WHY BOTHER AND LEARN RUST?

Periklis Tsirakidis

SHORT INTRODUCTION TO RUST LANG

- Systems Programming Language since 2004
- 1.0 since mid 2015, currently 1.18
- Promise I: Memory leak free programming
- Promise II: Safe multi-threading programming
- Promise III: Zero Cost Abstractions
- Promise I + II + III = Great Craftmanship

HOW RUST FULFILLS ITS PROMISES?

- Simple rules for pointer aliasing and data mutation
- Rules enforced through a sophisticated type system
- Front and mid end compiler support on top of LLVM
- Strong focus on programming language ergonomics
- Long research project history 2004 - 2008

LET'S DIVE IN RUST

MEMORY SAFETY I

No null pointer dereferences

NULL POINTER IN C/C++: A BILLION DOLLAR MISTAKE

```
obj* func_and_alloc() {...}

int main(int argc, char* argv[]) {
    obj* p = func_and_alloc();

    if (p != null) {
      // Do conditional stuff
    }

    // Do further stuff

    free(p); // Upps!!!
}
```

IN RUST WE LOVE RAII AND OPTION<T>

```
fn func() -> Option<T> {...}

fn main () {
    match func() {
        Some(t) => { // do something with t },
        None => { // do something with nothng }
    } // Deallocate Option and T here
}
```

MEMORY SAFETY II

No dangling pointers

C/C++: DANGLING POINTER STRIKES BACK

```
void borrow_to_friend(char* buf) {
    // process buffer
    free(buf);
}

int main(int argc, char* argv[]) {
    borrow_to_friend(arg[0]);

    // arg[0] now dangling
}
```

RUST I: MOVE ME, CLONE ME, DROP ME

```
fn give_to_friend(book: String) { } // Drop book

fn main {
    // Move temporary to str
    let book = String::from("Learn Rust in 21 days");

    // Assignment moves not copy
    let same_book = book;

    // Clone other_book into other_book
    let other_book = same_book.clone();

    // Move str1 to arg str
    give_to_friend(same_book);
} // Drop other_book, empty same_book and empty book
```

RUST II: BORROW ME A REFERENCE

```
// str is string slice type: &[]
fn borrow_to_a_friend(str: str) {
    borrow_to_friends_friend(&str.as_bytes());
}

fn borrow_to_a_friends_friend(buf: &[u8]){}

fn main() {
    // Move temporary to book_title
    let book_title = String::from("Alice in Wonderland");

    // Borrow a string slice reference here
    borrow_to_a_friend(&book_title);
}
```

RUST III: TAKE ME MUTABLE BUT NOBODY ELSE

```
fn add_apocalypse_date(str: mut str) {}

fn main() {
    let book_title = String::from("Apocalypse");

    add_apocalypse_date(&mut book_title);

    // ERROR
    let other_apocalypse = &mut book_title;
}
```

MEMORY SAFETY III

No buffer overruns

C/C++: PROGRAMMERS OVERRUN BUFFERS

```
void check_str(char* buf, int len) {
    for(int i = 0; i <= len; i++) {
        // buf[i]
        // Programmer overruns the buffer on i = len
    }
}

int main(int arc, char* argv[]) {
    char* book = "Learn Overruns in 21 Days";
    char* other = "Smash me in 21 Days";

    // Overrun book char buffer
    check_str(book, 25);
}
```

RUST: TAKE A SLICE LEAVE THE REST

```
fn check_str(str: str) {}

fn main() {
    let book = String::from("Learn Slicing in 21 Days");

    check_str(&[0..4]);
}
```

SAFE MULTI-THREADING BY DEFAULT

USE THE POWER OF ALL YOUR CORES AND SLEEP WELL

- Data races are another form of aliasing vs. mutation
- Simple rules:
  - Either you move contents into a thread
  - Or you borrow any immutable references
  - Or you borrow only once a mutable reference
- Let the compiler check the rules FTW
- MPSC: Share messages not data
- Mutex< T >, Arc< T >

WHY NOT ONLY FOR C/C++ DEVELOPERS?

- Wake up: We hit the end of Moore's Law by 2005
- Your PHP, Go, Ruby, etc. is as memory leak free as their impls/VMs/GCs
- Your PHP, Go, Ruby, etc. works on a costly abstraction above the OS/machine
- Use the power of a great cross-plattform std library and ecosystem

WHY NOT ONLY FOR C/C++ DEVELOPERS?

- Concurrent and parallel programming styles are going to stay, e.g.
  - Actors based concurrency
  - Futures/Tasks/Executors based concurrency
  - Map and reduce style data parallelism
  - SIMD
  - Offloading to GPU cores
- Rust enables productive writing of memory safe and concurrent/parallel applications

WHY IMPORTANT FOR WEB ENGINEERING?

- Rust enables the same productivity thanks to Cargo, Crates, Rustup, Rustc
- Targets futures/async/await like Scala/Finagle (Currently on Nightly only, est. release end 2017)
- Support for cross compilation targets: MSVC, IOS, ARM (IoT), WebAssembly & more

RUST KEY FACTS

- Express Ownership explicitly!
- No dangling but borrowing down the stack!
- No overruns but trust your slice!
- No Exceptions but Result< T >!
- No null but Option< T >!
- No garbage collection but Box, Rc, Arc, RefCell!
- No green threads but sane standard library support!

SO LONG AND THANKS FOR THE FISH!

Periklis Tsirakidis

Github: github.com/periklis

FURTHER READING

- Jim Blady - Why Rust?
- Aaron Turon - Standford Seminar on Rust Lang
- Repository - Rust-Learning
- Rust by Example
- Official Rust Land Documentation
- Official Cargo Guide
- Official Rust Lang Reference
- RustBelt: Securing the Foundations of the Rust Programming Language (PDF)