Applying the Rust Programming Language in ICS

Adam Crain
Software / Security Engineer

AUTOMATAK

The state of “system” software?

glyph
@glyph

“If you try really hard, you *can* write safe code in C/C++” is the flat-earth movement of software engineering

8:06 PM · Apr 22, 2019 · Twitter Web App

470 Retweets **1.8K** Likes

About Rust

Rust is a high-performance, memory-efficient, programming language. It compiles to machine code, executes without a runtime, and can call and expose a C ABI. It is suitable for embedded systems, yet it also provides powerful abstractions for high-level programming and code reuse.

Unlike other programming languages in its class, Rust's type system and ownership model provide **memory and thread safety**.

Execution model

interpreted

JIT / byte code

machine code

Memory Management

Automatic / GC

"Manual"

deterministic / real-time

Rust Security

Vulnerability		Mitigation
Dangling references (pointers)	✓	**compile-time** (borrow checker)
Use-after-free	✓	**compile-time** (ownership)
NULL pointer dereference	✓	**compile-time** (type system)
Out-of-bounds read/write	✓	**run-time bounds checking** (panic)
Multi-threading	✓	**compile-time** (type system)
Integer overflow/underflow	✓	**run-time*** (panic)

* only in debug build by default

Ownership

```
fn say_hello_by_value(name: String) {
    println!("Hello {}!", name);
}

fn say_hello_by_reference(name: &String) {
    println!("Hello {}!", name);
}

fn main() {
    let name = String::from("Jim"); // Heap allocate a string
    say_hello_by_reference(&name);  // “borrow” the value by immutable reference
    say_hello_by_value(name);       // transfer ownership into function
}
```

Hello Jim!
Hello Jim!

Ownership

```rust
fn main() {
    let name = String::from("Jim");
    say_hello_by_value(name);
    say_hello_by_reference(&name);
}
```

Switched order of these 2 function calls

```
error[E0382]: borrow of moved value: `name`
 --> ownership\src\main.rs:13:28
  |
11 |     let name = String::from("Jim");
  |         ---- move occurs because `name` has type `std::string::String`, which does not implement the `Copy` trait
12 |     say_hello_by_value(name);
  |                        ---- value moved here
13 |     say_hello_by_reference(&name);
  |                            ^^^^^ value borrowed here after move
```

Borrowing

```
struct Value {
    inner : i32
}

fn select_largest<'a>(a: &'a Value, b: &'a Value) -> &'a Value {
    if a.inner > b.inner { a } else { b }
}

fn main() {
    let x = Value { inner: 42 };
    let y = Value { inner: 77 };

    let largest = select_largest(&x, &y);
    println!("Largest is {}", largest.inner);
}
```

Largest is 77!

Borrowing

```
struct Value {
    inner : i32
}

fn select_largest<'a>(a: &'a Value, b: &'a Value) -> &'a Value {
    if a.inner > b.inner { a } else { b }
}

fn print_by_value(value : Value) {
    println!("value is {}", value.inner);
}

fn main() {
    let x = Value { inner: 42 };
    let y = Value { inner: 77 };

    let largest = select_largest(&x, &y);
    print_by_value(x);
    println!("Largest is {}", largest.inner);
}
```

Try to transfer ownership while value is “borrowed”

Borrowing

```
error[E0505]: cannot move out of `x` because it is borrowed
  --> borrowing\src\main.rs:19:20
   |
18 |     let largest = select_largest(&x, &y);
   |                                  -- borrow of `x` occurs here
19 |     print_by_value(x);
   |                    ^ move out of `x` occurs here
20 |     println!("Largest is {}", largest.inner);
   |                               ------------ borrow later used here
```

Concurrency

```
fn main() {
    let counter = std::rc::Rc::new(0u64);

    let mut handles : Vec<std::thread::JoinHandle<()>> = Vec::new();

    for _ in 0..10 {
        let counter = counter.clone();
        let handle = std::thread::spawn(move || {
            *counter += 1;
        });
        handles.push(handle);
    }


    for handle in handles {
        handle.join().unwrap();
    }

    println!("count is {}", *counter);
}
```

Concurrency

```
error[E0277]: `std::rc::Rc<u64>` cannot be sent between threads safely
   --> concurrency\src\main.rs:10:22
    |
10  |         let handle = std::thread::spawn(move || {
    |                      ^^^^^^^^^^^^^^^^^^^^^^^^^^ `std::rc::Rc<u64>` cannot be sent between threads safely
    |
    = help: within `[closure@concurrency\src\main.rs:10:41: 12:10 counter:std::rc::Rc<u64>]`, the trait
`std::marker::Send` is not implemented for `std::rc::Rc<u64>`
```

The Rust type system is *concurrency-aware*!

Concurrency

```
fn main() {
    let counter = std::sync::Arc::new(std::sync::Mutex::new(0u64));

    let mut handles : Vec<std::thread::JoinHandle<()>> = Vec::new();

    for _ in 0..10 {
        let counter = counter.clone();
        let handle = std::thread::spawn(move || {
            let mut num = counter.lock().unwrap();
            *num += 1;
        });
        handles.push(handle);
    }


    for handle in handles {
        handle.join().unwrap();
    }

    println!("count is {}", *counter.lock().unwrap());
}
```

```
count is 10!
```

So why adopt Rust?

"A spoonful of sugar helps the medicine go down" –Mary Poppins

Batteries Included !

- **Cargo -** build system and package manager
- **LLVM –** cross-compile to many different platforms
  - ARM / MIPS / PowerPC / WASM / etc …
  - no_std libs can compile to bare metal for MCUs
- **Rustdoc** – built-in documentation generator

crates.io
Rust Package Registry

Click or press 'S' to search...

Browse All Crates | Docs ▾ | Log in with GitHub

The Rust community's crate registry

Install Cargo

Getting Started

Instantly publish your crates and install them. Use the API to interact and find out more information about available crates. Become a contributor and enhance the site with your work.

1,982,794,029 Downloads

33,644 Crates in stock

New Crates

redarrow (0.8.3)

Most Downloaded

rand

Just Updated

allegro_examples (0.0.38)

Killer language features

- Compile-time and run-time polymorphism using *Traits*
- Rust uses a module system that will liberate you from header files and include guards.
- Rust defaults make sense, and there aren't nearly as many surprises and corner cases (e.g. C/C++ integer conversions)
- Async/await allows you to write synchronous looking code, that compiles down to state machines and events on top of non-blocking I/O.

When is Rust NOT a good fit?

- If you're already using a memory-safe enterprise language such as Java, C#, or Go in a product line, Rust will complicate things.
- Rust has a decent learning curve. While the language is simpler than C++, it is harder to learn and onboard developers than other languages.
- The Rust library ecosystem is still relatively young compared to many others. Library support is improving rapidly …

Rust in ICS NOW?

- If you're starting a greenfield project, consider Rust if the application is embedded, requires high-performance, has real-time constraints, involves networking services, or stability of the application is a MUST.

- If you're supporting or extending a C/C++ codebase, consider leveraging Rust's seamless interop with C to replace high-risk components such as protocol implementations. Write new functionality in Rust and integrate it into your existing codebase as a library that exposes a C API.

Open source surprise

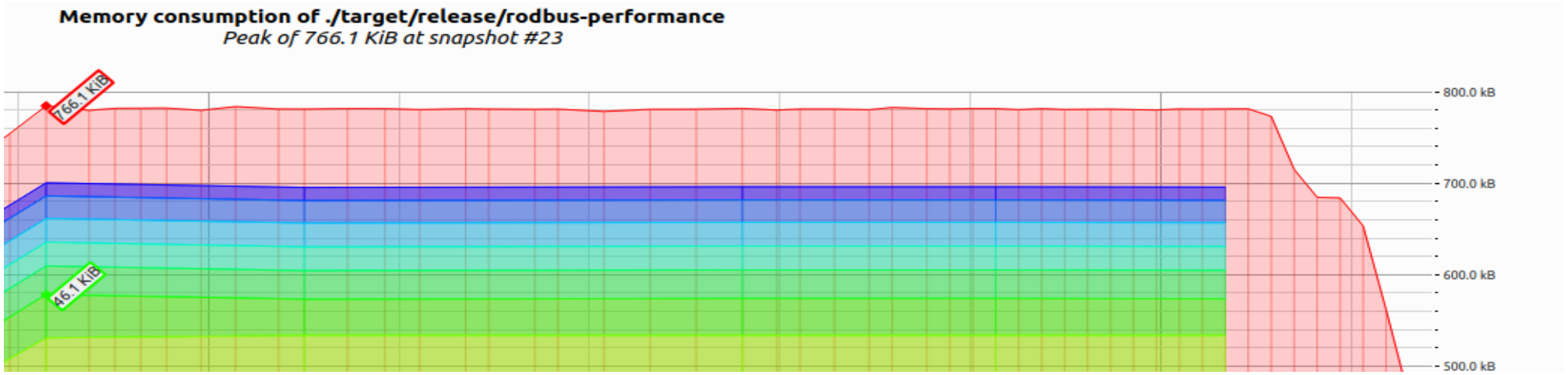
Rodbus!

- Modbus TCP client/server
- Uses async/await, scales for large master deployments
- C API for integration with legacy code

```
// every 5 seconds, perform a blocking read operation
for (int i = 0; i < 3; ++i) {
  Result result = read_coils(&session, 0, COUNT, &values);
  switch (result.status) {
  case (STATUS_OK): {
    printf("success!\n");
    for (uintptr_t i = 0; i < COUNT; ++i) {
      printf("value: %d\n", values[i]);
    }
    break;
  }
  case (STATUS_EXCEPTION):
    printf("Modbus exception: %d\n", result.exception);
    break;
  default:
    printf("error: %d \n", result.status);
    break;
  }
  sleep(5);
}
```

https://crates.io/crates/rodbus

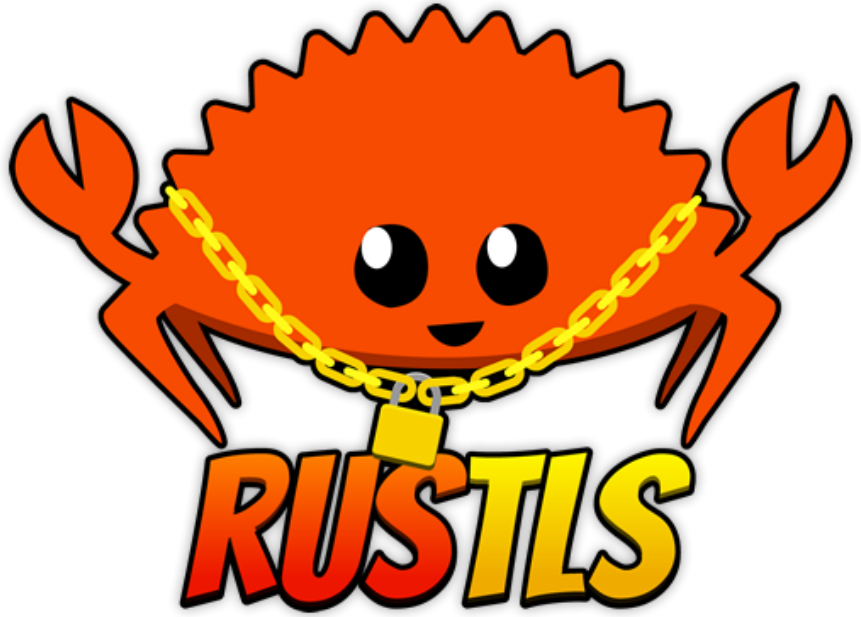
Benchmarks



- > 200,000 requests/sec using multi-core scheduler on developer workstation
- **100 concurrent TCP sessions** in **~800KB RAM** using async/await and the Tokio scheduler

Coming soon to Rodbus!

- Secure Modbus using RUSTLS
- Expose server and TLS APIs to C
- Bind the C API into other languages like Java, .NET, etc



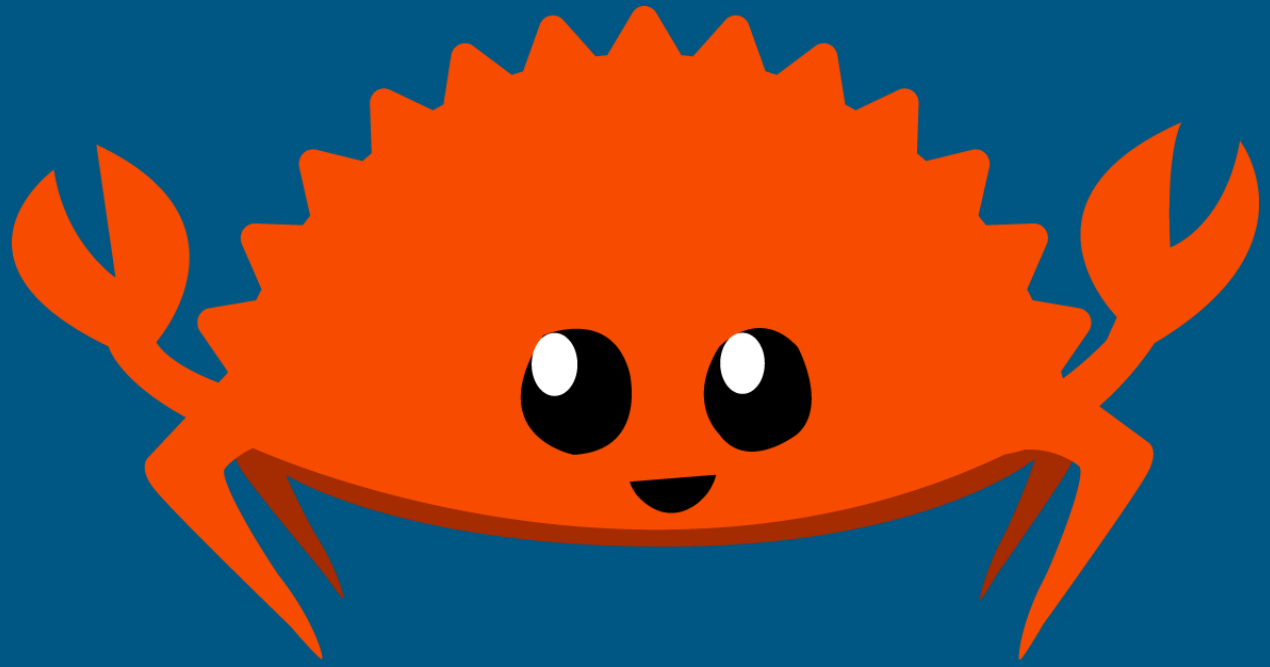
https://jbp.io/2019/07/01/rustls-vs-openssl-performance.html

- 15% quicker to send data.
- 5% quicker to receive data.
- 20-40% quicker to set up a client connection.
- 10% quicker to set up a server connection.
- 30-70% quicker to resume a client connection.
- 10-20% quicker to resume a server connection.
- uses **less than half** the memory of OpenSSL.

Summary

- Rust is a modern competitor to C/C++, providing advances in productivity, stability, and security without sacrificing control, determinism, or raw performance.
- The Rust ecosystem is “batteries included”, with a wealth of helpful tools and libraries.
- Rust doesn’t have to “replace” C/C++. Rust has seamless interop with C at the ABI level.

Questions?



AUTOMATAK