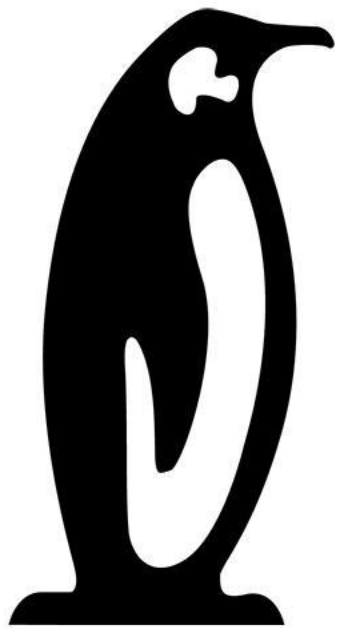


Linux Plumbers Conference

Dublin, Ireland September 12-14, 2022

Copy On Write, Get User Pages

... and Mysterious Counters

David Hildenbrand
david@redhat.com
12. September 2022

Red Hat

Agenda

Background

1. Copy On Write
2. Mysterious Counters
3. Get User Pages
4. What could go wrong?
5. More Mysterious Counters

Anonymous Memory: PageAnonExclusive

1. Overview
2. Other Applications
3. What's missing?

Discussion

1. Future of the Mapcount ?
2. Future of hugetlb COW-sharing during fork() ?
3. Future of Page Reuse during COW ?
4. Future of R/O Pinning ?

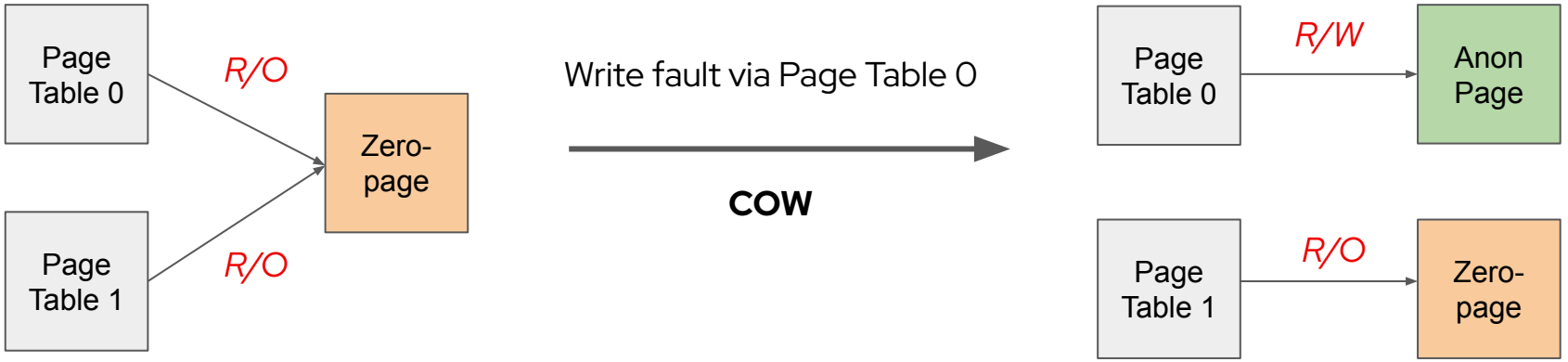
Background

1. Copy-on-Write (COW) (1)

... we'll focus on COW in private mappings (MAP_PRIVATE)

Avoid creating a private copy of a page as long as there are no modifications

- **Share page with COW semantics: map it R/O**
  - Zeropage, pagecache page, anonymous page, KSM page ...
- **Break COW on write fault**
  - Create private writable copy



Red Hat

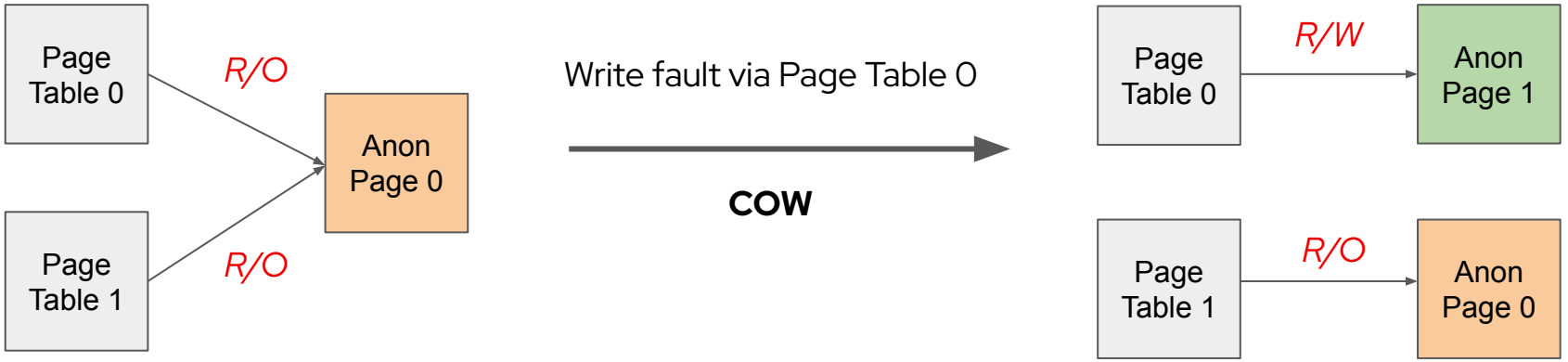
1. Copy-on-Write (COW) (2)

Shared anonymous pages? Important optimization for fork() ...

- **Share anonymous page R/O between parent and child**
  - Lazily copy on demand
- **On write fault ... always create a private copy?**
  - Wasteful: what if the child immediately quit?

Detecting possible sharing is a bit tricky ...

- **Traditional: "how many user page tables reference this page (in)directly?"**



Red Hat

2. Mysterious Counters

Refcount: one counter per folio (page_count)

- "how many tracked **references** to this **folio**"

Swapcount: one counter per subpage of a folio

- ... if the folio is in the swapcache
- "how many **swap PTEs** indirectly reference this **subpage**"

"Entire mapcount": one counter per folio

- "how often is this **entire folio** mapped into a user page table"
- ... and a **mapcount per subpage of a folio** and things get messy
  - page_mapcount(): entire mapcount + subpage mapcount

How to detect if an anonymous page is exclusive vs. shared?

- page_mapcount() + swap_count() == 1?
- page_count() == 1?

3. Get User Pages (GUP)

Lookup a page in a user page table and reference it for immediate/later use

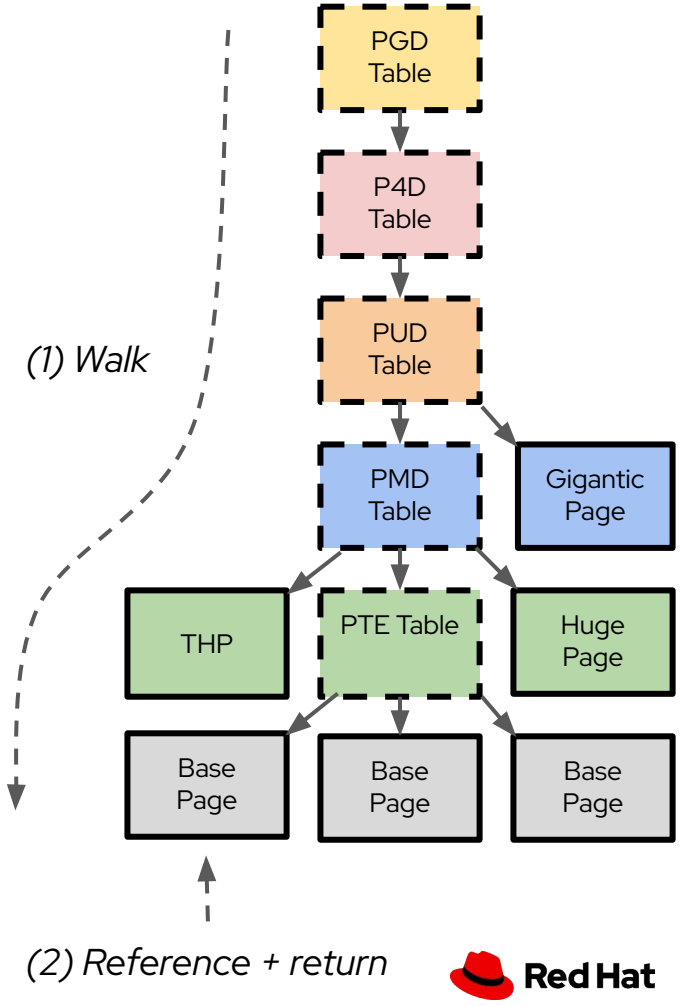
- Short term: O_DIRECT, ptrace, …
- Long term: VFIO, RDMA, io_uring fixed buffers, …

… with various flavors:

- FOLL_GET: “access struct page”
- **FOLL_PIN: “access page content”**
- FOLL_WRITE: R/W vs. R/O

… and various special cases:

- FOLL_FORCE: ignore VMA permissions (debug access)
- GUP-fast: don’t take any locks …



Red Hat

4. What could go wrong (1)

page_mapcount() + swap_count() == 1 ?

Parent		Child	
mem = mmap(MAP_PRIVATE)	*mapcount: 1*		
strcpy(mem, “Boring Data”)			
fork()	*mapcount: 2*		
		assert(!strcmp(mem, “Boring Data”));	*mapcount: 2*
		fds = pipe()	
		vmsplice(fds[1], mem)	*mapcount: 2*
		munmap(mem)	*mapcount: 1*
strcpy(mem, “Secret Data”)	*mapcount: 1* **-> No COW**		
		data = read(fds[0])	*mapcount: 1*
		assert(!strcmp(data, “Boring Data”));	**-> Fail**

CVE-2020-29374 (Jann Horn)

4. What could go wrong (2) page_count() == 1?

```
mem = mmap(pagesize, MAP_PRIVATE)

memset(mem, 0, pagesize);

iov.iov_base = mem;
iov.iov_len = size;
io_uring_register_buffers(&ring, &iov, 1);

/* page gets mapped R/O in the page table for reason X */

memset(mem, 0xff, pagesize);

io_uring_prep_write_fixed(..., fd, mem, pagesize);
io_uring_submit(...)
io_uring_wait(...)
```

P0, refcount: 1

P0, refcount: 2

COW → *P1, refcount: 1*

P0, refcount: 1 → **stale data written**

Note: The actual refcounts are slightly different

5. More Mysterious Counters

Pincount: one counter for large folios

- "how often was this **folio** pinned via GUP"
- Can be speculatively raised by GUP-fast

... there is no pincount for order-0 folios?

- Bits in "struct page" are rare
- Mangled into the **refcount**
  - GUP_PIN_COUNTING_BIAS = 1024

folio_maybe_dma_pinned() cannot have false negatives

- **... but false positives in both cases**

... COW decisions based on mysterious counters?

Anonymous Memory: PageAnonExclusive

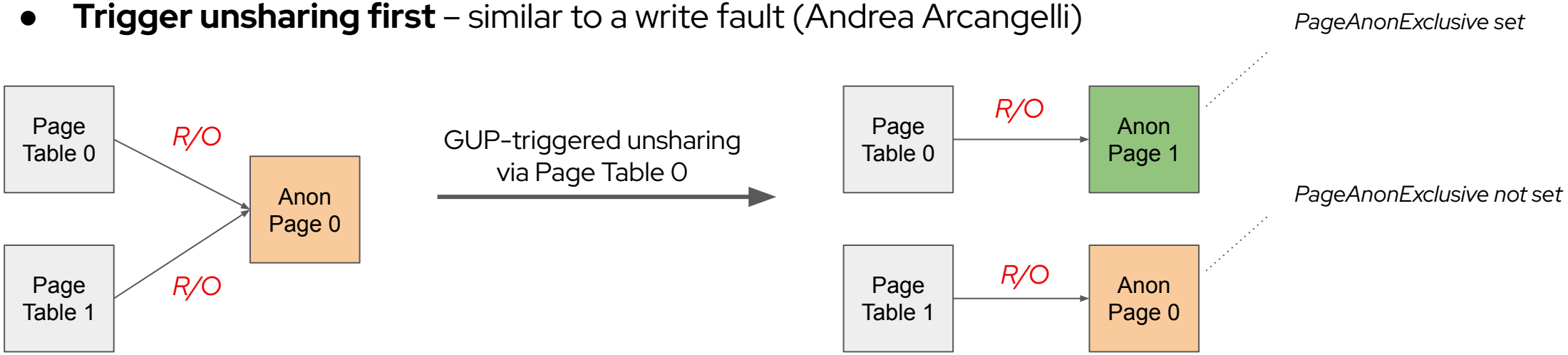
1. Overview

PageAnonExclusive: definitely exclusive vs. might be shared

- **Fresh or writable anonymous pages are exclusive** *(PageAnonExclusive set)*
- **Never pin an anonymous page that might be shared** *(PageAnonExclusive not set)*
- **Never share** (*clear PageAnonExclusive*) **an anonymous page that might be pinned**

R/O Pinning a R/O-mapped anonymous page that is not exclusive?

- **Trigger unsharing first** – similar to a write fault (Andrea Arcangelli)



Reuse page if not exclusive: if there is only one reference (page_count() == 1)

- Can result in **unnecessary copies** on other (e.g., speculative) references

Red Hat

2. Other Applications

mprotect(PROT_READ) -> mprotect(PROT_READ|PROT_WRITE)

- **Exclusive anonymous page: Map the page writable**
  - Avoid a write fault

FOLL_FORCE|FOLL_WRITE on MAP_PRIVATE VMA without PROT_WRITE

- Used for ptrace like */proc/self/mem* access
- **Exclusive anonymous page: Allow pinning/referencing the page**
- Fix for a security issue – Dirty-COW for SHMEM (CVE-2022-2590)

NUMA hinting (WIP)

- **Same as mprotect()**: replace pte_mk_savedwrite() ...

3. What's missing?

hugetlb

- **Still uses the mapcount to make COW decisions**
  - hugetlb cannot deal with unnecessary copies

O_DIRECT conversion FOLL_GET -> FOLL_PIN

- **Makes O_DIRECT/vmsplice/... with fork() fully functional**
- John Hubbard is on it

Preserve exclusive flag on more architectures in swp PTE

- **For now only x86-64, s390x, aarch64, ppc64/book3s**
- Others can lose the exclusive flag

GUP-fast handling

- **GUP-fast is tricky; one pending fix for PageAnonExclusive**

selftests

- **WIP :)**

Discussion

1. Future of the Mapcount ?

We no longer *need* the mapcount to make COW decisions

- well, hugetlb is an exception ...

... but it obviously has other users

- Detecting page table mappings: e.g., page_mapped()
- Detecting unknown references: e.g., mapcount != pagecount + 1
- Best-guess detection of “single page table mapping”: e.g., MADV_PAGEOUT

... which raises the questions

- **... do we still need full accuracy (~31 bit)?**
- **... do we still need a mapcount per THP subpage?**
  - ... PG_doublemap, total_mapcount() ...

2. Future of hugetlb COW-sharing during fork()?

COW-sharing of hugetlb pages is awkward

- Hugetlb reservation … **hugetlb pages cannot *really* be overcommitted**
- **Running out of hugetlb pages during COW?**
  - Let's just steal the page from the child process …

… can we rework it and avoid COW-sharing during fork() altogether?

- **Treat it as MADV_DONTFORK? :/**
- **Don't share but instead copy all pages for the child during fork (fail early)? :/**

Maybe we should never have added COW-sharing of hugetlb pages … can we deprecate?

3. Future of Page Reuse during COW ?

Anonymous Pages

- **We never reuse “maybe shared” anonymous pages if there is more than one reference**
  - On a PTE-mapped THP, we never reuse
- But how could we really optimize without the mapcount+swapcount?

KSM Pages

- **We never reuse KSM pages**
  - Have to remove the page from the KSM (stable) tree
  - Have to convert KSM page -> anonymous exclusive page
- 52d1e606ee73 (“mm: reuse only-pte-mapped KSM page in do_wp_page()”) implement that

Pagecache Pages

- **We never reuse pagecache pages**
  - Have to remove the page from the pagecache (only possible if clean? what else?)
  - Have to convert pagecache page -> anonymous exclusive page

... do we even care about optimizing these cases?

4. Future of R/O Pinning ?

We don't want to R/O pin COW-shared pages with LONGTERM semantics

- **The next write fault would break COW and turn the R/O PIN stale/unreliable**
- Current workaround: **FOLL_FORCE | FOLL_WRITE**

MAP_PRIVATE

- We have to break COW if we don't find an exclusive anonymous page
  - pagecache page, shared zeropage ...
- **"easy"**

MAP_SHARED

- We usually don't care about FS-handled COW (lazy allocation of disk blocks)
  - ... we just don't want the mapped page to change
- **DAX FS uses the shared zeropage to lazily allocate DAX pages .. any other cases?**
  - ... do we really care for now?
- **Implementing unsharing support in FS would be more involved**

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.


linkedin.com/company/red-hat


youtube.com/user/RedHatVideos


facebook.com/redhatinc


twitter.com/RedHat

Red Hat