Can a Rust closure be used by multiple threads?

Ask Question

Asked 5 years, 9 months ago

Active 3 years, 2 months ago

Viewed 5k times

9

2

I'd like the ability to have multiple threads evaluate the same closure. The application I have in mind is parallelized numerical integration, so a situation where the function domain can be easily split into N chunks and handed to threads.

This is a simple function that evaluates the provided closure multiple times and averages the result:

```
use std::sync::mpsc;
use std::thread;

const THREAD_COUNT: u64 = 4;

fn average<F: Fn(f64) -> f64>(f: F) -> f64 {
    let (tx, rx) = mpsc::channel();
    for id in 0..THREAD_COUNT {
        let thread_tx = tx.clone();
        thread::spawn(move || {
            thread_tx.send(f(id as f64));
        });
    }

    let mut total = 0.0;
    for id in 0..THREAD_COUNT {
        total += rx.recv().unwrap();
    }
    total / THREAD_COUNT as f64
}

fn main() {
    average(|x: f64| -> f64 { x });
}
```

When I compile I get this error:

```
error[E0277]: `F` cannot be sent between threads safely
  --> src/main.rs:10:9
   |
10 |         thread::spawn(move || {
   |         ^^^^^^^^^^^^^ `F` cannot be sent between threads safely
   |
   = help: within `[closure@src/main.rs:10:23: 12:10 thread_tx:std::sync::mpsc::Sender<f64>, f:F, id:u64]`, the trait `std::marker::Send` is not implemented for `F`
   = help: consider adding a `where F: std::marker::Send` bound
   = note: required because it appears within the type `[closure@src/main.rs:10:23: 12:10 thread_tx:std::sync::mpsc::Sender<f64>, f:F, id:u64]`
   = note: required by `std::thread::spawn`
```

So I add `+ Send` to the bounds on `F` and get a new error:

```
error[E0310]: the parameter type `F` may not live long enough
  --> src/main.rs:10:9
   |
6  | fn average<F: Fn(f64) -> f64 + Send>(f: F) -> f64 {
   |            -- help: consider adding an explicit lifetime bound `F: 'static`...
...
10 |         thread::spawn(move || {
   |         ^^^^^^^^^^^^^
   |
note: ...so that the type `[closure@src/main.rs:10:23: 12:10 thread_tx:std::sync::mpsc::Sender<f64>, f:F, id:u64]` will meet its required lifetime bounds
  --> src/main.rs:10:9
   |
10 |         thread::spawn(move || {
   |         ^^^^^^^^^^^^^
```

So I add `+ 'static` to `F` and get this:

```
error[E0382]: capture of moved value: `f`
  --> src/main.rs:11:28
   |
10 |         thread::spawn(move || {
   |                       ------- value moved (into closure) here
11 |             thread_tx.send(f(id as f64));
   |                            ^ value captured here after move
   |
   = note: move occurs because `f` has type `F`, which does not implement the `Copy` trait
```

So I add `+ Copy` to `F` and get:

```
error: the trait `core::marker::Copy` is not implemented for the type `[closure@src/test.rs:115:11: 115:26]
```

It seems every thread wants its own copy of the closure (because of `move`) but closures don't implement `Copy` so no luck. It seems strange to me because if the closures are never mutating state then what's the safety issue with multiple threads accessing them?

I can get the code to work by providing a regular function instead of a closure, but this makes my code non-generic, i.e. it only works for a specific function instead of for anything

Your privacy

By clicking “Accept all cookies”, you agree Stack Exchange can store cookies on your device and disclose information in accordance with our Cookie Policy.

Accept all cookies Customize settings

that's `Fn(f64) -> f64` . And for the type of integration I'm doing, the functions integrated often have certain fixed variables mixed with the variable of integration, so it would seem natural to capture the fixed variables with a closure.

Is there some way to make this kind of multithreaded function evaluation work in a generic manner? Am I just thinking about things wrong?

multithreading concurrency closures rust

Share
Improve this question
Follow

edited Oct 29 '18 at 20:11

Shepmaster
305k • 59 • 824 • 1083

asked Mar 24 '16 at 22:58

Josh Hansen
1,188 • 1 • 14 • 17

Add a comment

1 Answer

Active Oldest Votes

▲
10
▼

✓

The ultimate problem revolves around *who owns the closure*. The code as written states that ownership of the closure is transferred to `average` . This function then tries to give the closure to multiple threads, which fails as you have seen, as you can't give one item to multiple children.

> but closures don't implement `Copy` so no luck

As of Rust 1.26.0, closures *do* implement `Clone` and `Copy` if all of the captured variables do. This means your final example code now works as-is:

```
fn average<F: Fn(f64) -> f64 + Send + 'static + Copy>(f: F) -> f64 { /* ... */ }
```

However, it's possible that your closures won't implement `Copy` or `Clone` .

You cannot give out a reference to the closure owned by `average` because the thread created with `thread::spawn` may outlive the call to `average` . When `average` exits, any stack-allocated variables will be destroyed. Any use of them would cause memory unsafety, which Rust aims to prevent.

One solution is to use an `Arc` . This will allow multiple shared owners of a single resource in a multithreaded context. When the wrapped closure is cloned, only a new reference is created. When all references disappear, the object is freed.

```
use std::{
    sync::{mpsc, Arc},
    thread,
};

const THREAD_COUNT: u64 = 4;

fn average<F>(f: F) -> f64
where
    F: Fn(f64) -> f64 + Send + Sync + 'static,
{
    let (tx, rx) = mpsc::channel();
    let f = Arc::new(f);

    for id in 0..THREAD_COUNT {
        let thread_tx = tx.clone();
        let f = f.clone();
        thread::spawn(move || {
            thread_tx.send(f(id as f64)).unwrap();
        });
    }

    let mut total = 0.0;
    for _ in 0..THREAD_COUNT {
        total += rx.recv().unwrap();
    }

    total / THREAD_COUNT as f64
}

fn main() {
    average(|x| x);
}
```

A more standard solution is to use *scoped threads*. These threads are guaranteed to exit by a certain time, which allows you to pass references that outlive the threads to the threads.

See also:

- How can I pass a reference to a stack variable to a thread?
- How do I pass disjoint slices from a vector to different threads?

Share
Improve this answer

Follow

edited Oct 29 '18 at 20:15

answered Mar 25 '16 at 3:03

Shepmaster
305k ● 59 ● 824 ● 1083

---

crossbeam looks like what I'm looking for. One question: if crossbeam was basically booted from Rust 1.0 due to soundness issues, have those issues been resolved in the library as it now stands?
– Josh Hansen
Mar 25 '16 at 17:53

@JoshHansen yes. There's lots of background information for the curious. The original issue and the related RFC are very complete. IIRC, there was one issue that was fixed in Rust proper to allow some of these to be built on top, but I can't find the exact link now.
– Shepmaster
Mar 25 '16 at 18:10

Thanks for the additional context. I ended up using crossbeam which basically seems to be working.
– Josh Hansen
Mar 25 '16 at 22:05

Add a comment

Your Answer

Post Your Answer

By clicking "Post Your Answer", you agree to our terms of service, privacy policy and cookie policy

Not the answer you're looking for? Browse other questions tagged multithreading concurrency closures rust or ask your own question.

The Overflow Blog

- Sequencing your DNA with a USB dongle and open source code
- Don't push that button: Exploring the software that flies SpaceX rockets and...

Featured on Meta

- Providing a JavaScript API for userscripts
- Congratulations to the 59 sites that just left Beta

Linked

48
How can I pass a reference to a stack variable to a thread?

7
How do I pass disjoint slices from a vector to different threads?

Related

What is the difference between a 'closure' and a 'lambda'?

JavaScript closure inside loops – simple practical example

1415
How can I use threading in Python?

Android "Only the original thread that created a view hierarchy can touch its views."

How do I capture variables outside the scope of a closure in Rust?

Rust cloned closures expected closure, found different closure

How can I share or avoid sharing a websocket resource between two threads?

Awaiting a Number of Futures Unknown at Compile Time

Error: use of moved value: `path`. How to correct this code?

Hot Network Questions

Numbers, Racked Up

Have there been no deaths due to omicron in Africa?

What caused this crash landing?

Changing a color of a link

Remove the Times x from display in Inactivate expressions in V13

more hot questions

Question feed

STACK OVERFLOW

Questions
Jobs
Developer Jobs Directory
Salary Calculator
Help
Mobile

PRODUCTS

Teams
Talent
Advertising
Enterprise

COMPANY

About
Press
Work Here
Legal
Privacy Policy
Terms of Service
Contact Us
Cookie Settings
Cookie Policy

STACK EXCHANGE NETWORK

Technology
Culture & recreation
Life & arts
Science
Professional
Business
API
Data

Blog
Facebook
Twitter
LinkedIn
Instagram

site design / logo © 2021 Stack Exchange Inc; user contributions licensed under cc by-sa. rev 2021.12.22.41046