ORACLE

Shrinking Memmap

FOSDEM

Matthew Wilcox
Technical Advisor
Oracle Linux Development
2025-02-02

Safe harbor statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Memmap

- Typically 1.6% of memory reserved for memmap (64 bytes per 4KiB page)
  - Think of it as 16MB per gigabyte
- Virtualised systems pay this cost twice
  - We can't share between host & guest; they have different uses in each
- Memmap is an array of struct page

Size of struct page

- Too large
  - 16GB of a 1TB machine is wasted money!
- Too small
  - Many proposals for adding more information to struct page
  - struct page_ext exists, but is slow
- Just right
  - Same size as a cache line

Page allocator

- Fundamental memory allocator
  - You should probably use a different memory allocator (kmalloc, vmalloc, slab, percpu, CMA, page_pool, …)
- Calling `alloc_page()` gives you the 4KiB of memory *and* the struct page!
  - Some arcane restrictions on which parts you can use

struct page (v4.14)

```
unsigned long flags;
union {
    struct address_space *mapping;
    void *s_mem;
    atomic_t compound_mapcount;
    /* page_deferred_list().next */
};
union {
    pgoff_t index;
    void *freelist;
    /* page_deferred_list().prev */
};
union {
    unsigned long counters;
    struct {
        union {
            atomic_t _mapcount;
            struct { ... };
        };
        atomic_t _refcount;
   };
};
```

```
union {
    struct list_head lru;
    struct dev_pagemap *pgmap;
    struct {
        struct page *next;
        int pages;
        int pobjects;
    };
    struct rcu_head rcu_head;
    struct {
        unsigned long compound_head;
        unsigned int compound_dtor;
        unsigned int compound_order;
    };
    struct {
        unsigned long __pad;
        pgtable_t pmd_huge_pte;
    };
};
```

```
union {
    unsigned long private;
    spinlock_t ptl;
    struct kmem_cache *slab_cache;
};
struct mem_cgroup *mem_cgroup;
void *virtual;
void *shadow;
int _last_cpupid;
```

struct page (v4.19)

```
unsigned long flags;
union {
    struct /* folio */ {
        struct list_head lru;
        struct address_space *mapping;
        pgoff_t index;
        unsigned long private;
    };
    struct /* tail */ {
        unsigned long compound_head;
        unsigned char compound_dtor;
        unsigned char compound_order;
        atomic_t compound_mapcount;
    };
    struct /* Second tail */ {
        unsigned long _compound_pad_1;
        unsigned long _compound_pad_2;
        struct list_head deferred_list;
    };
```

```
struct /* slab */ {
    union {
        struct list_head slab_list;
        struct {
            struct page *next;
            int pages;
            int pobjects;
        };
    };
    struct kmem_cache *slab_cache;
    void *freelist;
    union {
        void *s_mem;
        unsigned long counters;
        struct { ... };
    };
};
struct /* ZONE_DEVICE */ {
    struct dev_pagemap *pgmap;
    unsigned long hmm_data;
};
```

```
struct /* Page table */ {
    unsigned long _pt_pad_1;
    pgtable_t pmd_huge_pte;
    unsigned long _pt_pad_2;
    union {
        struct mm_struct *pt_mm;
        atomic_t pt_frag_refcount;
    };
    spinlock_t ptl;
};
atomic_t _mapcount;
atomic_t _refcount;
struct mem_cgroup *mem_cgroup;
void *virtual;
void *shadow;
int _last_cpupid;
```

struct page (v6.12)

```
unsigned long flags;
union {
    struct /* folio */ {
        struct list_head lru;
        struct address_space *mapping;
        pgoff_t index;
        unsigned long private;
    };
    struct /* tail */ {
        unsigned long compound_head;
    };
```

```
struct /* ZONE_DEVICE */ {
    struct dev_pagemap *pgmap;
    void *zone_device_data;
};
struct /* page_pool */ {
    unsigned long pp_magic;
    struct page_pool *pp;
    unsigned long _pp_mapping_pad;
    unsigned long dma_addr;
    atomic_long_t pp_ref_count;
};
```

```
atomic_t _mapcount;
atomic_t _refcount;
struct mem_cgroup *mem_cgroup;
void *virtual;
void *shadow;
int _last_cpupid;
```

But how do we shrink struct page?

- Need to identify redundancies
- Multi-page allocations maintain almost all information in the first page
- If we move all that information into a dynamically allocated struct, we can point to it from each page
- Goal (eventual): 8 byte struct page
  - One pointer, plus 4 bits of metadata
- Goal (2025): 32 byte struct page

```
unsigned long flags;
unsigned long memdesc;
atomic_t _refcount;
unsigned long private;
```

What needs to happen in 2025?

- Finish folio conversions in filesystems
  - Or disable them with a Kconfig option for a developer preview
  - Not practical for production – btrfs, ceph, f2fs, nfs and others still have legacy code
- Remove uses of page→lru, page→mapping, page→index and page→memcg_data
  - Many independent projects here
- Remove uses of bh→b_page
  - I have a tree with this work completed
- Remove uses of folio→page
- Split out pagepool allocator into its own struct
- Dynamically allocate struct folio / slab / zsmalloc / ptdesc / page_pool / ...

Is everybody happy?

- struct page is smaller
- struct page is rarely modified
- struct folio can grow without affecting struct slab or vice versa
- Order 0 & 1 allocations will use more memory
- Order 2+ allocations use less memory, but need a slab allocation
- Possible cache miss when going from page to folio or vice versa
- We can shrink struct page further
- There's a lot more to this that I didn't have time to cover today, see https://kernelnewbies.org/MatthewWilcox/Memdescs/Path
-

Thanks

- Vlastimil Babka
- Christian Brauner
- Shakeel Butt
- Luis Chamberlain
- Jane Chu
- Hugh Dickins
- Yin Fengwei
- Andreas Gruenbacher
- Roman Gushchin
- Christoph Hellwig
- David Hildenbrand
- Michael Hocko
- David Howells
- Dev Jain
- Jan Kara
- Ryusuke Konishi
- Sidhartha Kumar
- Miaohe Lin
- Vishal Moola
- Andrew Morton
- Trond Myklebust
- Kent Overstreet
- ZhangPeng
- Pankaj Raghav
- Ryan Roberts
- Kiryl Shutsemau
- Lorenzo Stoakes
- Barry Song
- Kairui Song
- Muchun Song
- David Sterba
- Ted Ts'o
- Kefeng Wang
- Richard Weinberger
- Johannes Weiner
- Qu Wenruo
- Darrick Wong
- Zi Yan

ORACLE