

DEADLOCKED

MARA BOS

@m_ou_se

I'M GOING TO TELL YOU

A STORY



CHAPTER 1

PROBLEMS

MUTEX

MUTUAL EXCLUSION

LOCK
&MUT SOMETHING
UNLOCK

LOCK
&MUT SOMETHING
UNLOCK

IMPLEMENTING
IS HARD



THE OPERATING SYSTEM

PROVIDES THEM



LINUX, MAC, OTHER UNIXES

WINDOWS

LINUX, MAC, OTHER UNIXES

`pthread_mutex_t`

WINDOWS

LINUX, MAC, OTHER UNIXES

`pthread_mutex_t`
(Part of the POSIX standard)

WINDOWS

LINUX, MAC, OTHER UNIXES

`pthread_mutex_t`
(Part of the POSIX standard)

WINDOWS

LINUX, MAC, OTHER UNIXES

`pthread_mutex_t`
(Part of the POSIX standard)

WINDOWS

Critical Sections

LINUX, MAC, OTHER UNIXES

`pthread_mutex_t`

(Part of the POSIX standard)

WINDOWS

Critical Sections

SRW Locks (Slim Read-Write Locks)

SO, JUST WRAP THEM?

```
struct Mutex(SystemMutex);
```

AND, DONE?



NOPE

THESE WERE DESIGNED
FOR C

IN RUST, OBJECTS
CAN BE MOVED

Remarks

An SRW lock must be initialized before it is used. The `InitializeSRWLock` function is used to initialize a SRW lock dynamically. To initialize the structure statically, assign the constant `SRWLOCK_INIT` to the structure variable.

An SRW lock cannot be moved or copied. The process must not modify the object, and must instead treat it as logically opaque. Only use the SRW functions to manage SRW locks.

SO WE PUT THEM ON THE HEAP

```
struct Mutex(Box<SystemMutex>);
```

INEFFICIENT



NO CONST CONSTRUCTOR



IN RUST, 'WRONG' USAGE
MUST BE SAFE

```
let m = Mutex::new(123);
```

```
let a = m.lock();
```

```
    let b = m.lock(); // deadlock!
```

```
    ...
```

```
unlock(b);
```

```
unlock(a);
```

```
let m = Mutex::new(123);
```

```
let a = m.lock();
```

```
    let b = m.lock(); // deadlock!
```

```
    ...
```

```
unlock(b);
```

```
unlock(a);
```

```
let m = Mutex::new(123);
```

```
let a = m.lock();
```

```
    let b = m.lock(); // deadlock!
```

```
    ...
```

```
unlock(b);
```

```
unlock(a);
```

```
let m = Mutex::new(123);
```

```
let a = m.lock();
```

```
    let b = m.lock(); // deadlock!
```

```
    ...
```

```
unlock(b);
```

```
unlock(a);
```



```
let m = Mutex::new(123);
```

```
let a = m.lock();
```

```
    let b = m.lock(); // deadlock!
```

```
    ...
```

```
unlock(b);
```

```
unlock(a);
```

DEADLOCKING

IS SAFE

Mutex Type	Robustness	Relock	Unlock When Not Owner
NORMAL	non-robust	deadlock	undefined behavior
NORMAL	robust	deadlock	error returned
ERRORCHECK	either	error returned	error returned
RECURSIVE	either	recursive	error returned
		(see below)	
DEFAULT	non-robust	undefined	undefined behavior [†]
		behavior [†]	
DEFAULT	robust	undefined	error returned
		behavior [†]	

After a thread has ownership of a critical section, it can make additional calls to **EnterCriticalSection** or **TryEnterCriticalSection** without blocking its execution. This prevents a thread from deadlocking itself while waiting for a critical section that it already owns. The thread enters the critical section each time **EnterCriticalSection** and **TryEnterCriticalSection** succeed. A thread must call **LeaveCriticalSection** once for each time that it entered the critical section.

```
let m = Mutex::new(123);
```

```
let a = m.lock();
```

```
    let b = m.lock(); // maybe no deadlock?
```

```
        f(a, b); // undefined behaviour! 🤔
```

```
    drop(b);
```

```
drop(a);
```

```
let m = Mutex::new(123);
```

```
let a = m.lock();
```

```
    let b = m.lock(); // maybe no deadlock?
```

```
        f(a, b); // undefined behaviour! 🤔
```

```
    drop(b);
```

```
drop(a);
```

```
let m = Mutex::new(123);
```

```
let a = m.lock();
```

```
let b = m.lock(); // maybe no deadlock?
```

```
    f(a, b); // undefined behaviour! 🤔
```

```
drop(b);
```

```
drop(a);
```

```
let m = Mutex::new(123);
```

```
let a = m.lock();
```

```
let b = m.lock(); // maybe no deadlock?
```

```
f(a, b); // undefined behaviour! 🤔
```

```
drop(b);
```

```
drop(a);
```


IN RUST, FORGETTING THINGS
MUST BE SAFE

```
let m = Mutex::new(123);
```

```
let a = m.lock();
```

```
mem::forget(a); // safe!
```

```
drop(m); // dropped, but still locked!
```

```
let m = Mutex::new(123);
```

```
let a = m.lock();
```

```
mem::forget(a); // safe!
```

```
drop(m); // dropped, but still locked!
```

```
let m = Mutex::new(123);
```

```
let a = m.lock();
```

```
mem::forget(a); // safe!
```

```
drop(m); // dropped, but still locked!
```

```
let m = Mutex::new(123);
```

```
let a = m.lock();
```

```
mem::forget(a); // safe!
```

```
drop(m); // dropped, but still locked!
```

DESCRIPTION

The *pthread_mutex_destroy()* function shall destroy the mutex object referenced by *mutex*; the mutex object becomes, in effect, uninitialized. An implementation may cause *pthread_mutex_destroy()* to set the object referenced by *mutex* to an invalid value.

A destroyed mutex object can be reinitialized using *pthread_mutex_init()*; the results of otherwise referencing the object after it has been destroyed are undefined.

It shall be safe to destroy an initialized mutex that is unlocked. Attempting to destroy a locked mutex, or a mutex that another thread is attempting to lock, or a mutex that is being used in a *pthread_cond_timedwait()* or *pthread_cond_wait()* call by another thread, results in undefined behavior.

CONCLUSION

RUST IS NOT C

FITTING A C-SHAPED PEG
INTO A RUST-SHAPED HOLE.





CHAPTER 2

A SOLUTION?

SEPTEMBER 2018

[Sign Up](#)[Log In](#)

🔒 Standard library, synchronization primitives, and undefined behavior

■ libs



alexcrichton

Sep '18

Hey all! The standard library provides a number of synchronization primitives for all Rust programs to use in a cross-platform fashion. Ideally the standard library also provides *sound* implementations of these primitives so they can't stop working at runtime!

Currently the system primitives `Mutex`, `RwLock`, `Condvar`, and `ReentrantMutex` (internal to `libstd`) all wrap underlying primitives for each os (pthreads on non-Windows, corresponding objects on Windows). Unfortunately, though, using the OS primitives brings quite a few caveats:

- First off, once the primitive is used *its memory address cannot be changed* ⁵⁹. This requires our safe wrappers to use `Box` to contain the primitives (as Rust values can change memory addresses through moves).

Sep 2018

1 / 26

Sep 2018

[Back](#)

PARKING LOT

MUTEXES ARE ONE BYTE
AND NOT HEAP ALLOCATED



EVERYTHING IS
CONST-CONSTRUCTIBLE



DEADLOCKING AND FORGETTING
IS WELL-DEFINED



IS THIS THE SOLUTION
TO EVERYTHING?



created



Sep '18

last reply



Apr '19

25

replies

7.3k

views

13

users

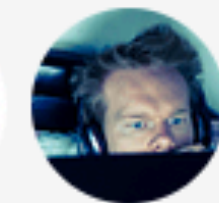
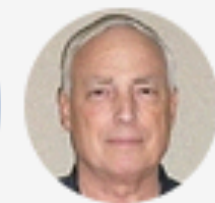
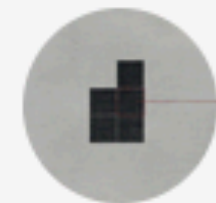
98

likes

31

links


Frequent Posters




DECEMBER 2018

Use the parking_lot locking primitives #56410

 Open

faern wants to merge 40 commits into `rust-lang:master` from `faern:add-parking-lot` 

 Conversation 1

 Commits 40

 Checks 1

 Files changed 64



faern commented on 1 Dec 2018

Contributor



This PR adds the `parking_lot` code as a git submodule and reimplements all the standard library locking primitives so they are thin wrappers over the `parking_lot` equivalents. The standard library public API is of course kept the same.

This has been discussed in <https://internals.rust-lang.org/t/standard-library-synchronization-primitives-and-undefined-behavior/8439/9>

Thanks @Amanieu for mentoring when doing this, and basically all the code is his as well of course.

Fixes [#35836](#)

Fixes [#53127](#)



37



31




32



5

DECEMBER 2018




Amanieu

on 1 Dec 2018

Member

This is an internal API which is only used in the internals of `parking_lot_core`. Regarding your question: I used "should" here because you don't want to issue a system call while holding a lock, to keep lock durations short.



Reply...

Unresolve conversation

faern marked this conversation as resolved.

src/libstd/sys_common/thread_parker.rs


⛶ Hide resolved

59 + }

60 +

61 + // Locks the parker to prevent the target thread from exiting. This is

62 + // necessary to ensure that thread-local ThreadData objects remain valid.

 Centril on 1 Dec 2018


Contributor

⋮

Suggested change ⓘ

- // necessary to ensure that thread-local ThreadData objects remain valid.

+ // necessary to ensure that thread-local `ThreadData` objects remain valid.


 faern on 2 Dec 2018

Author

Contributor

⋮

Fixed

 Reply...

Unresolve conversation


Centril marked this conversation as resolved.

src/libstd/sys_common/thread_parker.rs

Outdated

⛶ Hide resolved

```
21 + pub const IS_CHEAP_TO_CONSTRUCT: bool = true;
22 +
23 + pub fn new() -> ThreadParker {
24 +     ThreadParker {
```

 Centril on 1 Dec 2018

Contributor


😊 ...

Suggested change ⓘ

- ThreadParker {

+ Self {

There are a bunch of other places this applies in this PR.

 Reply...

Unresolve conversation

Centril marked this conversation as resolved.

src/libstd/sys_common/parking_lot_core/word_lock.rs

Outdated


Hide resolved

167 +


168 + #[cold]

169 + #[inline(never)]


170 + unsafe fn unlock_slow(&self) {

 Centril on 1 Dec 2018


Contributor

 ...


This is a lot of logic for an unsafe function and the invariants aren't stated; It would be more readable to split it up into smaller chunks imo and also state the invariants required. (Applies also to the function before this one).

 Amanieu on 1 Dec 2018


Member

 ...


In this case the invariants are pretty simple: don't lock if the current thread already owns the lock and don't unlock if the current thread doesn't own the lock.

 Centril on 1 Dec 2018 · edited

Contributor


 ...

@Amanieu Sure; but it should be stated in the code explicitly. :) and long unsafe fn s are more risky imo; if you break it down a bit things can be reasoned about better.


 faern on 4 Dec 2018

Author

Contributor

 ...

I have now managed to make three methods in this type not fully unsafe. And the only unsafe one left states why it is: unlock() must not be called on an already unlocked WordLock.

 Reply...

Unresolve conversation

faern marked this conversation as resolved.

```
src/libstd/sys_common/parking_lot_core/util.rs Outdated ⚙ Hide resolved
```

```
27 + unreachable!();
28 + } else {
29 +     enum Void {}
30 +     match *1 as *const Void {}
```


DECEMBER 2018

```
tidy error: /build/rust/src/parking_lot/benchmark/src/rwlock.rs:15: platform-specific cfg: cfg(unix)
tidy error: /build/rust/src/parking_lot/benchmark/src/rwlock.rs:96: platform-specific cfg: cfg(not(unix))
...
```

Here is the work in progress state of what I change in `parking_lot` to make this work: https://github.com/faern/parking_lot/compare/master...faern:as-libstd-submodule?expand=1

As you can see, it's quite hacky and quite a lot of conditional compilation going on. One way to reduce it might be to make `libstd` re-export itself under a module named `std` 🙌 That way `parking_lot` can use `use std::whatever` even when it's part of `libstd`.

One more still unsolved problem is the documentation tests in `parking_lot`. They also have the wrong use statements (`use parking_lot::Condvar` is not a valid path when compiled inside `libstd`). But I'm not sure we could conditionally compile that. And even if we could it would likely end up very ugly.

The way I'm bringing the code into `libstd` goes something like this at the moment:

```
#[path = "../parking_lot/core/src/lib.rs"]
#[allow(missing_debug_implementations, missing_docs, dead_code, unused_imports)]
mod parking_lot_core;
```



faern commented on 17 Dec 2018

Contributor Author 🗨️ ⋮

Small update on last post: Adding

```
mod std {
  pub use super::*;
}
```

to `libstd/lib.rs` is a bit awkward, but it allows code inside `libstd` to use the same paths to everything as if they were not inside `libstd`. This allows us to make way fewer changes to `parking_lot` itself. And possibly other crates we might want to include in a similar manner. 🙌



alexcrichton commented on 18 Dec 2018

Member 🗨️ ⋮

One obvious thing is that the path to `std` types are different if it's being compiled as a normal crate or if it's actually inside `libstd`

FWIW `stdsimd` ran into the same issue, and [local development uses a bit of a hack](#) to get `core`-looking paths working everywhere. I don't think this would work well in `parking_lot` though which is otherwise intended to compile on stable and also be more maintainable outside of `std`. The workaround you mentioned with `pub use super::*` if it works is probably the way to go.

Another problem is the tidy format check

Oh it's fine to ignore this submodule, the [whitelist to ignore is here](#)

As you can see, it's quite hacky and quite a lot of conditional compilation going on.

That part of the change I think is good to keep @Amanieu in the loop, as that's what'd ideally go upstream into `parking_lot` itself!

One more still unsolved problem is the documentation tests in `parking_lot`

This is a bummer indeed! We have tons of pain with this in `stdsimd`. The end goal I think is to skip all of `parking_lot`'s tests, including doctests, when included into the Rust repo. As to how to get there I'm not entirely certain...

FWIW I think all of these problems would be largely solved if `parking_lot` were a crate. I feel like that's still the best solution here if we can stomach it.

Would it be possible for `parking_lot` to have a `no_std` mode that has internal wrappers for types that `libstd` otherwise provides like `Instant`?



Amanieu commented on 18 Dec 2018

Member 🗨️ ⋮

As you can see, it's quite hacky and quite a lot of conditional compilation going on.

I don't think that shoehorning the existing `parking_lot` crate into compiling directly as part of `libstd` is a good idea. Unlike `stdsimd`, which is explicitly designed to work as part of the standard library, `parking_lot` is intended to be used as a normal crate. This is particularly noticeable when you look at `stdsimd`: there is a `stability` attribute on every function.

Would it be possible for `parking_lot` to have a `no_std` mode that has internal wrappers for types that `libstd` otherwise provides like `Instant`?

It's not just `Instant`, `parking_lot` also depends on `thread_local!` and `thread::yield`. We could just copy these types and functions into `parking_lot`, but then we're creating more maintenance problems than we are solving.

I personally feel that [hashbrown \(#56241\)](#) is better candidate for making `libstd` depend on an external crate, but even then, there are several obstacles to overcome (in particular API stability and stability attributes).



alexcrichton commented on 18 Dec 2018

Member 🗨️ ⋮

Oh so in terms of stability `stdsimd` is an odd-one-out. I think we'll want to always implement and document the stable interface in this repository, so even if we pull in implementation details from elsewhere we'll want the actual stable shims to be in this repository itself (aka `HashMap` would be a thin wrapper around `hashbrown` hash maps). With `stdsimd` it's an "odd

JANUARY 2019

crate on their own for their own rfc, and have that crate become a backing crate behind the facade rather than an albatross they have to continuously re-extract.

- "re-facading"
 - I wrote [🔗 Refactor std to improve ease of porting](#) rust-lang-nursery/portability-wg#1 (comment) with 3 independent tracks of library work.
 - People were wary of even temporarily making the alloc and std collections not unify, so I opened [🔥 RFC: Existential types with external definition](#) rfc#2492 to add a language feature to allow moving collections (and other things referring to global/singletons) into upstream crates without that compromise.
 - @glandium and I have some PRs for alloc to add allocator parameters (closed due to being blocked on LLVM min version issues and lib team bandwidth). This isn't strictly speaking re-facading, but I bring it up since `hashbrown` is the best candidate to stay out of tree, and I've long wanted all of the collections to also live out of tree since they many are so close to being stable code. (Unstable features that just gate an unstable API don't count!)

So I think the next steps are shepherding that RFC, rebooting the `core::io` stuff (which thankfully isn't blocked on any language change!), and the allocator stuff once LLVM is out of the way (or we figure out how to make some unstable feature whose existence is gated on the LLVM provided).

❤️ 2



RalfJung commented on 12 Jan 2019 • edited ↕

Member 🗨️ ⋮

So, what is the current status of this? @alexcrichton, @faern, @Amanieu is there a path forward without being blocked on the big refactorings mentioned by @Ericson2314?

I have to admit I feel a bit overwhelmed reviewing this PR. I can certainly lend my expertise in concurrent algorithms, and review the "high-level bits", like how to implement locks and condvars and whatnot in terms of the low-level "parking" mechanism. But I know very little about futexes or the other low-level platform-specific primitives that form the foundation of this PR, and I do not feel confident enough to r+ the entire PR on my own. Anyone up for sharing this PR, preferably someone with expertise to review the part below the "parking" abstraction?
EDIT: Ah I see others have already raised similar concerns, thank you :)



Ericson2314 commented on 12 Jan 2019

Contributor 🗨️ ⋮

To be clear I don't mean to propose blocking *anything*, just making the process of including something like a `hashbrown` or `parking_lot` easier in the future.



faern commented on 14 Jan 2019

Contributor Author 🗨️ ⋮

I have not had the time to work on this for quite some time now. I also feel it's kind of blocked on deciding how to get the code into libstd. My interpretation is that different people want different solutions. The options we have discussed are:

1. Copy the code into this repository. This is what this PR does right now. But it's not fully done yet as I paused while alternatives were discussed. Has the downside that we create a lot of duplicated code to maintain.
2. Include `parking_lot` as a git submodule. I experimented with this. This looks doable, but requires some hacks to be merged into `parking_lot`. My understanding was that @Amanieu was not too happy about them.
3. Make `parking_lot` into `#[no_std]` and make libstd depend on it from crates.io. Seems almost impossible. `parking_lot` really needs some types from the standard library, for example `Instant`.



alexcrichton commented on 14 Jan 2019

Member 🗨️ ⋮

@faern's [comment](#) is what I believe the current status to be, and we can poll other @rust-lang/libs folks if they have opinions on this as well.

I personally feel that we *really* want to avoid duplication here. I think there's also some policy-ish issues to work through in terms of reviewing code. In any case I think reviewing this is certainly much broader than "this technically looks good as is".



KodrAus commented on 14 Jan 2019

Member 🗨️ ⋮

Over the longer term, what do we expect the relationship between `std` and `parking_lot` to be? Do we expect more of `parking_lot` to make its way into `std` in a publicly visible way (effectively deprecating the external crate) or do we always expect there to be a case for using `parking_lot` as an external library?



alexcrichton commented on 15 Jan 2019


Member 🗨️ ⋮

I suspect that we'll forever want to be more conservative in the exposed surface area of libstd than in the crates.io crate. The crates.io crate can be far more ambitious and having breaking changes whereas libstd basically can't. I do suspect, though, that we'll want to grow the APIs of the types in `std::sync` if we move over to `parking_lot` and everything goes well, there's likely some very useful functionality to expose that we just can't right now (const initialization might be one...)

Relationship-wise we currently have a pretty high degree of review of any code going into libstd in this repository. We don't do a great job, however, for submodules. Submodules like `libc/stdsimd` aren't too high-risk because their APIs are defined by someone else and the implementations are pretty rigorously tested too. Submodules like `compiler-rt`, however, probably receive far less review on implementation than they should. I'd be a little worried that we'd be expanding this to the synchronization system in libstd, which seems like it's naturally full of tricky code that would benefit from a higher-than-average level of scrutiny. This is something I'm not entirely sure how we'd solve just yet.


👍 1

FEBRUARY 2019



This comment has been hidden.

Show comment




faern commented on 14 Feb 2019

Contributor Author


I now pushed a first version of this. A PR on `parking_lot` to make it work as a git submodule exists over at [Amanieu/parking_lot#119](#).

Currently my change to `sync/rwlock.rs` triggers a compiler ICE, so I got a bit stuck.



This comment has been hidden.

Show comment




faern commented on 14 Feb 2019

Contributor Author

The ICE is now visible in the build failure log from highfive:

```
[00:07:18] error: internal compiler error: src/librustc/hir/def.rs:257: attempted .def_id{} on invalid def: NonE
[00:07:18] thread 'rustc' panicked at 'Box<Any>', src/librustc_errors/lib.rs:588:9
```

The error looks quite similar to [#57889](#), but I can't tell if it's the same or not.




faern reviewed on 14 Feb 2019

View changes

`.gitmodules`

Show resolved




Centril commented on 14 Feb 2019 · edited

Contributor

May also be due to [#58110](#). Similar to [#58253](#).


cc @oli-obk



faern commented on 14 Feb 2019

Contributor Author

I got around the ICE. Removing the now invalid `self.inner.destroy()` calls from the `RwLock` fixed it. Moving on...




faern commented on 15 Feb 2019

Contributor Author

I'm stuck in ICEs again :/


Maybe someone is able to help me sort it out. The same error is printed as last time. But obviously not from the same code. The commit that starts producing the ICE is this one where I try to implement the Mutex with the `parking_lot` primitives:

[faern@ 6f5d37f](#)



This comment has been hidden.


Show comment



oli-obk commented on 15 Feb 2019

Member

So the only thing I noticed with this ICE is that it occurs whenever I should have gotten a resolve error. Mostly typos or accessing methods that don't exist.




oli-obk commented on 15 Feb 2019

Member


One way that could help you is to build stage 0 on the last working commit. Then readd the new commits and run `./x.py test --keep-stage 0 --stage 1`, since the ICE is fixed on master, stage1 won't have it.

1




This comment has been hidden.

Show comment




faern force-pushed the `faern:add-parking-lot` branch from `8fc6eeb` to `5ad6a2a` on 19 Feb 2019




This comment has been hidden.

Show comment



faern force-pushed the `faern:add-parking-lot` branch from `5ad6a2a` to `8e4adf5` on 21 Feb 2019



faern commented on 21 Feb 2019 · edited

Contributor Author

Now I got a lot further. Thanks [@oli-obk](#) for the ICE fix and workaround.

MARCH 2019

src/libstd/io/stdio.rs

498 + #[stable(feature = "rust1", since = "1.0.0")]

499 + impl UnwindSafe for Stdout {}

500 + #[stable(feature = "rust1", since = "1.0.0")]

501 + impl RefUnwindSafe for Stdout {}

faern on 4 Mar 2019 Author Contributor

I had to manually implement these traits, since that was being done for us via the old `sys_common::ReentrantMutex`. The `parking_lot::ReentrantMutex` does not make things automatically `UnwindSafe`.

pitdicker on 4 Mar 2019 · edited Contributor

This does not change the current behavior, as the stdio code just ignored the poisoning of `ReentrantMutex`.

pitdicker on 4 Mar 2019 Contributor

If a panic happens in `StdoutLock::write` (or `flush`) it can keep the `RefCell` borrow count at 1. After a `catch_unwind` it can then be observed, causing the next use of `Stdout` to panic. Not sure if that is a problem.

Reply...

Unresolve conversation faern marked this conversation as resolved.

This comment has been hidden. Show comment

retep998 added the renotes label on 4 Mar 2019

Centril added I-nominated T-release labels on 5 Mar 2019

faern mentioned this pull request on 5 Mar 2019

Simplify io::Lazy #58768 Closed

Mark-Simulacrum removed I-nominated T-release labels on 6 Mar 2019

This comment has been hidden. Show comment

This comment has been hidden. Show comment

faern mentioned this pull request on 18 Mar 2019

Add SGX thread parker Amanieu/parking_lot#123 Merged

faern force-pushed the faern:add-parking-lot branch from d6037bf to 1727169 on 22 Mar 2019

faern mentioned this pull request on 22 Mar 2019

Add wasm thread parker Amanieu/parking_lot#124 Merged

faern commented on 22 Mar 2019 · edited Contributor Author

The current status is that we need `ThreadParker` implementations in `parking_lot` for the platforms that currently has something smarter than just a spin lock in `std::sys`. Otherwise they will fall back to a spin lock implementation in `parking_lot`. And that would likely not be a desired regression for any platform.

These are the missing platforms I have identified:

SGX - Just got merged (Add SGX thread parker Amanieu/parking_lot#123)

Redox - Add Redox thread parker Amanieu/parking_lot#125. Ping @jackpot51, you seem to have contributed a lot of the current locking primitives. Maybe you are interested in following this thread, and/or help?

Cloudabi - Add CloudABI thread parker Amanieu/parking_lot#126

Wasm (only when atomic support is enabled) - I'm still not sure if we need this one or not. But the implementation seemed fairly trivial. So I have started that over at Add wasm thread parker Amanieu/parking_lot#124

3

This comment has been hidden. Show comment

This comment has been hidden. Show comment

matprec mentioned this pull request on 26 Mar 2019

Implement demo subscriber matprec/demo-subscriber#1 Open

APRIL 2019

Trying commit `e3888e7` with merge `1002bb9` ...

bors added a commit that referenced this pull request on 8 Apr 2019

Auto merge of `#56410` - faern:add-parking-lot, r=<try> ✓ 1002bb9

bors commented on 9 Apr 2019 Member

☀️ Try build successful - checks-travis
Build commit: `1002bb9`

Centril commented on 9 Apr 2019 Contributor

@craterbot run mode=build-and-test

craterbot commented on 9 Apr 2019 Member

🔥 Experiment `pr-56410` created and queued.
🤖 Automatically detected try build `1002bb9`
🔍 You can check out [the queue](#) and [this experiment's details](#).
📖 Crater is a tool to run experiments across parts of the Rust ecosystem. [Learn more](#)

craterbot added `S-waiting-on-crater` and removed `S-waiting-on-review` labels on 9 Apr 2019

craterbot commented on 9 Apr 2019 Member

🏠 Experiment `pr-56410` is now running on agent `aws-3-tnp`.
📖 Crater is a tool to run experiments across parts of the Rust ecosystem. [Learn more](#)

craterbot commented on 11 Apr 2019 Member

🔥 Experiment `pr-56410` is completed!
📊 18 regressed and 9 fixed (50551 total)
📖 [Open the full report](#).
⚠️ If you notice any spurious failure [please add them to the blacklist!](#)
📖 Crater is a tool to run experiments across parts of the Rust ecosystem. [Learn more](#)

craterbot added `S-waiting-on-review` and removed `S-waiting-on-crater` labels on 11 Apr 2019

faern commented on 11 Apr 2019 • edited Contributor Author

This is an early analysis of the crater results.

Possible regressions

These crates behave the same locally as on crater. So might actually be regressions. Will dig into the details of these later.

- `blocking_object_pool-0.1.0`
- `delay-queue-0.2.0` - EDIT: Was a bug in `parking_lot`. Being fixed here: [Fix was_last_thread value in the timeout callback of park\(\)](#) Amanieu/parking_lot#129

Spurious failures - Nondeterministic failures

These crates fail randomly on stable and nightly Rust for me. Should likely be blacklisted on crater.

- `atlas-coverage-core-0.1.0`
- `chef_api-0.2.0`
- `ci_info-0.2.1`
- `fromhетен.plato.5b6cc48fa80eb224eb632dd3ad2353a24abfa146`

Spurious failures(?) - Misc

- `doryen-rs-0.1.0` - The error on crater is about the linker not being able to allocate memory. So likely spurious?
- `fibers_rpc-0.2.17` - Not able to make it fail locally. The error on crater seems to be about network disconnects, so spurious?
- `kiteconnect-0.2.4` - Not able to make it fail locally. The error on crater seems to be about network disconnects, so spurious?
- `poston-0.3.1` - Not able to make the test failing on crater fail locally. Another test fails, but that one consistently fails on stable as well. The test failing on crater seems to be connection/timeout based, so spurious?
- `nattforni.touch.e1c41176da5755eb366ecd05ae33fc8c3c93f655` - Not able to make it fail locally. The failing test is about

MAY 2019

panicking and accessing the environment)

Are there places where it was not possible?

Make Mutex, Condvar, RwLock and Once basically just thin wrappers over their parking_lot equivalents.

What about `park` and `unpark` in libstd? Seems a little funny to depend on `parking_lot` of all crates and then still hand-roll a parking mechanism here -- even more so since that is implemented on top of Condvar, which is now implemented on top of parking. ;)

1

faern commented on 5 May 2019

ContributorAuthor🗨️⋮

Are there places where it was not possible?

I'm not able to find any at the moment. So the answer seems to be: No.

What about `park` and `unpark` in libstd? Seems a little funny to depend on `parking_lot` of all crates and then still hand-roll a parking mechanism here -- even more so since that is implemented on top of Condvar, which is now implemented on top of parking. ;)

I have not even given it a thought actually. The initial scope of what I intended to do here was just the first bullet in my list (fix the synchronization primitives). The rest just came along nicely in the process. So if we feature creep this even more and re-write `park + unpark` we would need another crater run. I feel this could be left for a separate PR?

RalfJung commented on 5 May 2019

Member🗨️⋮

I feel this could be left for a separate PR?

Sounds good.

faern commented on 5 May 2019

ContributorAuthor🗨️⋮

@RalfJung You nerd sniped me into implementing this now. Looks like it can be done quite easy. However, I do not trust myself to write this type of code at this hour. So I will let tests run over night and then I'll think it over again when I find time.

However, it's up to the libs team to decide if they think this would justify another crater run. If they do we might want to hold off pushing it here anyway, in order to not delay this PR even further.

6

mark-i-m commented on 6 May 2019

Member🗨️⋮

Is there any benefit to doing it in this PR? My understanding is that it is already quite large to review and the park/unpark issue can be fixed independently.

2

alexcrichon commented on 6 May 2019

Member🗨️⋮

I do not personally mind one way or another whether park/unpark is changed in this PR. After landing this there's quite a few follow-ups we'll want to do like enhancing the APIs of the various types, making methods `const`, moving away from `RawMutex` where possible in libstd, etc, etc.

Reviewing this PR is quite trivial, and I suspect that even if changes were made to park/unpark it would still be quite trivial to review. The real meat is in parking_lot to review as it's basically an entire crate.

Basically @faern it's up to you whether you'd like to include the changes here.

In terms of review, I will attempt to set aside this Friday (2019-05-10) for reviewing the parking_lot crate.

3

Amanieu commented on 6 May 2019

Member🗨️⋮

@alexcrichon Feel free to ping me on IRC/discord if you have any questions during the review.

lucab commented on 7 May 2019

Contributor🗨️⋮

I am interested in having the additional RwLock traits from `lock_api` (in particular, `RawRwLockTimed` and `RawRwLockFair`) in stdlib, but I think that this PR is not exposing them.

Is that work supposed to happen in a different RFC/PR, and if so is that tracked somewhere already?

faern commented on 7 May 2019

ContributorAuthor🗨️⋮

Is that work supposed to happen in a different RFC/PR, and if so is that tracked somewhere already?


JUNE 2019

JULY 2019

...

NOVEMBER 2019

...



faern commented on 15 Jul 2019

Contributor

Author


⌵

...

@faern

Is there any chance this PR will land into Rust 1.38?

Quite unlikely. It's currently blocked on a lot of work being done inside `parking_lot`. See the review feedback from Alex further up.



dnrusakov commented on 16 Jul 2019 · edited

⌵


...

@faern

Is there any chance this PR will land into Rust 1.38?

Quite unlikely. It's currently blocked on a lot of work being done inside `parking_lot`. See the review feedback from Alex further up.

@faern, thanks!
What ticket/tickets I could subscribe to to track the progress around this parking lot initiative? I see there is [faern/parking_lot#1](#), but the conversation there stopped more than a month ago. Are there any other tickets to track?



faern commented on 16 Jul 2019

Contributor

Author


⌵

...

What ticket/tickets I could subscribe to to track the progress around this parking lot initiative? I see there is [faern/parking_lot#1](#), but the conversation there stopped more than a month ago. Are there any other tickets to track?


That is basically the entire review of `parking_lot` and discussion around what we need to fix. Some of those things have been addressed by separate PRs and those PRs should be linked from the "review PR". But no, nothing has happened for over a month. I fixed some low hanging issues. The current large blocker is probably that `parking_lot` needs a lot of documentation to explain how it works and why certain `unsafe` functions are actually safe etc.

🔗

 jonas-schievink mentioned this pull request on 29 Jul 2019

Panic broken on Windows XP #34538

🔒 Closed



KronicDeth commented on 30 Jul 2019 · edited

⌵

...

Will using `parking_lot` break any compatibility with `wasm32-unknown-unknown`? Due to `parking_lot`'s use of `Instant::now()`, it doesn't work for `wasm32-unknown-unknown`:

```
panicked at 'Time system call is not implemented by WebAssembly host', src/libstd/sys/wasm/mod.rs:302:13

Stack:

Error:
   at Module.__wbg_new_59cb74e423758ede (webpack:///.../pkg/spawn_chain.js?:181:26)
   at __wbg_new_59cb74e423758ede (http://localhost:8080/bootstrap.js:65:102)
   at console_error_panic_hook::hook::h8df71f3722ab18fc (wasm-function[194]:292)
   at core::ops::function::Fn::call::h42e98a3c826ddf0b (wasm-function[776]:3)
   at std::panicking::rust_panic_with_hook::h3f94f83752aaaaa5 (wasm-function[314]:265)
   at std::panicking::begin_panic::h6beec87bc67d7532 (wasm-function[613]:48)
   at std::sys::wasm::TimeSysCall::perform::h27f1627f17fac1d9 (wasm-function[721]:13)
   at std::sys::wasm::time::Instant::now::h34de02a8c4aa47a2 (wasm-function[770]:3)
   at std::time::Instant::now::h70048c3feeca6e27 (wasm-function[791]:1)
   at parking_lot_core::parking_lot::HashTable::new::h1383e3e985aa35d8 (wasm-function[208]:53)
```


I know about the `wasm_syscall` flag, but without rebuilding with `xargo` I can't enable that, so I'm worried that using `parking_lot` could make the primitives incompatible with `wasm32-unknown-unknown`. Also @alexcrichon seemed to imply that `wasm_syscall` was highly-experimental and I got the impression that we shouldn't depend on it to make `Instant::now()` work with rust-wasm.

Edit:

Opened [Amanieu/parking_lot#166](#) too.

👍 1


🔗

 SimonSapin mentioned this pull request on 19 Oct 2019

standard lazy types rust-lang/rfcs#2788


🔓 Open

🔗

 Thomasdezeuw mentioned this pull request on 29 Oct 2019

Windows cleanup tokio-rs/mio#1112

🔒 Merged




cormacrelf commented on 1 Nov 2019

⌵

...

@KronicDeth I made a PR for some single-threaded RawMutex/RawRwLock implementations that hopefully avoid the entire parking & timers subsystems altogether. Needs heavy review. [Amanieu/parking_lot#187](#)


🔗

 Centril mentioned this pull request on 4 Nov 2019

Target tier policy rust-lang/rfcs#2803

🔒 Merged

🔗

 faern mentioned this pull request on 16 Nov 2019

WHAT'S GOING ON?

IT'S A HUGE AMOUNT OF
NEW CODE

(new to std)

FUTEX()?

UNDOCUMENTED WINDOWS API?

STABILITY GUARANTEES

TOO MANY THINGS TO
DISCUSS AT ONCE

JANUARY 2020

FEBRUARY 2020

Not meeting the quality standards of the standard library and won't be accepted as a part of it right now. See the review further up in this thread ([#56410](#) comment)). Everything in that review has to be fixed first, then maybe this can proceed.

👍 1

Dylan-DPC commented on 15 Feb 2020

Member 🗨️ ⋮

@faern how about closing this PR and reopening it once the parking lot issues are settled? Would probably be easier that way

😊 1

jethrogb commented on 16 Feb 2020

Contributor 🗨️ ⋮

Easier for whom? The triage team or people trying to stay abreast of progress on this issue?

d-e-s-o mentioned this pull request on 17 Feb 2020

RwLock: support upgrades and downgrades #69240

Open

Dylan-DPC commented on 18 Feb 2020

Member 🗨️ ⋮

Everyone. The author, reviewers, triagists

faern commented on 18 Feb 2020

Contributor Author 🗨️ ⋮

@Dylan-DPC It's not like I have the time to contribute to this now nor likely soon. I feel like many of the outstanding parking_lot issues are partially out of my hands / not related to integrating parking_lot into libstd to begin with.

For me personally it won't really be "easier" if this was closed. But nor would it be harder. Whenever a new PR is opened again we have to navigate back and forth between two PR threads, but that is going to be doable. Maybe better than having a PR open forever that no one contributes towards.

@jethrogb has a point about that people might want to stay up to date with this and not miss when/if it finally starts moving again. But they can still subscribe to this PR and if a new one is opened, someone can just post a comment in this PR linking to the new one.

👍 7

faern closed this on 18 Feb 2020

Dylan-DPC commented on 18 Feb 2020

Member 🗨️ ⋮

Yes @faern understandable. But even if kept open and it is ready to be merged, it will increase the number of conflicts this has with other prs.

shepmaster commented on 19 Feb 2020

Member 🗨️ ⋮

a new PR is opened again

You can reopen this one at that time.

👍 1

comex commented on 19 Feb 2020 • edited

Member 🗨️ ⋮

Incidentally, on Darwin, pthread mutexes use priority inheritance, while parking_lot does not intend to implement it. As long as that's true, switching the default mutex to parking_lot would be a regression in some ways.

eddyb commented on 19 Feb 2020

Member 🗨️ ⋮

You can reopen this one at that time.

Note of caution (we should have this somewhere linkable IMO, or GitHub could just make it simpler on their side): always reopen before pushing to the branch, otherwise GitHub will hide the option to reopen.

However, even if you pushed, you can still recover by finding the commit the branch was previously on, and force-pushing that, which will unlock reopening the PR.

👍 7 ❤️ 1

RalfJung mentioned this pull request on 25 Jun 2020

Feature request: Make Mutex::new/RwLock::new const. #73714

Open

OUT OF ENERGY

CONCLUSION

LARGE CHANGES PUSH
THINGS FORWARD
EVEN WHEN THEY FAIL



CHAPTER 3

STUCK

DEADLOCK

FAILED ATTEMPT
BECOMES AN OBSTACLE

CONTEXT

CONCLUSION



CHAPTER 4

TINY STEPS

TINY STEPS

Obstacle 1

STABILITY GUARANTEES

Relax promises about condition variable. #76932

Edit Open with ▾




 Merged bors merged 1 commit into rust-lang:master from fusion-engineering-forks:condvar-promise on 25 Sep 2020

 Conversation 9  Commits 1  Checks 11  Files changed 1

+5 -11 

Changes from all commits ▾ File filter ▾ Conversations ▾ Jump to ▾  ▾

0 / 1 files viewed ⓘ Review changes ▾

▼  16  library/std/src/sync/condvar.rs 

☐ Viewed ...

Obstacle 2


MOVABLE MUTEXES

Obstacle 2

MOVABLE MUTEXES

MicrosoftDocs / sdk-api

 Code

 Pull requests **36**

71

– An SRW lock cannot be moved or copied.

71

+ An SRW lock cannot be moved or copied while in use.

72

72

73


+ An SRW lock with no waiting threads is in its initial state and can be moved.



Merged

Unbox mutexes and condvars on some platforms #77380

 Merged

bors merged 12 commits into `rust-lang:master` from `fusion-engineering-forks:unbox-the-mutex`  on 4 Oct 2020

 Conversation 16

 Commits 12

 Checks 11

 Files changed 20



m-ou-se commented on 1 Oct 2020 • edited ▼

Member




Obstacle 3


NEW O.S. PRIMITIVES


Use futex-based thread::park/unpark on Linux. #76919


Edit Open with ▾


 Merged bors merged 8 commits into `rust-lang:master` from `fusion-engineering-forks:thread-parker`  on 1 Oct 2020

 Conversation 14

 Commits 8

 Checks 11

 Files changed 7

+276 -112 




m-ou-se commented on 19 Sep 2020 • edited ▾ Member 😊 ...


Reviewers 


Add fast WaitOnAddress-based thread parker for Windows. #77618


Edit Open with ▾


 Merged bors merged 8 commits into `rust-lang:master` from `fusion-engineering-forks:windows-parker`  on 14 Dec 2020

 Conversation 28

 Commits 8

 Checks 11

 Files changed 5

+301 -0 



m-ou-se commented on 6 Oct 2020 Member 😊 ...

Reviewers 

CONCLUSION

SMALLER CHUNKS
ARE EASIER TO CHEW



CHAPTER 5

MOVING FORWARD

PROCESS CHANGES?

MCPS:

MAJOR CHANGE PROPOSALS

Library API team

Designing and maintaining the standard library API and guarding its stability

Members



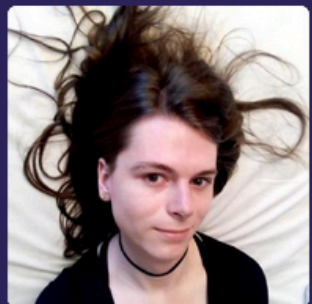
Amanieu d'Antras

GitHub: [Amanieu](#)



David Tolnay

GitHub: [dtolnay](#)



Mara Bos

GitHub: [m-ou-se](#)



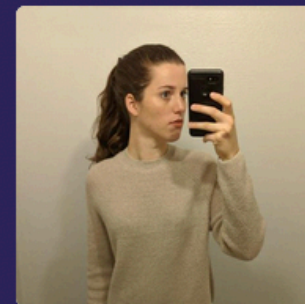
Andrew Gallant

GitHub: [BurntSushi](#)



Josh Triplett

GitHub: [joshtripllett](#)



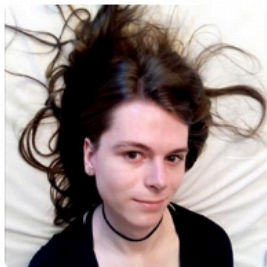
Jane Lusby

GitHub: [yaahc](#)

Library team

Managing and maintaining the Rust standard library and official rust-lang crates

Members



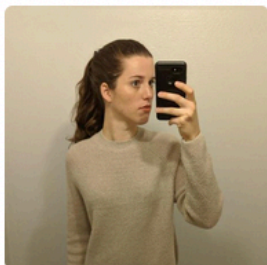
Mara Bos

GitHub: [m-ou-se](#)
Team leader



Josh Stone

GitHub: [cuviper](#)



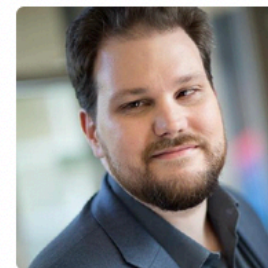
Jane Lusby

GitHub: [yaahc](#)



Amanieu d'Antras

GitHub: [Amanieu](#)



Josh Triplett

GitHub: [joshtripllett](#)

CONTRIBUTORS

KEEP MAKING
SMALL STEPS

THANK YOU

✨ Looking for more Rust on a pink background? ✨ https://twitter.com/m_ou_se ✨