mem::forget(completion) unsafety? #11

New issue

Closed **goffrie** opened this issue on Feb 7 · 11 comments

goffrie commented on Feb 7

It seems to me that the soundness of Completion relies on the fact that it waits for the operation to complete in its Drop impl. But per the leakpocalypse it's safe to just forget the completion, side-stepping that?

spacejam commented on Feb 8 Owner

Yeah. Personally, I consider `mem::forget` to be an incorrectly safe operation for this reason. I think this needs to be better documented.

👍 1

DemiMarie-parity commented on Feb 19

This can be avoided by returning `Pin<Box<Completion>>` and `impl !Unpin for Completion`. There should be other ways to do this as well, such as requiring a `Pin<&mut Completion>` argument in various places.

goffrie commented on Feb 19 Author

Can't you still mem::forget the box?
…

DemiMarie-parity commented on Feb 21

You can :(. The only solution I can think of is for `rio` to own all buffers. That will be needed anyway for preregistration, though, so it isn't all bad.

spacejam commented on Feb 21 • edited Owner

My personal feeling is that `mem::forget` is unlikely to be used on `Completion`. It's a dangerous tool in general that I believe most people rarely rely on. Can you folks think of a realistic reason someone may try to use it on this? I know this is not necessarily a water-tight argument, but personally I'm not sure we should significantly reduce the usability of an API due to an issue that you can only encounter if you do something kind of foolish to begin with. Are there cases where it doesn't seem foolish to use `mem::forget` over `drop`? I'm totally biased, because I understand what the current semantics are and I'm incentivized to justify them, but things like leakopalypse never really felt compelling as a reason to avoid relying on Drop to me.

twissel commented on Feb 21

For me mem::forget it's not the issue, but for example read or timeout is the issue, why do i need to wait until read is finished (drop called)?, passing ownership to kernel is the only way i think.

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Linked pull requests

Successfully merging a pull request may close this issue.

None yet

6 participants

Shnatsel commented on Apr 28

`mem::forget()` is a rather contrived example, but it's marked safe because you can get identical behavior by creating a reference cycle using `Rc` or `Arc` in safe code, and that cannot be fixed without introducing a garbage collector. And a leak caused by a reference cycle is a much more realistic scenario, especially seeing how prevalent `Arc` is in async code.

👍 2

stjepang commented on Apr 28

I like rio's API a lot! While it is technically unsound because it can cause undefined behavior in safe code, I don't think it needs a lot of changes to be sound.

All we need to do is require the user to open *one* line of unsafe code and to declare "I know what I'm doing". Rio's constructor could be made unsafe, or some kind of getter method on `Rio` that gives you its convenient API.

That way we get *technical* soundness and keep the convenience. Anyways... I'm thinking this is a lot of talk for what is a trivially easy problem to solve (:

DemiMarie-parity commented on Apr 28

@stjepang why is it bad for rio to own the buffers? That allows for preregistration (a large performance improvement).

👍 1

Shnatsel commented on May 6

withoutboats has published a blog post on io-uring and safe Rust wrappers for it. It discusses the issues with API reliant on Drop running beyond the obvious safety issues and proposes some possible solutions. https://boats.gitlab.io/blog/post/io-uring/

👍 1

spacejam commented 28 days ago • edited ▾ Owner

boats considers mem::forget to be a more realistic issue than I do. mem::forget is unsafe in real code, the same as a memory leak is unsafe in real code. I don't view the unsoundness of this library to be meaningful in real systems. I view Rust's design decisions as unfortunate to allow such bypasses of the borrow checker to become possible with leaks, but ultimately one that does not actually mean anything for people building real systems that need to guarantee that they do not contain leaks using LSAN anyway etc... Real systems have to use LSAN anyway because we can't rely on Rust to prevent leaks.

Just because some things are prevented by the Rust compiler, we still have to be responsible about engineering using Rust, and use these kinds of tools for anything that matters. Code is never sufficient. This is why I don't consider this form of unsoundness an issue - we have to use these tools that will catch issues anyway, and when we use them, meaningful unsoundness will almost always become clear quickly.

spacejam closed this 28 days ago

Repository owner locked and limited conversation to collaborators 28 days ago