Direct Map Management

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 825377
Direct map recap

- 1x1 mapping of the physical memory
  - Up to randomisation offsets with KASLR
- Direct map is the kernel page table
  - alloc_pages(), kmalloc() return addresses in the direct map
Direct map fragmentation

- Direct map uses large pages
  - 1GB and 2MB on x86
- Changes at 4k granularity
  - fragment the large pages
    - Unmapping some 4k page
    - Changing protection to RO
    - Performance degradation
    - Need to allocate page tables for the 4KB or 2MB levels
struct page *page = alloc_page(GFP_KERNEL);

/* physical address of the page is 0x1c004000 */
set_direct_map_invalid_noflush(page);
Example use-cases

- **Modules/BPF/ftrace/kprobes**
  - `set_memory_ro()` callers today

- **memfd_secret**
  - [https://git.kernel.org/torvalds/c/1507f51255c9ff07d75909a84e7c0d7f3c4b2f49](https://git.kernel.org/torvalds/c/1507f51255c9ff07d75909a84e7c0d7f3c4b2f49)

- **Protecting page tables with PKS**
  - [https://lore.kernel.org/all/20210830235927.6443-1-rick.p.edgecombe@intel.com](https://lore.kernel.org/all/20210830235927.6443-1-rick.p.edgecombe@intel.com)

- **AMD Secure Nested Paging (SEV-SNP) Hypervisor Support**
  - [https://lore.kernel.org/all/20210820155918.7518-1-brijesh.singh@amd.com](https://lore.kernel.org/all/20210820155918.7518-1-brijesh.singh@amd.com)
Direct map fragmentation benchmarks

- System: ThinkPad T480, Intel i7-8650U CPU, 32 GBytes of RAM
- Benchmarks: fs-mark, pgbench, redis, apache, kbuild
- Variants:
  - SSD vs tmpfs
  - HW vulnerabilities mitigations on vs off

- More comprehensive tests:
  https://lore.kernel.org/linux-mm/213b4567-46ce-f116-9cdf-bbd0c884eb3c@linux.intel.com/
Direct map fragmentation benchmarks

Mitigations off, SSD

Mitigations off, tmpfs

Mitigations on, SSD

Mitigations on, tmpfs

1G
2M
4K
Direct map management

- Reuse split large pages that are split anyway
  - Allocate large physically contiguous page
  - Cache large pages removed from the direct map
  - Resort to base page allocations as a fallback

- Cache design alternatives:
  - A cache (per user?) on top of page allocator
  - An extension to page allocator

RFC: https://lore.kernel.org/all/20210823132513.15836-1-rppt@kernel.org/
Cache design alternatives

Per-user caches

- Simpler implementation
- Better access control
  - Easier cache management
- Larger memory overhead
- Higher fragmentation
- Freeing via dedicated API

Page allocator extension

- More intrusive changes
- Cache is a black box
  - Allocations are unmovable
- More memory efficient
- Lower fragmentation
- Core MM integration
Thank you!