Core WG

Charter: https://github.com/tock/tock/blob/master/doc/wg/core/README.md
TockWorld 7 | June 2024

Goals

The goals of the Tock Core Working Group are to:

- Shepherd Tock project.
- Define and communicate project direction.
- Establish WGs and delegate responsibilities.
- Ensure WGs are accountable to their responsibilities
- Coordinate decisions that affect more than one working group.
- Facilitate communication and consensus among WGs.
- Coordinate project-wide changes to teams, structures, or processes.

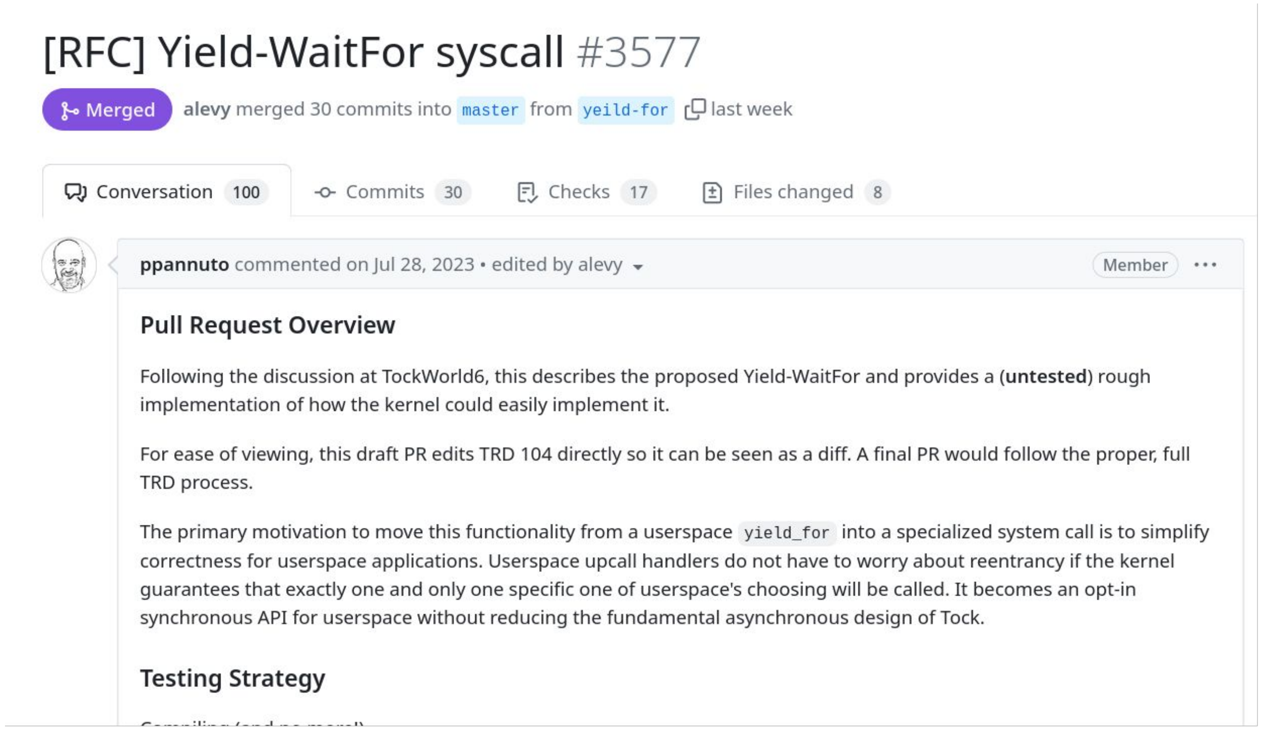
Members:

- Hudson Ayers,
- Brad Campbell
- Branden Ghena
- Philip Levis
- Amit Levy (Chair)
- Pat Pannuto
- Alexandru Radovici
- Leon Schuermann
- Johnathan Van Why

New Working Groups

- 2023-08-11 Network working group
- 2024-02-02 Documentation

Blocking I/O (YieldForWait)



[RFC] Yield-WaitFor syscall #3577

Merged alevy merged 30 commits into `master` from `yeild-for` last week

ppannuto commented on Jul 28, 2023 • edited by alevy

Pull Request Overview

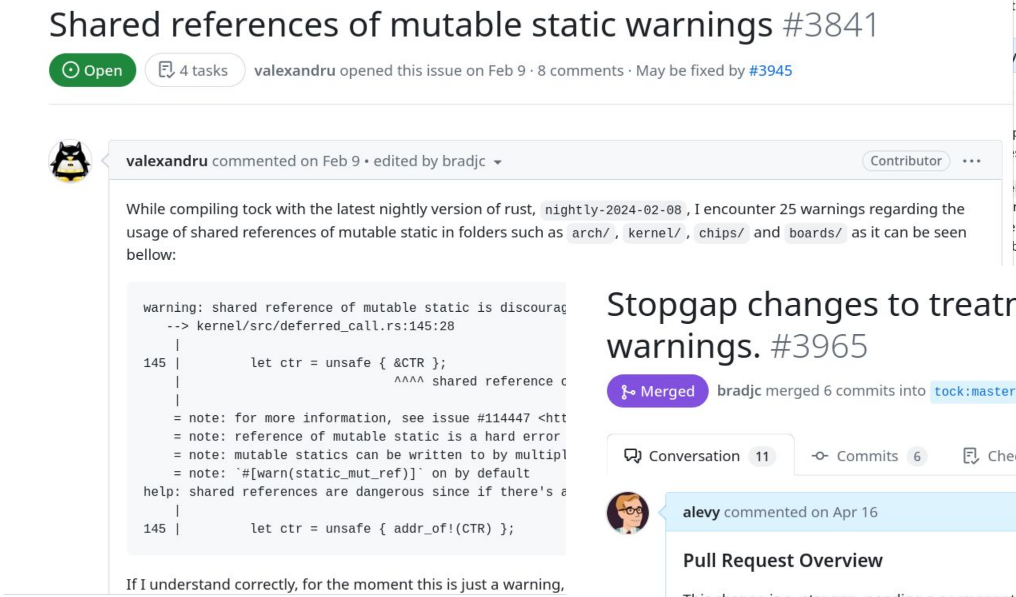
Following the discussion at TockWorld6, this describes the proposed Yield-WaitFor and provides a (**untested**) rough implementation of how the kernel could easily implement it.

For ease of viewing, this draft PR edits TRD 104 directly so it can be seen as a diff. A final PR would follow the proper, full TRD process.

The primary motivation to move this functionality from a userspace `yield_for` into a specialized system call is to simplify correctness for userspace applications. Userspace upcall handlers do not have to worry about reentrancy if the kernel guarantees that exactly one and only one specific one of userspace's choosing will be called. It becomes an opt-in synchronous API for userspace without reducing the fundamental asynchronous design of Tock.

Testing Strategy

`static mut`



Shared references of mutable static warnings #3841

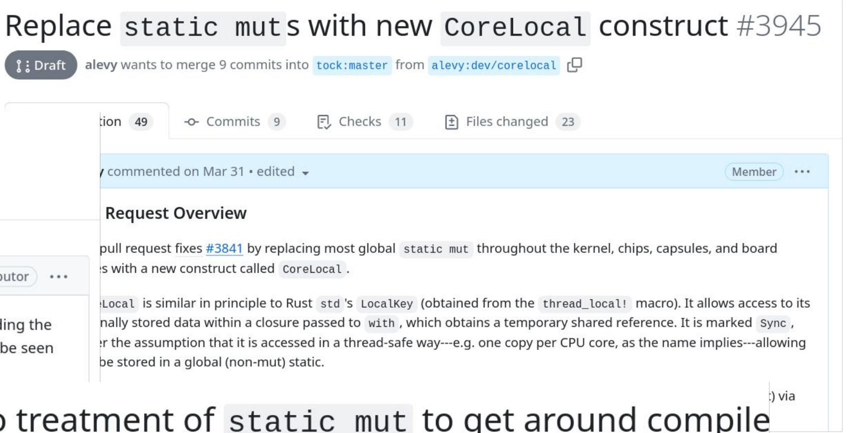
Open · 4 tasks · valexandru opened this issue on Feb 9 · 8 comments · May be fixed by #3945

valexandru commented on Feb 9 • edited by bradjc · Contributor

While compiling tock with the latest nightly version of rust, `nightly-2024-02-08`, I encounter 25 warnings regarding the usage of shared references of mutable static in folders such as `arch/`, `kernel/`, `chips/` and `boards/` as it can be seen bellow:

```
warning: shared reference of mutable static is discourag
   --> kernel/src/deferred_call.rs:145:28
    |
145 |         let ctr = unsafe { &CTR };
    |                            ^^^^ shared reference c
    |
    = note: for more information, see issue #114447 <htt
    = note: reference of mutable static is a hard error
    = note: mutable statics can be written to by multipl
    = note: `#[warn(static_mut_ref)]` on by default
help: shared references are dangerous since if there's a
    |
145 |         let ctr = unsafe { addr_of!(CTR) };
```

If I understand correctly, for the moment this is just a warning,



Replace `static mut`s with new `CoreLocal` construct #3945

Draft · alevy wants to merge 9 commits into `tock:master` from `alevy:dev/corelocal`

tion 49 · Commits 9 · Checks 11 · Files changed 23

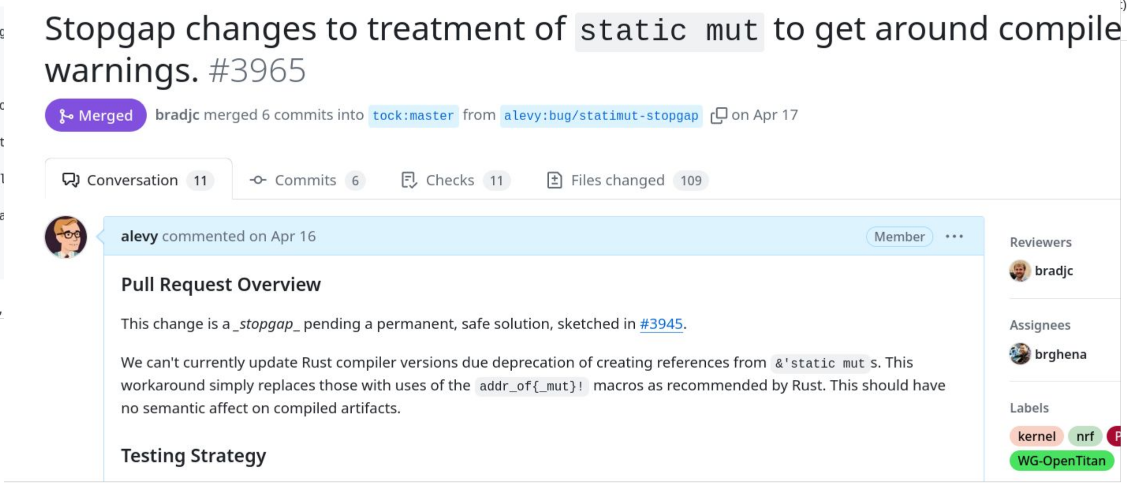
commented on Mar 31 • edited · Member

Request Overview

pull request fixes #3841 by replacing most global `static mut` throughout the kernel, chips, capsules, and board s with a new construct called `CoreLocal`.

`Local` is similar in principle to Rust `std`'s `LocalKey` (obtained from the `thread_local!` macro). It allows access to its nally stored data within a closure passed to `with`, which obtains a temporary shared reference. It is marked `Sync`, r the assumption that it is accessed in a thread-safe way---e.g. one copy per CPU core, as the name implies---allowing be stored in a global (non-mut) static.

) via



Stopgap changes to treatment of `static mut` to get around compile warnings. #3965

Merged · bradjc merged 6 commits into `tock:master` from `alevy:bug/statimut-stopgap` on Apr 17

Conversation 11 · Commits 6 · Checks 11 · Files changed 109

alevy commented on Apr 16 · Member

Pull Request Overview

This change is a _stopgap_ pending a permanent, safe solution, sketched in #3945.

We can't currently update Rust compiler versions due deprecation of creating references from `&'static mut`s. This workaround simply replaces those with uses of the `addr_of{_mut}!` macros as recommended by Rust. This should have no semantic affect on compiled artifacts.

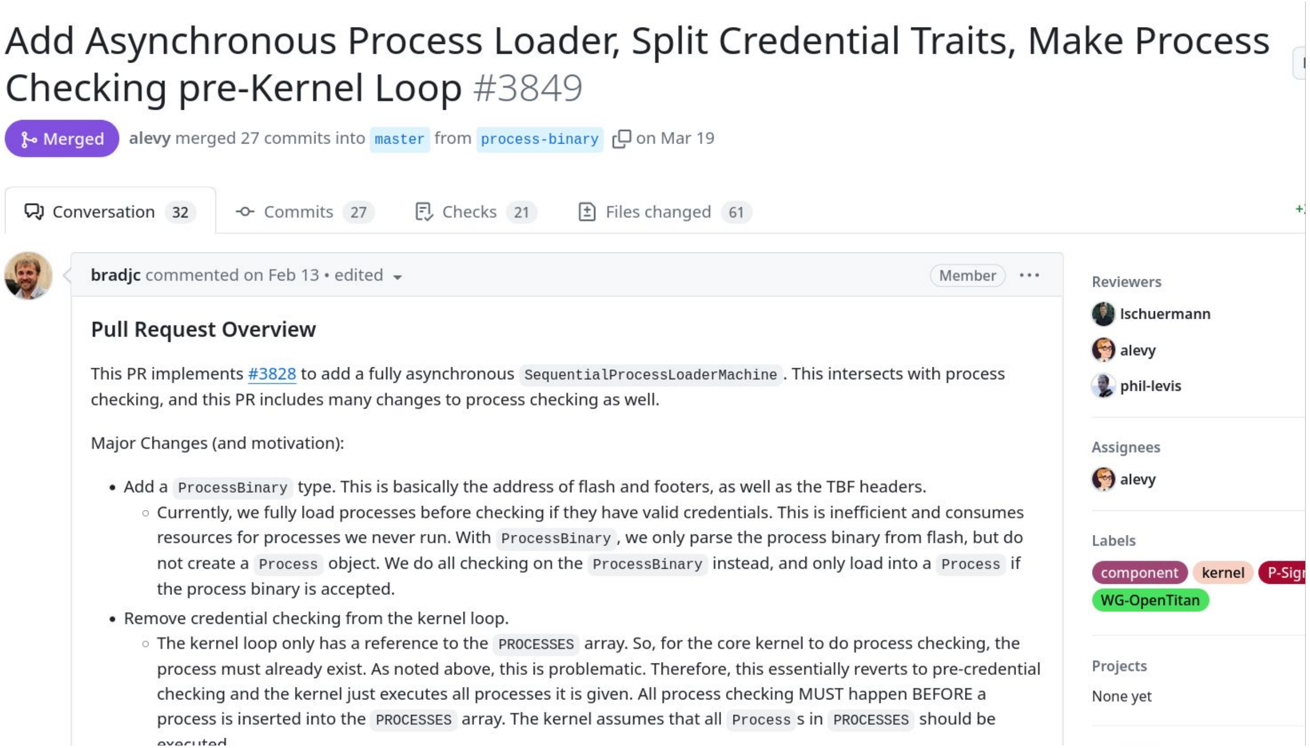
Testing Strategy

Reviewers: bradjc

Assignees: brghena

Labels: kernel, nrf, WG-OpenTitan

Asynchronous Process Loader



Add Asynchronous Process Loader, Split Credential Traits, Make Process Checking pre-Kernel Loop #3849

Merged **alevy** merged 27 commits into `master` from `process-binary` on Mar 19

Conversation 32 · Commits 27 · Checks 21 · Files changed 61

bradjc commented on Feb 13 • edited · Member

Pull Request Overview

This PR implements #3828 to add a fully asynchronous `SequentialProcessLoaderMachine`. This intersects with process checking, and this PR includes many changes to process checking as well.

Major Changes (and motivation):

- Add a `ProcessBinary` type. This is basically the address of flash and footers, as well as the TBF headers.
  - Currently, we fully load processes before checking if they have valid credentials. This is inefficient and consumes resources for processes we never run. With `ProcessBinary`, we only parse the process binary from flash, but do not create a `Process` object. We do all checking on the `ProcessBinary` instead, and only load into a `Process` if the process binary is accepted.
- Remove credential checking from the kernel loop.
  - The kernel loop only has a reference to the `PROCESSES` array. So, for the core kernel to do process checking, the process must already exist. As noted above, this is problematic. Therefore, this essentially reverts to pre-credential checking and the kernel just executes all processes it is given. All process checking MUST happen BEFORE a process is inserted into the `PROCESSES` array. The kernel assumes that all `Process`s in `PROCESSES` should be executed

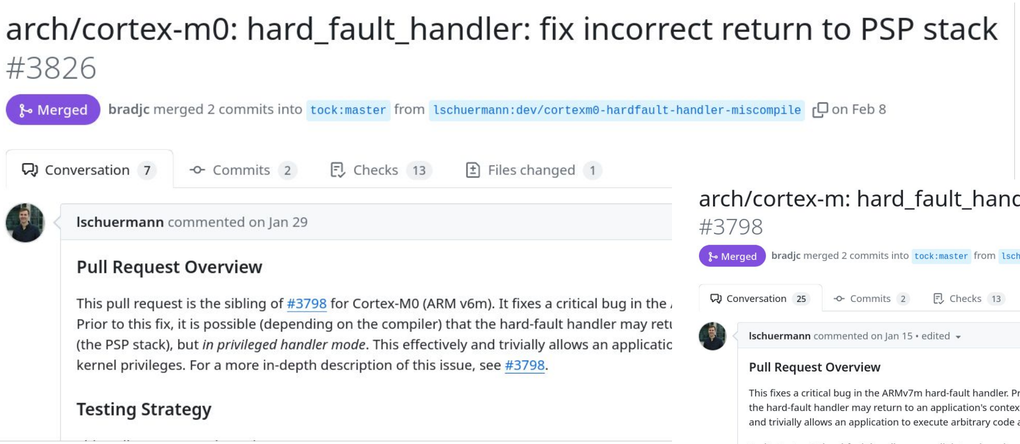
Reviewers
lschuermann
alevy
phil-levis

Assignees
alevy

Labels
component · kernel · P-Sign · WG-OpenTitan

Projects
None yet

Significant bug fixes in interrupt handler



arch/cortex-m0: hard_fault_handler: fix incorrect return to PSP stack #3826

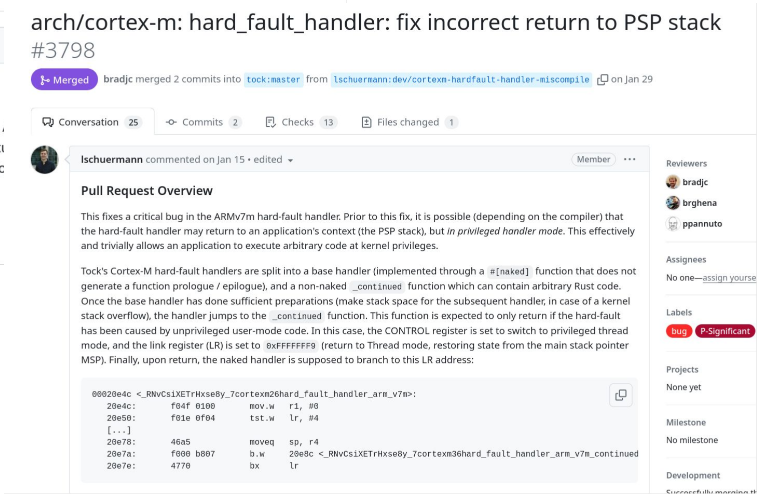
Merged bradjc merged 2 commits into `tock:master` from `lschuermann:dev/cortexm0-hardfault-handler-miscompile` on Feb 8

lschuermann commented on Jan 29

Pull Request Overview

This pull request is the sibling of #3798 for Cortex-M0 (ARM v6m). It fixes a critical bug in the
Prior to this fix, it is possible (depending on the compiler) that the hard-fault handler may retu
(the PSP stack), but *in privileged handler mode*. This effectively and trivially allows an applicatic
kernel privileges. For a more in-depth description of this issue, see #3798.

Testing Strategy



arch/cortex-m: hard_fault_handler: fix incorrect return to PSP stack #3798

Merged bradjc merged 2 commits into `tock:master` from `lschuermann:dev/cortexm-hardfault-handler-miscompile` on Jan 29

lschuermann commented on Jan 15 • edited

Pull Request Overview

This fixes a critical bug in the ARMv7m hard-fault handler. Prior to this fix, it is possible (depending on the compiler) that the hard-fault handler may return to an application's context (the PSP stack), but *in privileged handler mode*. This effectively and trivially allows an application to execute arbitrary code at kernel privileges.

Tock's Cortex-M hard-fault handlers are split into a base handler (implemented through a `#[naked]` function that does not generate a function prologue / epilogue), and a non-naked `_continued` function which can contain arbitrary Rust code. Once the base handler has done sufficient preparations (make stack space for the subsequent handler, in case of a kernel stack overflow), the handler jumps to the `_continued` function. This function is expected to only return if the hard-fault has been caused by unprivileged user-mode code. In this case, the CONTROL register is set to switch to privileged thread mode, and the link register (LR) is set to `0xFFFFFFF9` (return to Thread mode, restoring state from the main stack pointer MSP). Finally, upon return, the naked handler is supposed to branch to this LR address:

```
00020e4c <_RNvCsiXETrHxse8y_7cortexm26hard_fault_handler_arm_v7m>:
   20e4c:       f04f 0100       mov.w   r1, #0
   20e50:       f01e 0f04       tst.w   lr, #4
   [...]
   20e78:       46a5            moveq   sp, r4
   20e7a:       f000 b807       b.w     20e8c <_RNvCsiXETrHxse8y_7cortexm36hard_fault_handler_arm_v7m_continued
   20e7e:       4770            bx      lr
```

Restructure libtock-c

- Core `libtock` and separate `libtock-sync`
- Standardized namespaced functions
- Standardized mapping of system calls to C interfaces

2024 and Beyond

More working groups!

- Userland
- Testing and development infrastructure
- Kernel

Code-size

- Measure
- Inform
- Optimize