

An Introduction to OS-Level Verification with Asterinas and Verus

Kiwamu Okabe

kiwamu@metasepi.org

<https://metasepi.org/>



Asterinas: a Linux-compatible kernel in Rust

<https://github.com/asterinas/asterinas>

- Asterinas is a Linux-compatible kernel written in Rust.
- Most of the kernel is written in **safe Rust**.
- Low-level **unsafe Rust** is isolated in a small framework called **OSTD**.
- This keeps the memory-safety trusted code base small.



Verifying Asterinas OSTD with Verus

<https://github.com/asterinas/vostd>

- **OSTD** wraps low-level hardware operations in safe Rust APIs.
- Those low-level operations sometimes require **unsafe Rust**.
- **VOSTD** uses **Verus** to formally verify OSTD.
- Goal: make OSTD's safety guarantees stronger and more trustworthy.



Verus: verifying Rust code

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

- Write normal Rust code.
- Add **specifications** describing what the code must do.
- Verus checks that the code always satisfies those specifications.
- Verification is done **statically**.
- No extra run-time checks are required.



Example: `align_down` in VOSTD

```
$ cd ~/src/vostd
$ vi ostd/libs/align_ext/src/lib.rs
    #[inline]
    #[verus_spec(ret =>
        requires
        /// -- snip proof --
        ensures
        /// -- snip proof --
    )]

    /// ## Postconditions
    /// - `align` is a power of two `>= 2`.
    /// - The return value is the greatest multiple of `align`
    ///   that is less than or equal to `self`.
    fn align_down(self, align: Self) -> Self {
        /// -- snip proof --
        self & !(align - 1)
    }
}
```



Let's break VOSTD on purpose

```
$ git diff | cat
diff --git a/ostd/libs/align_ext/src/lib.rs b/ostd/libs/align_ext/src/lib.rs
index 3640e7c7c..288a0e4e6 100644
--- a/ostd/libs/align_ext/src/lib.rs
+++ b/ostd/libs/align_ext/src/lib.rs
@@ -212,7 +212,7 @@ macro_rules! impl_align_ext {
     assert((self & !mask) as nat == nat_align_down(self as nat, align as nat));
     lemma_nat_align_down_sound(self as nat, align as nat);
 }
- self & !(align - 1)
+ self & (align - 1)
 }
 }
)*
```



Verus catches the bug

```

$ make verify
--snip--
error: postcondition not satisfied
  --> ostd/libs/align_ext/src/lib.rs:186:25
    |
186 |         ret % align == 0,
    |         ^^^^^^^^^^^^^^^^^ failed this postcondition
...
215 |         self & (align - 1)
    |         ----- at the end of the function body
...
222 | / impl_align_ext! {
223 | |     u8,
224 | |     u16,
225 | |     u32,
226 | |     u64,
227 | |     usize,
228 | | }
    | |
    | | in this macro invocation

```



What does this error mean?

The specification says:

```
ensures
  align >= 2,
  is_pow2(align as int),
  ret <= self,
  ret % align == 0,
```

So the result **must be aligned**. But our broken code:

```
fn align_down(self, align: Self) -> Self {
  self & (align - 1)
}
```

- keeps only the low bits
- does **not** always return a multiple of `align`



But writing proof by hand is hard

```

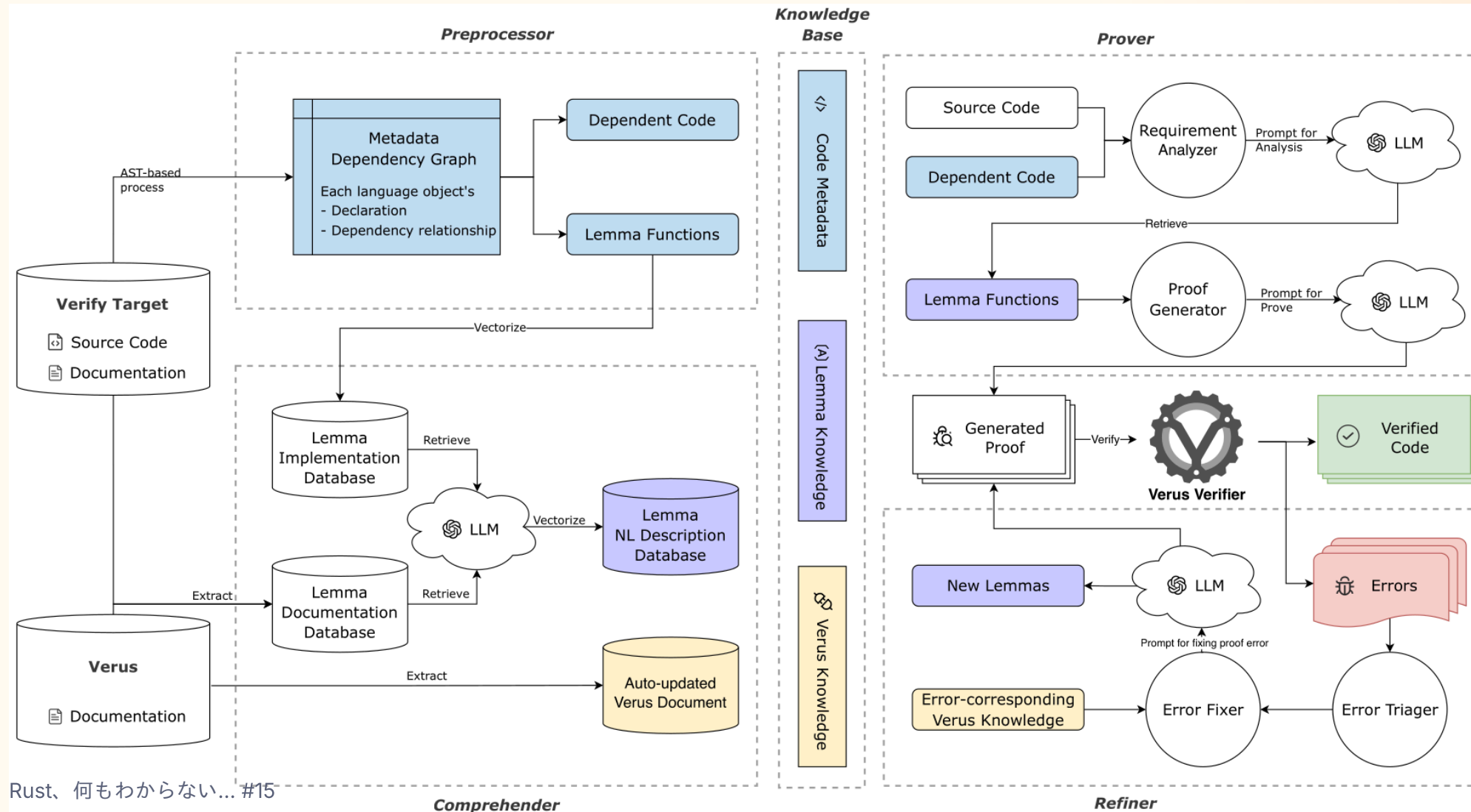
#[inline]
#[verus_spec(ret =>
  requires
    (is_pow2(align as int) && align >= 2) || may_panic(),
  ensures
    align >= 2,
    is_pow2(align as int),
    ret <= self,
    ret % align == 0,
    ret == nat_align_down(self as nat, align as nat),
    forall |n: nat| !(n<=self && #[trigger] (n % align as nat) == 0) || (ret >= n),
)]

fn align_down(self, align: Self) -> Self {
  vstd_extra::assert!(align.is_power_of_two() && align >= 2);
  proof!{
    is_pow2_equiv(align as int);
    lemma_low_bits_mask_values();
    let mask = (align - 1) as Self;
    let e = choose |e: nat| pow(2, e) == align;
    lemma_pow2(e);
    assert(e < $uint_type::BITS) by {
      if e >= $uint_type::BITS {
        lemma_pow2_strictly_increases($uint_type::BITS as nat, e);
        lemma2_to64();
      }
    }
    call_lemma_low_bits_mask_is_mod!($uint_type, self, e);
    assert(self == (self & mask) + (self & !mask)) by (bit_vector);
    assert((self & !mask) as nat == nat_align_down(self as nat, align as nat));
    lemma_nat_align_down_sound(self as nat, align as nat);
  }
  self & !(align - 1)
}

```



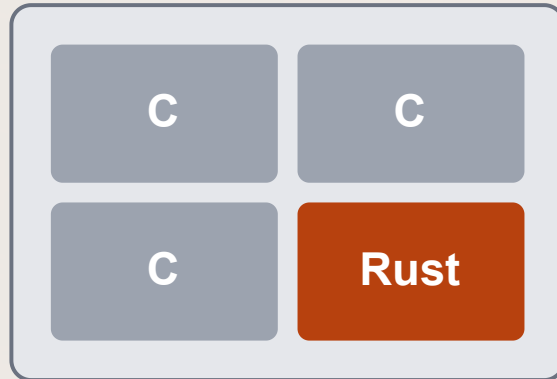
KVerus: LLMs help generate and repair proof



A quick plug: Takibi language

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

Rust for Linux



existing Linux kernel

Replace parts with Rust,
little by little

Asterinas + Verus



Linux-compatible kernel

Designed from scratch in Rust,
partly verified with Verus

Takibi



Linux-compatible kernel

Design an embedded language,
then write the kernel in it

incremental

most radical