Exploring a real life RCU use case for Rust

Vitaly Wool, Konsulko AB

Linux Plumbers 2025

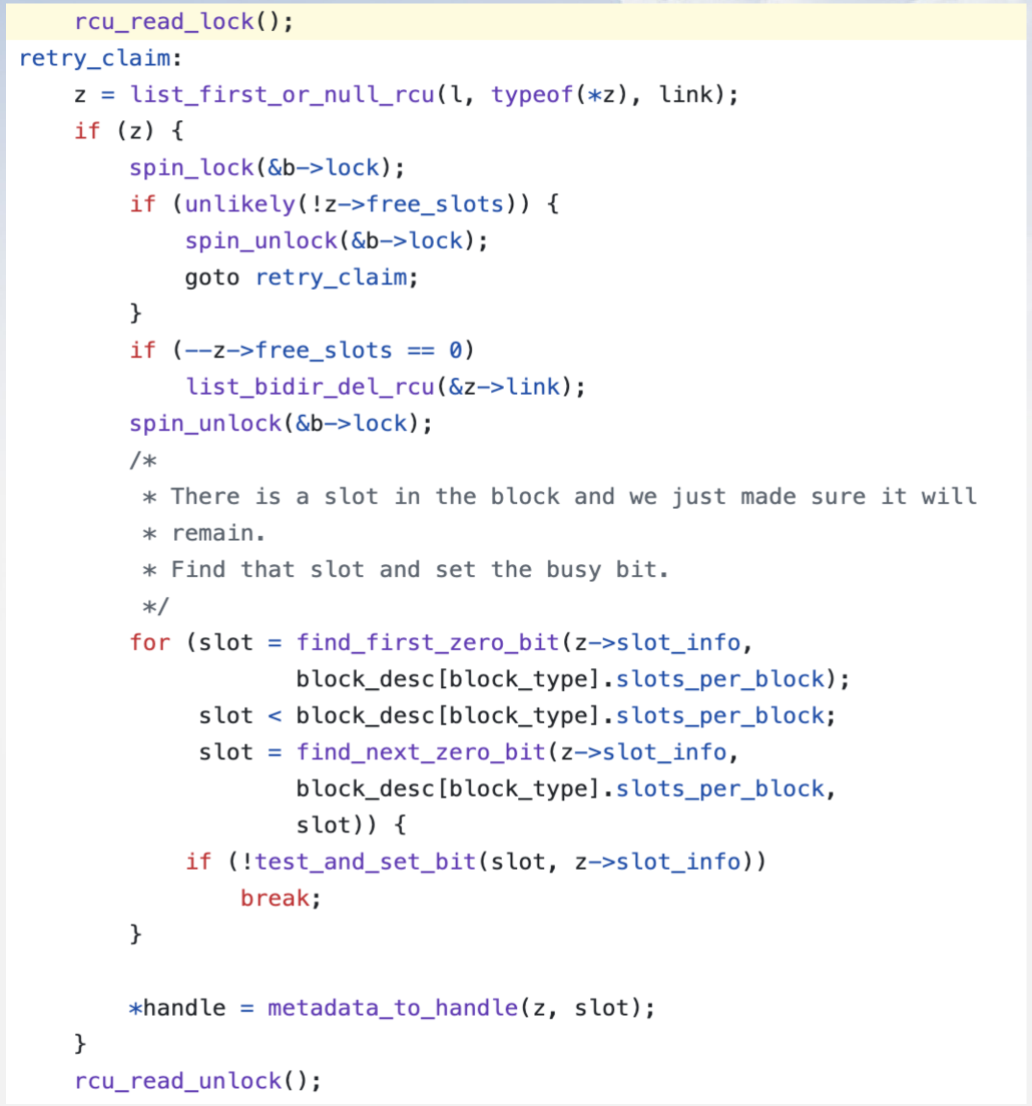
[z]blocks

Konsulko Group

- Allocator backend (zblock)
- Each block is divided into slots of equal size
  - Slots are occupied and freed
- Blocks can be empty (E), partially empty (PE)or full (F)
- Partially empty blocks are organized into a linked list
  - Blocks are created with a single occuppied slot => put into the list
  - PE Block becomes E => removed from the list and freed
  - PE Block becomes F => removed from the list
  - F Block becomes PE => put into the list

Read Mostly operation

- List operations should be protected
  - Spinlock in the Rust implementation of zblock
- List operations are:
  - relatively expensive
  - **read mostly** wrt slot operations
- Implemented as an RCU list in C variant of zblock



```
    rcu_read_lock();
retry_claim:
    z = list_first_or_null_rcu(l, typeof(*z), link);
    if (z) {
        spin_lock(&b->lock);
        if (unlikely(!z->free_slots)) {
            spin_unlock(&b->lock);
            goto retry_claim;
        }
        if (--z->free_slots == 0)
            list_bidir_del_rcu(&z->link);
        spin_unlock(&b->lock);
        /*
         * There is a slot in the block and we just made sure it will
         * remain.
         * Find that slot and set the busy bit.
         */
        for (slot = find_first_zero_bit(z->slot_info,
                    block_desc[block_type].slots_per_block);
             slot < block_desc[block_type].slots_per_block;
             slot = find_next_zero_bit(z->slot_info,
                    block_desc[block_type].slots_per_block,
                    slot)) {
            if (!test_and_set_bit(slot, z->slot_info))
                break;
        }

        *handle = metadata_to_handle(z, slot);
    }
    rcu_read_unlock();
```

What’s there for Rust

Konsulko Group

- Very basic RCU implementation
  - rust/kernel/sync/rcu.rs
- Can be extended with rcu_dereference()/rcu_assign_pointer() analog
  - https://share.google/dumrrPHshvbsgygU1
- zblock honestly uses native Rust *List* implementation
  - nothing to dereference

Alternative 1: C RCU list

- use rculist.h for a new Rust helper (helpers/rculist.c)
- use bindings::list_add_rcu() and friends directly in zblock
- use the existing RCU Rust implementation for rcu_read_lock()
- what about safety?
  - not too good
- looks quite ugly to be honest

Alternative 2: Rust RCU list

Konsulko Group

- Basically like Alternative 1, but with a Rust RCU list implementation
- several variants for this one
  - use bindings::rcu_dereference() ← where?
  - use UnsafeRcu::dereference() ← where?
- /me confused

What do we actually need?

Konsulko Group

- list_first_or_null_rcu()
- list_add_rcu()
- list_bidir_del_rcu()
- deferred free after list_bidir_del_rcu()
- mutual exclusion between list_add_rcu() and list_bidir_del_rcu()
- we **don't** need to walk through the list

How do we get there?

- list_first_or_null_rcu()
  - ListRcu<T: ?Sized> { first: AtomicPtr<T>, … }
  - return Option<&’b T> ?
- list_add_rcu()
  - ListRcu::push_back()
- list_bidir_del_rcu()
  - ListRcu::remove()
  - we can get stale &T above but we know how to deal with it
- deferred free after list_bidir_del_rcu()
  - how do we do this one?
  - need a Rust variant of kvfree_rcu()

Please contribute!

E-mail: vitaly.wool@konsulko.com

LINUX PLUMBERS CONFERENCE 2025
TOKYO, JAPAN