2.19 内存管理数据结构和 API

请画出内存管理中常用的数据结构的关系图，比如 mm_struct、vma、vaddr、page、pfn、pte、zone、paddr 和 pg_data 等。请思考如下转换关系：

1. *由 mm 数据结构和虚拟地址 vaddr 找到对应的 VMA？*
2. *由 page 和 VMA 找到虚拟地址 vaddr?*
3. *由 page 找到所有映射的 VMA？*
4. *由 VMA 和虚拟地址 vaddr，找出相应的 page 数据结构？*
5. *page 和 pfn 之间的互换*
6. *pfn 和 paddr 之间的互换*
7. *page 和 pte 之间的互换*
8. *zone 和 page 之间的互换*
9. *zone 和 pg_data 之间的互换*

2.19.1 内存管理数据结构的关系图

现在大部分 Linux 系统中，内存设备的初始化一般是在 BIOS 或者 bootloader 中，然后把 DDR 的大小传递给 Linux 内核，因此从 Linux 内核角度来看 DDR 的话，其实就是一段物理内存空间。在 Linux 内核中和内存硬件物理特性相关的一些数据结构主要是集中在 MMU（处理器中内存管理单元），比如页表、cache/TLB 操作等。因此有大部分的 Linux 内核中关于内存管理的相关数据结构都是软件的概念，比如 mm、vma、zone、page、pg_data 等。Linux 内核中的内存管理中的数据结构错综复杂，有必要进行归纳总结，如图所示：

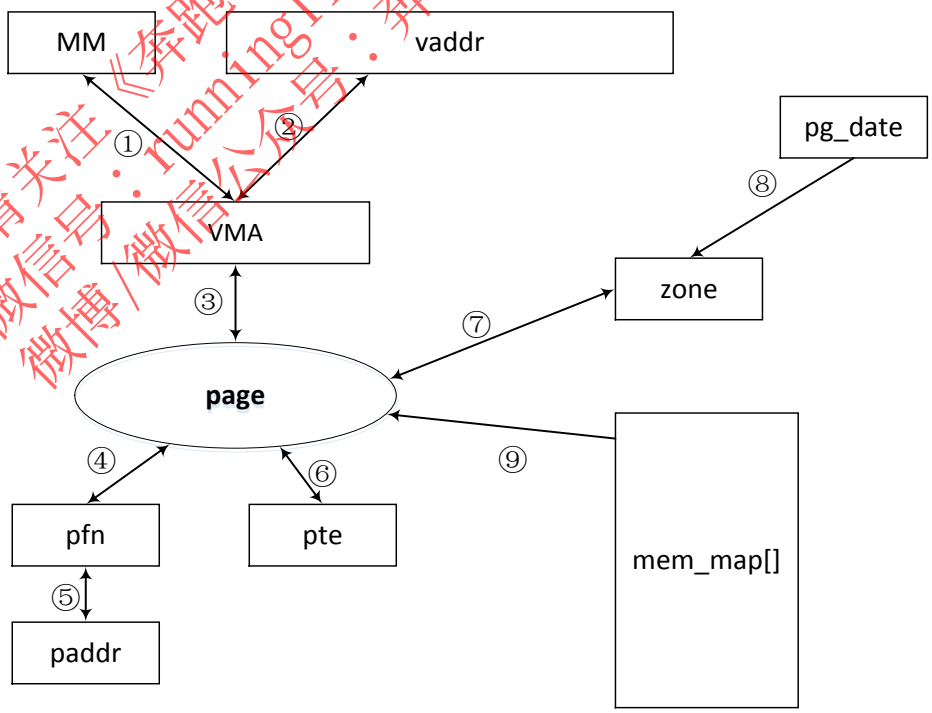

图 2.35 内存管理数据结构关系图

①：由 mm 数据结构和虚拟地址 vaddr 找到对应的 VMA。

内核提供相当多的 API 来查找 VMA。

```
struct vm_area_struct * find_vma(struct mm_struct * mm, unsigned long addr);

struct vm_area_struct * find_vma_prev(struct mm_struct * mm, unsigned long addr,
struct vm_area_struct **pprev);

struct vm_area_struct * find_vma_intersection(struct mm_struct * mm, unsigned long
start_addr, unsigned long end_addr)
```

由 VMA 得出 MM 数据结构。struct vm_area_struct 数据结构有一个指针指向 struct mm_struct。

```
struct vm_area_struct {
  …
      struct mm_struct *vm_mm;
  …
}
```

②：由 page 和 VMA 找到虚拟地址 vaddr。

```
[mm/rmap.c]

//这个只针对匿名页面, KSM 页面见第 2.17.2 章
unsigned long vma_address(struct page *page, struct vm_area_struct *vma)
=>pgoff = page->index; 表示在一个 vma 中 page 的 index
=>vaddr = vma->vm_start + ((pgoff - vma->vm_pgoff) << PAGE_SHIFT);
```

③：由 page 找到所有映射的 VMA。

```
通过反向映射 rmap 系统来实现。rmap_walk()

对于匿名页面来说:
=>由 page->mapping 找到 anon_vma 数据结构
=>遍历 anon_vma->rb_root 红黑树，取出 avc 数据结构
=>每个 avc 数据结构中指向每个映射的 VMA
```

由 VMA 和虚拟地址 vaddr，找出相应的 page 数据结构。

```
[include/linux/mm.h]

struct page *follow_page(struct vm_area_struct *vma, unsigned long vaddr, unsigned
int foll_flags)

=>由虚拟地址 vaddr 通过查询页表找出 pte
=>由 pte 找出页帧号 pfn，然后在 mem_map[]找到相应的 struct page 结构
```

④：page 和 pfn 之间的互换

```
[include/asm-generic/memory_model.h]
```

```
由 page 到 pfn: page_to_pfn()
    #define __page_to_pfn(page)    ((unsigned long)((page) - mem_map) + \
                    ARCH_PFN_OFFSET)

由 pfn 到 page
#define __pfn_to_page(pfn)      (mem_map + ((pfn) - ARCH_PFN_OFFSET))
```

⑤：pfn 和 paddr 之间的互换

```
[arch/arm/include/asm/memory.h]

由 paddr 和 pfn
#define  __phys_to_pfn(paddr) ((unsigned long)((paddr) >> PAGE_SHIFT))

由 pfn 到 paddr
#define  __pfn_to_phys(pfn)    ((phys_addr_t)(pfn) << PAGE_SHIFT)
```

⑥：page 和 pte 之间的互换

```
由 page 到 pte：
=>先由 page 到 pfn
=>然后再由 pfn 到 pte

由 pte 到 page：
#define pte_page(pte)       (pfn_to_page(pte_pfn(pte)))
```

⑦：zone 和 page 之间的互换

```
zone 到 page
  zone 数据结构有 zone->start_pfn 指向 zone 起始的页面，然后由 pfn 找到 page 数据结构。

page 到 zone
   page_zone()函数返回 page 所属的 zone。通过 page->flags 布局来实现的。
```

⑧：zone 和 pg_data 之间的互换

```
由 pd_data 到 zone：
    pg_data_t->node_zones

由 zone 到 pg_data:
    zone->zone_pgdat
```


2.19.2 内存管理中常用 API

内存管理错综复杂，除了从用户态的相关 API 来窥探和理解 Linux 内核内存是如何运作的以外，总结 Linux 内核中常用的内存管理相关的 API 也是很有帮助的。刚才已经总结了内存管理相关的数据结构之间错综复杂的关系，下面总结内存管理中内核常用的 API。

2.19.2.1 页表相关

页表相关的 API 可以概括为这类几类：

- 查询页表
- 判断页表项的状态位
- 修改页表
- page 和 pfn 的关系

```
//查询页表
#define pgd_offset_k(addr) pgd_offset(&init_mm, addr)
#define pgd_index(addr)          ((addr) >> PGDIR_SHIFT)
#define pgd_offset(mm, addr)   ((mm)->pgd + pgd_index(addr))
#define pte_index(addr)          (((addr) >> PAGE_SHIFT) & (PTRS_PER_PTE - 1))
#define pte_offset_kernel(pmd,addr)(pmd_page_vaddr(*(pmd)) + pte_index(addr))
#define pte_offset_map(pmd,addr)   (__pte_map(pmd) + pte_index(addr))
#define pte_unmap(pte)                 __pte_unmap(pte)
#define pte_offset_map_lock(mm, pmd, address, ptlp)

//判断页表项的状态位
#define pte_none(pte)       (!pte_val(pte))
#define pte_present(pte)    (pte_isset((pte), L_PTE_PRESENT))
#define pte_valid(pte)      (pte_isset((pte), L_PTE_VALID))
#define pte_accessible(mm, pte)      (mm_tlb_flush_pending(mm) ? pte_present(pte) :
pte_valid(pte))
#define pte_write(pte)      (pte_isclear((pte), L_PTE_RDONLY))
#define pte_dirty(pte)      (pte_isset((pte), L_PTE_DIRTY))
#define pte_young(pte)          (pte_isset((pte), L_PTE_YOUNG))
#define pte_exec(pte)       (pte_isclear((pte), L_PTE_XN))

//修改页表
#define mk_pte(page,prot) pfn_pte(page_to_pfn(page), prot)
static inline pte_t pte_mkdirty(pte_t pte)
static inline pte_t pte_mkold(pte_t pte)
static inline pte_t pte_mkclean(pte_t pte)
static inline pte_t pte_mkwrite(pte_t pte)
static inline pte_t pte_wrprotect(pte_t pte)
static inline pte_t pte_mkyoung(pte_t pte)
static inline void set_pte_at(struct mm_struct *mm, unsigned long addr,
                    pte_t *ptep, pte_t pteval)
int ptep_set_access_flags(struct vm_area_struct *vma,
                unsigned long address, pte_t *ptep,
                pte_t entry, int dirty)

//page 和 pfn 的关系
#define pte_pfn(pte)        ((pte_val(pte) & PHYS_MASK) >> PAGE_SHIFT)
#define pfn_pte(pfn,prot)   __pte(__pfn_to_phys(pfn) | pgprot_val(prot))
```

2.19.2.2 内存分配

内核中常用的内存分配 API 如下：

```
//分配和释放页面
static inline struct page * alloc_pages(gfp_t gfp_mask, unsigned int order)
unsigned long __get_free_pages(gfp_t gfp_mask, unsigned int order)
struct page *
__alloc_pages_nodemask(gfp_t gfp_mask, unsigned int order,
                struct zonelist *zonelist, nodemask_t *nodemask)
void free_pages(unsigned long addr, unsigned int order)
```

```
void __free_pages(struct page *page, unsigned int order)

//slab 分配器
struct kmem_cache *
kmem_cache_create(const char *name, size_t size, size_t align,
                  unsigned long flags, void (*ctor)(void *))
void kmem_cache_destroy(struct kmem_cache *s)
void *kmem_cache_alloc(struct kmem_cache *cachep, gfp_t flags)
void kmem_cache_free(struct kmem_cache *cachep, void *objp)
static void *kmalloc(size_t size, gfp_t flags)
void kfree(const void *objp)

//vmalloc 相关
void *vmalloc(unsigned long size)
void vfree(const void *addr)
```

2.19.2.3 VMA 操作相关

```
struct vm_area_struct * find_vma(struct mm_struct * mm, unsigned long addr);
struct vm_area_struct * find_vma_prev(struct mm_struct * mm, unsigned long addr,
                                      struct vm_area_struct **pprev);
struct vm_area_struct * find_vma_intersection(struct mm_struct * mm, unsigned long
start_addr, unsigned long end_addr);
static int find_vma_links(struct mm_struct *mm, unsigned long addr,
           unsigned long end, struct vm_area_struct **pprev,
           struct rb_node ***rb_link, struct rb_node **rb_parent);
int insert_vm_struct(struct mm_struct *mm, struct vm_area_struct *vma);
```

2.19.2.4 页面相关

内存管理当中最复杂的地方就是和页面相关的操作，内核中常用的 API 函数可以归纳如下几类：

- PG_XXX 标志位操作
- page 引用计数操作
- 匿名页面和 KSM 页面
- 页面操作
- 页面映射
- 缺页中断
- LRU 和页面回收

```
//PG_XXX 标志位操作
PageXXX()
SetPageXXX()
ClearPageXXX()
TestSetPageXXX()
TestClearPageXXX()
void lock_page(struct page *page)
int trylock_page(struct page *page)
void wait_on_page_bit(struct page *page, int bit_nr);
void wake_up_page(struct page *page, int bit)
static inline void wait_on_page_locked(struct page *page)
static inline void wait_on_page_writeback(struct page *page)
```

```
//page 引用计数操作
void get_page(struct page *page)
void put_page(struct page *page);
#define page_cache_get(page)        get_page(page)
#define page_cache_release(page)   put_page(page)
static inline int page_count(struct page *page)
static inline int page_mapcount(struct page *page)
static inline int page_mapped(struct page *page)
static inline int put_page_testzero(struct page *page)

//匿名页面和 KSM 页面
static inline int PageAnon(struct page *page)
static inline int PageKsm(struct page *page)
struct address_space *page_mapping(struct page *page)
void page_add_new_anon_rmap(struct page *page,
      struct vm_area_struct *vma, unsigned long address)

//页面操作
struct page *follow_page(struct vm_area_struct *vma,
            unsigned long address, unsigned int foll_flags)
struct page *vm_normal_page(struct vm_area_struct *vma, unsigned long addr,
                        pte_t pte)
long get_user_pages(struct task_struct *tsk, struct mm_struct *mm,
            unsigned long start, unsigned long nr_pages, int write,
            int force, struct page **pages, struct vm_area_struct **vmas)

//页面映射
void create_mapping_late(phys_addr_t phys, unsigned long virt,
                          phys_addr_t size, pgprot_t prot)
unsigned long do_mmap_pgoff(struct file *file, unsigned long addr,
                  unsigned long len, unsigned long prot,
                  unsigned long flags, unsigned long pgoff,
                  unsigned long *populate)
int remap_pfn_range(struct vm_area_struct *vma, unsigned long addr,
                 unsigned long pfn, unsigned long size, pgprot_t prot)

//缺页中断
int do_page_fault(unsigned long addr, unsigned int fsr, struct pt_regs *regs)
int handle_pte_fault(struct mm_struct *mm,
                  struct vm_area_struct *vma, unsigned long address,
                  pte_t *pte, pmd_t *pmd, unsigned int flags)
static int do_anonymous_page(struct mm_struct *mm, struct vm_area_struct *vma,
            unsigned long address, pte_t *page_table, pmd_t *pmd,
            unsigned int flags)
static int do_wp_page(struct mm_struct *mm, struct vm_area_struct *vma,
            unsigned long address, pte_t *page_table, pmd_t *pmd,
            spinlock_t *ptl, pte_t orig_pte)
static int do_wp_page(struct mm_struct *mm, struct vm_area_struct *vma,
            unsigned long address, pte_t *page_table, pmd_t *pmd,
            spinlock_t *ptl, pte_t orig_pte)

//LRU 和页面回收
void lru_cache_add(struct page *page)
#define lru_to_page(_head) (list_entry((_head)->prev, struct page, lru))
bool zone_watermark_ok(struct zone *z, unsigned int order, unsigned long mark,
                   int classzone_idx, int alloc_flags)
```