Unsafe and strerror() - impossible to fix?

help

jbe March 13, 2023, 12:54pm 1

Inspired by a comment on a supposedly unnecessary `Mutex` in my code, I want to review all `unsafe` occurrences in `mmtkvdb` and add proper `// SAFETY` comments. I got already stuck at the first occurrence of `unsafe`:

```
/// Converts an error code from LMDB into a `Result`
fn check_err_code(code: c_int) -> io::Result<()> {
   if code > 0 {
       Err(io::Error::from_raw_os_error(code))
   } else if code < 0 {
       Err(io::Error::new(
           io::ErrorKind::Other,
           unsafe { CStr::from_ptr(lmdb::mdb_strerror(code)) }
               .to_str()
               .unwrap_or("unknown error (LMDB error string is not valid UTF-8)"),
       ))
   } else {
       Ok(())
   }
}
```

It should always be okay to call `mdb_strerror(code)` with any integer (maybe it would be safer to demand that the error code is valid, though). However, even though not *explicitly* documented in the LMDB API, `mdb_strerror(code)` doesn't seem to be thread-safe as in it could "return a pointer to an internal static buffer that could be overwritten by calls from other threads". This is because it internally directly uses `strerror()` without synchronizing (source), and `strerror()` is specified to "return a pointer to an internal static buffer that could be overwritten by calls from other threads" (at least on FreeBSD).

So how to fix this? I guess I could use a `static Mutex` to synchronize *my* calls of `mdb_strerror`. But this doesn't fix my problem as any other crate in any other thread could invoke `strerror()`, right?

Is this impossible to fix?

Is it okay to use Mutex<bool>?

Cerber-Ursi March 13, 2023, 1:10pm 2

I guess this might be as unfixable as the `set_var` unsoundness (for the exact same reason).

3 Likes

jbe March 13, 2023, 1:10pm 3

Okay, but… what do I do now? I think the only "solution" is to refrain from using the `mdb_strerror` function. I guess I could simply return an integer error code (which isn't nice), or I copy the table from the LMDB source into my own source? Unfortunately that table is declared `static`, so I really would need to copy it from the source (unless I patch LMDB).

I found this entry in LMDB's bugzilla. It basically states that `mdb_strerror` *is* thread-safe, but only if you call it with LMDB's error codes (and not system error codes or invalid/unknown codes). This is also what I would deduce from the source code here.

It's not documented in the API specification though.

However, if I could rely on that, then I could simply mark my own `fn check_err_code` as `unsafe` and demand the `code` must be a valid code. If it's a system error, then I branch into this anyway:

jbe:

```
       if code > 0 {
          Err(io::Error::from_raw_os_error(code))
```

The only way where `strerror` could actually be called is when the error code is invalid.

So to conclude:

My function `check_err_code` *is* unsound. It should be marked `unsafe` and demand that `code` is a valid error code. Luckily that (private) function can't be called with invalid error codes *(edit: from outside the module)*, so the overall module should be sound.

My "solution":

```
 /// Converts an error code from LMDB into a `Result`
-fn check_err_code(code: c_int) -> io::Result<()> {
+///
+/// # Safety
+///
+/// * If `code` is negative, it must be a valid LMDB error code.
+/// * `lmdb::mdb_strerror` is assumed to be thread-safe for valid negative LMDB
+///   error codes. This is currently not guaranteed by LMDB's API
+///   documentation. See also: `https://bugs.openldap.org/show_bug.cgi?id=8361`
+unsafe fn check_err_code(code: c_int) -> io::Result<()> {
    if code > 0 {
        Err(io::Error::from_raw_os_error(code))
    } else if code < 0 {
        Err(io::Error::new(
            io::ErrorKind::Other,
+            // SAFETY: because `code` is negative and an existent LMDB error
+            // code, `lmdb::mdb_strerror` returns a pointer to a static
+            // C string (not guaranteed by the API docs, but believed to not
+            // change in future versions of LMDB)
            unsafe { CStr::from_ptr(lmdb::mdb_strerror(code)) }
                .to_str()
                .unwrap_or("unknown error (LMDB error string is not valid UTF-8)"),
```

jbe March 13, 2023, 2:50pm 4

Now this is also scary:

```
#ifdef _WIN32
       /** HACK: pad 4KB on stack over the buf. Return system msgs in buf.
        *        This works as long as no function between the call to mdb_strerror
        *        and the actual use of the message uses more than 4K of stack.
        */
#define MSGSIZE        1024
#define PADSIZE        4096
       char buf[MSGSIZE+PADSIZE], *ptr = buf;
#endif
```

A workaround would be to copy the returned C string as soon as possible, instead of passing a `&str` to `std::io::Error::new`. However, this problem also doesn't seem to affect `mmtkvdb` because that hackery `buf` is only used for positive or non-existent error codes. The positive error codes are passed to `std::io::Error::from_raw_os_error`, which should be sound, of course. And negative non-existent error codes should not occur (and for which reason the `check_err_code` function will be declared `unsafe` in future, as explained in my previous post).

My overall fear is that similar "hacks" could be found elsewhere in LMDB too.

Is it okay to use Mutex<bool>?

Is it okay to use Mutex<bool>?

SebastianJL March 14, 2023, 12:12pm 5

Couldn't you also `assert!(code < 0)`? That way you don't have to mark the function unsafe and just document that it panics for non-negative numbers.

That way you don't get UB but a panic instead. Much more visible.

jbe March 14, 2023, 12:16pm 6

Actually the `code < 0` case is the one that makes problems:

- If `code == 0`, then `Ok(())` is returned.
- If `code > 0`, then `Err(io::Error::from_raw_os_error(code))` is returned.
- Only if `code < 0`, then `mdb_strerror` is called.

Luckily, if the `code` is an existent negative error code, `mdb_strerror` supposedly returns a static C string (even though not guaranteed in the API documentation). The problem is a negative error code which does *not* exist. Then `mdb_strerror` falls back to call the thread-unsafe `strerror` function.

SebastianJL March 14, 2023, 12:18pm 7

Ah sorry yes■ Ignore what I said then.

SebastianJL March 14, 2023, 12:19pm 8

Wait, so how do you know you have a valid error code to pass to the function?

jbe March 14, 2023, 12:20pm 9

I don't know. That is why I propose to make `check_err_code` unsafe and demand:

```
/// # Safety
///
/// * If `code` is negative, it must be a valid LMDB error code.
```

By marking `check_err_code` as being `unsafe` and documenting the requirements, the caller must ensure that `code` is valid (if negative), e.g. by only passing `code`s to `check_err_code` which have been returned by LMDB.

SebastianJL March 14, 2023, 12:24pm 10

Yeah, sounds like a reasonable solution then

1 Like

jbe March 14, 2023, 12:27pm 11

jbe:

> […] the caller must ensure that `code` is valid (if negative) […]

Currently, this was always ensured already, because the only places where `check_err_code` was called is when passing an LMDB error code to the function. Moreover, `check_err_code` is private, so it wasn't possible to call this function with any non-valid error code. Nonetheless, it still should be marked `unsafe` to avoid future use which might result in UB.

1 Like

SebastianJL March 14, 2023, 12:29pm 12

Yes yes, I agree. I meant your proposed change earlier with unsafe and the safety documentation sounds reasonable.

system Closed June 12, 2023, 12:30pm 13

This topic was automatically closed 90 days after the last reply. We invite you to open a new topic if you have further questions or comments.

Related topics

Topic		Replies	Views	Activity
Avoiding unsafe when using a C library: LMDB help	18	1141	August 15, 2022	
Impossible to safely wrap thread-unsafe FFI?	17	2311	January 12, 2023	
How unsafe is mmap, for a database type lib? help	69	2066	September 20, 2023	
Safely wrapping libc's `__errno_location` function code review	3	1028	April 26, 2023	
Is there a checklist for review of unsafe code? help	18	1822	September 3, 2019	