CYBERUS
TECHNOLOGY

Advanced Unit Testing in Hedron

Julian Stecklina

Web: https://x86.lol/
Podcast: https://syslog.show/
Matrix @js:ukvly.org
Twitter: @blitzclone

What's Hedron?

Microhypervisor with capability-based security model written in C++

Focus on:

- Simplicity, readability, testability.
- x86-64 virtualization

Lives on Github developed by Cyberus Technology

Fork of NOVA microhypervisor.

Check out https://www.cyberus-technology.de/blog.html to learn more.

Not going to talk about any of this! 😅

CYBERUS TECHNOLOGY

Why (Unit) Test?

A healthy software project

- is easy to change by multiple people
- with confidence that it doesn't break.

Good unit test coverage helps:

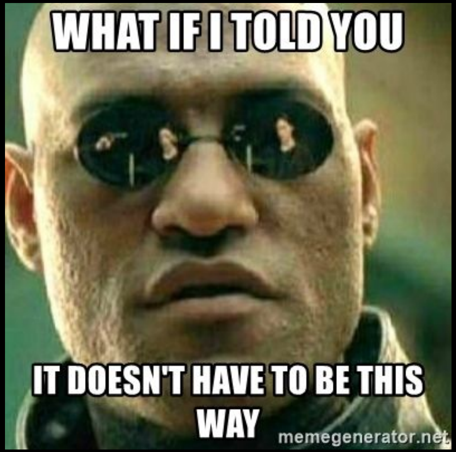
- Tests can run anywhere,
- developer feedback in *seconds*,
- sanitizers (UBSAN, ASAN, …)!

Kernels Are Not Doing Well

OS kernels are particularly hostile:

- strange programming environments
- interaction with hardware
- mindset / lack of education

Result: Usually extremely poor (unit) tests.



Let's spread some testing ideas!

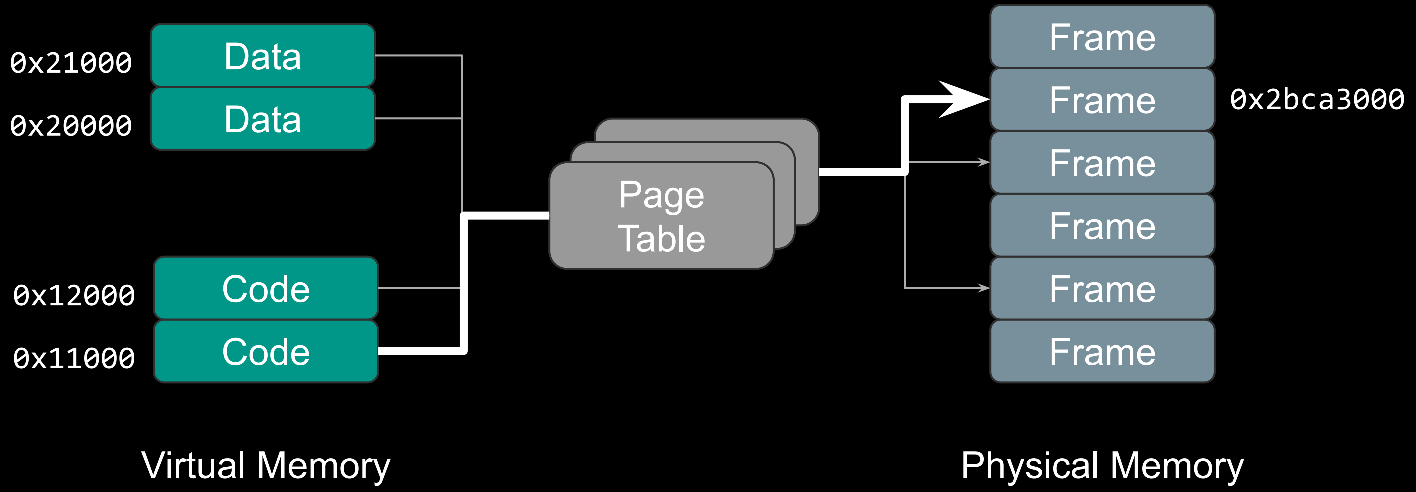
Example: Page Table Manipulation

Needed to modify Hedron’s page table. No existing tests!

- Important piece of code in a microkernel.
- Bugs are extremely hard to debug.

How to get good test coverage? We decided to redesign it.

Recap: Address Spaces



Recap: Page Tables (x86-64)

0xFFFF8C1A80401023 -> 0x80401023

9	9	9	9	12

L0	L1	L2	L3	Offset
0x178	0x06a	0x002	0x001	0x023

Intel SDM Vol. 3 Ch. 4

Unit Testing Challenges

1. Code uses kernel-internal APIs.

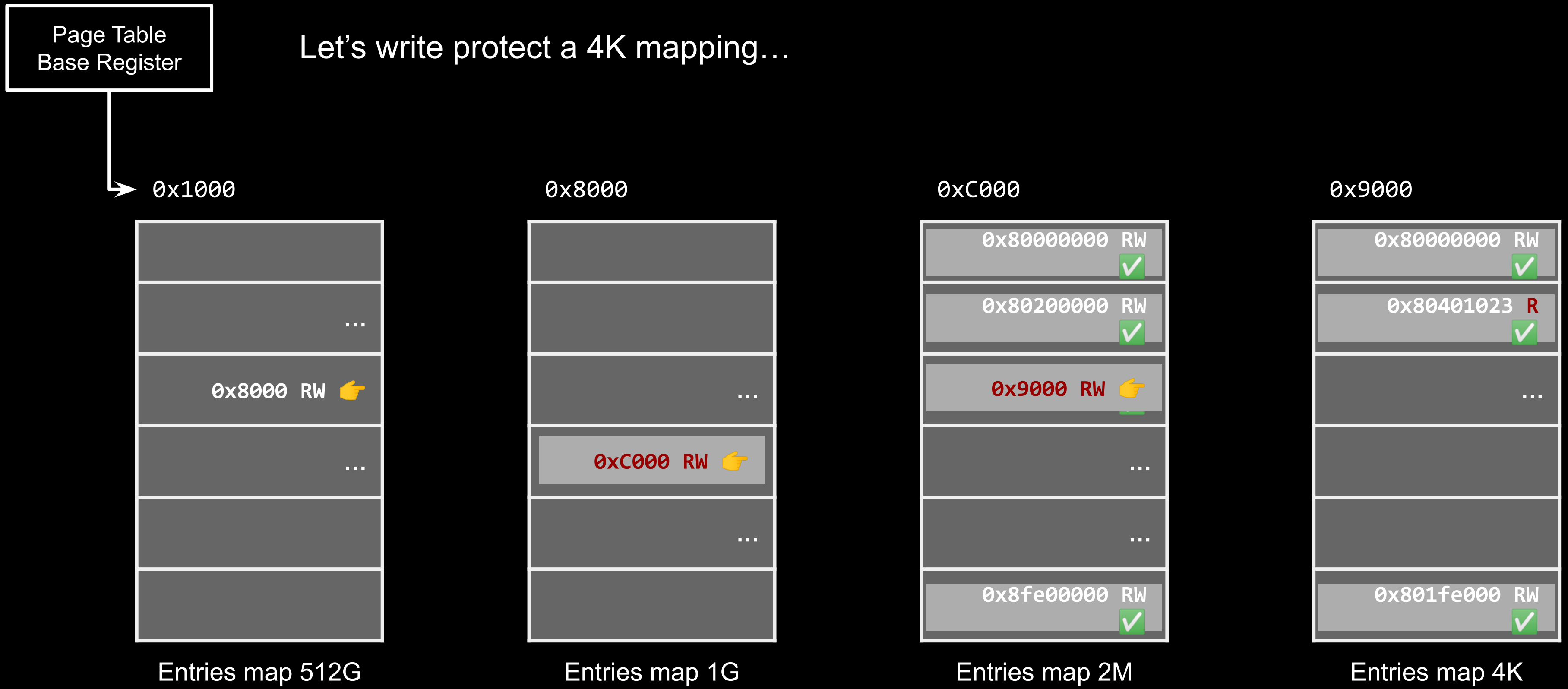
   Prevents us from compiling the code for Linux.

2. CPU reads page tables while they are being modified.

   Checking result when everything is done: Insufficient!

Updating Page Tables (x86-64)

Let's write protect a 4K mapping…



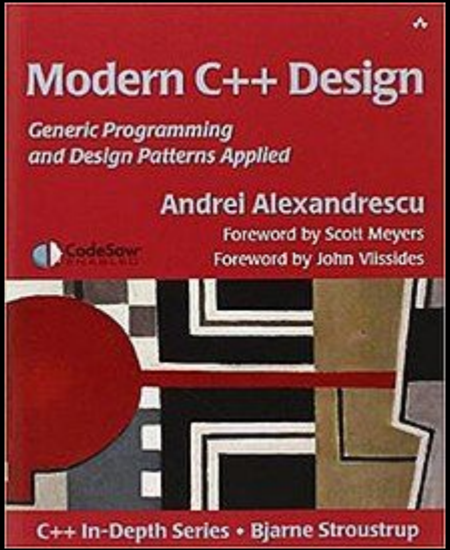
Idea: Record Observable Side Effects

In unit tests, we want to:

- Record all visible side-effects while the code is running.
- Check all transient states for validity.

Applying Policy-Based Design

```
template <int BITS_PER_LEVEL, typename ENTRY, typename MEMORY, typename CACHE_FLUSH,
          typename PAGE_ALLOC, typename DEFERRED_CLEANUP, typename ATTR>
class Generic_page_table
{
    private:

        MEMORY      memory_;

        // ...

        // Use a superpage from the given level to fill out a new page table one
        // hierarchy deeper with the same mappings.
        void fill_from_superpage(pte_pointer_t new_table, pte_t superpage_pte, level_t cur_level)
        {
            assert_slow (is_superpage(cur_level, superpage_pte));

            pte_t attr_mask {cur_level == 1 ? static_cast<pte_t>(ATTR::PTE_S) : 0};

            for (size_t i {0}; i < static_cast<size_t>(1) << BITS_PER_LEVEL; i++) {
                pte_t offset {i << (PAGE_BITS + (cur_level - 1) * BITS_PER_LEVEL)};

                memory_.write (new_table + i, (superpage_pte & ~attr_mask) | offset);
            }

            flush_cache_page (new_table);
        }
```



```
class Atomic_access_policy
{
    public:
        static entry read  (pointer ptr)          { return Atomic::load(*ptr); }
        static void  write (pointer ptr, entry e) { Atomic::store(*ptr, e); }
```

```
class Fake_memory
{
        using location = std::pair<pointer, entry>;
        using memory_list = std::forward_list<location>;

        // Keep a list of address/value pairs that can be prepended with new
        // content.
        memory_list memory_;

    public:

        void write(pointer ptr, entry e)
        {
            memory_.emplace_front (ptr, e);
        }

        entry read(pointer ptr) const
        {
            auto it {std::find_if (memory_.cbegin(), memory_.cend(),
                                   [ptr] (auto const &pair) { return pair.first == ptr; } )};

            if (it == memory_.cend()) {
                // Reading unwritten memory. Fake_page_alloc relies on this being
                // zero.
                return 0;
            }

            return it->second;
        }
```

```
TEST_CASE("Superpage splitting is correct") {
    Fake_hpt hpt {4, 3};

    // What we initially have in the page table.
    Fake_hpt::Mapping const initial_1gb_mapping {0, 0x80000000, Fake_attr::PTE_P, onegb_order};

    // Allowed transient mappings.
    Fake_hpt::Mapping const trans_2mb_mapping {0, 0x80000000, Fake_attr::PTE_P, twomb_order};
    Fake_hpt::Mapping const trans_4k_mapping {0, 0x80000000, Fake_attr::PTE_P, PAGE_BITS};

    // Final mapping.
    Fake_hpt::Mapping const page_mapping {0, 0xC0000000, Fake_attr::PTE_P | Fake_attr::PTE_W, PAGE_BITS};

    // Create initial state.
    hpt.update (initial_1gb_mapping);
    auto before_mapping {hpt.memory().now()};

    // Map 4K over 1GB page.
    hpt.update(page_mapping);

    // Look through all intermediate states of the memory.
    for (auto it {hpt.memory().now()}; it != before_mapping; ++it) {
        auto const cur_map {rewind (it, hpt).lookup (0)};

        REQUIRE((cur_map == initial_1gb_mapping ||
                 cur_map == trans_2mb_mapping ||
                 cur_map == trans_4k_mapping ||
                 cur_map == page_mapping));
    }
}
```

Taking It Further

We can use this technique to test other properties:

- Are page tables disconnected before they are deallocated?
- Do we handle atomic-compare-exchange failures?
- Do we read/write memory exactly as often as needed?
- …

We only started to tap the potential.

C++ 20 Concepts: Better Error Messages

Incompatible policy classes lead to hard to read C++ error messages…

```
template <typename T, typename P, typename V>
concept MemoryAccess = requires(T v, P p, V v)
{
    {v.read(p)} -> std::convertible_to<uint64_t>;
    {v.write(p, v)};

    // ...
};
```

Somewhat similar to traits in Rust.

Summary

We re-wrote Hedron’s page table code:

- after deciding what we want to test,
- using policy-based design,
- to unit test otherwise hard-to-test properties,
- by recording all observable side effects of operations,
- with a concept that’s applicable to C++ and Rust.

Let’s improve kernel testing: Please share **your** test stories!

CYBERUS
TECHNOLOGY

https://cyberus-technology.de/blog.html

https://github.com/cyberus-technology

@CyberusTech

Personal

https://x86.lol/

https://github.com/blitz

syslog

Podcast about Systems Topics:
https://syslog.show/

@blitzclone

@ukvly

[m]

@js:ukvly.org

#uvkly:ukvly.org