Imposing memory safety in C

while not rewriting to Rust

Maria Matejka · Feb 02, 2025

cz.nic CZ DOMAIN REGISTRY

The Quest

- years old C codebase
- memory safety concerns
- better rewrite to Rust?

Requirements

- keep what works
- automatable refactoring
- lead developers the right way
- allow footguns but make them obvious

It should be hard to write bad code that passes … but not impossible.

What should happen with unsafe code

- obvious build error
- static analyzer error
- unit test error
- stick out in plain sight

What should happen with unsafe code

- obvious build error
- static analyzer error
- unit test error
- stick out in plain sight

Innocent return must be actually innocent.

The easy part: defaults

- local is good, global is bad
- `const` everywhere (what about adding `mutable` to C, SC22/WG14?)
- nonpure function must have a reason to exist
- no `void *`, anywhere

Getting rid of globals

- context → pass it as a context in an argument
- read-only global info → explicit access
- really shared data → *locked* explicit access

Void pointer eradication

- use unions instead
- generated code is safer than `void *` (hello, M4)
- own linked list / hashtable / … type for every member type
- no plain typecast, anywhere (pack these into macros)

Locally acquired resources

- explicit releasing is unreliable (return from a locked context)
- cleanup hooks → can be packed in macros
- end-of-task hooks
- different types for different allocation scopes
- marking stack-allocated data by naming convention

Example: unlock macro usage

```c
int table_get_size(rtable *tpub) {
  // unlocked, tpriv not available
  int raw_size;
  LOBJ_LOCKED(tpub, tpriv, rtable, rtable) {
    // locked, tpriv available
    raw_size = tpriv->size;
    if (trivial_case) return raw_size;
  }
  // unlocked, tpriv not available
  return table_size_adjusted(raw_size);
}
```

Example: unlock macro definition

```
#define CLEANUP(fun) __attribute__((cleanup(fun)))

#define LOBJ_LOCKED(_obj, _pobj, _stem, _level) \
  for (CLEANUP(LOBJ_UNLOCK_CLEANUP_NAME(_stem)) \
  struct _stem##_private *_pobj = LOBJ_LOCK_SIMPLE(_obj, _level); \
      _pobj ? (_pobj->locked_at = &_pobj) : NULL; \
      LOBJ_UNLOCK_CLEANUP_NAME(_stem)(&_pobj), _pobj = NULL)
```

Example: unlock cleanup hook

```c
#define LOBJ_UNLOCK_CLEANUP(_stem, _level) \
  static inline void LOBJ_UNLOCK_CLEANUP_NAME(_stem)(struct _stem##_private **obj) { \
    if (!*obj) return; \
    ASSERT_DIE(LOBJ_IS_LOCKED((*obj), _level)); \
    ASSERT_DIE((*obj)->locked_at == obj); \
    (*obj)->locked_at = NULL; \
    UNLOCK_DOMAIN(_level, (*obj)->lock); \
  }
```

Memory allocation strategy

- use what fits your project
- BIRD: hierarchical pools keeping track of everything
- `tmp_alloc` = gets freed at end of task

Temporarily getting a global resource

- find / reference / allocate it
- schedule an end-of-task event to release it
- safe to use, not safe to store
- currently: too much explicit code

Arrays and their items

- always store the array lengths and check ranges
- macros and simple libs can do this easier
- checkable by static analysis and plain sight (and grep)

The Event Loop

- an infinite cycle around `poll()`
- yes, we have a custom one
- end-of-task = run after the current block of events
- ensures temporary resource cleanup

Global data structures

- full references (backpointers) → allows for proper checks
- usecounting is hard to check
- cleanup hooks to auto-unref on deallocation
- expecting an awful lot of M4-generated code in future

Where to see this

- BIRD Internet Routing Daemon version 3
- not yet completely imposed
- working on a split of the BIRDlib for public use
- https://gitlab.nic.cz/labs/bird/tree/stable-v3.0
- also LibUCW https://www.ucw.cz/libucw/

Ad Hominem

- Maria Matejka, CZ.NIC
- maria.matejka@nic.cz
- Expert in C, performance, multithreading
- developer / maintainer / team leader

:wq⏎

Maria Matejka • Feb 02, 2025

cz.nic | CZ DOMAIN REGISTRY