mobica
a cognizant company

workshop_2.rs

Dawid Królak

26.09.2024

mobica
a cognizant company

$whoami_

Dawid Królak

mobica
a cognizant company

- Expert engineer @ mobica a cognizant company
- Rust developer
- Passionate about space exploration industry

- https://twitter.com/Taavicjusz
- https://github.com/taavit
- https://mastodon.social/@taavit

Agenda

mobica
a cognizant company

- Rust 101
- Async/Sync
- Embassy 101
  - setup
  - hello world
  - tasks
  - channel
- Signal generator

mobica
a cognizant company

Rust 101

Basics definition

Cloning, Copying
Reference
Ownership
Naming
Loops

Cloning, copying

mobica
a cognizant company

Copying

- When memory can be copied used `memcpy`
- Everything that can be copied, can be cloned.
- Implicit
- eg. primitives: u8, [f64; 8]

Cloning

- When duplicating requires special actions like memory allocation
- Explicit:
  - let s = String::new();
  - s.clone();
- eg. std::String, std::Vec

Ownership

mobica
a cognizant company

Single variable can live only in a single scope (won't compile):

```
struct Foo(u8);
fn main() {
    let mut a = Foo(0);
    do_something(a);
    do_something(a);
}

fn do_something(a: Foo) {
    // something
}
```

No need to remember order of constructors like in c++ ... (copying, cloning, default etc.)

References

- single mutable reference
- many read-only references
- Can reference to a part of array

```
struct Foo(u8);
fn main() {
    let mut a = Foo(0);
    do_something(&a);
    do_something(&a);
}

fn do_something(a: &Foo) {
    // something
}
```

References

mobica
a cognizant company

Cannot outlives variable it references

```
struct Foo(u8);
fn main() {
    let a = Foo;
    let b = &a;
    something(a);
    different(b);
}
fn something(f: Foo) {}
fn different(f: &Foo) {}
```

```
error[E0505]: cannot move out of `a` because it is borrowed
note: if `Foo` implemented `Clone`, you could clone the value
```

Naming

mobica
a cognizant company

into_* Convert consumes
to_* Convert clone
as_* Representation (mostly ref)
***_mut** Returned type is mutable
blocking_* To distinguish between async and sync functions

Loops

```
let a = vec![1, 2, 3];
// Change to iter_mut()
for v in a.iter() {
    dbg!(v);
}

for (idx, v) in a.iter().enumerate() {
    dbg!(idx, v);
}

for i in (0..4) {
    dbg!(i);
}

let mut a = 0;
loop {
    dbg!("a");
    if a > 10 {
        break;
    }
    a += 1;
}
```

Structs

mobica
a cognizant company

Various formats of structs

```
struct Foo;
struct Bar(u8);
struct Qoo {
 baz: u8,
}
struct Baz<T> {
   zoo: T,
   sth: bool,
}
```

Enums

Ok, Result

Pretty much like any other data type

```
enum SignalType {
    Sine(f32),
    Square(f32),
}

enum PublishSignalType {
    Sine(f32, f32),
    Square(f32, f32),
}
```

mobica
a cognizant company

Failure is not an **Option<T>**

It's a **Result<T, E>**

~*Generic Kranz*

Result, Option

.unwrap().unwrap().unwrap()?.unwrap().unwrap().unwrap()?

Option

```
pub enum Option<T> {
    None,
    Some(T),
}
```

Result

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

Pattern matching

mobica
a cognizant company

There is no switch case

- All branches have to be covered (if not explicitly defined)
- Can be nested, and nested, and nested.

```
let foo: Option<u8> = Some(4);
match foo {
    Some(0) => { println!("NOTHING") }
    Some(1) => { println!("ONE") }
    Some(_) => { println!("MANY") }
    None => { println!("No idea") }
}
```

Might be useful

mobica
a cognizant company

Formating text in #[no_std], timers

```
// Imports traits
use core::fmt::Write;

// Create a buffer (String come from heapless)
let mut buf = String::<64>::new();

// Print to buffer. In case overflow, code will panic. Formatting comes from defmt-03
crates/features
core::write!(&mut buf, "SINE,{},{}\r\n", raw, filtered).unwrap();

// Handles delay
use embassy_time::{Delay, Duration};
embassy_time::Timer::after(Duration::from_millis(50)).await;
```

mobica
a cognizant company

Async/Await

Sequential vs parallel vs concurrent

mobica
a cognizant company

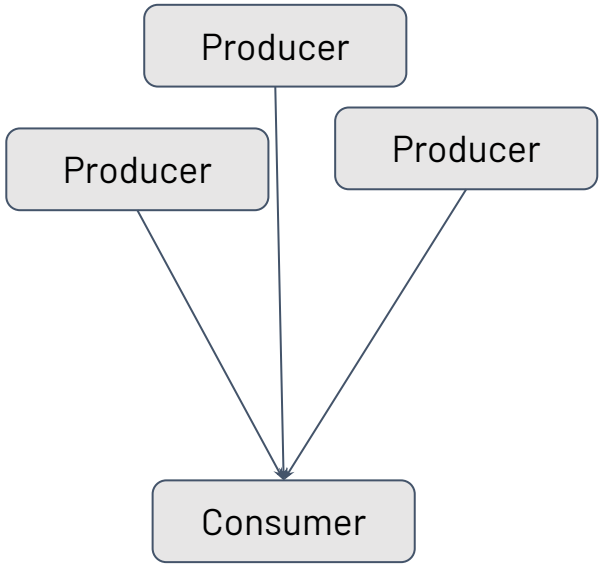
Sequential

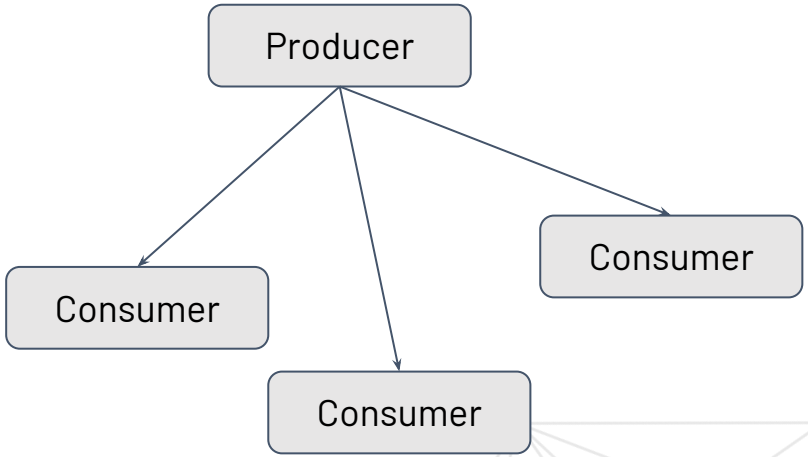
Parallel

Concurrent

Communication - Channels

mpsc (multiple producers, single consumer), pub sub (multiple consumers, single producer)

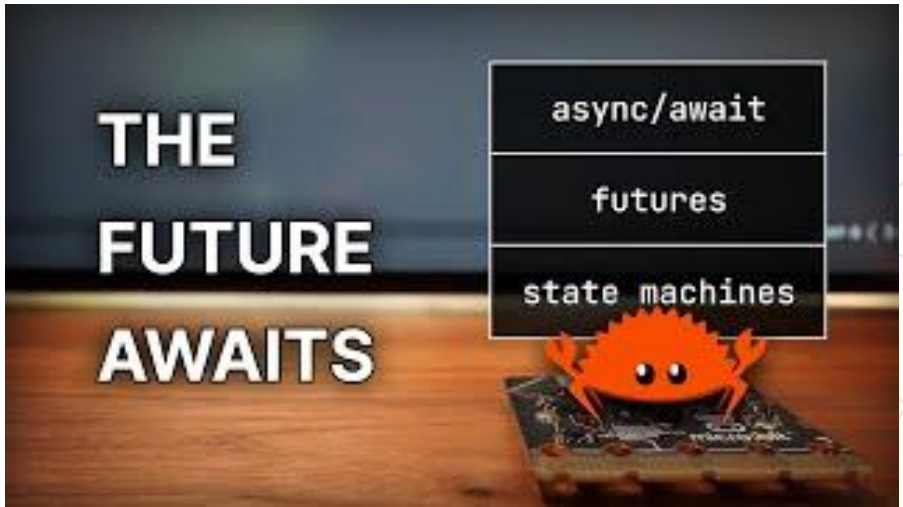




Async await

mobica
a cognizant company

- Stores current state (state must be Send)
- Schedule task
- Yields to executor
- Wait for interrupt



mobica
a cognizant company

Embassy

Embassy structure

mobica
a cognizant company

- Embassy framework
  - Timers
  - Synchronization (Mutex, Channels)
  - Async executor
- Embassy embedded-hal-async implementations
  - stm32
  - rp2040
  - nrf

Hello embassy...

mobica
a cognizant company

```rust
#![no_std]
#![no_main]

use embassy_executor::Spawner;
use embassy_rp::gpio;
use embassy_time::Timer;
use gpio::{Level, Output};
use {defmt_rtt as _, panic_probe as _};

#[embassy_executor::main]
async fn main(_spawner: Spawner) {
    let p = embassy_rp::init(Default::default());
    let mut led = Output::new(p.PIN_25, Level::Low);

    loop {
        led.set_high();
        Timer::after_secs(1).await;
        led.set_low();
        Timer::after_secs(1).await;
    }
}
```

MPSC Channel

mobica
a cognizant company

embassy_sync::channel::Channel

A bounded channel for communicating between asynchronous tasks with backpressure.

The channel will buffer up to the provided number of messages. Once the buffer is full, attempts to send new messages will wait until a message is received from the channel.

All data sent will become available in the same order as it was sent.

https://docs.embassy.dev/embassy-sync/git/default/channel/struct.Channel.html

Channel - send

mobica
a cognizant company

```rust
/// Send a value, waiting until there is capacity.
///
/// Sending completes when the value has been pushed to the channel's queue.
/// This doesn't mean the value has been received yet.
pub fn send(&self, message: T) -> SendFuture<'_, M, T, N> {
    SendFuture {
        channel: self,
        message: Some(message),
    }
}
```

Channel receive

mobica
a cognizant company

```rust
/// Receive the next value.
///
/// If there are no messages in the channel's buffer, this method will
/// wait until a message is sent.
pub fn receive(&self) -> ReceiveFuture<'_, M, T, N> {
    ReceiveFuture { channel: self }
}
```

mobica
a cognizant company

Project

Signal generator

Plan

Task #1 - Generate sinusoidal signal + noise

Task #2 - Generate square signal + noise

Task #3 - Receive signal, filter it, send processed data

Task #4 - Send filtered signal via uart

Template - Basic enums, statics

mobica
a cognizant company

```
static SIGNAL_CHANNEL: Channel<ThreadModeRawMutex, SignalType, 4> = Channel::new();
static PUBLISH_CHANNEL: Channel<ThreadModeRawMutex, PublishSignalType, 4> = Channel::new();
bind_interrupts!(pub struct Irqs {
    ADC_IRQ_FIFO => InterruptHandler;
    UART0_IRQ  => UARTInterruptHandler<UART0>;
});
enum SignalType {
    Sine(f32),
    Square(f32),
}
enum PublishSignalType {
    Sine(f32, f32),
    Square(f32, f32),
}
```

Template - Main function

mobica
a cognizant company

```rust
#[embassy_executor::main]
async fn main(spawner: Spawner) {
    let p = embassy_rp::init(Default::default());
    // Needed for random number generation for noise
    let mut adc = Adc::new(p.ADC, Irqs, Config::default());
    let mut p28 = AdcChannel::new_pin(p.PIN_28, Pull::None);
    let uart = uart::Uart::new(
        p.UART0,
        p.PIN_0,
        p.PIN_1,
        Irqs,
        p.DMA_CH0,
        p.DMA_CH1,
        uart::Config::default(),
    );

    unwrap!(spawner.spawn(sine_generator()));
    unwrap!(spawner.spawn(square_generator()));
    unwrap!(spawner.spawn(filter_data()));
    unwrap!(spawner.spawn(send_to_pc(uart)));
}
```

Template tasks

mobica
a cognizant company

```rust
#[embassy_executor::task]
async fn sine_generator() {
    todo!();
}

#[embassy_executor::task]
async fn square_generator() {
    todo!();
}

#[embassy_executor::task]
async fn filter_data() {
    todo!();
}

#[embassy_executor::task]
async fn send_to_pc(mut uart: uart::Uart<'static, UART0, uart::Async>) {
    todo!();
}
```

Links

mobica
a cognizant company

- embassy - https://embassy.dev
- embedded-hal: https://github.com/rust-embedded/embedded-hal
- Rust based OS for automotive: https://oxidos.io/
- Critical safety systems in Rust (Compiler for ESA based LEON cpu): https://ferrous-systems.com/blog/rust-for-mission-critical-applications/
- Aerorust community: https://aerorust.org/

Ok(())