LINUX系统高级调试和优化
-- 内存管理

张银奎

2016/12/8

1

物理内存

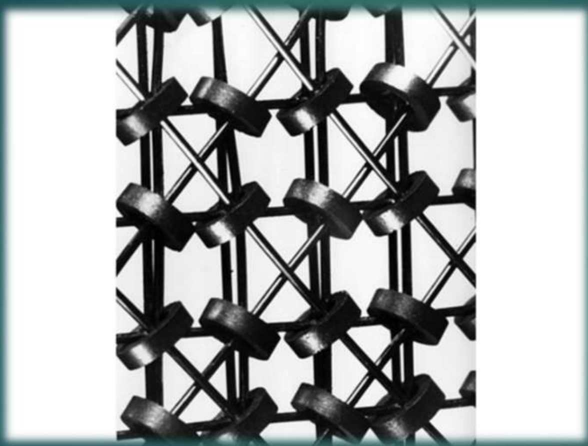
页表管理

虚拟内存

内核态池

故障调试

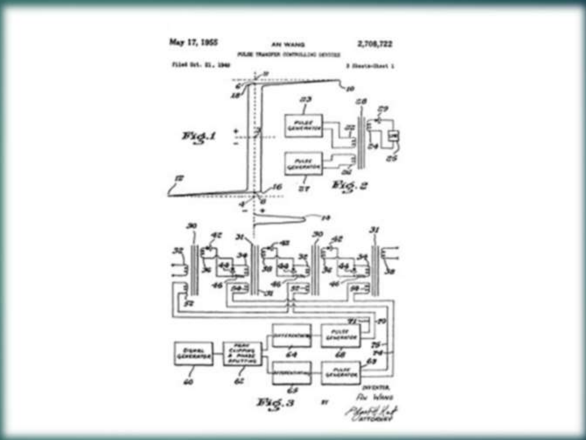
2



Magnetic Core memory
1953年MIT的旋风计算机首次使用

Core memory stack, equivalent to 28,672 bytes
Contains 229,376 ferrite cores, organized as 4096 56-bit "words"

http://ljkrakauer.com/LJK/essays/bits.htm



王安和他的Core内存专利，1955年IBM为其支付$500,000

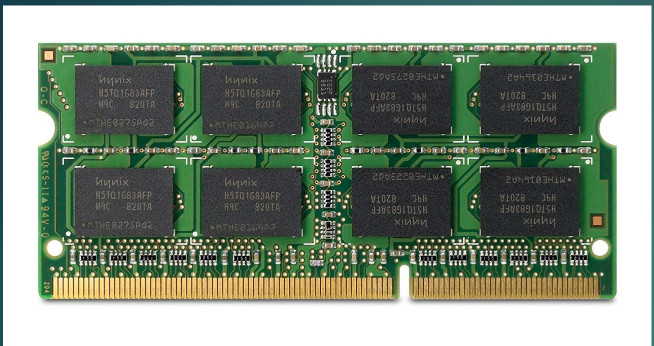
It took about seven years to convince people in the industry that magnetic core memory would work, and it took the next seven years to convince them that they had not all thought of it first.
—Jay Forrester, MIT

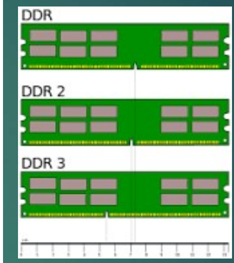
Frederick Viehe filed a core memory patent in 1947
An Wang filed one in 1949

RCA's Jan Rajchman and MIT's Jay Forrester filed in 1950 and 1951 respectively

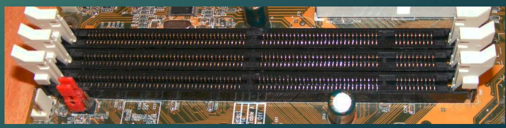
In 1964, after years of legal wrangling, IBM paid MIT $13 million for rights to Forrester's patent

* http://www.computerhistory.org/revolution/memory-storage/8/253





HP VH641AT 4-GB DDR3 RAM

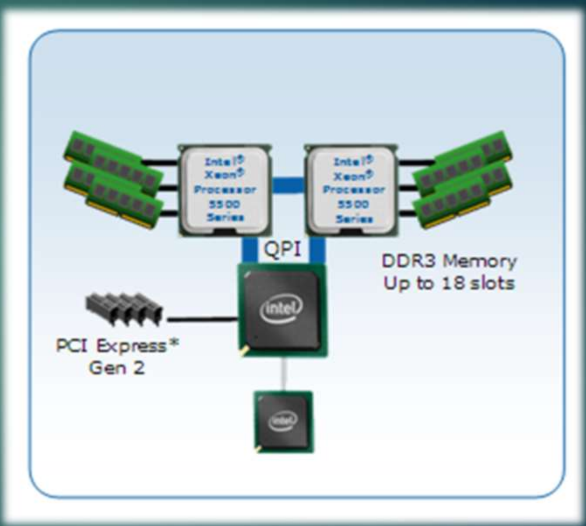


Double data rate dynamic random-access memory

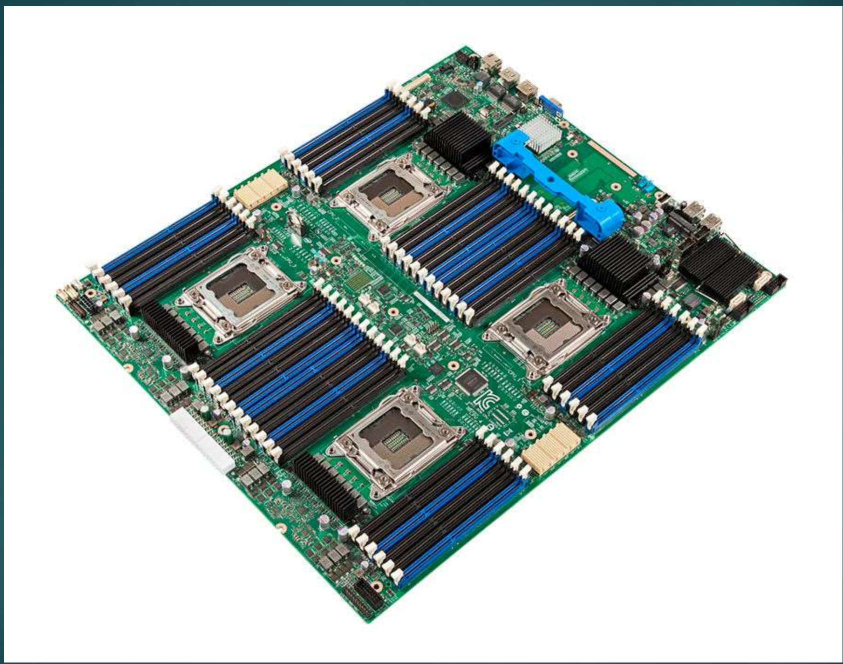
synchronous

NUMA

- Non-Uniform Memory Access
- 每个CPU有自己的内存（local memory）
- 一个CPU也可以访问其它CPU的内存，但是访问速度要比访问自己的内存慢很多
- 相对于UMA架构
  - 目前在使用的大多数x86 PC系统
- IA架构从代号为Nehalem的CPU开始采用NUMA



8

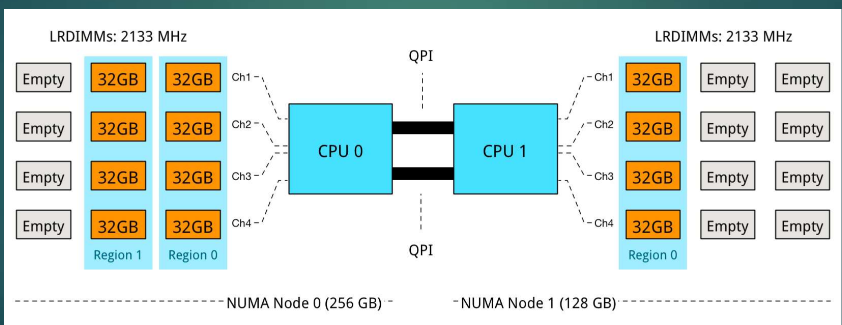


SSI EEB motherboard / Intel® / Intel® Xeon E5 / DDR3 SDRAM - S4600LH2, S4600LT2 series

9

Memory Bank

- 也被称为Memory Node
- 对于普通PC，只有一个Memory Bank/Node
- 不妨就翻译为内存银行



10

struct node

```
struct node {
	struct device	dev;

#if defined(CONFIG_MEMORY_HOTPLUG_SPARSE) && defined(CONFIG_HUGETLBFS)
	struct work_struct	node_work;
#endif
};
```

```
 * This is mainly for topological representation. We define the
 * basic 'struct node' here, which can be embedded in per-arch
 * definitions of processors.
```

- D:\bench\linux-4.4.14\include\linux\node.h


```
typedef struct pglist_data {
	struct zone node_zones[MAX_NR_ZONES];
	struct zonelist node_zonelists[MAX_ZONELISTS];
	int nr_zones;
	unsigned long node_start_pfn;
	unsigned long node_present_pages; /* total number of physical pages */
	unsigned long node_spanned_pages; /* total size of physical page
					     range, including holes */
	int node_id;
	wait_queue_head_t kswapd_wait;
	wait_queue_head_t pfmemalloc_wait;
	struct task_struct *kswapd;	/* Protected by
					   mem_hotplug_begin/end() */
	int kswapd_max_order;
	enum zone_type classzone_idx;
```

<linux/mmzone.h>

Denote a higher-level memory zone than the zone denotes.

Zones

- Each node is divided into a number of blocks called *zones*, which represent ranges within memory
- DMA-capable memory
  - Platform dependent
  - First 16MB of RAM on the x86 for ISA devices
  - PCI devices have no such limit
- Normal memory
- High memory
  - Platform dependent
  - > 32-bit addressable range

ZONE_DMA

```
#ifdef CONFIG_ZONE_DMA
        /*
         * ZONE_DMA is used when there are devices that are not able
         * to do DMA to all of addressable memory (ZONE_NORMAL). Then we
         * carve out the portion of memory that is needed for these devices.
         * The range is arch specific.
         *
         * Some examples
         *
         * Architecture            Limit
         * ---------------------------
         * parisc, ia64, sparc     <4G
         * s390                    <2G
         * arm             Various
         * alpha           Unlimited or 0-16MB.
         *
         * i386, x86_64 and multiple other arches
         *                 <16M.
         */
        ZONE_DMA,
#endif
```

```
ge@gewubox:/boot$ cat config-3.12.2 | grep ZONE_DMA
# CONFIG_ZONE_DMA32 is not set
CONFIG_ZONE_DMA=y
CONFIG_ZONE_DMA_FLAG=1
ge@gewubox:/boot$
```

- <linux/mmzone.h>
- 兼容老的PC设备

```
#ifdef CONFIG_ZONE_DMA32
	/*
	 * x86_64 needs two ZONE_DMAs because it supports devices that are
	 * only able to do DMA to the lower 16M but also 32 bit devices that
	 * can only do DMA areas below 4G.
	 */
	ZONE_DMA32,
#endif
	/*
	 * Normal addressable memory is in ZONE_NORMAL. DMA operations can be
	 * performed on pages in ZONE_NORMAL if the DMA devices support
	 * transfers to all addressable memory.
	 */
	ZONE_NORMAL,
#ifdef CONFIG_HIGHMEM
	/*
	 * A memory area that is only addressable by the kernel through
	 * mapping portions into its own address space. This is for example
	 * used by i386 to allow the kernel to address the memory beyond
	 * 900MB. The kernel will set up special mappings (page
	 * table entries on i386) for each page that the kernel needs to
	 * access.
	 */
	ZONE_HIGHMEM,
#endif
	ZONE_MOVABLE,
#ifdef CONFIG_ZONE_DEVICE
	ZONE_DEVICE,
#endif
	__MAX_NR_ZONES

};
```



```
struct zone {
	/* Read-mostly fields */
#ifdef CONFIG_NUMA
	int node;
#endif

	struct pglist_data	*zone_pgdat;

	/* zone_start_pfn == zone_start_paddr >> PAGE_SHIFT */
	unsigned long		zone_start_pfn;

	unsigned long		managed_pages;
	unsigned long		spanned_pages;
	unsigned long		present_pages;

	const char		*name;
```

Each zone is described by a zone struct. zone structs keep track of information like page usage statistics, free area information and locks.

<linux/mmzone.h>

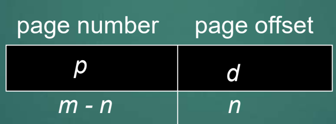
Page

- Physical and Virtual Memory divided into chunks of the same size called pages (4 KB on x86)
- use of page tables for translation easier translation
- each page has unique page frame number (**PFN**)
- an address consists of offset and (virtual) PFN ⇒ look up
- (physical) PFN and access at correct offset
- translation lookaside buffer (TLB)
- flags indicate if the page is in real memory
- Swapping/Paging

17

分页内存寻址

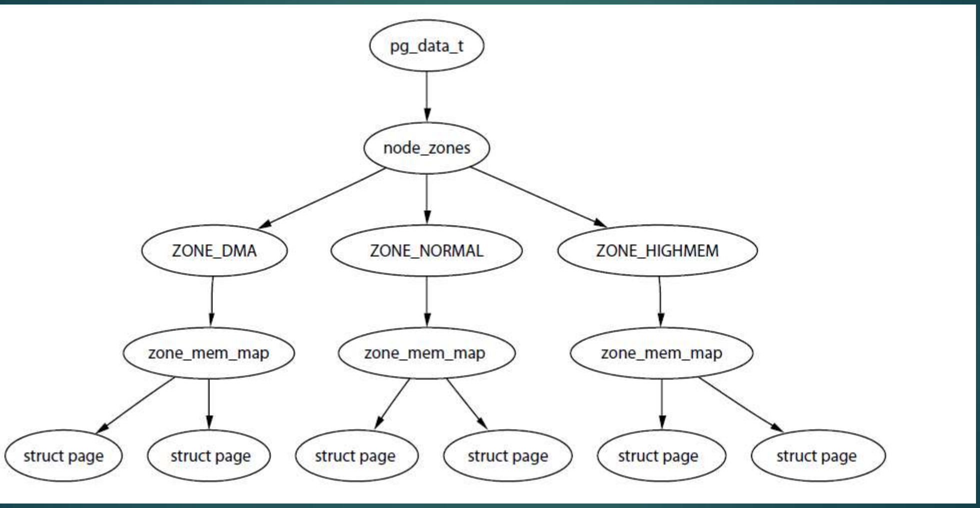
- The MMU causes every memory reference instruction address to contain a:
  - **Page number (*p*)** – index into a *page table* array containing the base address of every frame in physical memory
  - **Page offset (d)** – Offset into a physical frame



  - Logical addresses contain m bits, n of which are a displacement. There are 2^{m-n} pages of size 2^n
  - **Advantage:** No external fragmentation

18

Nodes, Zones and Pages



Physical Pages Allocation API

- struct page * alloc page(unsigned int gfp mask)
  - Allocates a single page and returns a struct address.
- struct page * alloc pages(unsigned int gfp mask, unsigned int order)
  - Allocates 2order number of pages and returns a struct page.
- unsigned long get free page(unsigned int gfp mask)
  - Allocates a single page, zeros it, and returns a virtual address.
- unsigned long get free page(unsigned int gfp mask)
  - Allocates a single page and returns a virtual address.
- unsigned long get free pages(unsigned int gfp mask, unsigned int order)
  - Allocates 2order number of pages and returns a virtual address.
- struct page * get dma pages(unsigned int gfp mask, unsigned int order)
  - Allocates 2order number of pages from the DMA zone and returns a struct page.

The `alloc_pages` Interface

- Core Linux page allocator function

```
struct page *alloc_pages_node(int nid, unsigned int flags,
                              unsigned int order);
```

  - `nid`: NUMA node ID
- Two higher level macros

```
struct page *alloc_pages(unsigned int flags,
                         unsigned int order);
struct page *alloc_page(unsigned int flags);
```

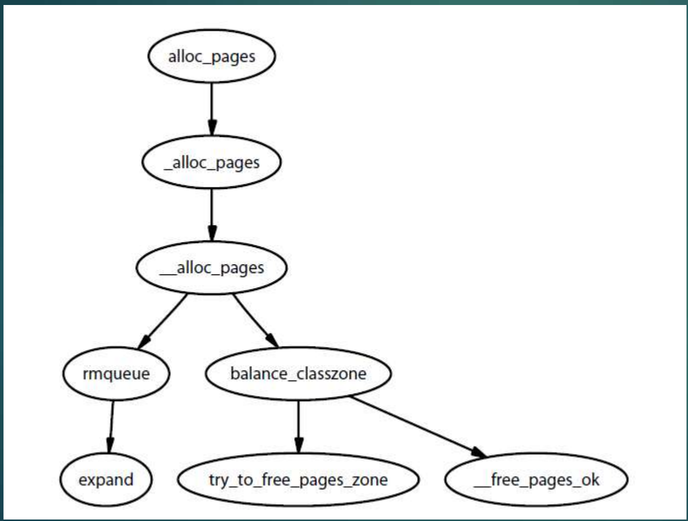
  - Allocate memory on the current NUMA node

The `alloc_pages` Interface

- To release pages, call

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

/* optimized calls for cache-resident or non-cache-resident pages */
void free_hot_page(struct page *page);
void free_cold_page(struct page *page);
```

alloc_pages()



- UMA (function in mm/page alloc.c)
- NUMA(function in mm/numa.c)

23

drivers\android\binder.c

```
*page = alloc_page(GFP_KERNEL | __GFP_HIGHMEM | __GFP_ZERO);
if (*page == NULL) {
      pr_err("%d: binder_alloc_buf failed for page at %p\n",
            proc->pid, page_addr);
      goto err_alloc_page_failed;
}
```

- GFP HIGHMEM Allocate from ZONE HIGHMEM if possible.

24

sd.c

```
page = alloc_page(GFP_ATOMIC | __GFP_ZERO);
if (!page)
      return BLKPREP_DEFER;
```

```
/**
 * sd_setup_discard_cmnd - unmap blocks on thinly provisioned device
 * @sdp: scsi device to operate one
 * @rq: Request to prepare
 *
 * Will issue either UNMAP or WRITE SAME(16) depending on preference
 * indicated by target device.
 **/
static int sd_setup_discard_cmnd(struct scsi_cmnd *cmd)
{
        struct request *rq = cmd->request;
        struct scsi_device *sdp = cmd->device;
        struct scsi_disk *sdkp = scsi_disk(rq->rq_disk);
        sector_t sector = blk_rq_pos(rq);
        unsigned int nr_sectors = blk_rq_sectors(rq);
        unsigned int nr_bytes = blk_rq_bytes(rq);
        unsigned int len;
        int ret;
        char *buf;
        struct page *page;

        sector >>= ilog2(sdp->sector_size) - 9;
        nr_sectors >>= ilog2(sdp->sector_size) - 9;

        page = alloc_page(GFP_ATOMIC | __GFP_ZERO);
        if (!page)
                return BLKPREP_DEFER;
```

- D:\bench\linux-4.4.14\drivers\scsi\sd.c

Get Free Page (GFP) Flags

```
/* Plain integer GFP bitmasks. Do not use this directly. */
#define ___GFP_DMA              0x01u
#define ___GFP_HIGHMEM          0x02u
#define ___GFP_DMA32            0x04u
#define ___GFP_MOVABLE          0x08u
#define ___GFP_RECLAIMABLE      0x10u
#define ___GFP_HIGH             0x20u
#define ___GFP_IO               0x40u
#define ___GFP_FS               0x80u
#define ___GFP_COLD             0x100u
#define ___GFP_NOWARN           0x200u
#define ___GFP_REPEAT           0x400u
#define ___GFP_NOFAIL           0x800u
#define ___GFP_NORETRY          0x1000u
#define ___GFP_MEMALLOC         0x2000u
#define ___GFP_COMP             0x4000u
#define ___GFP_ZERO             0x8000u
#define ___GFP_NOMEMALLOC       0x10000u
#define ___GFP_HARDWALL         0x20000u
#define ___GFP_THISNODE         0x40000u
#define ___GFP_ATOMIC           0x80000u
#define ___GFP_NOACCOUNT        0x100000u
#define ___GFP_NOTRACK          0x200000u
#define ___GFP_DIRECT_RECLAIM   0x400000u
#define ___GFP_OTHER_NODE       0x800000u
#define ___GFP_WRITE            0x1000000u
#define ___GFP_KSWAPD_RECLAIM   0x2000000u
```

- /linux/gfp.h

物理内存

页表管理

虚拟内存

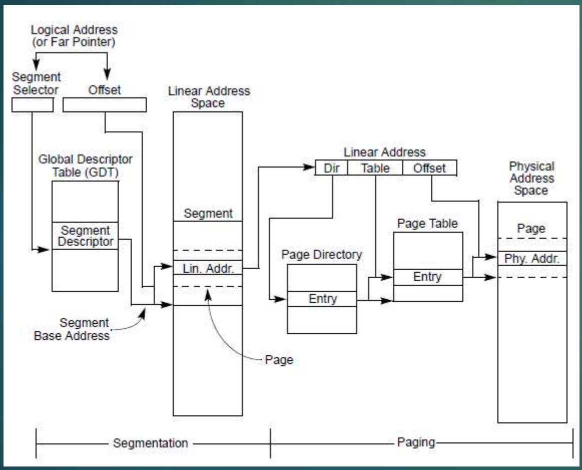
内核态池

故障调试

保护模式下的内存管理

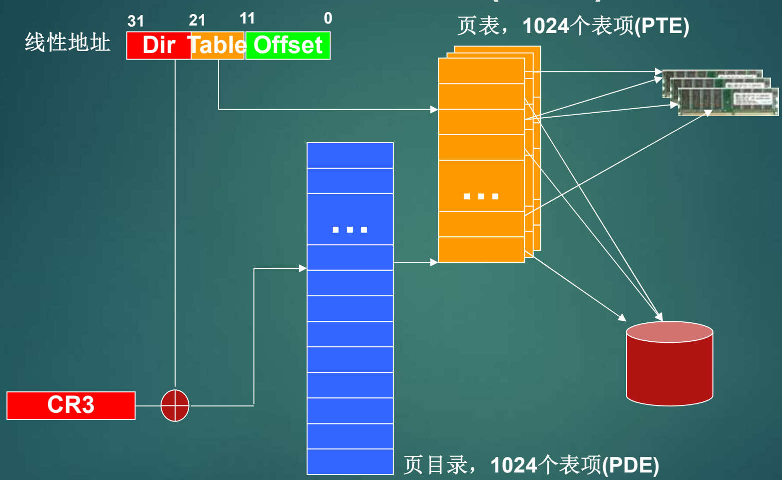
- 保护模式
  - 保护系统中的每个任务
  - 每个任务有自己的地址空间，在同一任务空间中保护搞特权的代码
- 保护模式下，进程空间中的代码和数据使用的都是虚拟地址
- CPU负责把虚拟地址翻译为物理地址

X86 CPU的内存管理



- 两种机制
- 段机制和页机制
- 段机制不可以禁止
- 页机制可以启用或者禁止

从线性地址到物理地址(x86)



- 上面是以4KB内存页(未启用PAE)为例
- 1024PDE*1024PTE=2^{20}页

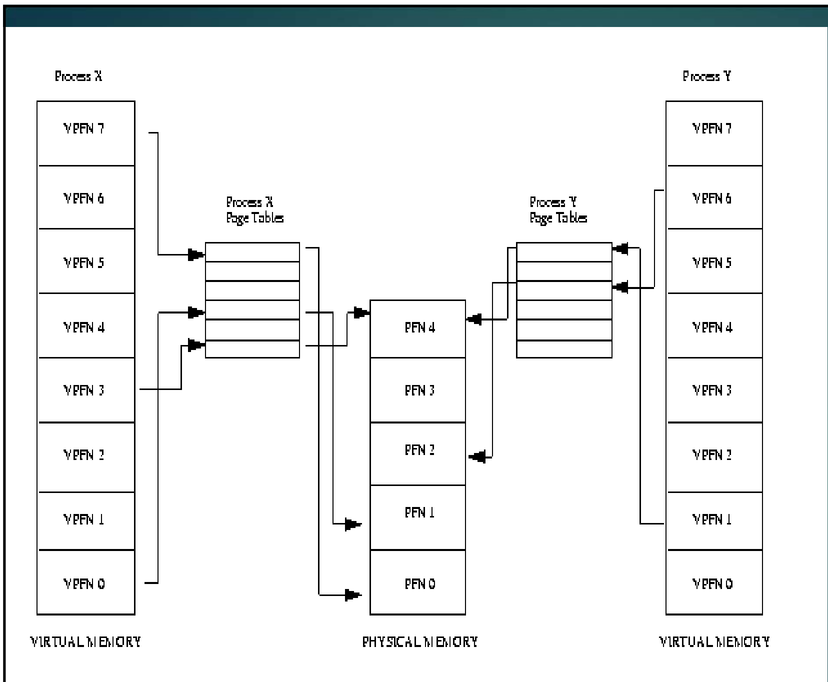
CR3寄存器

- IA32 CPU用来记录当前页目录表的物理基地址的寄存器，简称PDBR (Page Directory Base Register)。
- 每个进程的最重要属性之一。
- 切换任务时，系统会将前一个任务的CR3作为上下文信息（context）的一部分保存起来。在开始执行新任务前，系统会恢复寄存器状态，包括CR3, EFLAGS, EIP, 等。
- 切换CR3寄存器意味着切换地址空间。
- 不同进程拥有不同的地址空间（CR3内容）——隔离与保护

lkd> !process 0 0
****** NT ACTIVE PROCESS DUMP ******
PROCESS 89e32830 SessionId: none Cid: 0004   Peb: 00000000 ParentCid: 0000
DirBase: 006f1000 ObjectTable: e1000c98 HandleCount: 463.
Image: System

Each process is a pointer (mm struct→pgd) to its own *PGD* which is a physical page frame.
task_size = 3221225472, highest_vm_end = 3220901888, pgd = 0xd7e2e000,

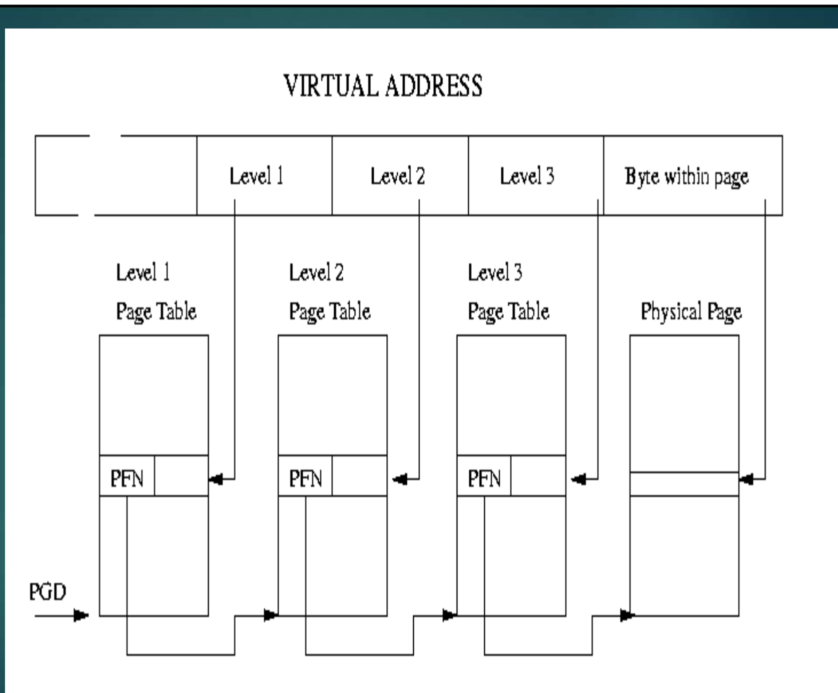
31



*Introduction to Memory Management*黄偉修

Virtual memory model

- Linux Page
  - *__.There level page Tables__*
  - *__. Independent to processor__*
    - *- X86 32位经典分页模式 2-level*
    - *- Alpha 3-level*
- *each platform must provide translation macros that allow the kernel to traverse the page tables for a particular process*



Virtual memory model

- Address Transfer
- . Ex: ***386 processor***

A 32-bit Linear address is divided as follows:

31 22	21 12	11 0
DIR	TABLE	OFFSET

Physical address is then computed (in hardware) as:

CR3 + DIR points to the table_base.

table_base + TABLE points to the page_base.

physical_address page_base + OFFSET

Format for Page directory and Page table entry:

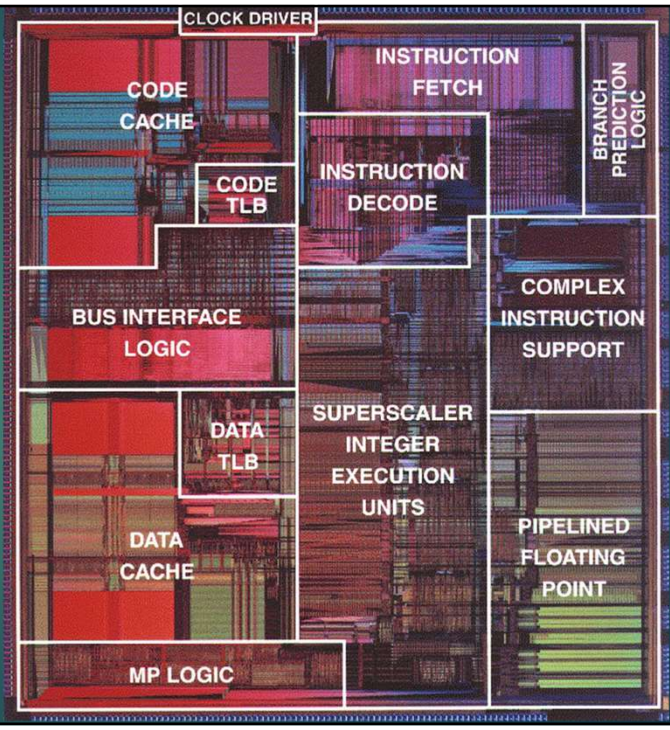
31 …… 12	11 .. 9	8	7	6	5	4	3	2	1	0
ADDRESS	OS	0	0	D	A	0	0	U/S	R/W	P

D 1 means page is dirty (undefined for page directory entry).
R/W 0 means readonly for user.
U/S 1 means user page.
P 1 means page is present in memory.
A 1 means page has been accessed (set to 0 by aging).
OS bits can be used for LRU etc, and are defined by the OS.

I.e :When a page is swapped, bits 1-31 are used to mark where a page is stored in swap (bit 0 must be 0).

TLB

- **Translation Lookaside Buffer**
- a small associative memory that caches virtual to physical page table resolutions



__flush_tlb

- On the x86, the process page table is loaded by copying mm struct→pgd into the cr3 register, which has the side effect of flushing the TLB. In fact, this is how the function __flush_tlb() is implemented in the architecture-dependent code.

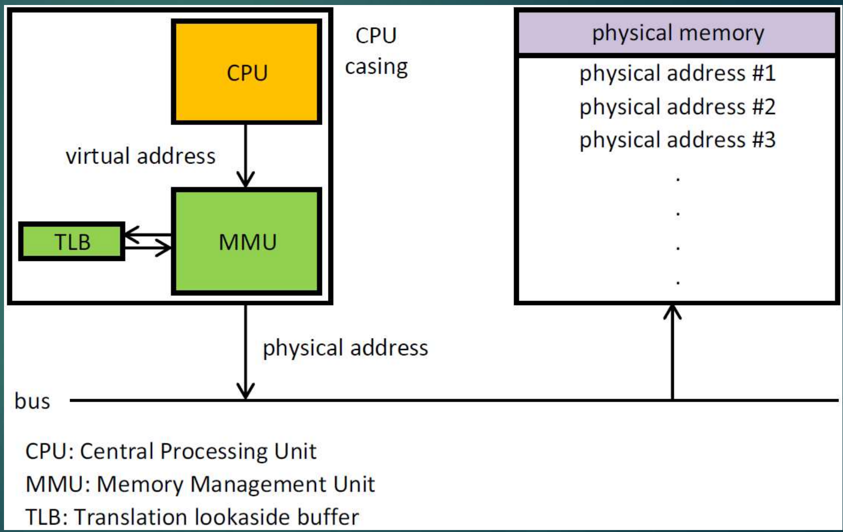
分页(Paging)

Definition: A page is a fixed-sized block of logical memory, generally a power of 2 in length between 512 and 8,192 bytes
Definition: A frame is a fixed-sized block of physical memory. Each frame corresponds to a single page
Definition: A Page table is an array that translates from pages to frames

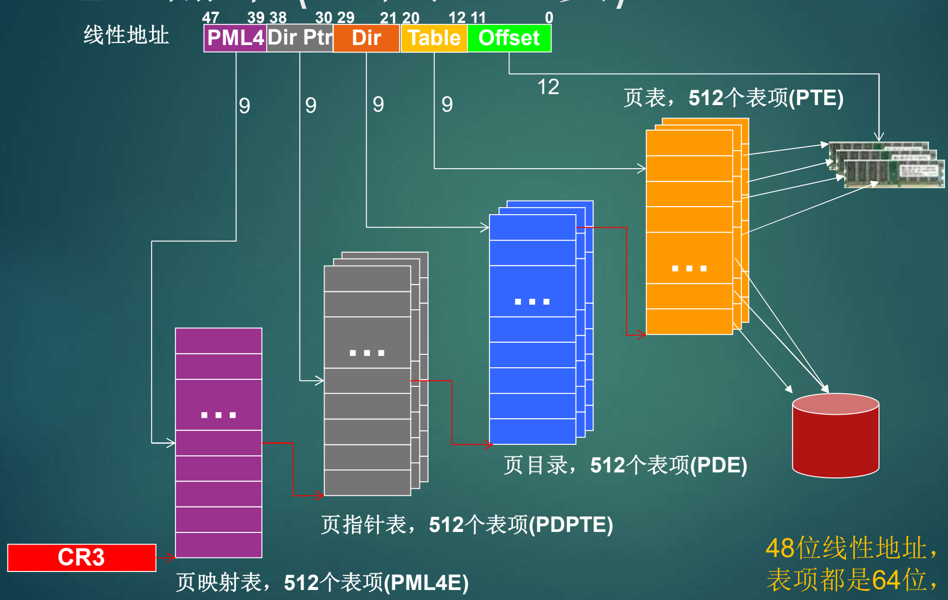
- **Operating System responsibilities**
  - Maintain the page table
  - Allocate sufficient pages from free frames to execute a program
- **Benefit:** Logical address space of a process can be noncontiguous and allocated as needed
- **Issue:** Internal fragmentation

MMU

- **Memory Management Unit (MMU)**: Device mapping logical (virtual) addresses to physical addresses
- **Logical address** – process view of memory
- **Physical address** – MMU view of memory

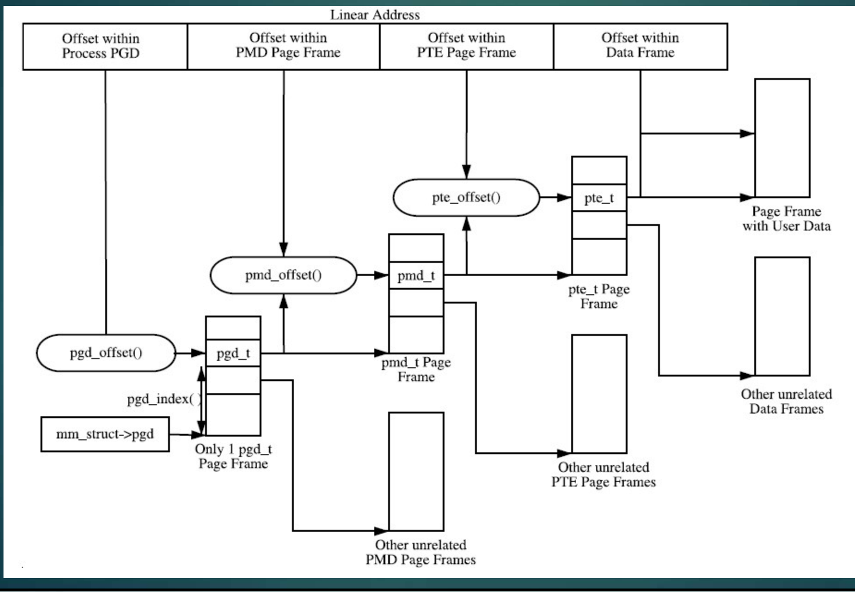


地址翻译 (64位, 4KB页)



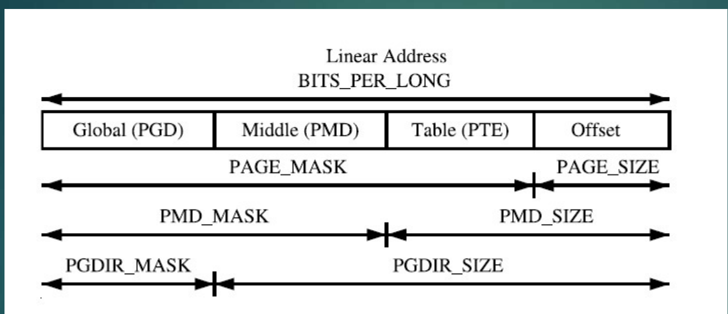
48位线性地址，52位物理地址
表项都是64位，8字节

Linux的抽象页表结构



- 三层表结构

宏

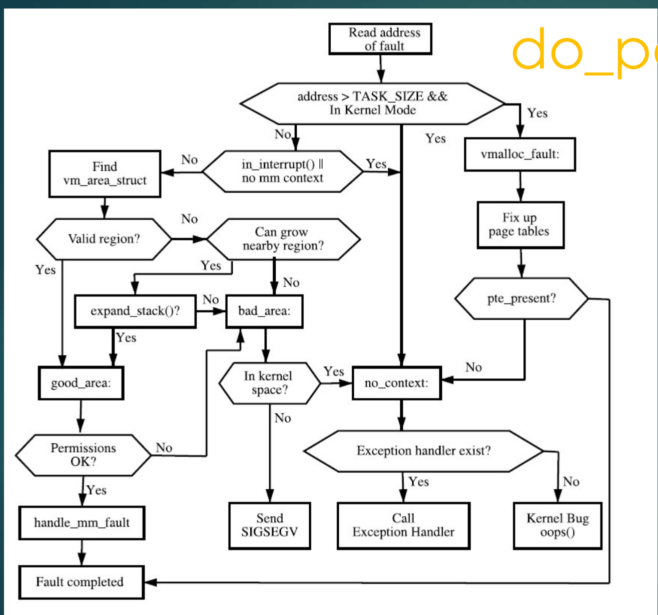


- Linear Address Size and Mask Macros

<linux/mm.h>

```
/*
 * vm_fault is filled by the the pagefault handler and passed to the vma's
 * ->fault function. The vma's ->fault is responsible for returning a bitmask
 * of VM_FAULT_xxx flags that give details about how the fault was handled.
 *
 * pgoff should be used in favour of virtual_address, if possible.
 */
struct vm_fault {
	unsigned int flags;		/* FAULT_FLAG_xxx flags */
	pgoff_t pgoff;			/* Logical page offset based on vma */
	void __user *virtual_address;	/* Faulting virtual address */

	struct page *cow_page;		/* Handler may choose to COW */
	struct page *page;		/* ->fault handlers should return a
					 * page here, unless VM_FAULT_NOPAGE
					 * is set (which is also implied by
					 * VM_FAULT_ERROR).
					 */
	/* for ->map_pages() only */
	pgoff_t max_pgoff;		/* map pages for offset from pgoff till
					 * max_pgoff inclusive */
	pte_t *pte;			/* pte entry associated with ->pgoff */
};
```


do_page_fault()



细分

Minor PF

- 不需要从磁盘换进页
- 例如：访问没有分配过的malloc内存块

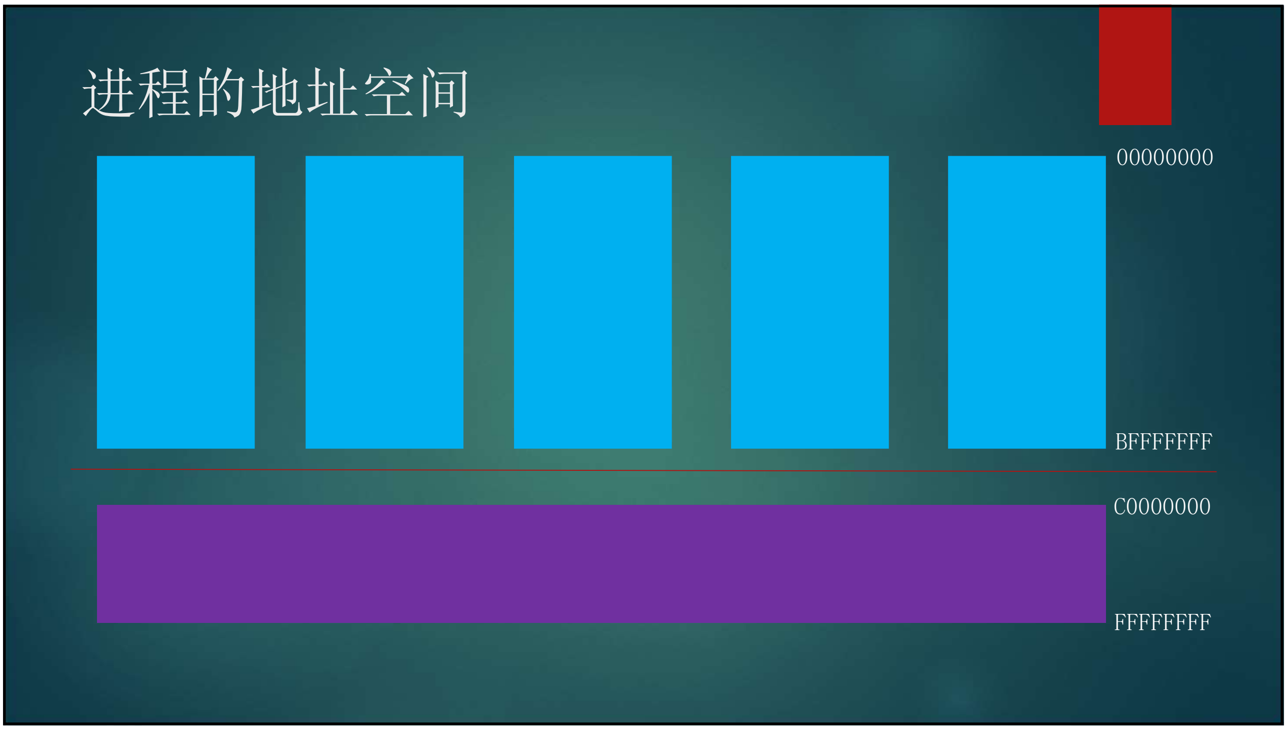
Major PF

- 需要从磁盘读取内容
- 情况1：换进交换出去的页
- 情况2：访问文件映射的页

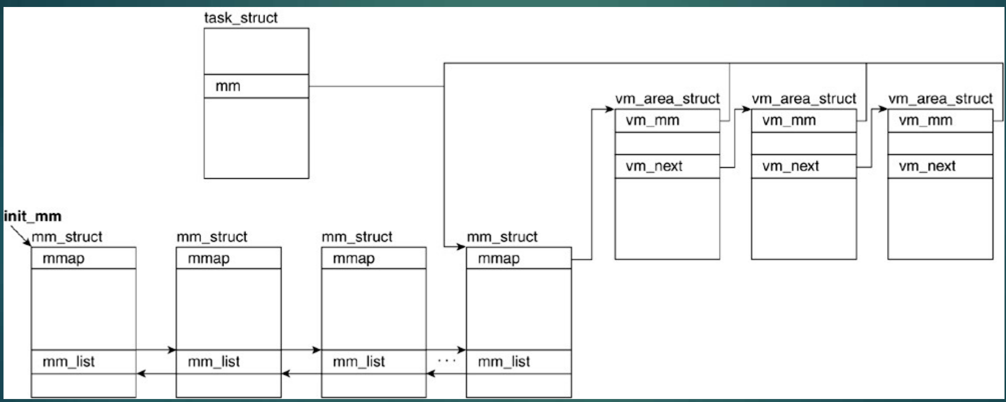

```
Fields Management for window 1:Def, whose current sort field is %CPU
   Navigate with Up/Dn, Right selects for move then <Enter> or Left commits,
   'd' or <Space> toggles display, 's' sets sort.  Use 'q' or <Esc> to end!

* PID     = Process Id             TIME    = CPU Time
* USER    = Effective User Name    SWAP    = Swapped Size (KiB)
* PR      = Priority               CODE    = Code Size (KiB)
* NI      = Nice Value             DATA    = Data+Stack (KiB)
* VIRT    = Virtual Image (KiB)    nMaj    = Major Page Faults
* RES     = Resident Size (KiB)    nMin    = Minor Page Faults
* SHR     = Shared Memory (KiB)    nDRT    = Dirty Pages Count
* S       = Process Status         WCHAN   = Sleeping in Function
* %CPU    = CPU Usage              Flags   = Task Flags <sched.h>
* %MEM    = Memory Usage (RES)     CGROUPS = Control Groups
* TIME+   = CPU Time, hundredths   SUPGIDS = Supp Groups IDs
* COMMAND = Command Name/Line      SUPGRPS = Supp Groups Names
  PPID    = Parent Process pid     TGID    = Thread Group Id
  UID     = Effective User Id      ENVIRON = Environment vars
  RUID    = Real User Id           vMj     = Major Faults delta
  RUSER   = Real User Name         vMn     = Minor Faults delta
  SUID    = Saved User Id          USED    = Res+Swap Size (KiB)
  SUSER   = Saved User Name        nsIPC   = IPC namespace Inode
  GID     = Group Id               nsMNT   = MNT namespace Inode
  GROUP   = Group Name             nsNET   = NET namespace Inode
  PGRP    = Process Group Id       nsPID   = PID namespace Inode
  TTY     = Controlling Tty        nsUSER  = USER namespace Inode
  TPGID   = Tty Process Grp Id     nsUTS   = UTS namespace Inode
  SID     = Session Id
  nTH     = Number of Threads
  P       = Last Used Cpu (SMP)
```

需要较高版本的TOP命令(WSL的可以)


物理
内存
页表
管理
虚拟
内存
内核
态池
故障
调试


进程的地址空间
00000000
BFFFFFFF
C0000000
FFFFFFFF

task_struct->mm



Lister - [D:\bench\linux-3.16.3\include\linux\mm_types.h]

File Edit Options Help

```
};

struct kioctx_table;
struct mm_struct {
        struct vm_area_struct *mmap;            /* list of VMAs */
        struct rb_root mm_rb;
        u32 vmacache_seqnum;                   /* per-thread vmacache */
#ifdef CONFIG_MMU
        unsigned long (*get_unmapped_area) (struct file *filp,
                                unsigned long addr, unsigned long len,
                                unsigned long pgoff, unsigned long flags);
#endif
        unsigned long mmap_base;                /* base of mmap area */
        unsigned long mmap_legacy_base;         /* base of mmap area in bottom-up allocations */
        unsigned long task_size;                /* size of task vm space */
        unsigned long highest_vm_end;           /* highest vma end address */
        pgd_t * pgd;
        atomic_t mm_users;                      /* How many users with user space? */
        atomic_t mm_count;                      /* How many references to "struct mm_struct" (users count as 1) */
        atomic_long_t nr_ptes;                  /* Page table pages */
        int map_count;                          /* number of VMAs */

        spinlock_t page_table_lock;             /* Protects page tables and some counters */
        struct rw_semaphore mmap_sem;

        struct list_head mmlist;                /* List of maybe swapped mm's.  These are globally strung
                                                 * together off init_mm.mmlist, and are protected
                                                 * by mmlist_lock
                                                 */


        unsigned long hiwater_rss;      /* High-watermark of RSS usage */
        unsigned long hiwater_vm;       /* High-water virtual memory usage */
```

```
(gdb) p *mm
$17 = {mmap = 0xd8b05900, mm_rb = {rb_node = 0xd1e7e8b0},
 mmap_cache = 0xe368ca80,
 get_unmapped_area = 0xc114bd60 <arch_get_unmapped_area_topdown>,
 mmap_base = 3078336512, mmap_legacy_base = 1073741824,
 task_size = 3221225472, highest_vm_end = 3220901888, pgd = 0xd7e2e000,
 mm_users = {counter = 2}, mm_count = {counter = 1}, map_count = 29,
 page_table_lock = {{rlock = {raw_lock = {{head_tail = 4626, tickets = {
       head = 18 '\022', tail = 18 '\022'}}}}}}, mmap_sem = {count = 1,
  wait_lock = {raw_lock = {{head_tail = 0, tickets = {head = 0 '\000',
      tail = 0 '\000'}}}}, wait_list = {next = 0xeef01e3c,
   prev = 0xeef01e3c}}, mmlist = {next = 0xeef01e44, prev = 0xeef01e44},
 hiwater_rss = 170, hiwater_vm = 617, total_vm = 601, locked_vm = 0,
 pinned_vm = 0, shared_vm = 524, exec_vm = 509, stack_vm = 34, def_flags = 0,
 nr_ptes = 5, start_code = 134512640, end_code = 134539028,
 start_data = 134545172, end_data = 134546796, start_brk = 166912000,
 brk = 167047168, start_stack = 3220901136, arg_start = 3220901681,
 arg_end = 3220901705, env_start = 3220901705, env_end = 3220901870,
 saved_auxv = {32, 3078194196, 33, 3078193152, 16, 126614527, 6, 4096, 17,
  100, 3, 134512692, 4, 32, 5, 8, 7, 3078197248, 8, 0, 9, 134518128, 11, 0,
  12, 0, 13, 0, 14, 0, 23, 0, 25, 3220901371, 31, 3220901870, 15,
  3220901387, 0, 0, 0, 0, 0, 0}, rss_stat = {count = {{counter = 153}, {
    counter = 34}, {counter = 0}}}, binfmt = 0xc198373c,
 cpu_vm_mask_var = {{bits = {0}}}, context = {ldt = 0x0, size = 0, lock = {
---Type <return> to continue, or q <return> to quit---
```



```
(gdb) p *task->mm
$15 = {mmap = 0xdd9ef420, mm_rb = {rb_node = 0xdd9efeb0},
 mmap_cache = 0xdd9ef420,
 get_unmapped_area = 0xc114bd60 <arch_get_unmapped_area_topdown>,
 mmap_base = 3077722112, mmap_legacy_base = 1073741824,
 task_size = 3221225472, highest_vm_end = 3213553664,
 pgd = 0xeef99000, mm_users = {counter = 1}, mm_count = {
  counter = 1}, map_count = 29, page_table_lock = {{rlock = {
    raw_lock = {{head_tail = 4626, tickets = {head = 18 '\022',
       tail = 18 '\022'}}}}}}, mmap_sem = {count = 0,
  wait_lock = {raw_lock = {{head_tail = 0, tickets = {
       head = 0 '\000', tail = 0 '\000'}}}}, wait_list = {
   next = 0xd8bdd3bc, prev = 0xd8bdd3bc}}, mmlist = {
  next = 0xd8bdd3c4, prev = 0xd8bdd3c4}, hiwater_rss = 171,
 hiwater_vm = 617, total_vm = 601, locked_vm = 0, pinned_vm = 0,
 shared_vm = 524, exec_vm = 509, stack_vm = 34, def_flags = 0,
 nr_ptes = 5, start_code = 134512640, end_code = 134539028,
 start_data = 134545172, end_data = 134546796,
 start_brk = 157958144, brk = 158093312, start_stack = 3213547312,
 arg_start = 3213553457, arg_end = 3213553481,
 env_start = 3213553481, env_end = 3213553646, saved_auxv = {32,
  3077579796, 33, 3077578752, 16, 126614527, 6, 4096, 17, 100, 3,
  134512692, 4, 32, 5, 8, 7, 3077582848, 8, 0, 9, 134518128, 11,
  0, 12, 0, 13, 0, 14, 0, 23, 0, 25, 3213547547, 31, 3213553646,
```

```
15, 3213547563, 0, 0, 0, 0, 0, 0}, rss_stat = {count = {{
      counter = 153}, {counter = 35}, {counter = 0}}},
binfmt = 0xc198373c, cpu_vm_mask_var = {{bits = {0}}}, context = {
  ldt = 0x0, size = 0, lock = {count = {counter = 1},
    wait_lock = {{rlock = {raw_lock = {{head_tail = 0, tickets = {
              head = 0 '\000', tail = 0 '\000'}}}}}},
    wait_list = {next = 0xd8bdd4f4, prev = 0xd8bdd4f4},
    owner = 0x0, spin_mlock = 0x0}, vdso = 0xb7702000},
flags = 207, core_state = 0x0, ioctx_lock = {{rlock = {
      raw_lock = {{head_tail = 0, tickets = {head = 0 '\000',
          tail = 0 '\000'}}}}}}, ioctx_table = 0x0,
owner = 0xd7e03400, exe_file = 0xeed563c0, mmu_notifier_mm = 0x0,
pmd_huge_pte = 0x0, uprobes_state = {xol_area = 0x0}}


(gdb) p task->mm->mmap->vm_file->f_path->dentry->d_iname
$11 = "syslogd\000
```


vma

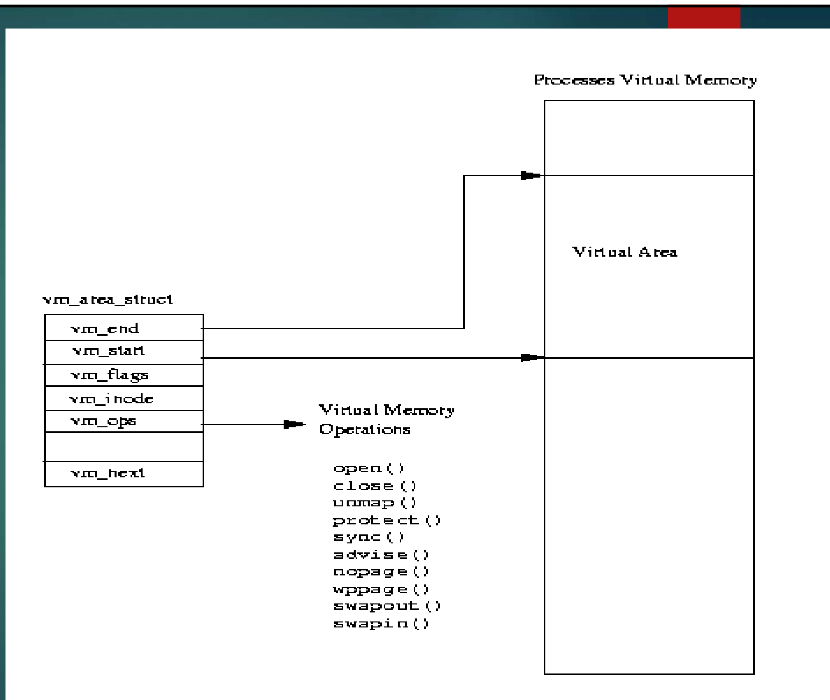
```
(gdb) p *task->mm->mmap
$16 = {vm_start = 134512640, vm_end = 134541312,
  vm_next = 0xdd9efa80, vm_prev = 0x0, vm_rb = {
    __rb_parent_color = 3718183569, rb_right = 0x0, rb_left = 0x0},
  rb_subtree_gap = 134512640, vm_mm = 0xd8bdd380, vm_page_prot = {
    pgprot = 37}, vm_flags = 134219893, shared = {linear = {rb = {
        __rb_parent_color = 3718183601, rb_right = 0x0,
        rb_left = 0x0}, rb_subtree_last = 6}, nonlinear = {
      next = 0xdd9efab1, prev = 0x0}}, anon_vma_chain = {
    next = 0xdd9ef460, prev = 0xdd9ef460}, anon_vma = 0x0,
  vm_ops = 0xc16a5ebc, vm_pgoff = 0, vm_file = 0xeed563c0,
  vm_private_data = 0x0}

  (gdb) p sizeof(*vma->vm_mm)
  $12 = 428
```

arch\x86\include\asm\mmu.h

```
typedef struct {
	void *ldt;
	int size;

#ifdef CONFIG_X86_64
	/* True if mm supports a task running in 32 bit compatibility mode. */
	unsigned short ia32_compat;
#endif

	struct mutex lock;
	void __user *vdso;
} mm_context_t;
```


内存映射

- Memory Mappin
- *把文件映射到虚拟地址空间*
- *1或者多个vm_area_struct结构体*



maps

```
ge@gewubox:/proc/2411$ cat maps
08048000-08124000 r-xp 00000000 08:01 393220     /bin/bash
08124000-08125000 r--p 000db000 08:01 393220     /bin/bash
08125000-0812a000 rw-p 000dc000 08:01 393220     /bin/bash
0812a000-0812f000 rw-p 00000000 00:00 0
08aaa000-08aab000 rw-p 00000000 00:00 0          [heap]
08aab000-08aac000 rw-p 00000000 00:00 0          [heap]
08aac000-08aad000 rw-p 00000000 00:00 0          [heap]
08aad000-08aae000 rw-p 00000000 00:00 0          [heap]
08aae000-08aaf000 rw-p 00000000 00:00 0          [heap]
08aaf000-08ab0000 rw-p 00000000 00:00 0          [heap]
08ab0000-08ab1000 rw-p 00000000 00:00 0          [heap]
08ab1000-08ab2000 rw-p 00000000 00:00 0          [heap]
08ab2000-08ab3000 rw-p 00000000 00:00 0          [heap]
08ab3000-08ab4000 rw-p 00000000 00:00 0          [heap]
08ab4000-08ab5000 rw-p 00000000 00:00 0          [heap]
08ab5000-08ab6000 rw-p 00000000 00:00 0          [heap]
08ab6000-08ab7000 rw-p 00000000 00:00 0          [heap]
08ab7000-08ab8000 rw-p 00000000 00:00 0          [heap]
```



```
b73b4000-b73bb000 r--s 00000000 08:01 395274     /usr/lib/i386-linux-gnu/gconv/gconv-modules.cache
b73bb000-b75bb000 r--p 00000000 08:01 268965     /usr/lib/locale/locale-archive
b75bb000-b75bc000 rw-p 00000000 00:00 0
b75bc000-b775f000 r-xp 00000000 08:01 154504     /lib/i386-linux-gnu/libc-2.15.so
b775f000-b7761000 r--p 001a3000 08:01 154504     /lib/i386-linux-gnu/libc-2.15.so
b7761000-b7762000 rw-p 001a5000 08:01 154504     /lib/i386-linux-gnu/libc-2.15.so
b7762000-b7765000 rw-p 00000000 00:00 0
b7765000-b7766000 rw-p 00000000 00:00 0
b7766000-b7769000 r-xp 00000000 08:01 154496     /lib/i386-linux-gnu/libdl-2.15.so
b7769000-b776a000 r--p 00002000 08:01 154496     /lib/i386-linux-gnu/libdl-2.15.so
b776a000-b776b000 rw-p 00003000 08:01 154496     /lib/i386-linux-gnu/libdl-2.15.so
b776b000-b7787000 r-xp 00000000 08:01 132170     /lib/i386-linux-gnu/libtinfo.so.5.9
b7787000-b7789000 r--p 0001b000 08:01 132170     /lib/i386-linux-gnu/libtinfo.so.5.9
b7789000-b778a000 rw-p 0001d000 08:01 132170     /lib/i386-linux-gnu/libtinfo.so.5.9
b7790000-b7797000 r-xp 00000000 08:01 154511     /lib/i386-linux-gnu/libnss_compat-2.15.so
b7797000-b7798000 r--p 00006000 08:01 154511     /lib/i386-linux-gnu/libnss_compat-2.15.so
b7798000-b7799000 rw-p 00007000 08:01 154511     /lib/i386-linux-gnu/libnss_compat-2.15.so
b7799000-b779a000 r--p 005e0000 08:01 268965     /usr/lib/locale/locale-archive
b779a000-b779c000 rw-p 00000000 00:00 0
b779c000-b779d000 r-xp 00000000 00:00 0          [vdso]
b779d000-b77bd000 r-xp 00000000 08:01 154510     /lib/i386-linux-gnu/ld-2.15.so
b77bd000-b77be000 r--p 0001f000 08:01 154510     /lib/i386-linux-gnu/ld-2.15.so
b77be000-b77bf000 rw-p 00020000 08:01 154510     /lib/i386-linux-gnu/ld-2.15.so
bfdf5000-bfe16000 rw-p 00000000 00:00 0          [stack]
```

物理内存 | 页表管理 | 虚拟内存 | 内核态池 | 故障调试

kmalloc

- Does not clear the memory
- Allocates consecutive virtual/physical memory pages
  - Offset by **`PAGE_OFFSET`**
- Tries its best to fulfill allocation requests
  - Large memory allocations can degrade the system performance significantly

The Flags Argument

- **`kmalloc`** prototype

```
#include <linux/slab.h>
void *kmalloc(size_t size, int flags);
```

- **`GFP_KERNEL`** is the most commonly used flag
  - Eventually calls **`__get_free_pages`**
    - The origin of the GFP prefix
  - Can put the current process to sleep while waiting for a page in low-memory situations
  - Cannot be used in atomic context

The Flags Argument

  - To obtain more memory
    - Flush dirty buffers to disk
    - Swapping out memory from user processes
- **`GFP_ATOMIC`** is called in atomic context
  - Interrupt handlers, tasklets, and kernel timers
  - Does not sleep
  - If the memory is used up, the allocation fails
    - No flushing and swapping

The Flags Argument

- Other flags are available
  - Defined in **`<linux/gfp.h>`**
- **`GFP_USER`** allocates user pages; it may sleep
- **`GFP_HIGHUSER`** allocates high memory user pages
- **`GFP_NOIO`** disallows I/Os
- **`GFP_NOFS`** does not allow making FS calls
  - Used in file system and virtual memory code
  - Disallow **`kmalloc`** to make recursive calls

Priority Flags

- Used in combination with GFP flags (via ORs)
  - **`__GFP_DMA`** requests allocation to happen in the DMA-capable memory zone
  - **`__GFP_HIGHMEM`** indicates that the allocation may be allocated in high memory
  - **`__GFP_COLD`** requests for a page not used for some time (to avoid DMA contention)
  - **`__GFP_NOWARN`** disables **`printk`** warnings when an allocation cannot be satisfied

Priority Flags

- **__GFP_HIGH** marks a high priority request
  - Not for **kmalloc**
- **__GFP_REPEAT**
  - Try harder
- **__GFP_NOFAIL**
  - Failure is not an option (strongly discouraged)
- **__GFP_NORETRY**
  - Give up immediately if the requested memory is not available

Memory Zones

- If **__GFP_DMA** is specified
  - Allocation will only search for the DMA zone
- If nothing is specified
  - Allocation will search both normal and DMA zones
- If **__GFP_HIGHMEM** is specified
  - Allocation will search all three zones

The Size Argument

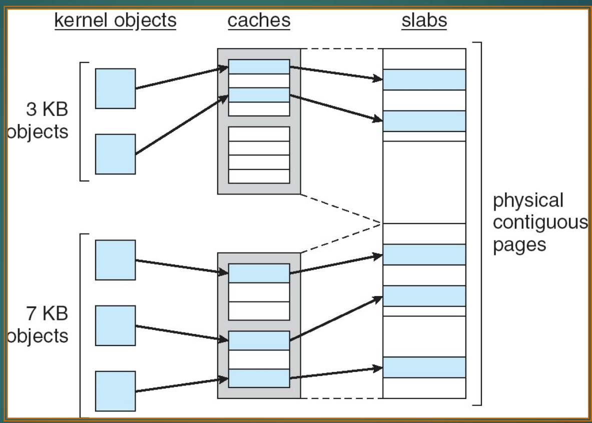
- Kernel manages physical memory in pages
  - Needs special management to allocate small memory chunks
- Linux creates pools of memory objects in predefined fixed sizes (32-byte, 64-byte, 128-byte memory objects)
- Smallest allocation unit for `kmalloc` is 32 or 64 bytes
- Largest portable allocation unit is 128KB

Slab Memory Allocation

Slab: One or more physically contiguous pages

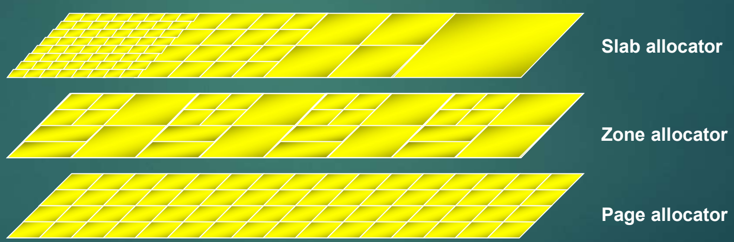
- **Slab cache**
  - Contains of one or more slabs
  - A cache exists for each unique kernel data structure
  - Single cache for each unique kernel data structure
    - A cache initially contains a group instantiated data structure objects
    - The cache is initialized with objects marked as **free**
    - Allocated objects are marked as **used**
  - A new Slab is added to a cache when no more free objects
- **Benefits**
  - No fragmentation
  - Fast allocation

Slab Allocation



三个层次的分配器

- Slab allocator groups allocations by sizes to reduce internal memory fragmentation



```
gedu@gedu-VirtualBox:~$ sudo cat /proc/slabinfo
slabinfo - version: 2.1
# name            <active_objs> <num_objs> <objsize> <objperslab> <pagesperslab> : tunables <limit> <batch
ext4_groupinfo_4k    252    252    144   28    1 : tunables    0    0    0 : slabdata      9      9      0
ip6-frags              0      0    224   18    1 : tunables    0    0    0 : slabdata      0      0      0
tw_sock_TCPv6          0      0    272   15    1 : tunables    0    0    0 : slabdata      0      0      0
request_sock_TCPv6     0      0    328   12    1 : tunables    0    0    0 : slabdata      0      0      0
TCPv6                 30     30   2112   15    8 : tunables    0    0    0 : slabdata      2      2      0
kcopyd_job             0      0   3312    9    8 : tunables    0    0    0 : slabdata      0      0      0
dm_uevent              0      0   2632   12    8 : tunables    0    0    0 : slabdata      0      0      0
bsg_cmd                0      0    312   13    1 : tunables    0    0    0 : slabdata      0      0      0
mqueue_inode_cache    18     18    896   18    4 : tunables    0    0    0 : slabdata      1      1      0
fuse_request          40     40    400   20    2 : tunables    0    0    0 : slabdata      2      2      0
fuse_inode            38     38    832   19    4 : tunables    0    0    0 : slabdata      2      2      0
ecryptfs_key_record_cache      0      0    576   14    2 : tunables    0    0    0 : slabdata      0      0      0
ecryptfs_sb_cache     42     42   1152   14    4 : tunables    0    0    0 : slabdata      3      3      0
ecryptfs_inode_cache      0      0   1024   16    4 : tunables    0    0    0 : slabdata      0      0      0
ecryptfs_auth_tok_list_item      0      0    832   19    4 : tunables    0    0    0 : slabdata      0      0      0
fat_inode_cache        0      0    728   22    4 : tunables    0    0    0 : slabdata      0      0      0
fat_cache              0      0     40  102    1 : tunables    0    0    0 : slabdata      0      0      0
squashfs_inode_cache      0      0    704   23    4 : tunables    0    0    0 : slabdata      0      0      0
jbd2_journal_head   1088   1088    120   34    1 : tunables    0    0    0 : slabdata     32     32      0
jbd2_revoke_table_s    256    256     16  256    1 : tunables    0    0    0 : slabdata      1      1      0
ext4_inode_cache    2874   4980   1080   15    4 : tunables    0    0    0 : slabdata    332    332      0
ext4_allocation_context     64     64    128   32    1 : tunables    0    0    0 : slabdata      2      2      0
```

73

Lookaside Caches (Slab Allocator)

- 与硬件cache和TLB无关，内存池概念
- Useful for USB and SCSI drivers
  - Improved performance
- To create a cache for a tailored size

```
#include <linux/slab.h>

kmem_cache_t *
kmem_cache_create(const char *name, size_t size,
                  size_t offset, unsigned long flags,
                  void (*constructor) (void *, kmem_cache_t *,
                                       unsigned long flags));
```

74

分配和释放

- To allocate an memory object from the memory cache, call

```
void *kmem_cache_alloc(kmem_cache_t *cache, int flags);
```

  - **cache**: the cache created previously
  - **flags**: same flags for **kmalloc**
  - Failure rate is rather high
    - Must check the return value
- To free an memory object, call

```
void kmem_cache_free(kmem_cache_t *cache,
                     const void *obj);
```


Lookaside Caches (Slab Allocator)

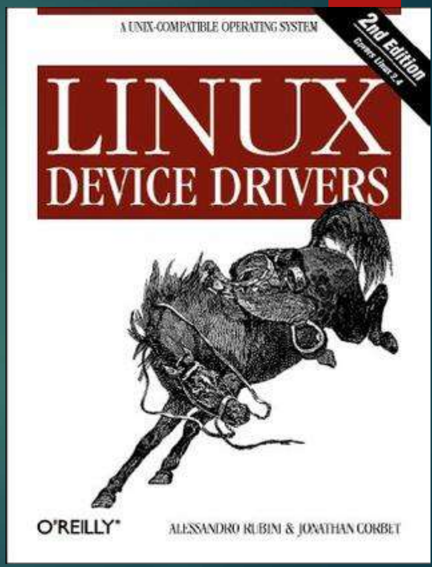
- To free a memory cache, call

```
int kmem_cache_destroy(kmem_cache_t *cache);
```

  - Need to check the return value
  - Failure indicates memory leak
- Slab statistics are kept in **/proc/slabinfo**

SCULL

- Simple character utility for loading localities
- 《LINUX设备驱动程序》一书的驱动程序示例
- 一个操作内存区域的字符设备驱动程序，这片内存区域就相当于一个字符设备



A **scull** Based on the Slab Caches: **scullc**

- Declare slab cache

```
kmem_cache_t *scullc_cache;
```

- Create a slab cache in the init function

```
/* no constructor */
scullc_cache
  = kmem_cache_create("scullc", scullc_quantum, 0,
                      SLAB_HWCACHE_ALIGN, NULL);
if (!scullc_cache) {
  scullc_cleanup();
  return -ENOMEM;
}
```

scull（simple character utility for loading localities）

《linux设备驱动程序》

分配和释放

- To allocate memory quanta

```
if (!dptr->data[s_pos]) {
  dptr->data[s_pos] = kmem_cache_alloc(scullc_cache,
                                       GFP_KERNEL);
  if (!dptr->data[s_pos])
    goto nomem;
  memset(dptr->data[s_pos], 0, scullc_quantum);
}
```

- To release memory

```
for (i = 0; i < qset; i++) {
  if (dptr->data[i]) {
    kmem_cache_free(scullc_cache, dptr->data[i]);
  }
}
```


销毁

- To destroy the memory cache at module unload time

```
/* scullc_cleanup: release the cache of our quanta */
if (scullc_cache) {
  kmem_cache_destroy(scullc_cache);
}
```

I915实例

```
void
i915_gem_load(struct drm_device *dev)
{
	struct drm_i915_private *dev_priv = dev->dev_private;
	int i;

	dev_priv->objects =
		kmem_cache_create("i915_gem_object",
				  sizeof(struct drm_i915_gem_object), 0,
				  SLAB_HWCACHE_ALIGN,
				  NULL);
	dev_priv->vmas =
		kmem_cache_create("i915_gem_vma",
				  sizeof(struct i915_vma), 0,
				  SLAB_HWCACHE_ALIGN,
				  NULL);
	dev_priv->requests =
		kmem_cache_create("i915_gem_request",
				  sizeof(struct drm_i915_gem_request), 0,
				  SLAB_HWCACHE_ALIGN,
				  NULL);
```


Memory Pools

- Similar to memory cache
  - Reserve a pool of memory to guarantee the success of memory allocations
  - Can be wasteful
- To create a memory pool, call

```
#include <linux/mempool.h>

mempool_t *mempool_create(int min_nr,
                          mempool_alloc_t *alloc_fn,
                          mempool_free_t *free_fn,
                          void *pool_data);
```

参数

- **`min_nr`** is the minimum number of allocation objects
- **`alloc_fn`** and **`free_fn`** are the allocation and freeing functions

```
typedef void *(mempool_alloc_t)(int gfp_mask,
                                void *pool_data);
typedef void (mempool_free_t)(void *element,
                              void *pool_data);
```

- **`pool_data`** is passed to the allocation and freeing functions

实例

```
static int init_resync(struct r10conf *conf)
{
	int buffs;
	int i;

	buffs = RESYNC_WINDOW / RESYNC_BLOCK_SIZE;
	BUG_ON(conf->r10buf_pool);
	conf->have_replacement = 0;
	for (i = 0; i < conf->geo.raid_disks; i++)
		if (conf->mirrors[i].replacement)
			conf->have_replacement = 1;
	conf->r10buf_pool = mempool_create(buffs, r10buf_pool_alloc, r10buf_pool_free, conf);
	if (!conf->r10buf_pool)
		return -ENOMEM;
	conf->next_resync = 0;
	return 0;
}
```

- /drivers/md/raid10.c

使用slab分配器

- To allow the slab allocator to handle allocation and deallocation, use predefined functions

```
cache = kmem_cache_create(...);
pool = mempool_create(MY_POOL_MINIMUM, mempool_alloc_slab,
                      mempool_free_slab, cache);
```

- To allocate and deallocate a memory pool object, call

```
void *mempool_alloc(mempool_t *pool, int gfp_mask);
void mempool_free(void *element, mempool_t *pool);
```


实例

```
sd_cdb_cache = kmem_cache_create("sd_ext_cdb", SD_EXT_CDB_SIZE,
                                  0, 0, NULL);
if (!sd_cdb_cache) {
    printk(KERN_ERR "sd: can't init extended cdb cache\n");
    err = -ENOMEM;
    goto err_out_class;
}

sd_cdb_pool = mempool_create_slab_pool(SD_MEMPOOL_SIZE, sd_cdb_cache);
if (!sd_cdb_pool) {
    printk(KERN_ERR "sd: can't init extended cdb pool\n");
    err = -ENOMEM;
    goto err_out_cache;
}
```

改变大小和销毁

- To resize the memory pool, call

```
int mempool_resize(mempool_t *pool, int new_min_nr,
                   int gfp_mask);
```

- To deallocate the memory poll, call

```
void mempool_destroy(mempool_t *pool);
```


get_free_page and Friends

- For allocating big chunks of memory, it is more efficient to use a page-oriented allocator
- To allocate pages, call

```
/* returns a pointer to a zeroed page */
get_zeroed_page(unsigned int flags);

/* does not clear the page */
__get_free_page(unsigned int flags);

/* allocates multiple physically contiguous pages */
__get_free_pages(unsigned int flags, unsigned int order);
```

`get_free_page` and Friends

- **`flags`**
  - Same as **`flags`** for **`kmalloc`**
- **`order`**
  - Allocate 2^{order} pages
    - **`order`** = 0 for 1 page
    - **`order`** = 3 for 8 pages
  - Can use **`get_order(size)`** to find out **`order`**
  - Maximum allowed value is about 10 or 11
  - See **`/proc/buddyinfo`** statistics

`get_free_page` and Friends

- Subject to the same rules as **`kmalloc`**
- To free pages, call

```
void free_page(unsigned long addr);
void free_pages(unsigned long addr, unsigned long order);
```

- Make sure to free the same number of pages
  - Or the memory map becomes corrupted

A `scull` Using Whole Pages: `scullp`

- Memory allocation

```
if (!dptr->data[s_pos]) {
  dptr->data[s_pos] =
    (void *) __get_free_pages(GFP_KERNEL, dptr->order);
  if (!dptr->data[s_pos])
    goto nomem;
  memset(dptr->data[s_pos], 0, PAGE_SIZE << dptr->order);
}
```

- Memory deallocation

```
for (i = 0; i < qset; i++) {
  if (dptr->data[i]) {
    free_pages((unsigned long) (dptr->data[i]), dptr->order);
  }
}
```

91

Vmalloc系列

- 分配虚拟地址连续的内存
- 物理地址未必连续
  - Each page retrieved with a separate **alloc_page** call
    - Less efficient
  - Can sleep (cannot be used in atomic context)
  - Returns 0 on error, or a pointer to the allocated memory
  - Its use is discouraged

使用kmalloc分配物理地址连续的内存，DMA时需要物理地址连续的内存块

92

vmalloc

- **vmalloc**-related prototypes

```
#include <linux/vmalloc.h>

void *vmalloc(unsigned long size);
void vfree(void *addr);

#include <asm/io.h>
void *ioremap(unsigned long offset, unsigned long size);
void iounmap(void *addr);
```

93

A scull vmalloc

- This module allocates 16 pages at a time
- To obtain new memory

```
if (!dptr->data[s_pos]) {
  dptr->data[s_pos]
    = (void *) vmalloc(PAGE_SIZE << dptr->order);
  if (!dptr->data[s_pos]) {
    goto nomem;
  }
  memset(dptr->data[s_pos], 0,
         PAGE_SIZE << dptr->order);
}
```

```
for (i = 0; i < qset; i++) {
  if (dptr->data[i]) {
    vfree(dptr->data[i]);
  }
}
```

94

ioremap

- **`ioremap`** builds page tables
  - Does not allocate memory
  - Takes a physical address (**`offset`**) and return a virtual address
    - Useful to map the address of a PCI buffer to kernel space
  - Should use **`readb`** and other functions to access remapped memory

Per-CPU Variables

- Each CPU gets its own copy of a variable
  - Almost no locking for each CPU to work with its own copy
  - Better performance for frequent updates
- Example: networking subsystem
  - Each CPU counts the number of processed packets by type
  - When user space requests to see the value, just add up each CPU's version and return the total

Per-CPU Variables

- To create a per-CPU variable

```
#include <linux/percpu.h>
DEFINE_PER_CPU(type, name);
```

- `name`: an array

```
DEFINE_PER_CPU(int[3], my_percpu_array);
```

  - Declares a per-CPU array of three integers
- To access a per-CPU variable
  - Need to prevent process migration

```
get_cpu_var(name); /* disables preemption */
put_cpu_var(name); /* enables preemption */
```


Per-CPU Variables

- To access another CPU's copy of the variable, call

```
per_cpu(name, int cpu_id);
```

- To dynamically allocate and release per-CPU variables, call

```
void *alloc_percpu(type);
void *__alloc_percpu(size_t size);

void free_percpu(const void *data);
```

Per-CPU Variables

- To access dynamically allocated per-CPU variables, call

```
per_cpu_ptr(void *per_cpu_var, int cpu_id);
```

- To ensure that a process cannot be moved out of a processor, call **get_cpu** (returns cpu ID) to block preemption

```
int cpu;

cpu = get_cpu()
ptr = per_cpu_ptr(per_cpu_var, cpu);
/* work with ptr */
put_cpu();
```


Per-CPU Variables

- To export per-CPU variables, call

```
EXPORT_PER_CPU_SYMBOL(per_cpu_var);
EXPORT_PER_CPU_SYMBOL_GPL(per_cpu_var);
```

- To access an exported variable, call

```
/* instead of DEFINE_PER_CPU() */
DECLARE_PER_CPU(type, name);
```

- More examples in **<linux/percpu_counter.h>**

启动时分配内存

- 获取大内存区的一种方法
- To allocate, call one of these functions

```
#include <linux/bootmem.h>

void *alloc_bootmem(unsigned long size);

/* need low memory for DMA */
void *alloc_bootmem_low(unsigned long size);

/* allocated in whole pages */
void *alloc_bootmem_pages(unsigned long size);
void *alloc_bootmem_low_pages(unsigned long size);
void free_bootmem(unsigned long addr, unsigned long size);
```


```
ge@gewubox:~/work/heap$ cat /proc/meminfo
MemTotal:        766212 kB //总可用物理内存
MemFree:          90056 kB //总空闲内存=HighFree+LowFree
Buffers:          12236 kB    //缓冲区用量
Cached:          275296 kB  //磁盘缓存diskcache-swapc
SwapCached:         348 kB //交换文件中的缓存
Active:          306888 kB // 最近使用过的
Inactive:        304720 kB
Active(anon):    156212 kB
Inactive(anon):  182932 kB
Active(file):    150676 kB
Inactive(file):  121788 kB
Unevictable:          0 kB
Mlocked:              0 kB
HighTotal:            0 kB //高于~860MB
HighFree:             0 kB
LowTotal:        766212 kB
LowFree:          90056 kB
SwapTotal:       783356 kB
SwapFree:        778140 kB
Dirty:               28 kB // 修改过的
Writeback:            0 kB
AnonPages:       323768 kB //匿名页
Mapped:           93660 kB
Shmem:            15068 kB
Slab:             44596 kB
SReclaimable:     32576 kB
SUnreclaim:       12020 kB
KernelStack:       3008 kB
PageTables:        7428 kB
NFS_Unstable:         0 kB
Bounce:               0 kB
WritebackTmp:         0 kB
CommitLimit:    1166460 kB
Committed_AS:   2400960 kB
VmallocTotal:    249912 kB
VmallocUsed:      20264 kB
VmallocChunk:    221168 kB
HardwareCorrupted:    0 kB
AnonHugePages:        0 kB
HugePages_Total:      0
HugePages_Free:       0
HugePages_Rsvd:       0
HugePages_Surp:       0
Hugepagesize:      2048 kB
DirectMap4k:      34752 kB
DirectMap2M:     751616 kB
```


free

```
ge@gewubox:~/work/heap$ free -l -t
             total       used       free     shared    buffers     cached
Mem:        766212     676192      90020          0      12348     275304
Low:        766212     676192      90020
High:            0          0          0
-/+ buffers/cache:     388540     377672
Swap:       783356       5216     778140
Total:     1549568     681408     868160
```

top

```
top - 17:05:20 up  9:10,  3 users,  load average: 0.13, 0.09, 0.28
Tasks: 162 total,   1 running, 161 sleeping,   0 stopped,   0 zombie
Cpu(s):  3.1%us,  0.7%sy,  0.0%ni, 96.2%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    766212k total,   677876k used,    88336k free,    12392k buffers
Swap:   783356k total,     5216k used,   778140k free,   275400k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1065 root      20   0  142m  94m  20m S  4.0 12.6   1:32.29 Xorg
 2406 ge        20   0 96536  16m  10m S  2.3  2.2   0:18.07 gnome-terminal
 1917 ge        20   0  119m  12m 9984 S  0.3  1.7   0:06.95 metacity
 1936 ge        20   0  254m  53m  29m S  0.3  7.1   0:32.82 unity-2d-shell
 3804 ge        20   0  152m  35m  27m S  0.3  4.7   0:02.81 unity-2d-spread
    1 root      20   0  3520 1784 1228 S  0.0  0.2   0:03.21 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S  0.0  0.0   0:00.29 ksoftirqd/0
    5 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 kworker/0:0H
    7 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    8 root      20   0     0    0    0 S  0.0  0.0   0:00.00 rcu_bh
    9 root      20   0     0    0    0 S  0.0  0.0   0:00.84 rcu_sched
   10 root      RT   0     0    0    0 S  0.0  0.0   0:00.59 watchdog/0
   11 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 khelper
   12 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kdevtmpfs
   13 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 netns
   14 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 writeback
   15 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 kintegrityd
   16 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 bioset
   17 root       0 -20     0    0    0 S  0.0  0.0   0:00.10 kworker/u3:0
   18 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 kblockd
   19 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 ata_sff
   20 root      20   0     0    0    0 S  0.0  0.0   0:00.05 khubd
```

105

Source-code organization

High-level 'mm' code is processor-independent
(It's in the '/usr/src/linux/mm' subdirectory)

Low-level 'mm' code is processor-specific
(It's in the '/usr/src/linux/arch/' subdirectories)

i386/mm **ppc/mm** **mips/mm** **alpha/mm** . . .

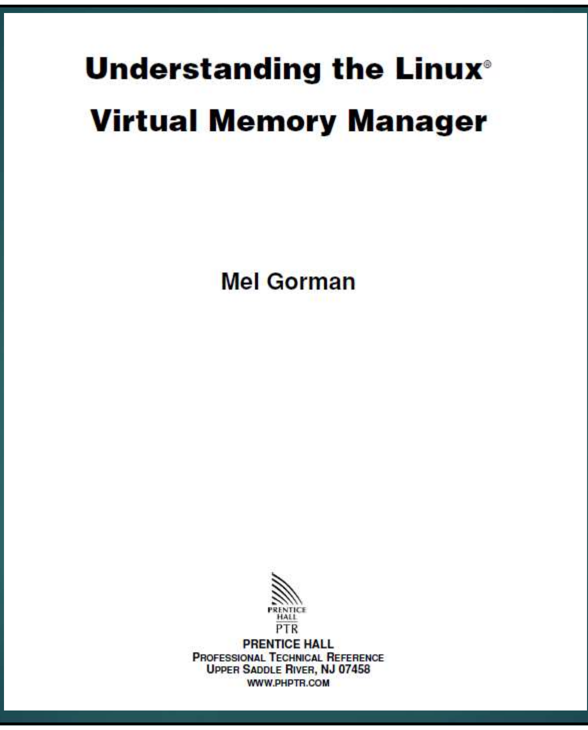
106

文档

d:\bench\linux-4.4.14\Documentation\vm*.*

↑Name	Ext	Size	Date	Attr
[..]		<DIR>	06/25/2016 01:18	—
.gitignore		20	06/25/2016 01:18	-a–
00-index		1,359	06/25/2016 01:18	-a–
active_mm	txt	3,806	06/25/2016 01:18	-a–
balance		5,325	06/25/2016 01:18	-a–
cleancache	txt	14,111	06/25/2016 01:18	-a–
frontswap	txt	15,384	06/25/2016 01:18	-a–
highmem	txt	5,809	06/25/2016 01:18	-a–
hugetlbpage	txt	17,163	06/25/2016 01:18	-a–
hwpoison	txt	5,975	06/25/2016 01:18	-a–
idle_page_tracking	txt	4,916	06/25/2016 01:18	-a–
ksm	txt	5,540	06/25/2016 01:18	-a–
numa		8,913	06/25/2016 01:18	-a–
numa_memory_policy	txt	23,098	06/25/2016 01:18	-a–
overcommit-accounting		2,573	06/25/2016 01:18	-a–
page_migration		6,601	06/25/2016 01:18	-a–
page_owner	txt	3,701	06/25/2016 01:18	-a–
pagemap	txt	6,970	06/25/2016 01:18	-a–
remap_file_pages	txt	1,554	06/25/2016 01:18	-a–
slub	txt	13,179	06/25/2016 01:18	-a–
soft-dirty	txt	1,782	06/25/2016 01:18	-a–
split_page_table_lock		3,619	06/25/2016 01:18	-a–
transhuge	txt	18,422	06/25/2016 01:18	-a–
unevictable-lru	txt	29,388	06/25/2016 01:18	-a–
userfaultfd	txt	7,114	06/25/2016 01:18	-a–
zsmalloc	txt	3,093	06/25/2016 01:18	-a–
zswap	txt	5,282	06/25/2016 01:18	-a–

107



经典之笔，针对2.4内核

108

Q & A

张银奎