Bindgen improvements for Rust for Linux

John Baublitz
Principal Software Engineer

Red Hat

Talk outline:

- Macro expansion in bindgen

  Issue #753

- Raw pointer access for bitfields

  Issue #2674

- Safe and unsafe conversions for Rustified enums

  Issue #2646

Macro expansion in bindgen

Problem

Complex macros that evaluate to constants were not included

```
#define CONSTANT 5 // expanded

#define CONSTANT UINT32_C(5) // not expanded
```

Solution

Use clang to evaluate macro invocations in temporary file

- Capture the name of the macro
- Create temporary file invoking the macro
- Use clang to evaluate the invocation and return the result
  - Nothing if it does not evaluate to a literal
  - Literal otherwise

Usage

Opt-in

- `--clang-macro-fallback`
- `--clang-macro-fallback-build-dir=DIR`

Performance

Final performance: 3 - 5 seconds

- Performance testing was run on a consolidated header of all kernel constants created by Vadzim Dambrouski

Performance

Final performance: 3 - 5 seconds

- Initial prototype created a new file and `TranslationUnit` for each macro that couldn't be parsed by cexpr
  - Performance was unacceptably bad (35m for consolidated header)
  - Likely due to all of the IO required

Performance

Final performance: 3 - 5 seconds

- Second prototype reparsed the header for each macro, but reused the `TranslationUnit` so only one temporary file needed to be written to the filesystem
  - Started with suggestion from ChatGPT from Vadzim Dambrouski
    - ChatGPT seems to have hallucinated; good starting point
    - Claimed that headers in reused translation units were not reparsed
    - Did not appear to be true based on performance
  - Performance was much better; still not good enough

Performance

Final performance: 3 - 5 seconds

- Final prototype reused the `TranslationUnit` and took advantage of precompiled headers to avoid parsing the header for each macro
  - This performance was acceptable
  - Slight compilation time increase

Development hurdles

- Bug was filed against original PR by `@SeleDreams`
- Two problems with the original PR
  - Clang will only accept one precompiled header; I used multiple
    - Other precompiled headers are ignored
  - `CFLAGS` were not passed into `TranslationUnit`
    - Breakage for `#include` among other things

Status

Released in 0.70.0

- Potential future work
  - Add clang API to maintain macro information on parse

Raw pointer access for bitfields

Problem

In some cases, accessors for bitfields were not sufficient due to Rust aliasing rules

```
#[repr(C)]
struct foo {
    // this is a bitfield
    field: u8,
    mtx: mutex_t,
    // mutex protected, bitfield
    protected: u8
}
```

```
// this is how you reference the type normally
struct FooWrapper(UnsafeCell<foo>);
```

Example from `@tgross35`

Problem

In some cases, accessors for bitfields were not sufficient due to Rust aliasing rules

Consider the case in the previous data structure:

- Context A creates `&foo` and access `field`
- Preemption hits, context blocks `mtx`, changes protected, and unlocks
- Context A resumes and `&foo` points to changed data, rustc has no way of knowing about it
- Context A accesses `field`

Example from `@tgross35`

Solution

Add accessors that operate on raw pointers

```rust
#[inline]
pub unsafe fn available_raw(
    this: *const Self
) -> ::std::os::raw::c_uint {
    ::std::mem::transmute(__BindgenBitfieldUnit::raw_get(
        addr_of!((*this)._bitfield_1),
        0usize,
        1u8,
    ) as u32)
}
```

```rust
#[inline]
pub unsafe fn set_available_raw(
    this: *mut Self, val: ::std::os::raw::c_uint
) {
    let val: u32 = ::std::mem::transmute(val);
    __BindgenBitfieldUnit::raw_set(
        addr_of!((*this)._bitfield_1),
        0usize, 1u8, val as u64
    )
}
```

Usage

Automatic

Status

Rust for Linux code review done, waiting on maintainer code review

Safe and unsafe conversions for Rustified enums

Problem

C allows any constant to be passed as an enum; no checks on validity of value

```c
#include <stdio.h>

enum my_enum {
	CONSTANT1 = 1,
	CONSTANT2 = 2,
};
```

```c
int takes_enum(enum my_enum e) {
	printf("%d\n", e);
}

int main() {
	takes_enum(2);
	takes_enum(3);
}
```

Example from @tgross35

Solution

Add safe conversions from raw integers to Rust enums

```
pub const foo_one: foo = 1;
pub const foo_two: foo = 2;
pub const foo_three: foo = 3;
pub type foo_ctype = ::std::os::raw::c_uint;
```

```
#[repr(u32)]
#[derive(Debug, Copy, Clone, Hash, PartialEq, Eq)]
pub enum foo {
    one = 1,
    two = 2,
    three = 3,
}
```

Example from @tgross35

Solution

Add safe conversions from raw integers to Rust enums

```
struct FooError(foo_ctype);

impl TryFrom<foo_ctype> for foo {
    type Error = FooError;
    fn try_from(value: foo_ctype) -> Result<foo, FooError> {
        match value {
            1 => Ok(foo::one),
            2 => Ok(foo::two),
            3 => Ok(foo::three),
            _ => Err(FooError(value)),
        }
    }
}
```

Example from @tgross35

Solution

Add safe conversions from raw integers to Rust enums

```
impl foo {
    const unsafe fn from_ctype_unchecked(value: foo_ctype) -> Self {
        std::mem::transmute(value)
    }
}
```

Example from `@tgross35`

Usage

Opt-in, breaking change

- `--rustified-enum=[REGEX](=([non_exhaustive|try_from_raw|from_raw_unchecked]),*)?`
- `--rustified-enum-non-exhaustive-enum` merged into `--rustified-enum`

Development hurdles

- CLI (`REGEX[=(option),*]`)
  - Some prior art for argument format but very uncommon and needed extension
- Needed to merge `rust` and `rust_non_exhaustive`
  - Previously separate options via CLI
- A lot of the internals for enum generation assume exactly one type, translated or untranslated
  - Needed to extend internals to handle receiving both

Status

In progress

Discussion questions:

- Currently the PR adds constants representing enum values to code that previously did not have them
  - Thoughts?
- There are some cases where new constant names collide with existing constant names
  - Thoughts about namespacing?
  - `[ENUM_NAME]_[VARIANT]` does not appear to be sufficient
    - There are still collisions with existing constants
  - Perhaps `_cval` suffix?

Thank you

Referenced PRs

- https://github.com/rust-lang/rust-bindgen/pull/2779
- https://github.com/rust-lang/rust-bindgen/pull/2823
- https://github.com/rust-lang/rust-bindgen/pull/2876
- https://github.com/rust-lang/rust-bindgen/pull/2908


linkedin.com/company/red-hat


youtube.com/user/RedHatVideos


facebook.com/redhatinc


twitter.com/RedHat

Red Hat