Rust, TEE, and OS (2)

Mingshen

Recap

- ***What is TEE?***
  - Security properties: data confidentiality, data integrity, code integrity, etc.
  - TEE implementations
  - Applications of TEE
- ***Why need an OS in TEE?***
  - TEE Development Model
  - Security vs Usability
- ***Does language matter?***
  - Memory safety
  - How does Rust guarantee the memory safety?
  - Why use Rust for TEE development?

Homework

- Writing an OS in Rust: bare bones and interrupts
- Run sample code
- Rewrite on RISC-V
- Writing an OS in Rust: https://os.phil-opp.com
- The Adventures of OS: Making a RISC-V Operating System using Rust: http://osblog.stephenmarz.com/

Homework

rustOS链接提交 最近保存 21:35

插入 常规 .0 默认字体 10

D16

	A	B	C
1	姓名	github链接	备注
2	邹泽桦	https://github.com/HuaHuaY/Rust_TEE_OS	
3	鲍志远	https://github.com/bzy-debug/blog_os	
4	彭宇科	https://github.com/sdww0/rustOS	
5	周翊澄	https://github.com/Zhou-Yicheng/rust_os	
6	张开源	https://github.com/imporoutco586/RUST-OS	
7	孙永康	https://github.com/Konata-CG/Rust_TEE-OS	

Most of the people has done the first two tasks.

Sadly, I didn't see the RISC-V rewrite.

Homework

The Adventures of OS: Making a RISC-V Operating System using Rust

Running title: RISC-V OS using Rust

26 September 2019

Purpose

RISC-V ("risk five") and the Rust programming language both start with an R, so naturally they fit together. In this blog, we will write an operating system targeting the RISC-V architecture in Rust (mostly). If you have a sane development environment for RISC-V, you can skip the setup parts right to bootloading. Otherwise, it'll be fairly difficult to get started.

This tutorial will progressively build an operating system from start to something that you can show your friends or parents -- if they're significantly young enough. Since I'm rather new at this I decided to make it a "feature" that each blog post will mature as time goes on. More details will be added and some will be clarified. I look forward to hearing from you!

The Road Ahead...

+ Chapter 0: Setup and pre-requisites (UPDATED 2020: Rust out-of-the-box!)
+ Chapter 1: Taking control of RISC-V
+ Chapter 2: Communications
+ Chapter 3.1: Page-grained memory allocation
+ Chapter 3.2: Memory Management Unit
+ Chapter 4: Handling interrupts and traps
+ Chapter 5: External interrupts
+ Chapter 6: Process memory
+ Chapter 7: System calls
+ Chapter 8: Starting a process
+ Chapter 9: Block driver
+ Chapter 10: Filesystems
+ Chapter 11: Userspace Processes

Next week
- Read the first three chapters.
- Implement by your self.

Two weeks later
- Come back to the original homework to rewrite into RISC-V.

Rust's Ownership, Borrowing, and Lifetime

Ownership and Borrowing

- In Rust, every value has a **single, statically-known, owning path** in the code, at any time.

- Pointers to values have limited duration, known as a "**lifetime**", that is also **statically tracked**.

- All pointers to all values are known **statically**.

Ownership

Alice


```
fn main() {
    let alice = vec![1, 2, 3];;
    {
        let bob = alice;
        println!("bob: {}", bob[0]);
    }
    println!("alice: {}", alice[0]);
}
```

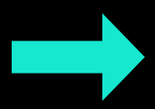
Ownership (T)

Alice **Bob**


```
fn main() {
    let alice = vec![1, 2, 3];;
    {
        let bob = alice;
        println!("bob: {}", bob[0]);
    }
    println!("alice: {}", alice[0]);
}
```

Ownership (T)

Alice **Bob**



```
fn main() {
    let alice = vec![1, 2, 3];;
    {
        let bob = alice;
        println!("bob: {}", bob[0]);
    }
    println!("alice: {}", alice[0]);
}
```

Ownership (T)

Alice


```
fn main() {
    let alice = vec![1, 2, 3];;
    {
        let bob = alice;
        println!("bob: {}", bob[0]);
    }
    println!("alice: {}", alice[0]);
}
```

Ownership (T)

Alice

```
error[E0382]: use of moved value: `alice`
 --> src/main.rs:7:27
  |
4 |         let bob = alice;
  |             --- value moved here
...
7 |     println!("alice: {}", alice[0]);
  |                           ^^^^^ value used here after move
  |
  = note: move occurs because `alice` has type
`std::vec::Vec<i32>`, which does not implement the `Copy` trait
```

```
fn main() {
    let alice = vec![1, 2, 3];;
    {
        let bob = alice;
        println!("bob: {}", bob[0]);
    }
    println!("alice: {}", alice[0]);
}
```

Ownership (T)

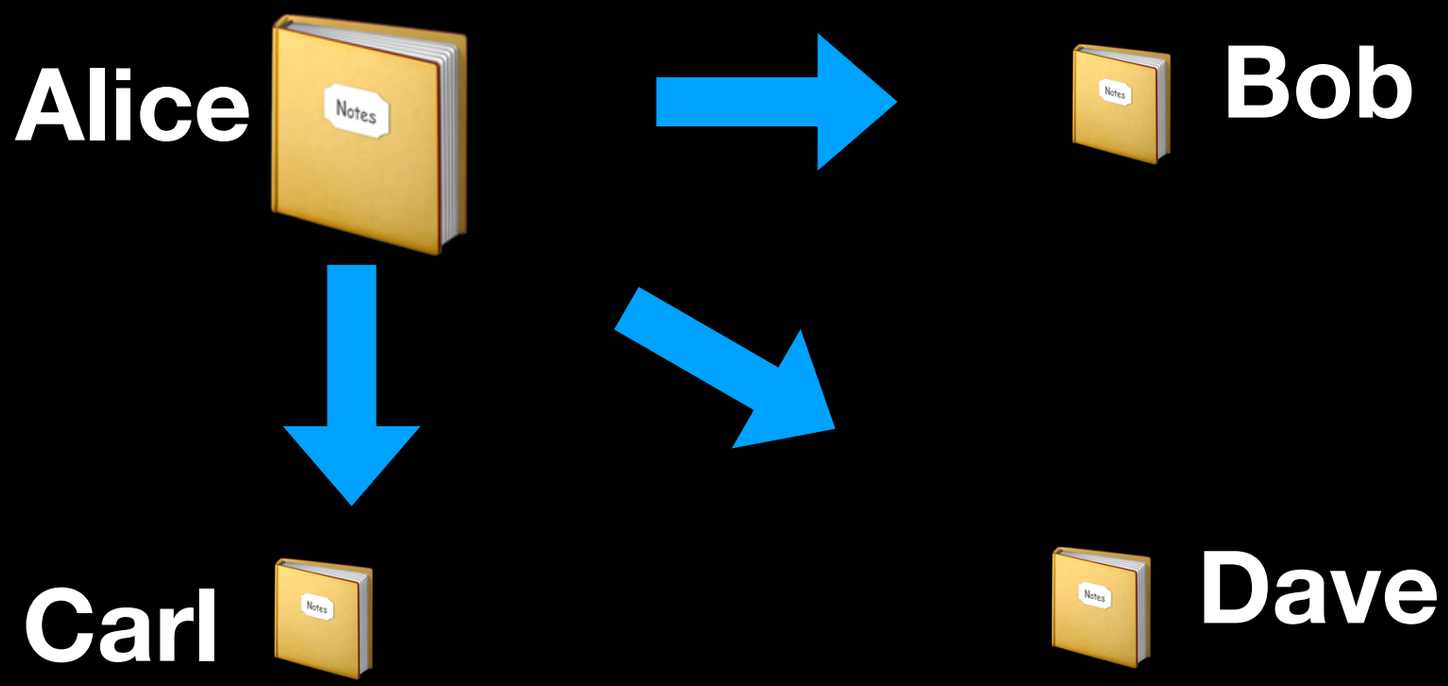
Alice

```
error[E0382]: use of moved value: `alice`
 --> src/main.rs:7:27
  |
4 |         let bob = alice;
  |             --- value moved here
...
7 |     println!("alice: {}", alice[0]);
  |                           ^^^^^ value used here after move
  |
  = note: move occurs because `alice` has type
`std::vec::Vec<i32>`, which does not implement the `Copy` trait
```

```
fn main() {
    let mut alice = vec![1, 2, 3];;
    {
        let mut bob = alice;
        println!("bob: {}", bob[0]);
    }
    println!("alice: {}", alice[0]);
}
```

Shared Borrow (&T)

Aliasing + ~~Mutation~~



Mutable Borrow (&mut T)

Alice


```
fn main() {
    let mut alice = 1;
    {
        let bob = &mut alice;
        *bob = 2;
        println!("bob: {}", bob);
    }
    println!("alice: {}", alice);
}
```

Mutable Borrow (&mut T)

Alice **Bob**


```
fn main() {
    let mut alice = 1;
    {
        let bob = &mut alice;
        *bob = 2;
        println!("bob: {}", bob);
    }
    println!("alice: {}", alice);
}
```

Mutable Borrow (&mut T)

Alice


```
fn main() {
    let mut alice = 1;
    {
        let bob = &mut alice;
        *bob = 2;
        println!("bob: {}", bob);
    }
    println!("alice: {}", alice);
}
```

Mutable Borrow (&mut T)

Alice


```
fn main() {
    let mut alice = 1;
    {
        let bob = &mut alice;
        *bob = 2;
        println!("bob: {}", bob);
    }
    println!("alice: {}", alice);
}
```

Mutable Borrow (&mut T)

~~Aliasing~~ + Mutation

Alice

The lifetime of a borrowed reference should end before the lifetime of the owner object does.

Rust's Ownership & Borrowing

~~Aliasing~~ + ***~~Mutation~~***

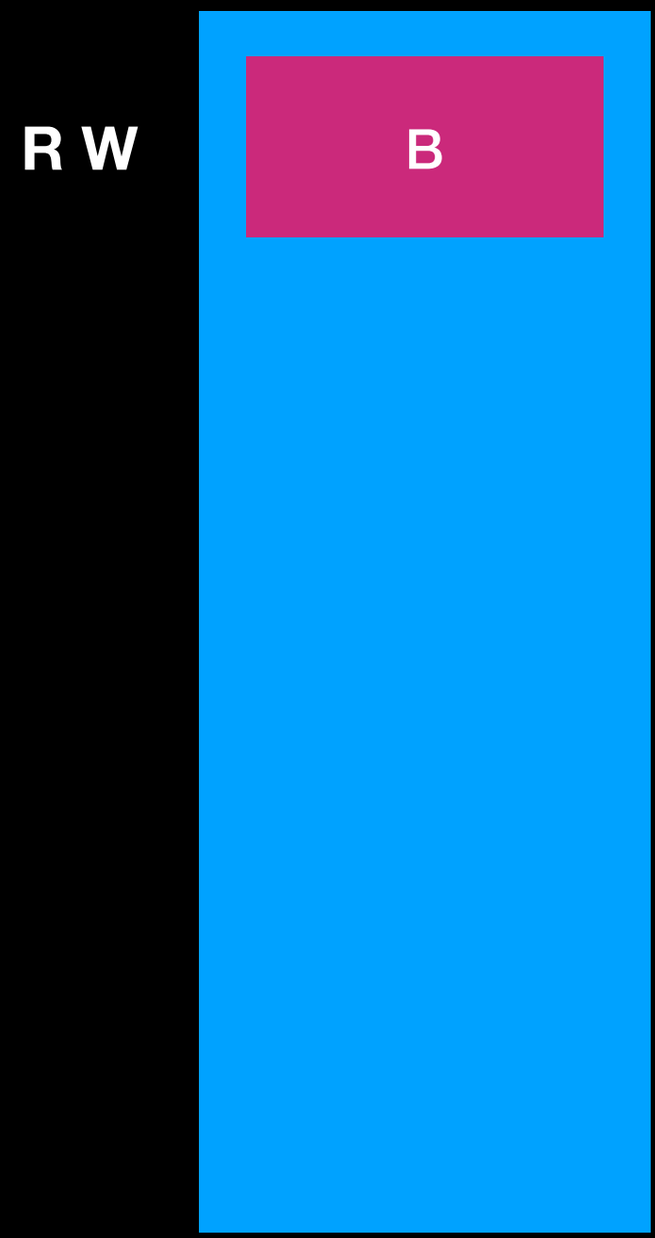
- Compiler enforced:
  - Every resource has a unique **owner**
  - Others can **borrow** the resource from its owner (e.g., create an **alias**) with restrictions
  - Owner **cannot** free or mutate its resource while it is borrowed

Ownership & Borrowing

Owership	`T`	"owned"
Exclusive access	`&mut T`	"mutable"
Shared access	`&T`	"read-only"

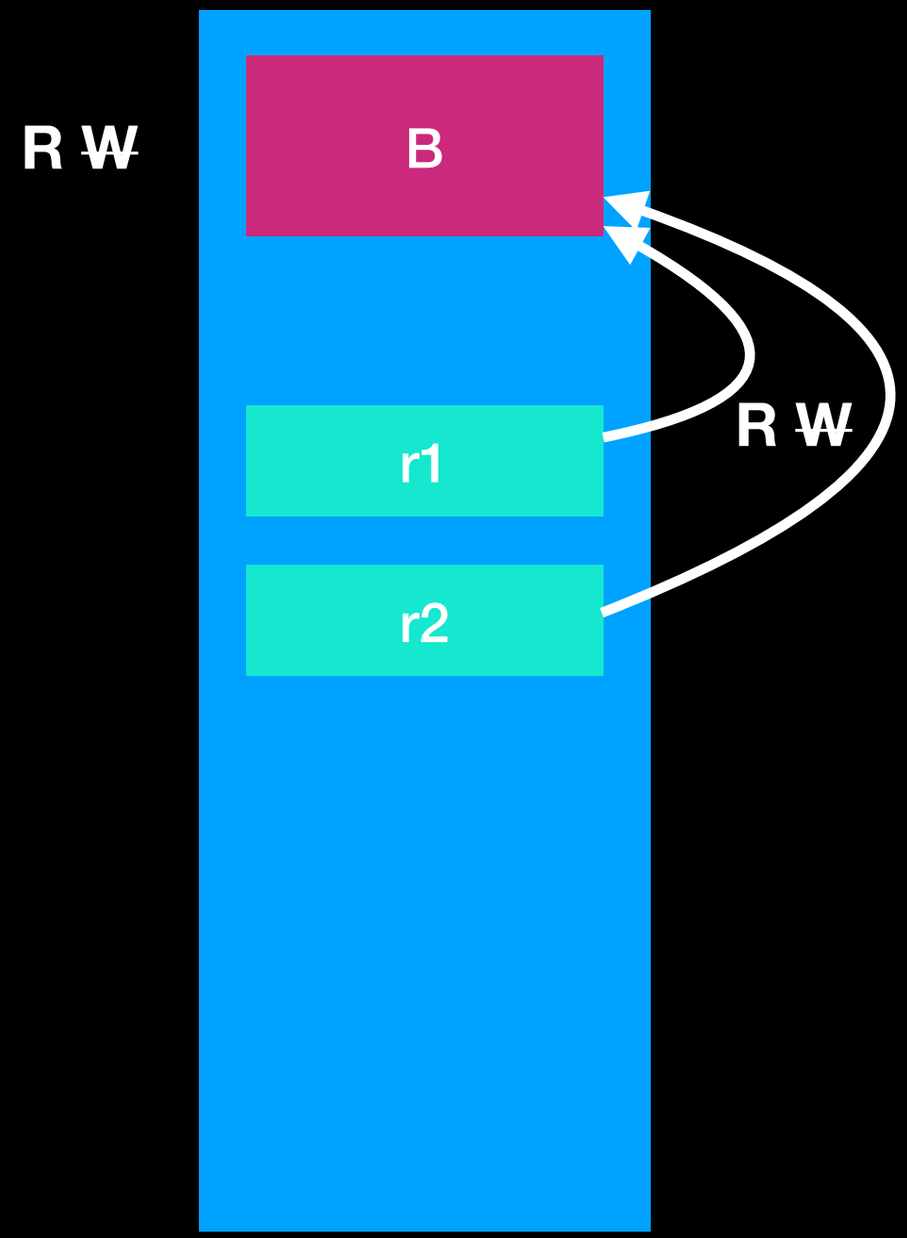
Stack allocation

```
let b = B::new();
```



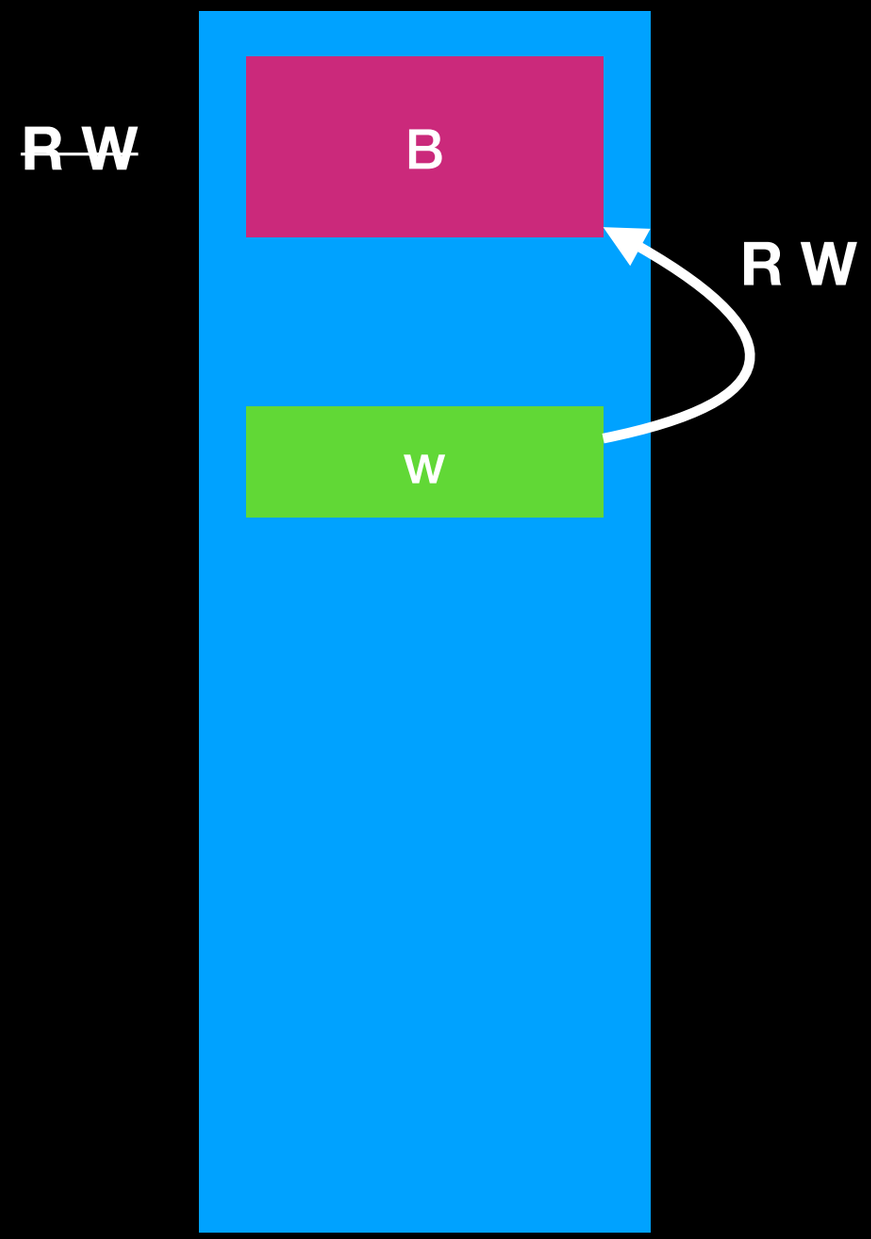
Stack allocation

```
let b = B::new();

let r1: &B = &b;
let r2: &B = &b;

// stack allocation and
// immutable borrows, b has
// lost write capability
```



Stack allocation

```
let b = B::new();

let w: &mut B = &mut b;

// stack allocation and mutable
borrows, b has temporarily lost
both read and write
capabilities
```

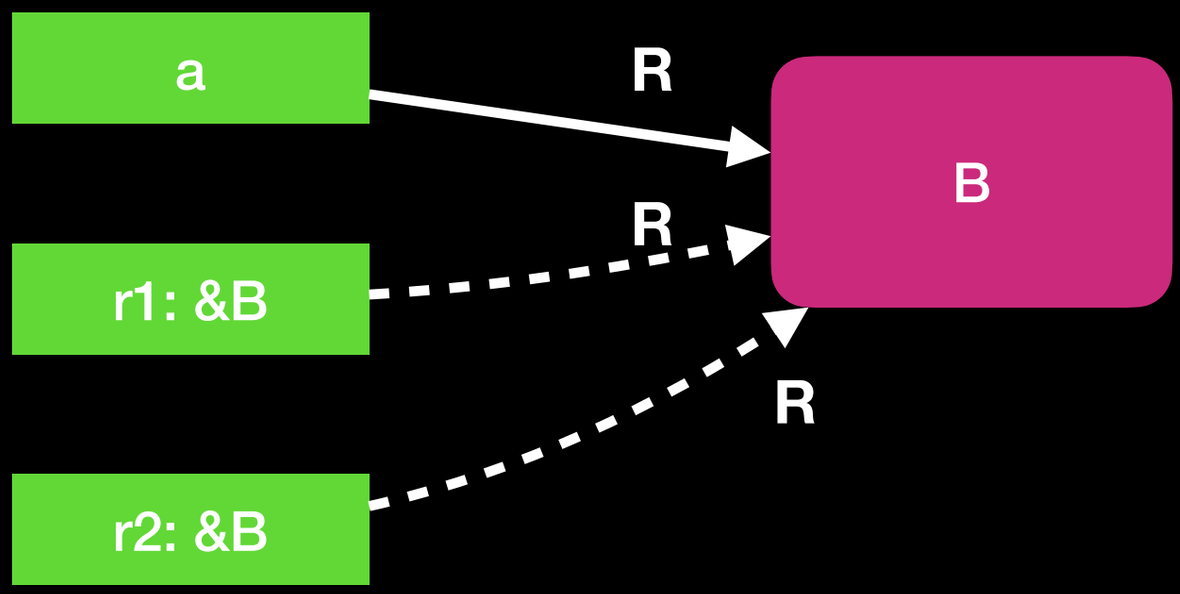


Heap allocation

```
let a = Box::new(B::new());

// Boxed B, a (as owner) has both read and write
capabilities.
```

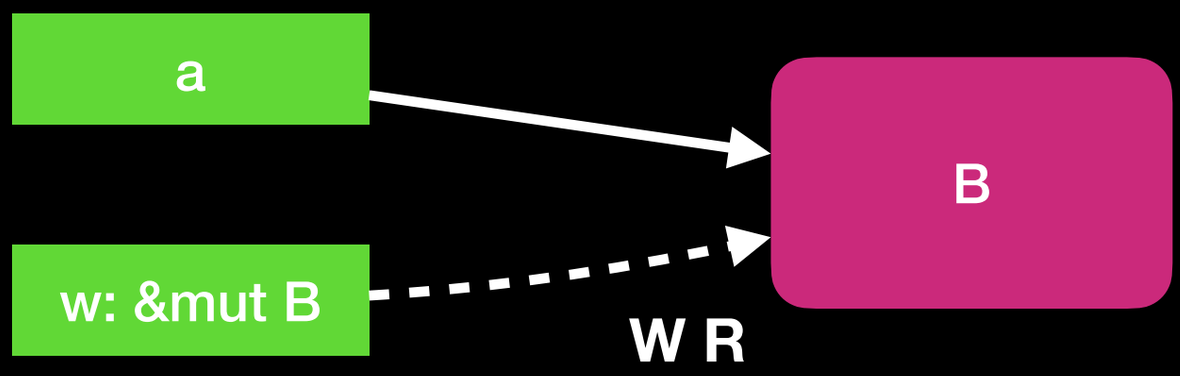
Immutably borrowing a box

```
let a = Box::new(B::new());
let r_of_box: &Box<B> = &a; // not directly a ref of B

let r1: &B = &*a;
let r2: &B = &a;  // <-- coercion!

// immutable borrows of heap-allocated B, a retains
read capabilities (has temporarily lost write)
```



Mutably borrowing a box

```
let a = Box::new(B::new());
let r_of_box: &Box<B> = &a; // not directory a ref of B

let w: &mut B = &mut a; // (again, coercion here)

// mutable borrow of heap-allocated B, a has
temporarily lost both read and write capabilities
```



Lifetime

```
{
    let r;

    {
        let x = 5;
        r = &x;
    }

    println!("r: {}", r);
}
```

Lifetime

```
error[E0597]: `x` does not live long enough
  --> src/main.rs:7:5
   |
6  |           r = &x;
   |                 - borrow occurs here
7  |       }
   |       ^ `x` dropped here while still borrowed
...
10 | }
   | - borrowed value needs to live until here
```

Borrow Checker

```
{
    let r;                // ---------+-- 'a
                          //          |
    {                     //          |
        let x = 5;        // -+-- 'b  |
        r = &x;           //  |       |
    }                     // -+       |
                          //          |
    println!("r: {}", r); //          |
}                         // ---------+
```

Borrow Checker

```
{
    let x = 5;                  // ----------+-- 'b
                                //           |
    let r = &x;                 // --+-- 'a  |
                                //   |       |
    println!("r: {}", r);       //   |       |
                                // --+       |
}                               // ----------+
```

Lifetime in Functions

```
fn main() {
    let string1 = String::from("abcd");
    let string2 = "xyz";

    let result = longest(string1.as_str(), string2);
    println!("The longest string is {}", result);
}

fn longest(x: &str, y: &str) -> &str {
    if x.len() > y.len() {
        x
    } else {
        y
    }
}
```

Lifetime in Functions

```
error[E0106]: missing lifetime specifier
 --> src/main.rs:1:33
  |
1 | fn longest(x: &str, y: &str) -> &str {
  |                                 ^ expected lifetime
parameter
  |
  = help: this function's return type contains a
borrowed value, but the
signature does not say whether it is borrowed from `x`
or `y`
```

Lifetime in Functions

```
fn main() {
    let string1 = String::from("abcd");
    let string2 = "xyz";

    let result = longest(string1.as_str(), string2);
    println!("The longest string is {}", result);
}

fn longest(x: &str, y: &str) -> &str {
    if x.len() > y.len() {
        x
    } else {
        y
    }
}
```

Lifetime in Functions

```
fn main() {
    let string1 = String::from("abcd");
    let string2 = "xyz";

    let result = longest(string1.as_str(), string2);
    println!("The longest string is {}", result);
}

fn longest<'a>(x: &'a str, y: &'a str) -> &'a str {
    if x.len() > y.len() {
        x
    } else {
        y
    }
}
```

Lifetime in Functions

```
fn main() {
    let string1 = String::from("long string is long");

    {
        let string2 = String::from("xyz");
        let result = longest(string1.as_str(), string2.as_str());
        println!("The longest string is {}", result);
    }
}


fn longest<'a>(x: &'a str, y: &'a str) -> &'a str {
    if x.len() > y.len() {
        x
    } else {
        y
    }
}
```

Lifetime in Functions

```
fn main() {
    let string1 = String::from("long string is long");
    let result;
    {
        let string2 = String::from("xyz");
        result = longest(string1.as_str(), string2.as_str());
    }
    println!("The longest string is {}", result);
}


fn longest<'a>(x: &'a str, y: &'a str) -> &'a str {
    if x.len() > y.len() {
        x
    } else {
        y
    }
}
```

Lifetime in Functions

```
error[E0597]: `string2` does not live long enough
  --> src/main.rs:15:5
   |
14 |         result = longest(string1.as_str(), string2.as_str());
   |                                             ------- borrow
occurs here
15 |     }
   |     ^ `string2` dropped here while still borrowed
16 |     println!("The longest string is {}", result);
17 | }
   | - borrowed value needs to live until here
```

Use-After Free in C/Rust

C/C++

```
void func() {
    int *used_after_free = malloc(sizeof(int));

    free(used_after_free);

    printf("%d", *used_after_free);
}
```

Rust

```
fn main() {
    let name = String::from("Hello World");
    let mut name_ref = &name;
    {
        let new_name = String::from("Goodbye");
        name_ref = &new_name;
    }
    println!("name is {}", &name_ref);
}
```

Use-After Free in Rust

```
error[E0597]: `new_name` does not live long enough
 --> main.rs:7:5
  |
6 |         name_ref = &new_name;
  |                    --------- borrow occurs here
7 |     }
  |     ^ `new_name` dropped here while still borrowed
8 |     println!("name is {}", &name_ref);
9 | }
  | - borrowed value needs to live until here

error: aborting due to previous error
```

Rust Syntax

Rust Syntax

- Rust has a C-style syntax with influences from functional languages.

- Specific functionality will be covered later.

Basic

```
// Function declaration
fn add_them(first: i32, second: i32) -> i32 {
    first + second
}

fn main() {
    // Mutable variable
    let mut some_value = 1;
    // Immutable, explict type
    let explicitly_typed: i32 = 1;

    // Function call
    some_value = add_them(some_value, explicitly_typed);

    // Macro, note the !
    println!("{}", some_value)
}
```

if

```
fn main() {
    let value = 2;

    if value % 2 == 0 {
        // ...
    } else if value == 5 {
        // ...
    } else { /* ... */ }
}
```

match

```
fn main() {
    let maybe_value = Some(2);
    match maybe_value {
        Some(value) if value == 2 => {
            // ...
        }
        Some(value) => {
            // ...
        },
        None => {
            // ...
        },
    }
}
```

if let

```
fn main() {
    let maybe_value = Some(2);

    if let Some(value) = maybe_value {
        // ...
    } else { /* ... */ }
}
```

loop and while

```
fn main() {
    let mut value = 0;
    // Loop with break
    loop {
        if value >= 10 {
            break;
        }
        value += 1;
    }
    // Break on conditional
    while value <= 10 {
        value += 1;
        // ...
    }
}
```

for and while let

```
fn main() {
    // Loop over iterator
    let range = 0..10;
    for i in range {
        // ...
    }
    // while let
    let mut range = 0..10;
    while let Some(v) = range.next() {
        // ...
    }
}
```

struct, type, and enum

```
struct Empty;

struct WithFields {
    foo: i32,
    bar: Choice,
}

type Explanation = String;

enum Choice {
    Yes,
    No,
    Maybe(Explanation),
}

fn main() {}
```

impl and trait

```
trait Bar {
    // This can be overridden
    fn default_implementation(&self) -> bool {
        true
    }
    fn required_implementation(&self);
}

impl Bar for Foo {
    fn required_implementation(&self) {
        // ...
    }
}

impl Foo {
    fn new() -> Self { Foo }
}
```

Borrowing

```
// &mut denotes a mutable borrow
fn accepts_borrow(thing: &mut u32) {
    *thing += 1
}

fn main() {
    let mut value = 1;
    accepts_borrow(&mut value);
    println!("{}", value)
}
```

Lifetimes

```
fn with_lifetimes<'a>(thing: &'a str) -> &'a str {
    thing
}

fn main() {
    let foo = "foo";
    println!("{}", with_lifetimes(foo))
}
```

Scopes: Rust is block scoped. Scopes can return values.

```
fn main() {
    let foo = 1;
    let bar = {
        // Shadows earlier declaration.
        let foo = 2;
        foo
    };
    println!("{}", foo);
    println!("{}", bar);
}
```

Closures

```
fn main() {
    // Shorthand
    let value = Some(1).map(|v| v + 1);
    // With a block
    let value = Some(1).map(|v| {
        v + 1
    });
    // Explict return type
    let value = Some(1).map(|v| -> i32 {
        v + 1
    });
    // Declared
    let closure = |v| v + 1;
    let value = Some(1).map(closure);
}
```

Generics

```
// Inline syntax
fn generic_inline<S: AsRef<str>>(thing: S) -> S {
    thing
}

// Where syntax
fn generic_where<Stringish>(thing: Stringish) -> Stringish
where Stringish: AsRef<str> {
    thing
}

// Enums too!
struct GenericStruct<A> {
    value: A,
}

fn main() {
    let foo = "foo";
    generic_inline(foo);
    generic_where(foo);
}
```

use and mod

```
use foo::foo;

mod foo {
    pub fn foo() {
        // ...
    }
}

// Will try to open `./bar.rs` relative to this file.
pub mod bar;

fn main() {
    foo()
}
```

Attributes

- Rust attributes are used for a number of different things. There is a full list of attributes in the reference.

```
#[derive(Clone, Copy)]
struct Foo;

#[inline(always)]
fn bar() {}

fn main() {}
```

Attributes

- Comparison traits: Eq, PartialEq, Ord, PartialOrd
- Clone, to create T from &T via a copy.
- Copy, to give a type 'copy semantics' instead of 'move semantics'
- Hash, to compute a hash from &T.
- Default, to create an empty instance of a data type.
- Debug, to format a value using the {:?} formatter.

Error Handling

- Rust groups errors into two major categories:
  - recoverable -> `Result<T, E>`
  - unrecoverable errors -> `panic!`

Unrecoverable Errors with `panic!`

- print a failure message
- unwind and clean up the stack, and then quit
- occurs when a bug of some kind has been detected and it's not clear to the programmer how to handle the error.

```
fn main() {
    panic!("crash and burn");
}

$ cargo run
   Compiling panic v0.1.0 (file:///projects/panic)
    Finished dev [unoptimized + debuginfo] target(s) in 0.25 secs
     Running `target/debug/panic`
thread 'main' panicked at 'crash and burn', src/main.rs:2:4
note: Run with `RUST_BACKTRACE=1` for a backtrace.
```

Recoverable Errors with `Result<T, E>`

```
enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

Recoverable Errors with `Result<T, E>`

```
enum Result<T, E> {
    Ok(T),
    Err(E),
}

fn foo() -> Result<usize, std::io::Error>

match foo() {
    Ok(size) => println!("size: {}", size);
    Err(e) => panic!("panic: {:?}", e);
}
```

Recoverable Errors with `Result<T, E>`

```
enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

```
std::io::Stdin

pub fn read_line
    (&self, buf: &mut String) -> Result<usize>

type std::io::Result<T> =
    std::result::Result<T, std::io::Error>;
```

Recoverable Errors with `Result<T, E>`

```
use std::io;

fn get_string() -> io::Result<String> {
    let mut buffer = String::new();

    match io::stdin().read_line(&mut buffer) {
        Ok(_) => {},
        Err(e) => return Err(e)
    }

    Ok(buffer)
}
```

Recoverable Errors with `Result<T, E>`

```
use std::io;

fn get_string() -> io::Result<String> {
    let mut buffer = String::new();

    io::stdin().read_line(&mut buffer)?;

    Ok(buffer)
}
```

Getting Started

- Installation: https://rustup.rs/

rustup is an installer for
the systems programming language **Rust**

Run the following in your terminal, then follow
the onscreen instructions.

```
curl https://sh.rustup.rs -sSf | sh
```

You appear to be running Unix. If not, display all supported installers.

Need help? Ask on #rust-beginners.

rustup is an official Rust project.

other installation options · about rustup

Hello, World!

```
fn main() {
    println!("Hello, world!");
}
```

```
$ rustc main.rs
$ ./main
Hello, world!
```

Cargo

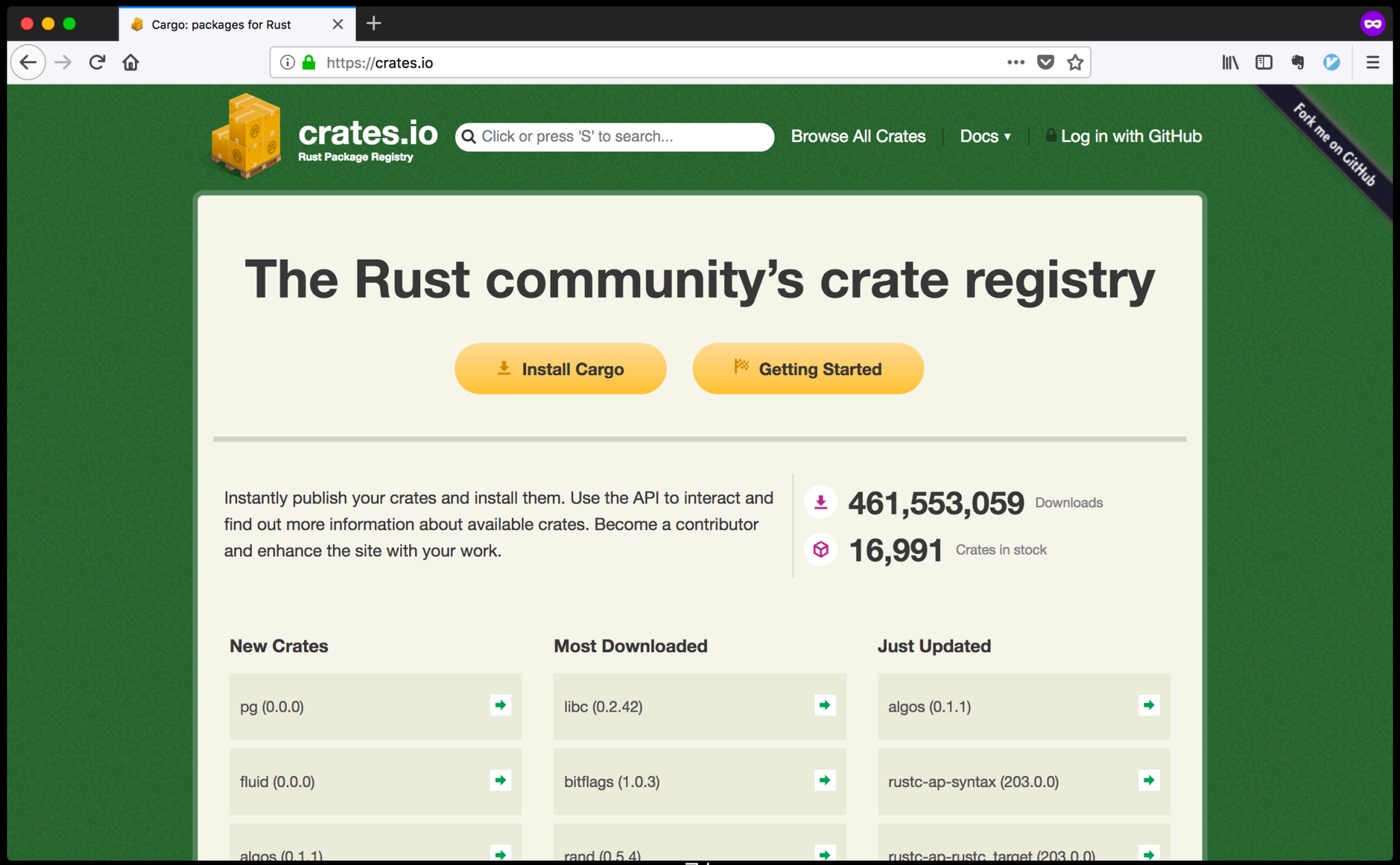
Cargo

Cargo is the Rust package manager. Cargo downloads your Rust project’s dependencies, compiles your project, makes packages, and upload them to crates.io, the Rust community’s package registry.

Cargo

- cargo new
- cargo build
- cargo run
- cargo XXX

crates.io

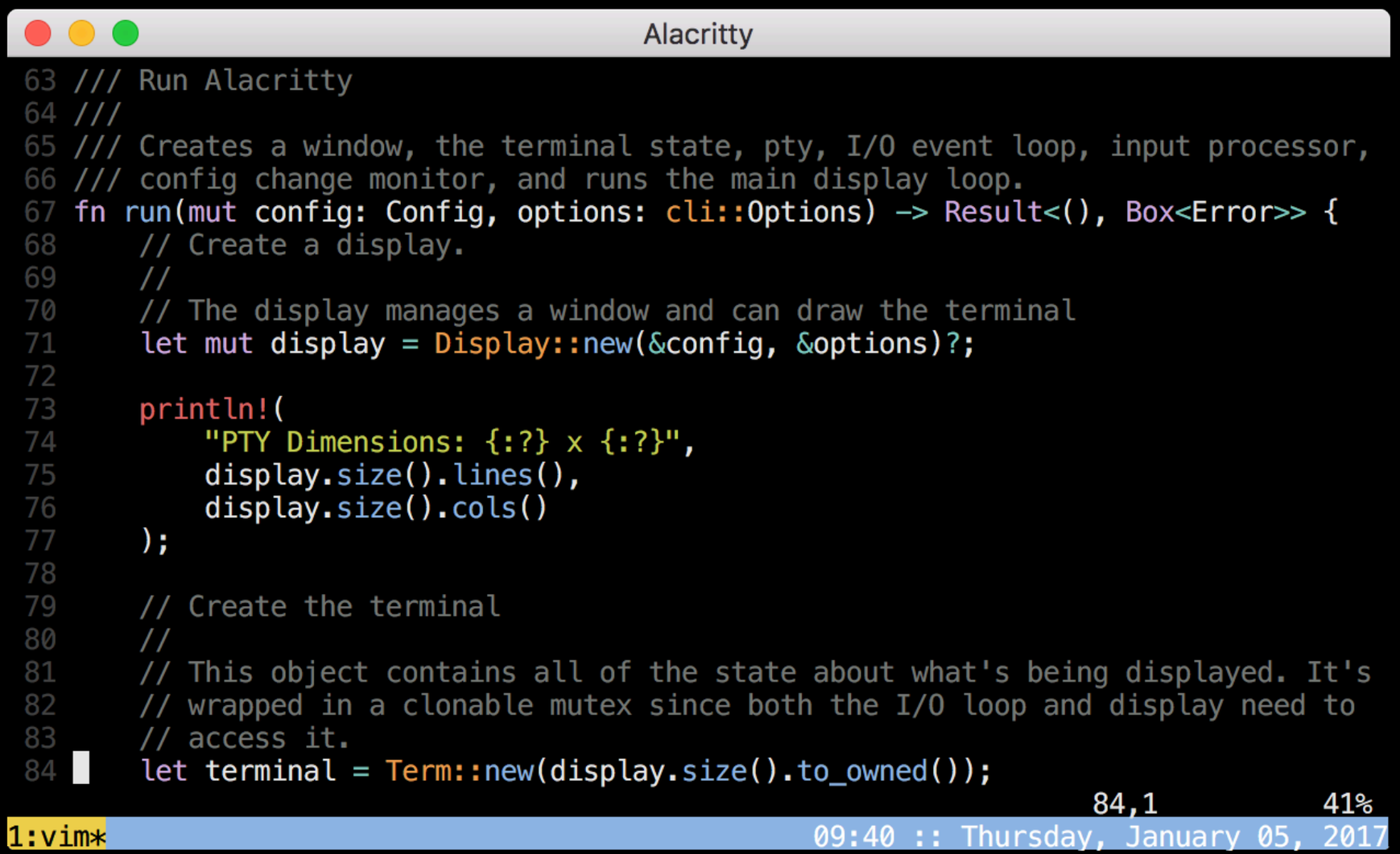

Cargo: packages for Rust
https://crates.io
crates.io
Rust Package Registry
Click or press 'S' to search...
Browse All Crates
Docs
Log in with GitHub
Fork me on GitHub
The Rust community's crate registry
Install Cargo
Getting Started
Instantly publish your crates and install them. Use the API to interact and find out more information about available crates. Become a contributor and enhance the site with your work.
461,553,059 Downloads
16,991 Crates in stock
New Crates
pg (0.0.0)
fluid (0.0.0)
algos (0.1.1)
Most Downloaded
libc (0.2.42)
bitflags (1.0.3)
rand (0.5.4)
Just Updated
algos (0.1.1)
rustc-ap-syntax (203.0.0)
rustc-ap-rustc_target (203.0.0)

Major projects

- Rust compiler and Cargo
- **Servo** – Mozilla's new parallel web browser engine
- **Redox OS** – a microkernel operating system
- **TockOS** – an embedded operating system
- **ripgrep** - text search provider in VS code

Alacritty: a GPU-accelerated terminal emulator

- Alacritty is the fastest terminal emulator in existence.



Xi

Xi.md

```
# xi-editor

The xi editor project is an attempt to build a high quality text
editor, using modern software engineering techniques. It is initially
built for Mac OS X, using Cocoa for the user interface. There are also
frontends for other operating systems available from third-party
developers.

Goals include:

* Incredibly high performance. All editing operations should commit and
paint in under 16ms. The editor should never make you wait for
anything.

* Beauty. The editor should fit well on a modern desktop, and not look
like a throwback from the '80s or '90s. Text drawing should be done
with the best technology available (Core Text on Mac, DirectWrite on
Windows, etc.), and support Unicode fully.

* Reliability. Crashing, hanging, or losing work should never happen.

* Developer friendliness. It should be easy to customize xi editor,
whether by adding plug-ins or hacking on the core.

## Getting started
```

Redox OS

Documentation Community News Screenshots
Donate GitLab RSoC

Redox is a Unix-like Operating System written in **Rust**, aiming to bring the innovations of Rust to a modern microkernel and full set of applications.

View Releases

Pull from GitLab

- Implemented in Rust
- Microkernel Design
- Includes optional GUI - Orbital
- Supports Rust Standard Library
- MIT Licensed
- Drivers run in Userspace
- Includes common Unix commands
- Newlib port for C programs

Redox running Orbital