Protection Keys, Supervisor (PKS)

Ira Weiny and Rick Edgecombe

LINUX
PLUMBERS
CONFERENCE > September 20-24, 2021

Outline

- Why are we doing this?
- PKS Hardware overview
- PMEM Stray Write Protection
- Write Protected Page Tables
- PKS core software
- Status, next steps, and acknowledgements

Why are we doing this?

Some use cases

- PMEM stray write protection
- Write protected page tables
- Additional use cases?
  - Harden sensitive data like kernel keys

- But why not just use Page tables???

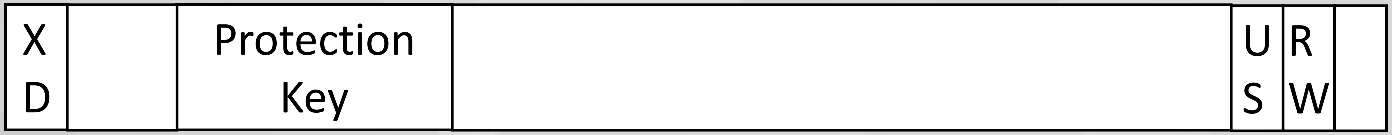
PKS Hardware Overview

PKS Hardware

- A protection key in each PTE
- Adds a Per-thread Model-specific Register (MSR) to control the permissions of those keys
- Changes to access are “fast”
  - No page table walks
  - TLB flushes are not required
  - MSR is non-serializing
  - Thread local

Page Table Entry

- ➢ Simple addition to the page table protections
- ➢ Like user space keys but applicable to kernel pages
  - ➢ U/S bit == 0
- ➢ Associate each mapping (PTE) with a protection key (4 bits)



XD

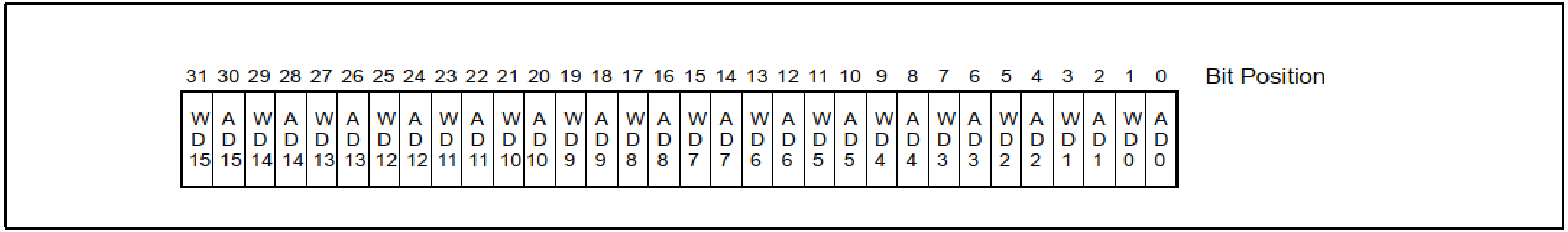
Protection Key

US

RW

Per-Thread MSR

- A single **per-thread register** defines the accessibility for all the keys
  - Bits 63-32 reserved; 31-0 define permissions for Pkey 0-15
  - Thread local
  - Not XSAVE managed



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Bit Position
WD15	AD15	WD14	AD14	WD13	AD13	WD12	AD12	WD11	AD11	WD10	AD10	WD9	AD9	WD8	AD8	WD7	AD7	WD6	AD6	WD5	AD5	WD4	AD4	WD3	AD3	WD2	AD2	WD1	AD1	WD0	AD0	

Figure 2-9. Format of Protection-Key Rights Registers

PKS advantages

- This hardware overlays additional protections on large domains of pages
- With a single place to change the protections on the entire domain quickly
  - MSR write is relatively fast
- Changes are thread local
  - Protection key in PTE is constant

PMEM Stray Write Protection

PMEM stray writes

- Persistent memory is vulnerable to ‘stray writes’
  - PMEM is mapped in the direct map but is not really ‘allowed’
  - A write could permanently corrupt user’s data
- Changing PTEs is troublesome and PKS is ‘fast’
  - Just a simple MSR write, right?
- Applying PKS protections = easy
- Toggling PKS protections = hard

PMEM…

- Default protections restrict any access (no reads or writes)
  - Works well with default PKS permissions
- Direct access is limited to the PMEM and a few other drivers
- General kernel access is wrapped with kmap*
  - Turns out kmap was more difficult to alter than expected

Kmap issues

- kmap() was not thread local…
  - Global updates were difficult
- kmap_thread() → kmap_local_page()
  - Preemptable, thread local kmap
- Drove the need for a ‘relaxed’ mode
  - Which was later expanded

Write Protected Page Tables

Write Protected Page Tables

- Purpose: prevent writes to page tables except through dedicated helpers
  - Default RO
- Hardening/debugging
- Toggling PKS protections = easy
- Applying PKS protections = hard

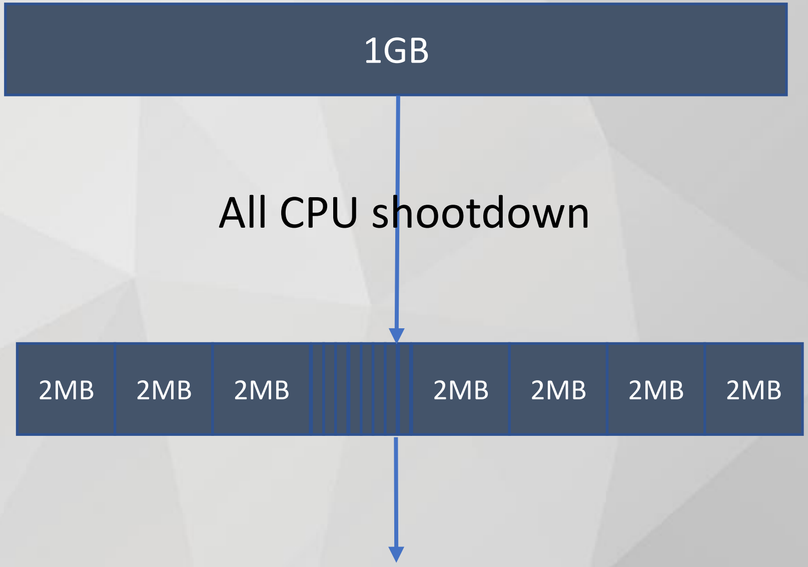
Toggle inside helpers:

```
void set_pte_at(struct mm_struct *mm, unsigned long addr,
                pte_t *ptep, pte_t pte);
pte_t ptep_get_and_clear(struct mm_struct *mm, unsigned long addr,
                         pte_t *ptep);
int ptep_test_and_clear_young(struct vm_area_struct *vma,
                              unsigned long addr, pte_t *ptep);
…etc
```

Allocating Tables

- Pmem usage applies protection on mapping
- Page tables are allocated dynamically at runtime
- Changing kernel memory permissions is *expensive*
  - All CPU TLB shootdown
  - Break direct map large pages for surrounding memory
- Many page table allocations...

Worst case single page allocation

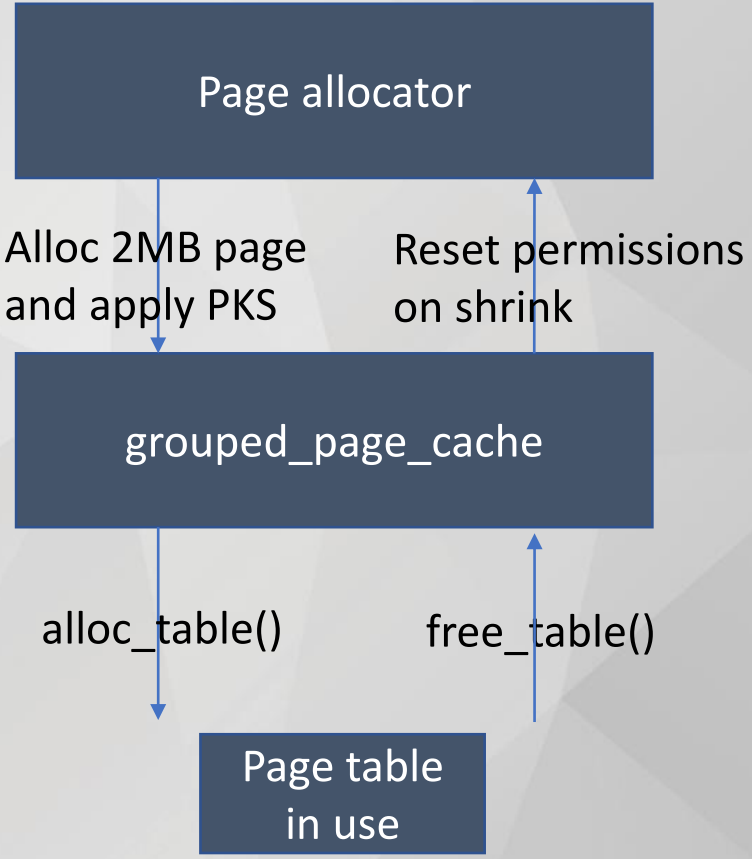


Single new page table

LINUX PLUMBERS CONFERENCE › September 20-24, 2021

Allocating Tables

- Not first thing with this problem
  - Many RFCs by me around other kernel memory permission usages
  - Secretmem unmapping direct map
- Approaches
  - Convert memory in batch and cache
  - Reset direct map on shrink



Direct Map Page Tables

- If cache runs out of page tables, need to convert some more
  - …usually requires breaking large direct map pages
  - …which needs a page for a table
  - …but there are none
- Chicken and egg

Direct Map Page Tables

Options

- Allocate table from break and new table from same high order allocation
- Break direct map to 4K at boot
- Reserve enough page tables to map the entire direct map at 4k at boot and pre-convert them to PKS
  - Current solution

Core Software Support

Key allocation

- Keys are statically allocated
- This works well as the number of users is not anticipated to be large

```
enum pks_pkey_consumers
{
    PKS_KEY_DEFAULT,
    PKS_KEY_MY_FEATURE,
    PKS_KEY_NR_CONSUMERS
};

…
consumer_defaults[PKS_KEY_DEFAULT] = 0;
consumer_defaults[PKS_KEY_MY_FEATURE] =
                                        PKR_DISABLE_WRITE;
…
```

Thread and Exceptions

- XSAVE not supported
- ‘struct thread_struct’ contains a cached msr
- First implementations skipped exception save support
- Eventually Andy Lutomirski came up with a cleaver idea
  - Use extra space on the stack for ‘struct extended_pt_regs’

‘Relaxed’ Mode

- Both of the current use cases desired a ‘chicken switch’
- PMEM -- ‘memremap.pks_fault_mode’
- Write Protected Page Tables – ‘pkstablesoft’

‘Relaxed’ Mode

- So there has been a PKS fault…
- Walk tables to get the key
- But it could be in an interrupt…
- Kernel address space page table frees
  - Memory hot unplug
- Once the key has been determined, the kernel can decide what to do

CPU 0	CPU 1
Memory hot unplug	
Gather page tables	
	PKS fault!
synchronize_rcu()	rcu_read_lock()
	Walk tables
	Get key
	rcu_read_unlock()
Free tables	

Status, next steps, and acknowledgements

Status

- V7 patches: core and PMEM use case
  - https://lore.kernel.org/lkml/20210804043231.2655537-1-ira.weiny@intel.com/
  - Documentation/core-api/protection-keys.rst
- RFC V2: Page table support
  - https://lore.kernel.org/lkml/20210830235927.6443-1-rick.p.edgecombe@intel.com/

Test it out

- Don’t need any special HW to develop/test PKS features
- QEMU TCG support >6.0.0
- -cpu qemu64,+pks

PMEM next steps

- Continue to remove kmap() users
- At some point make pks_fault_mode ‘strict’

PKS Tables Next Steps

- RFCv2
  - Protect all known page tables
  - Handle direct map
  - Memory hotplug/unplug
  - Relaxed mode
- Plans
  - Performance
  - Mike Rapoport page allocation effort

Acknowledgements

- The following people had large input on the series and we would like to thank them:
- Dave Hansen
- Dan Williams
- Peter Zijlstra
- Thomas Gleixner
- Fenghua Yu
- Sean Christopherson
- Christoph Hellwig
- Andy Lutomirski