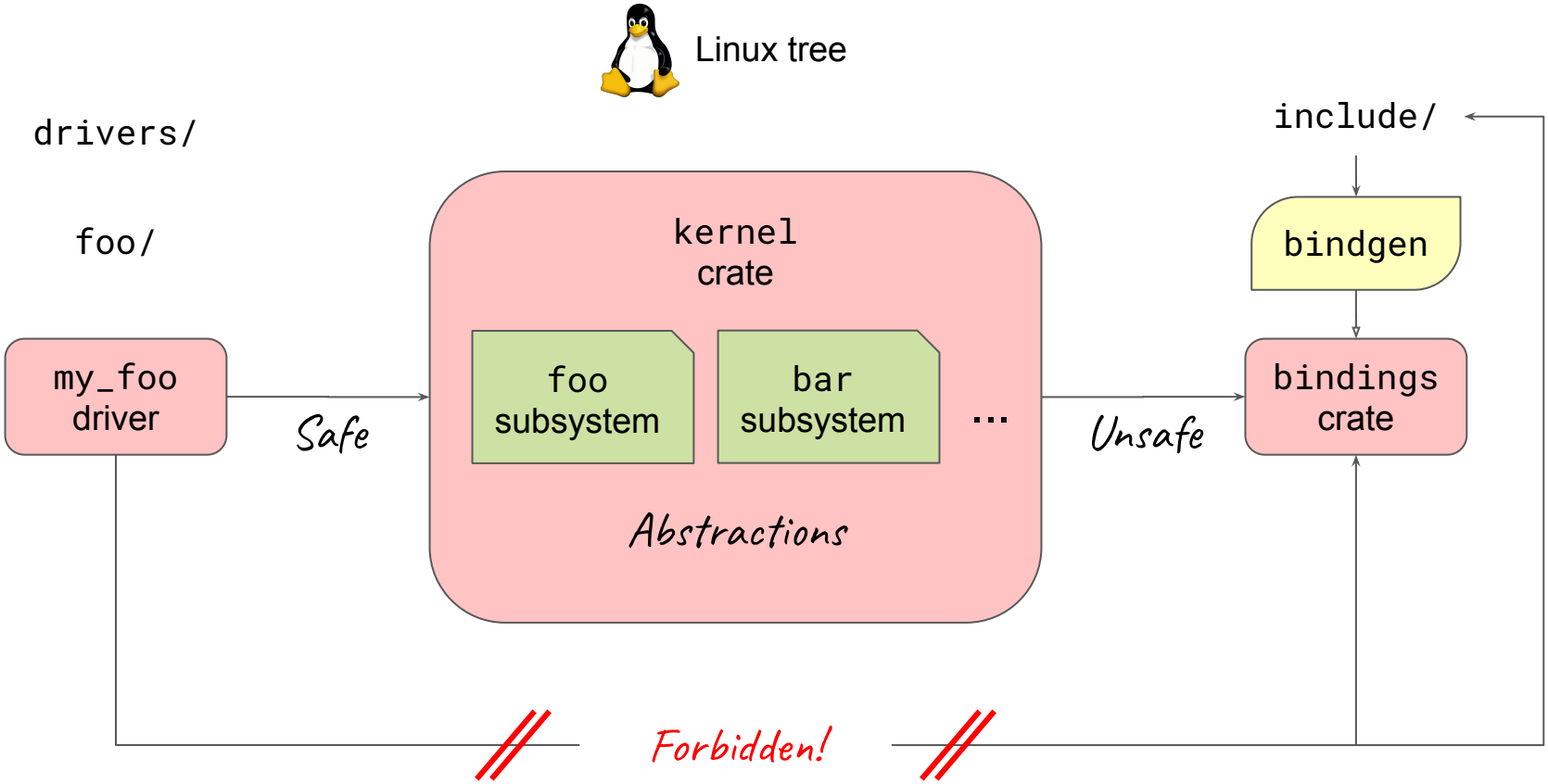
Wrap lines

```
error[E0616]: field `ptr` of struct `foo::Foo` is private
  --> <source>:39:12
   |
39 |     my_foo.ptr = 42 as *mut i32;
   |            ^^^ private field

error: aborting due to previous error

For more information about this error, try `rustc --explain E0616`.
Compiler returned: 1
```

Rust support in the kernel


Rust tree
Linux tree
library/
rust/
include/
core crate
alloc crate
alloc crate
kernel crate
macros crate
builtins crate
exports
helpers
bindgen
Module
bindings crate

Driver point of view


Linux tree
drivers/
foo/
my_foo
driver
Safe
kernel
crate
foo
subsystem
bar
subsystem
...
Abstractions
Unsafe
include/
bindgen
bindings
crate
Forbidden!

Supported architectures

`arm`	(`armv6` only)
`arm64`	
`powerpc`	(`ppc64le` only)
`riscv`	(`riscv64` only)
`x86`	(`x86_64` only)

See `Documentation/rust/arch-support.rst`

Supported architectures

`arm`	(`armv6` only)
`arm64`	
`powerpc`	(`ppc64le` only)
`riscv`	(`riscv64` only)
`x86`	(`x86_64` only)

...so far!

32-bit and other restrictions should be easy to remove

Kernel LLVM builds work for `mips` and `s390`

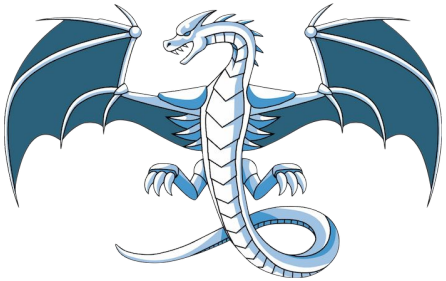
GCC codegen paths should open up more

See `Documentation/rust/arch-support.rst`

Rust codegen paths for the kernel

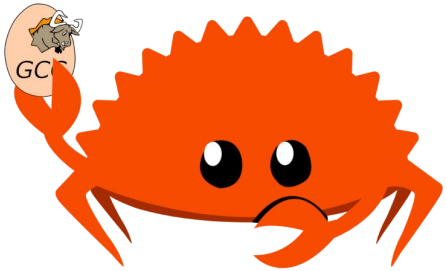
`rustc_codegen_gcc`

Already passes most rustc tests



`rustc_codegen_llvm`

Main one



Rust GCC

Expected in 1-2 years (rough estimate)

Documentation

Crate kernel [-][src]

[-] The `kernel` crate.

This crate contains the kernel APIs that have been ported or wrapped for usage by Rust code in the kernel and is shared by all of them.

In other words, all the rest of the Rust code in the kernel (e.g. kernel modules written in Rust) depends on `core`, `alloc` and this crate.

If you need a kernel C API that is not ported or wrapped yet here, then do so first instead of bypassing this crate.

Modules

buffer	Struct for writing to a pre-allocated buffer with the `write!` macro.
c_types	C types for the bindings.
chrdev	Character devices.
file	Files and file descriptors.
file_operations	File operations.
io_buffer	Buffers used in IO.
iov_iter	IO vector iterators.
linked_list	Linked lists.
miscdev	Miscellaneous devices.
of	Devicetree and Open Firmware abstractions.
pages	Kernel page allocation and management.
platdev	Platform devices.
prelude	The `kernel` prelude.
print	Printing facilities.

Struct kernel::sync::Mutex [-][src]

```
pub struct Mutex<T: ?Sized> { /* fields omitted */ }
```

[-] Exposes the kernel's `struct mutex`. When multiple threads attempt to lock the same mutex, only one at a time is allowed to progress, the others will block (sleep) until the mutex is unlocked, at which point another thread will be allowed to wake up and make progress.

A `Mutex` must first be initialised with a call to `Mutex::init` before it can be used. The `mutex_init` macro is provided to automatically assign a new lock class to a mutex instance.

Since it may block, `Mutex` needs to be used with care in atomic contexts.

Implementations

[-] `impl<T> Mutex<T>` [src]

[-] `pub unsafe fn new(t: T) -> Self` [src]

Constructs a new mutex.

Safety

The caller must call `Mutex::init` before using the mutex.

[-] `impl<T: ?Sized> Mutex<T>` [src]

```
/// end.
///
/// Used for interoperability with kernel APIs that take C strings.
#[repr(transparent)]
pub struct CStr([u8]);

impl CStr {
    /// Returns the length of this string excluding `NUL`.
    #[inline]
    pub const fn len(&self) -> usize {
        self.len_with_nul() - 1
    }

    /// Returns the length of this string with `NUL`.
    #[inline]
    pub const fn len_with_nul(&self) -> usize {
        // SAFETY: This is one of the invariant of `CStr`.
        // We add a `unreachable_unchecked` here to hint the optimizer that
        // the value returned from this function is non-zero.
        if self.0.is_empty() {
            unsafe { core::hint::unreachable_unchecked() };
        }
        self.0.len()
    }
```

Struct Mutex

Methods

lock

new

Trait Implementations

Lock

NeedsLockClass

Send

Sync

Auto Trait Implementations

!Unpin

Blanket Implementations

Any

Borrow<T>

BorrowMut<T>

From<T>

All crates

pr

?

Results for pr

In Names (176)	In Parameters (0)	In Return Types (0)

kernel::print	Printing facilities.
kernel::platdev::PlatformDriver::probe	Platform driver probe.
kernel::pr_err	Prints an error-level message (level 3).
kernel::pr_cont	Continues a previous log message in the same line.
kernel::pr_crit	Prints a critical-level message (level 2).
kernel::pr_info	Prints an info-level message (level 6).
kernel::pr_warn	Prints a warning-level message (level 4).
kernel::prelude	The `kernel` prelude.
kernel::pr_alert	Prints an alert-level message (level 1).
kernel::pr_emerg	Prints an emergency-level message (level 0).
kernel::linked_list::CursorMut::peek_prev	Returns the element immediately before the one the cursor ...
kernel::pr_notice	Prints a notice-level message (level 5).
kernel::prelude::Vec::swap_remove	Removes an element from the vector and returns it.
kernel::prelude::Box::is_prefix_of	
kernel::prelude::Box::strip_prefix_of	
alloc::prelude	The alloc Prelude
core::prelude	The libcore prelude
core::iter::Product	Trait to represent types that can be created by ...
core::iter::Product::product	Method which takes an iterator and generates `Self` from ...
core::iter::Iterator::product	Iterates over the entire iterator, multiplying all the ...
core::option::Option::product	Takes each element in the [`Iterator`]: if it is a [None], ...

Documentation code

```rust
/// Wraps the kernel's `struct task_struct`.
///
/// # Invariants
///
/// The pointer `Task::ptr` is non-null and valid. Its reference count is also non-zero.
///
/// # Examples
///
/// The following is an example of getting the PID of the current thread with
/// zero additional cost when compared to the C version:
///
/// ```
/// # use kernel::prelude::*;
/// use kernel::task::Task;
///
/// # fn test() {
/// Task::current().pid();
/// # }
/// ```
pub struct Task {
    pub(crate) ptr: *mut bindings::task_struct,
}
```

Conditional compilation

Rust code has access to conditional compilation based on the kernel config

```
#[cfg(CONFIG_X)]       // `CONFIG_X` is enabled (`y` or `m`)
#[cfg(CONFIG_X="y")]   // `CONFIG_X` is enabled as a built-in (`y`)
#[cfg(CONFIG_X="m")]   // `CONFIG_X` is enabled as a module   (`m`)
#[cfg(not(CONFIG_X))] // `CONFIG_X` is disabled
```

Coding guidelines

No direct access to C bindings

No undocumented public APIs

No implicit `unsafe` block

Docs follows Rust standard library style

`// SAFETY` proofs for all `unsafe` blocks

Clippy linting enabled

Automatic formatting enforced

Rust 2018 edition & idioms

No unneeded panics

No infallible allocations

...

Coding guidelines

No direct access to C bindings

No undocumented public APIs

No implicit `unsafe` block

Docs follows Rust standard library style

`// SAFETY` proofs for all `unsafe` blocks

Clippy linting enabled

Automatic formatting enforced

Rust 2018 edition & idioms

No unneeded panics

No infallible allocations

...

Aiming to be as strict as possible

Abstractions code

```rust
/// Wraps the kernel's `struct file`.
///
/// # Invariants
///
/// The pointer `File::ptr` is non-null and valid.
/// Its reference count is also non-zero.
pub struct File {
   pub(crate) ptr: *mut bindings::file,
}
```

```rust
impl File {
    /// Constructs a new [`struct file`] wrapper from a file descriptor.
    ///
    /// The file descriptor belongs to the current process.
    pub fn from_fd(fd: u32) -> Result<Self> {
        // SAFETY: FFI call, there are no requirements on `fd`.
        let ptr = unsafe { bindings::fget(fd) };
        if ptr.is_null() {
            return Err(Error::EBADF);
        }

        // INVARIANTS: We checked that `ptr` is non-null, so it is valid.
        // `fget` increments the ref count before returning.
        Ok(Self { ptr })
    }

    // ...
}
```

Driver code

```c
static int pl061_resume(struct device *dev)
{
    int offset;

    struct pl061 *pl061 = dev_get_drvdata(dev);

    for (offset = 0; offset < PL061_GPIO_NR; offset++) {
        if (pl061->csave_regs.gpio_dir & (BIT(offset)))
            pl061_direction_output(&pl061->gc, offset,
                    pl061->csave_regs.gpio_data &
                    (BIT(offset)));
        else
            pl061_direction_input(&pl061->gc, offset);


    }

    writeb(pl061->csave_regs.gpio_is, pl061->base + GPIOIS);
    writeb(pl061->csave_regs.gpio_ibe, pl061->base + GPIOIBE);
    writeb(pl061->csave_regs.gpio_iev, pl061->base + GPIOIEV);
    writeb(pl061->csave_regs.gpio_ie, pl061->base + GPIOIE);

    return 0;
}
```

```rust
fn resume(data: &Ref<DeviceData>) -> Result {


    let inner = data.lock();
    let pl061 = data.resources().ok_or(Error::ENXIO)?;

    for offset in 0..PL061_GPIO_NR {
        if inner.csave_regs.gpio_dir & bit(offset) != 0 {
            let v = inner.csave_regs.gpio_data & bit(offset) != 0;
            let _ = <Self as gpio::Chip>::direction_output(
                data, offset.into(), v);
        } else {
            let _ = <Self as gpio::Chip>::direction_input(
                data, offset.into());
        }
    }

    pl061.base.writeb(inner.csave_regs.gpio_is, GPIOIS);
    pl061.base.writeb(inner.csave_regs.gpio_ibe, GPIOIBE);
    pl061.base.writeb(inner.csave_regs.gpio_iev, GPIOIEV);
    pl061.base.writeb(inner.csave_regs.gpio_ie, GPIOIE);

    Ok(())
}
```

— https://lwn.net/Articles/863459/

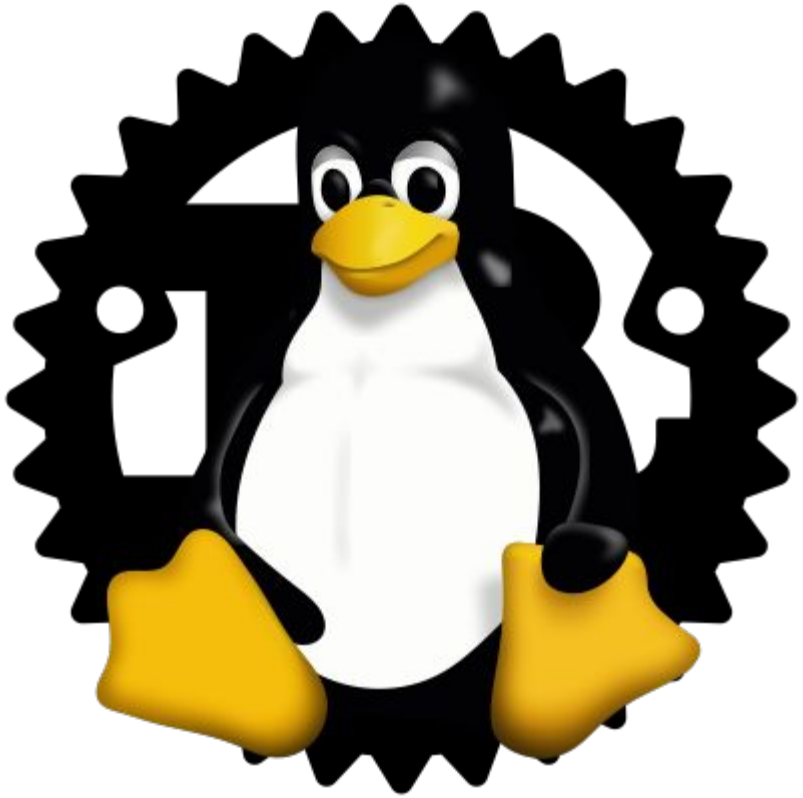
Testing code

```rust
fn trim_whitespace(mut data: &[u8]) -> &[u8] {
    // ...
}

#[cfg(test)]
mod tests {
   use super::*;

   #[test]
   fn test_trim_whitespace() {
       assert_eq!(trim_whitespace(b"foo    "), b"foo");
       assert_eq!(trim_whitespace(b"    foo"), b"foo");
       assert_eq!(trim_whitespace(b"  foo  "), b"foo");
   }
}
```

```
/// Getting the current task and storing it in some struct. The reference count is automatically
/// incremented when creating `State` and decremented when it is dropped:
///
/// ```
/// # use kernel::prelude::*;
/// use kernel::task::Task;
///
/// struct State {
///     creator: Task,
///     index: u32,
/// }
///
/// impl State {
///     fn new() -> Self {
///         Self {
///             creator: Task::current().clone(),
///             index: 0,
///         }
///     }
/// }
/// ```
```

How to proceed?

https://github.com/Rust-for-Linux/linux



Rust for Linux

Miguel Ojeda

ojeda@kernel.org

Backup slides

C Charter

6. **Keep the spirit of C.** The Committee kept as a major goal to preserve the traditional spirit of C. There are many facets of the spirit of C, but the essence is a community sentiment of the underlying principles upon which the C language is based. The C11 revision added a new facet **f** to the original list of facets. The new spirit of C can be summarized in phrases like:

(a) *Trust the programmer.*
(b) *Don't prevent the programmer from doing what needs to be done.*
(c) *Keep the language small and simple.*
(d) *Provide only one way to do an operation.*
(e) *Make it fast, even if it is not guaranteed to be portable.*
(f) *Make support for safety and security demonstrable.*

— N2086 C2x Charter - Original Principles

12. ***Trust the programmer*, as a goal, is outdated in respect to the security and safety programming communities.** While it should not be totally disregarded as a facet of the spirit of C, the C11 version of the C Standard should take into account that programmers need the ability to check their work.

— N2086 C2x Charter - Additional Principles for C11

COMPILER EXPLORER Add... More Sponsors intel. PC-lint Share Policies Other

C source #1

Save/Load Add new... Vim C

```
1  int f(const int x) {
2      return x + 1 > x;
3  }
4
```

x86-64 gcc 11.2 (C, Editor #1, Compiler #2)

x86-64 gcc 11.2 -std=gnu89 -O2 -Wall -Wextra -Wpedantic

Output... Filter... Libraries Add new... Add tool...

```
1  f:
2          mov     eax, 1
3          ret
```

Output (0/0) x86-64 gcc 11.2 - 681ms (2676B) ~171 lines filtered

C source #2

Save/Load Add new... Vim C

```
1  int f(const int x) {
2      return x + 1 > x;
3  }
4
```

x86-64 gcc 11.2 (C, Editor #2, Compiler #1)

x86-64 gcc 11.2 -std=gnu89 -O2 -Wall -Wextra -Wpedantic -fwrapv

Output... Filter... Libraries Add new... Add tool...

```
1  f:
2          xor     eax, eax
3          cmp     edi, 2147483647
4          setne   al
5          ret
```

Output (0/0) x86-64 gcc 11.2 - 435ms (2741B) ~172 lines filtered

C source #3

Save/Load Add new... Vim C

```
1  int f(const int x) {
2      return x + 1 > x;
3  }
4
```

x86-64 gcc 11.2 (C, Editor #3, Compiler #3)

x86-64 gcc 11.2 -std=gnu89 -O2 -Wall -Wextra -Wpedantic -ftrapv

Output... Filter... Libraries Add new... Add tool...

```
1  f:
2          push    rbx
3          mov     esi, 1
4          mov     ebx, edi
5          call    __addvsi3
6          cmp     eax, ebx
7          pop     rbx
8          setg    al
9          movzx   eax, al
10         ret
```

Output (0/0) x86-64 gcc 11.2 - 722ms (4034B) ~252 lines filtered

Undefined Behavior

3.4.3

1 **undefined behavior**

behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this document imposes no requirements

2 **Note 1 to entry:** Possible undefined behavior ranges from ignoring the situation completely with unpredictable results, to behaving during translation or program execution in a documented manner characteristic of the environment (with or without the issuance of a diagnostic message), to terminating a translation or execution (with the issuance of a diagnostic message).

3 **Note 2 to entry:** J.2 gives an overview over properties of C programs that lead to undefined behavior.

4 **EXAMPLE** An example of undefined behavior is the behavior on dereferencing a null pointer.

— N2596 C2x Working Draft

Example of UB

— The value of the second operand of the / or % operator is zero (6.5.5).

```
int f(int a, int b) {
    return a / b;
}
```

Example of UB

— The value of the second operand of the / or % operator is zero (6.5.5).

```
int f(int a, int b) {
    return a / b;
}
```

UB ∀x f(x, 0);

Example of UB

Any other inputs that trigger UB?

```
int f(int a, int b) {
    return a / b;
}
```

Example of UB

Any other inputs that trigger UB?

```
int f(int a, int b) {
    return a / b;
}
```

```
UB f(INT_MIN, -1);
```

Instances of UB

Instances of UB

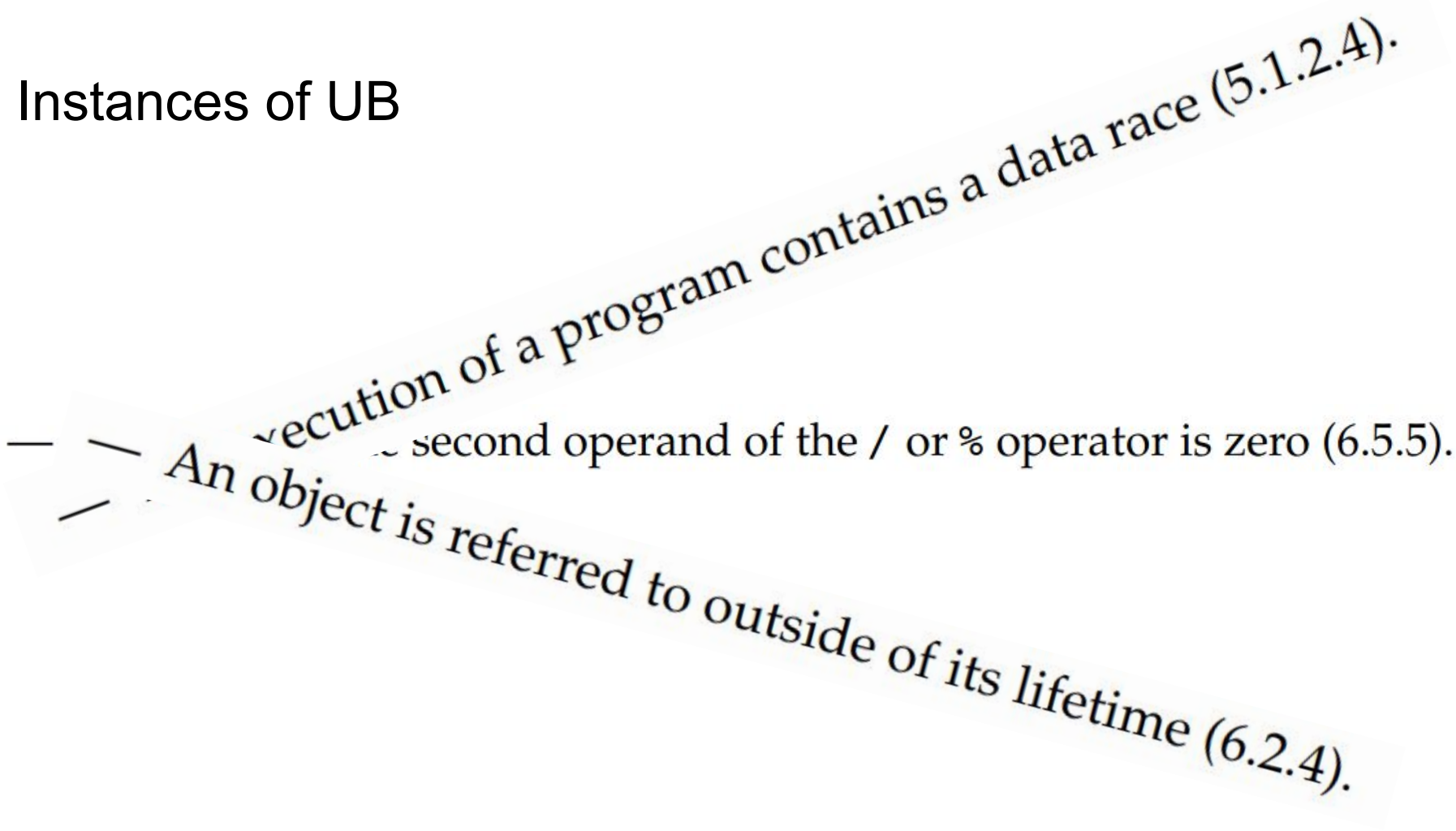
— The value of the second operand of the `/` or `%` operator is zero (6.5.5).

Instances of UB

— The execution of a program contains a data race (5.1.2.4).

— Th second operand of the / or % operator is zero (6.5.5).

Instances of UB



Instan[illegible] UB

— The value of a pointer to an object whose lifetime has ended is used (6.2.4).

— [illegible]ecution of a p[illegible]am contains a data race (5.1.2.4).

— [illegible] second operand of t[illegible] operator is zero (6.5.5).

— An object is referred to outside of its lifetime (6.2.4).

Instan UB

— The value of a pointer

— The value of an object with automatic storage duration is used while it is indeterminate (6.2.4, 6.7.9, 6.8).

— — ecution of a p ct whose lifetime has ended is used (6.2.4).

— — second operand of t operator is zero (6.5.5).

— An object is referred to outside of its lifetime (6.2.4).

a data race (5.1.2.4).

Instan UB

— The value of a point

— A trap representation is read by an lvalue expression that does not have character type (6.2.6.1).

— The value of an object with duration is used while it is indeterminate (6.2.4, 6.7.9, 6.8).

— — execution of a program whose lifetime has ended is used (6.2.4).

— — second operand of the operator is zero (6.5.5).

— An object is referred to outside of its lifetime (6.2.4).

— a data race (5.1.2.4).

Instan UB

— The value of a point

— A trap representation is read by an lvalue expression that does not have character type (6.2.6.1).

— The value of an object wit duration is used while it is indeterminate (6.2.4, 6.7.9, 6.8).

— — ecution of a p ct whose lifetime operator i

— An object is r second operand of t

— Pointers that do not point into, or just beyond, the same array object are subtracted (6.5.6).

— — ded is used (6.2.4).

— — u to outside of its lifetime (6.2.4).

— — a data race (5.1.2.4).

Instan UB

— The value of a point a data race (5.1.2.4).

— A trap representation is read by an lvalue expression that does not have character type (6.2.6.1).

— The value of an object wit duration is used while it is indeterminate (6.2.4, 6.7.9, 6.8).

— — ecution of a p ct whose lifetime operator i

— — second operand of t

— An object is r ded is used (6.2.4).

— Pointers that do not point into, or just beyond, the same array object are subtracted (6.5.6).

u to outside of its lifetime (6.2.4).

Avoiding UB

```
int f(int a, int b) {
    if (b == 0)
        abort();

    if (a == INT_MIN && b == -1)
        abort();

    return a / b;
}
```

Avoiding UB

```
int f(int a, int b) {
    if (b == 0)
        abort();

    if (a == INT_MIN && b == -1)
        abort();

    return a / b;
}
```

f is a safe function

Safe function

Not C

```
int f(int a, int b) [[safe]] {
    if (b == 0)
        abort();

    if (a == INT_MIN && b == -1)
        abort();

    return a / b;
}
```

f is a safe function

Safe function

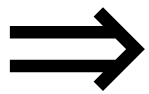
Not C

(yet? N2659)

```
int f(int a, int b) [[safe]] {
    if (b == 0)
        abort();

    if (a == INT_MIN && b == -1)
        abort();

    return a / b;
}
```

f is a safe function

Safety examples

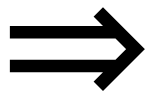


`abort()`s in C

are

Rust-safe

Safety examples



`abort()`s in C

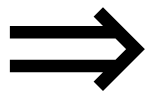
are

Rust-safe

Even if your company goes bankrupt.

Safety examples

`abort()`s in C



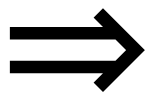
are

Rust-safe

Even if your company goes bankrupt.

Even if somebody is injured.

Safety examples

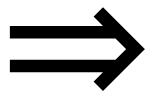


Rust panics

are

Rust-safe

Safety examples

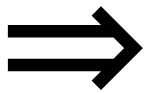


Kernel panics

are

Rust-safe

Safety examples



Uses after free, null derefs, double frees,

OOB accesses, uninitialized memory reads,

invalid inhabitants, data races...

are **not**

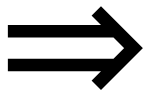
Rust-safe

Safety examples

Uses after free, null derefs, double frees,

OOB accesses, uninitialized memory reads,

invalid inhabitants, data races...

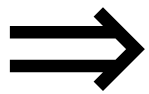


are **not**

Rust-safe

Even if your system still works.

Safety examples

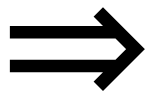


Race conditions

are

Rust-safe

Safety examples

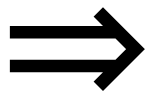


Memory leaks

are

Rust-safe

Safety examples

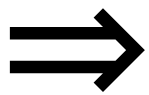


Deadlocks

are

Rust-safe

Safety examples

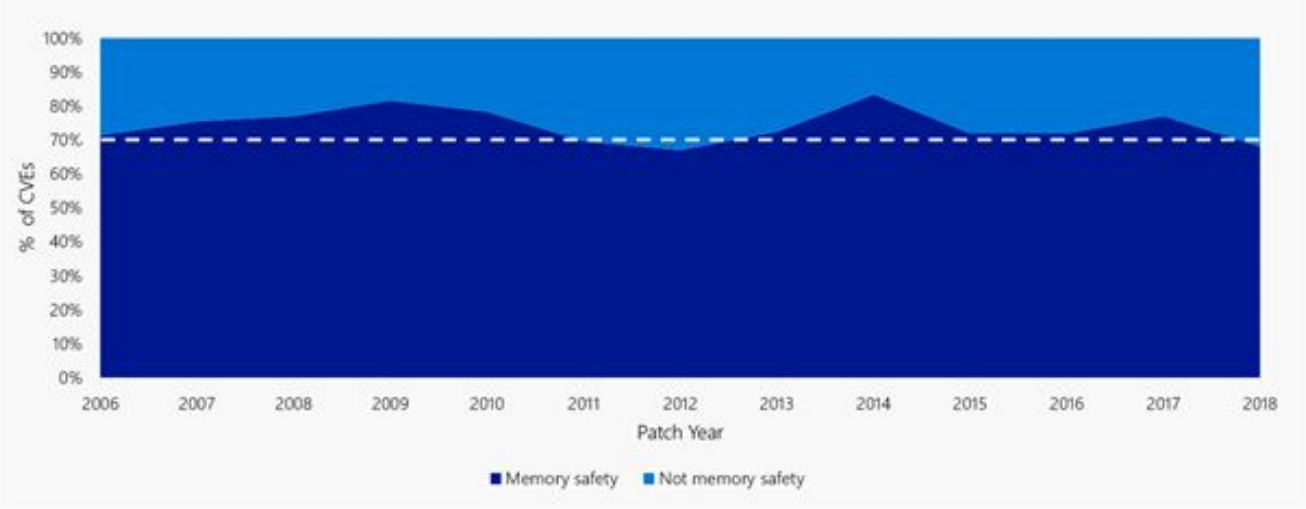


Integer overflows

are

Rust-safe

Is avoiding UB that important?



~70% of the vulnerabilities Microsoft assigns a CVE each year continue to be memory safety issues

— https://msrc-blog.microsoft.com/2019/07/18/we-need-a-safer-systems-programming-language/

Is avoiding UB that important?

Mojave (aka macOS 10.14)

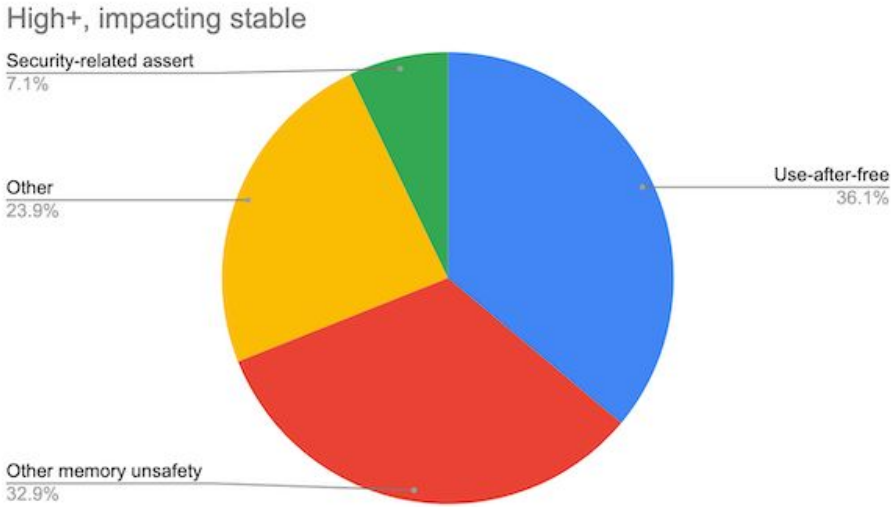
Apple released macOS 10.14 Mojave on September 24, 2018 and subsequently has issued 6 point releases.

Total CVE Count	Memory Unsafety Bugs	Percentage	Release
44	36	81.8%	10.14.6
45	40	88.9%	10.14.5
38	20	52.6%	10.14.4
23	22	95.7%	10.14.3
13	11	84.6%	10.14.2
71	40	56.3%	10.14.1
64	44	68.8%	10.14

— https://langui.sh/2019/07/23/apple-memory-safety/

Is avoiding UB that important?

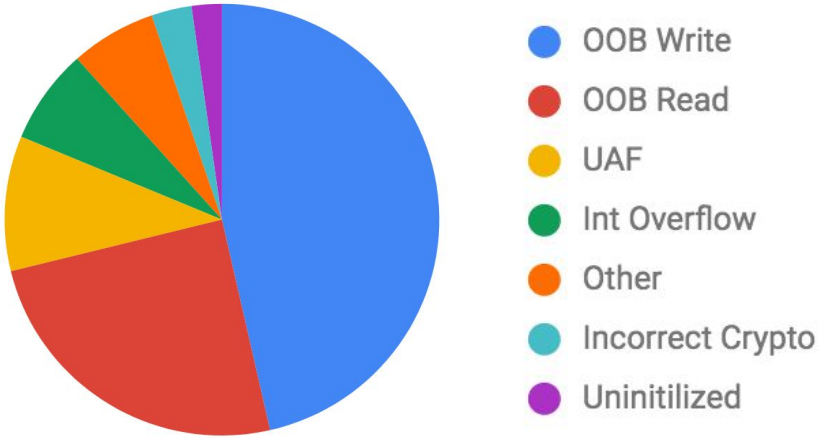
The Chromium project finds that around 70% of our serious security bugs are memory safety problems. Our next major project is to prevent such bugs at source.



— https://www.chromium.org/Home/chromium-security/memory-safety

Is avoiding UB that important?

Most of Android's vulnerabilities occur in the media and bluetooth components. Use-after-free (UAF), integer overflows, and out of bounds (OOB) reads/writes comprise 90% of vulnerabilities with OOB being the most common.



— https://security.googleblog.com/2019/05/queue-hardening-enhancements.html

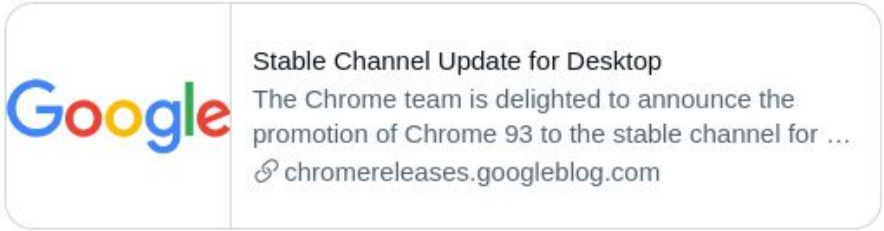
Is avoiding UB that important?

Fish in a Barrel @LazyFishBarrel · Sep 9

5/8 vulnerabilities fixed in Firefox 92 are memory unsafety mozilla.org/en-US/security… #memoryunsafety

Fish in a Barrel @LazyFishBarrel · Sep 1

13/19 (5/5 high) vulnerabilities fixed in Google Chrome 93.0.4577.63 are memory unsafety chromereleases.googleblog.com/2021/08/stable… #memoryunsafety



Does Rust help?

Does Rust help?

I took a look at this spreadsheet published a couple weeks ago...

Does Rust help?

I took a look at this spreadsheet published a couple weeks ago...

Fuzzing 100+ open source projects with OSS-Fuzz - lessons learned.

31st August, 2021

David Korczynski & Adam Korczynski,
Security Research & Security Engineering

34	Project specs			Monorail public stats			
35	Project name	Github URL	Language	Bugs	Security Bugs	Bugs verified (fixed)	Security bugs verified (fixed)
36	apache-httpd			11	2	11	2
37	blackfriday			1	0	1	0
38	caddy			8	2	1	0
39	cascadia			5	11	1	0
40	cctz			1	0	0	0
41	cfengine			2	0	0	0
42	cilium			0	5	0	0
43	Civetweb			1	0	1	0
44	Clib			11	0	4	0
45	containerd			3	3	1	0
46	dgraph			3	3	1	0

— https://adalogics.com/blog/fuzzing-100-open-source-projects-with-oss-fuzz

Does Rust help?

I filled the language column and plotted...

Does Rust help?

I filled the language column and plotted...