第8章 Linux 存储器管理

北京理工大学计算机学院

8.1 进程地址空间的管理

- 32位机，每个进程的地址空间为4GB。
- Linux把地址空间分成两部分。进程的私有空间是前3G，进程的公有空间是后1G的内核虚空间。
- 内核1GB虚空间中的前896MB用来映射物理内存的前896MB，因此前896MB的内存物理地址等于内核虚地址减去0xc0000000。后128MB的虚空间实现对超过896MB的物理内存的映射。

8.1.1 Linux中的分段

- Intel微处理器中的段方案允许程序员把程序划分成逻辑实体，例如程序段和数据段。
- Linux以非常有限的方式使用段，因为当所有的进程使用相同的段寄存器值时，内存管理变得更加简单，即它们共享同样的线性地址空间。

段机制被屏蔽了

段	段基地址	段最大偏移
用户代码段	0x00000000	0xfffff×4KB
用户数据段	0x00000000	0xfffff
内核代码段	0x00000000	0xfffff
内核数据段	0x00000000	0xfffff

8.1.2 虚拟内存区域

- 对于进程的地址空间，是一些为程序的可执行代码、程序的初始化数据、程序的未初始化数据、用户栈、所需共享库的可执行代码和数据、由程序动态请求内存的堆等分配保留的虚空间。
- 需用一组虚拟内存区域描述符来描述进程地址空间的使用情况。虚拟内存区域描述符的结构类型**vm_area_struct**。

vm_area_struct

```
struct vm_area_struct {
    struct mm_struct * vm_mm; 虚拟内存描述符
    unsigned long vm_start;      /*起始地址*/
    unsigned long vm_end;        /*结束地址*/
    struct vm_area_struct  *vm_next;
    struct rb_node  vm_rb;        /*红-黑树*/
    struct file * vm_file;  指向文件映射对象或为NULL
    struct raw_prio_tree_node  prio_tree_node;
内存映射文件时，用此结构构造radix优先级搜索树
    ……}
```

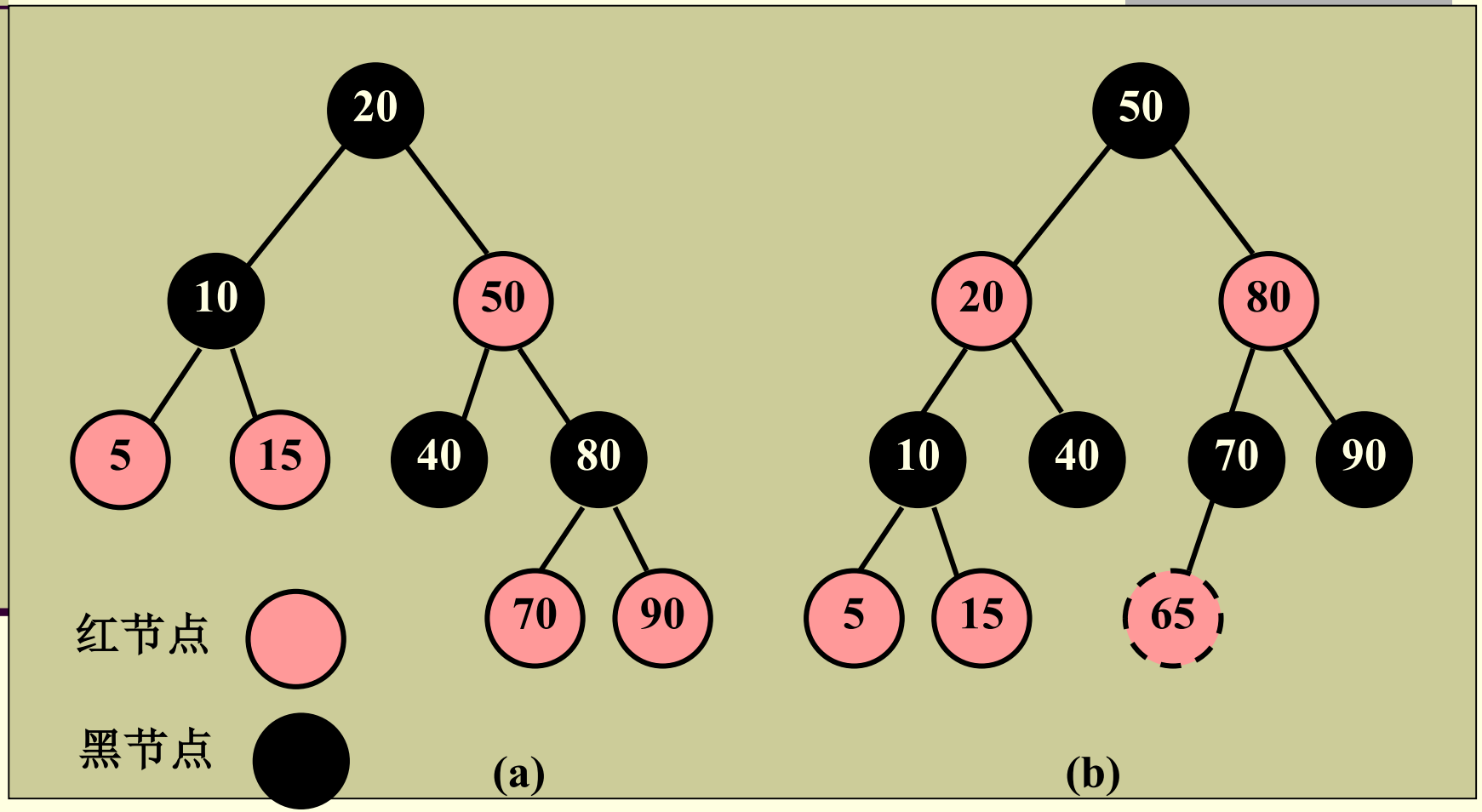
单向链/红黑树

- 进程通过一个单链表把所拥有的各个虚拟内存区域按照地址递增顺序链接在一起。当进程需要的虚拟内存区域数较少，即只有一二十个时，使用链表来管理虚拟内存区域是很方便的。
- 当进程需要的虚拟内存区域较多，如成百上千时，使用简单的链表进行管理就变得非常低效。为此，Linux使用红黑树来管理虚拟内存区域。

红黑树

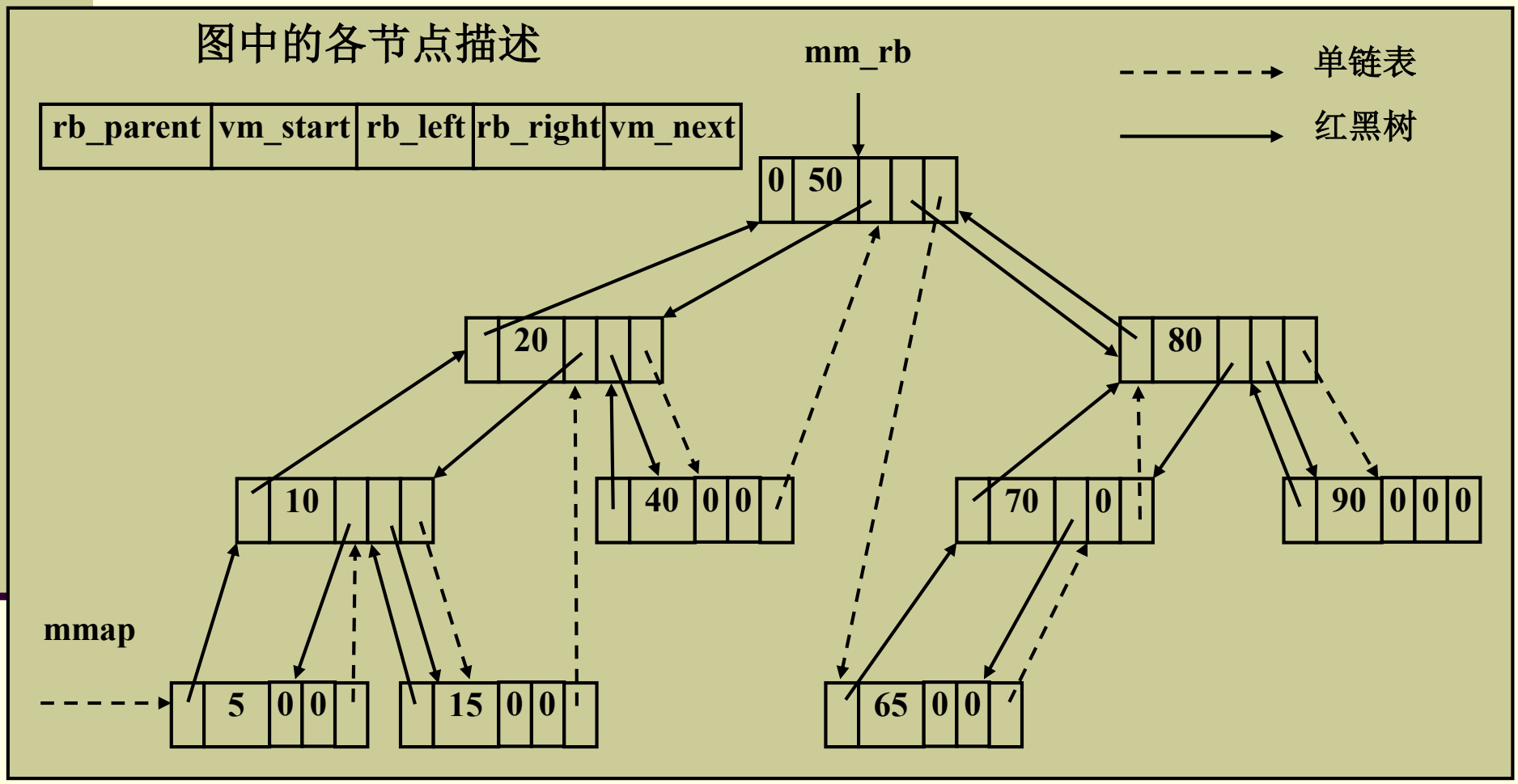
- 红黑树是一棵排好序的平衡二叉树。
- 必须满足四条规则：①树中的每个节点或为红或为黑；②树的根节点必须为黑；③红节点的孩子必须为黑；④从一个节点到后代诸叶子节点的每条路经，都包含相同数量的黑节点，在统计黑节点个数时，空指针也算作黑节点。
- 这四条规则保证：具有n个节点的红黑树，其高度至多为2*log(n+1)。

红黑树



- 当插入或删除一个虚拟内存区域时，通过红黑树搜索其相邻节点，并用搜索结果快速更新单链表。
- 链接虚拟内存区域的单链表和红黑树的链接图如下所示：

红黑树


图中的各节点描述
rb_parent
vm_start
rb_left
rb_right
vm_next
mm_rb
单链表
红黑树
0
50
20
80
10
40
0
0
70
0
90
0
0
0
mmap
5
0
0
15
0
0
65
0
0

分配或释放虚拟内存区域

- 分配：

do_mmap(file, addr, len, prot, flag, offset);

当实现文件映射时，需要文件描述符指针**file**和文件偏移量**offset**，否则他们为空。

- 释放：

do_munmap(mm, start, len);

8.1.3 虚拟内存描述符

- 管理进程地址空间中的所有保留的虚拟内存区域。
- 虚拟内存描述符mm_struct。
- mm_struct的mmlist字段，把所有进程的内存描述符链接成一个双向链表。链表中的第一个元素是init_mm，它是0号进程所使用的内存描述符。

mm_struct

```
struct mm_struct {
    struct vm_area_struct *mmap;
    struct rb_root  mm_rb;  /*指向红-黑树的根*/
    pgd_t *pgd;                    /*指向页目录表*/
    atomic_t mm_users;         /*次使用计数器*/
    atomic_t mm_count;         /*主使用计数器*/
    struct list_head mmlist;   双向链表
    unsigned long start_code, end_code;  /*可
执行代码所占用的地址区间*/
    ……};
```

mm、active_mm

- task_struct：mm、active_mm。mm指针指向进程所拥有的内存描述符，active_mm指针指向进程在运行态时所使用的内存描述符。对于普通进程，mm和active_mm指针值相同。
- 由于内核线程不拥有内存描述符，所以它的mm字段总为NULL。当内核线程运行时，它的active_mm字段就被初始化为刚运行过进程的active_mm字段值。

mm_struct

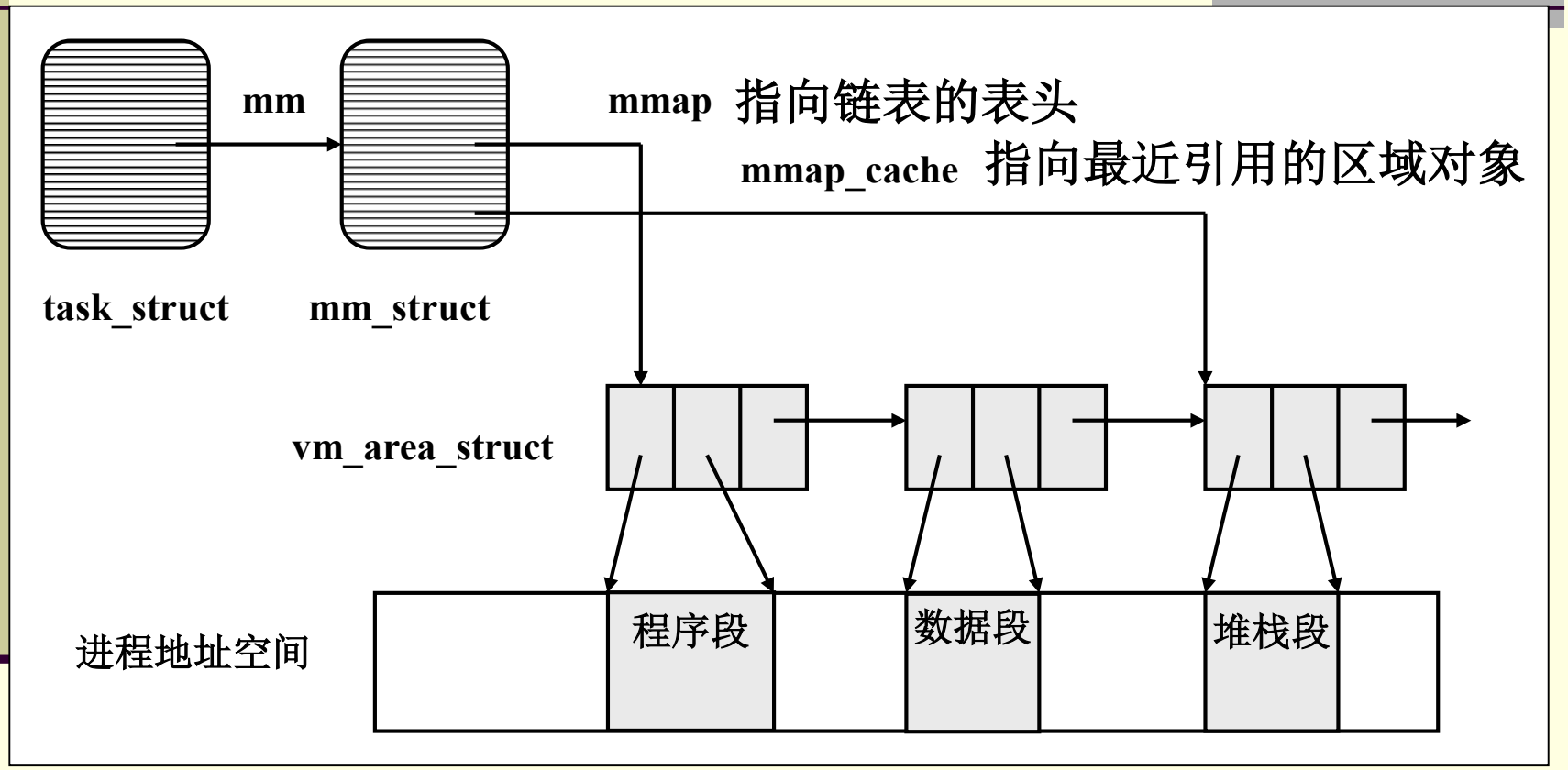


图8.3 进程地址空间所涉及的诸描述符之间的关系

主、次使用计数器

- mm_users次使用计数器记录共享mm_struct的轻量级进程数。本进程中的所有轻量级进程在mm_count中只作为一个单位。
- 当一个内存描述符由两个轻量级进程共享时，它的mm_users为2，mm_count为1。若把内存描述符暂时借给一个内核线程使用，则mm_count值增1。这时，即使轻量级进程都结束，mm_users值为0，只要mm_count递减后的值不为0，这个内存描述符就不被释放。

延迟方式

- 当处于核心态的一个进程为高于3G的高端虚拟地址修改了页表项时，系统中所有进程页表的相应页表项都应该被修改。Linux采用一种延迟方式来处理。
- 延迟方式：每当一个高端地址必须被重新映射时，内核就更新init_mm中的pgd字段所指向的页表。当某一进程运行时，内核对pgd的修改能传递到由进程实际使用的页目录中。

8.1.4 创建进程的地址空间

- 当创建一个新进程时，内核调用**copy_mm()**函数建立新进程的页表和内存描述符，以此来创建进程的地址空间。
- 通常，每个进程都有自己独立的地址空间。但是，**clone()**函数创建轻量级进程时，如果设置了**CLONE_VM**标志，则采用写时复制方法与父进程共享同一地址空间。

8.1.5 堆的管理

- 每个UNIX进程都拥有一个特殊的虚拟内存区域——堆(heap)。堆用于满足进程的动态内存请求。在虚拟内存描述符中，start_brk字段指定了堆的起始地址，brk字段指定了堆的结束地址。
- malloc(size)：请求size个字节的动态内存。
- free(addr)：释放内存。

8.2 物理内存管理

- 当进程请求内存时，系统只是在进程地址空间中分配一个新的虚拟内存区域。缺页中断时，才通过缺页中断处理程序分配物理页框。
- 调用__get_free_pages()或alloc_pages()从分区页框分配器中获得页框。
- 调用kmem_cache_alloc()或kmalloc()使用slab分配器为对象分配几十或几百个字节的内存块。
- 调用vmalloc()或vmalloc_32()从高端内存获得一块非连续的内存区。

8.2.1 物理内存布局

- 页框0由BIOS使用，存放加电自检期间检查到的系统硬件配置。
- 从0x000a0000到0x000fffff的物理地址通常留给BIOS例程，并且映射ISA图形卡上的内部内存。这个区域就是众所周知的所有IBM兼容PC上从640KB到1MB之间的洞，对应的页框不能由操作系统使用。
- 为了避免把内核装入一组不连续的页框里，Linux跳过RAM的第一个MB的空间。

RAM

- 处理机所支持的RAM容量是受连接到地址总线上的地址管脚数限制的。
- 理论上，在32位处理机系统上，可以安装高达4GB的RAM。实际上，由于用户进程对虚地址空间的需求，Linux内核不能直接对1GB以上的RAM进行寻址。

8.2.2 物理页框管理

- 页框大小为**4KB**。
- 页框描述符为**struct page**。所有页框描述符存放在**mem_map**数组中。
- 内核必须记录每个物理页框的当前状态：哪些页框属于进程的页，哪些页框是内核代码或内核数据页，哪些页框是空闲的。

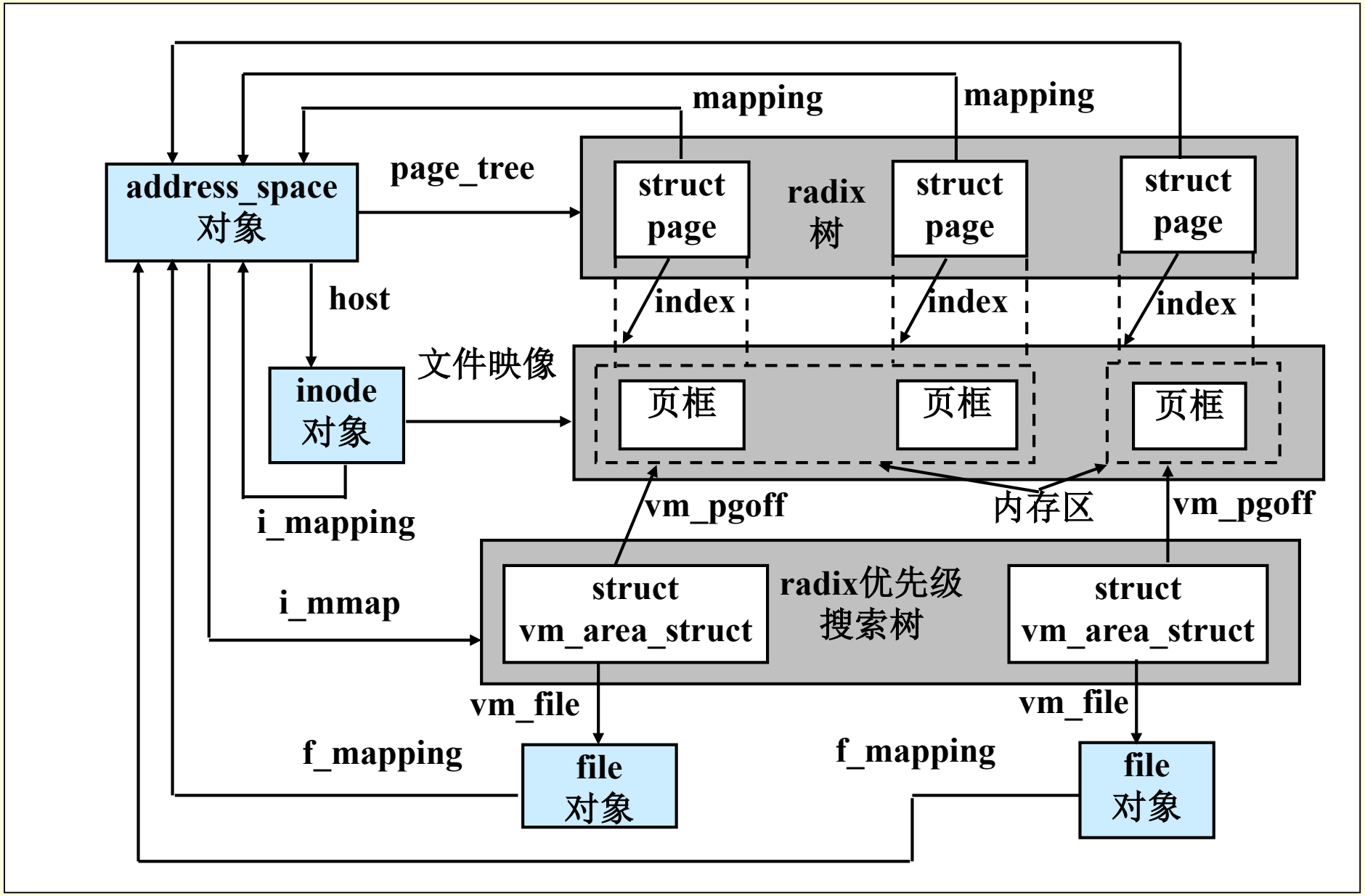
页框描述符

```
struct page {
    unsigned long flags;  /*页框状态标志*/
    atomic_t  _count;       /*页框的引用计数*/
    atomic_t  _mapcount; /*对应的页表项数目*/
    unsigned long private; /*由伙伴系统使用*/
    struct address_space *mapping;页高速缓存
    pgoff_t  index;  在页高速缓存中以页为单位偏移
    struct list_head lru; 链入活动页框链表或非活动..
    void *virtual;     /*页框所映射的内核虚地址*/
};
```

mapping

- mapping字段的结构类型为address_space，它是页高速缓存的核心数据结构，嵌入在页所有者的主存索引节点对象（inode结构）中。属于同一个文件的缓存页被链入同一个address_space对象中，构成radix树。
- 文件的读写操作大都依赖于页高速缓存。文件的直接I/O模式将数据在用户态地址空间和磁盘间直接传送而不通过页高速缓存。

图11.3 实现文件内存映射的各数据结构之间的关系



lru

- 页框描述符中的lru字段是一个页框双向链表。根据页框的最近被访问情况，按照LRU策略把页框链入活动页框链表（active_list）或非活动页框链表（inactive_list）。回收页框时，直接扫描非活动页框链表。

virtual

- virtual字段主要用于超过896MB的高端物理页框的映射。
- 经过映射的页框才能被内核访问。
- 没有超出内核虚空间的页框有线性映射，而超出内核虚空间的高端页框只能进行动态映射，因此需要增加存放内核虚地址的字段virtual。

8.2.3 非一致内存访问机制

- 通常假设内存是一种均匀的一致的共享资源，即CPU对内存中不同存储单元的访问所需时间相同。
- Linux 2.6支持非一致内存访问（Non-Uniform Memory Access, NUMA）。
- 在X86体系结构中，IBM兼容PC使用一致内存访问（UMA）模型。为了增加内核代码的可移植性，Linux还是使用了节点，不过只有一个单独的节点，包含了系统中所有物理内存。

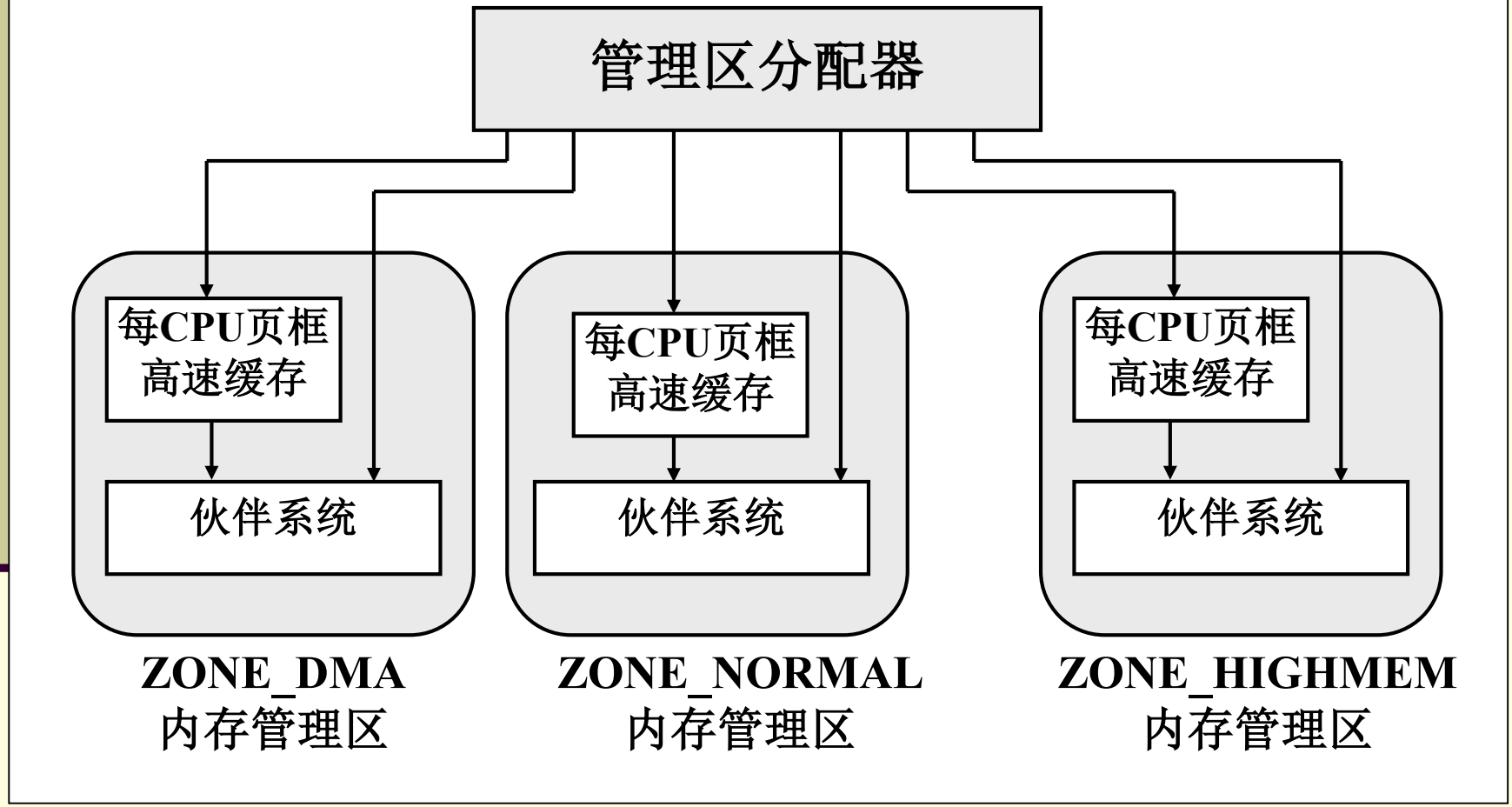
3个管理区

- Linux把内存节点划分为3个管理区zone。
  - ZONE_DMA：包含低于16MB的常规内存页框。用于对老式的基于ISA设备的DMA支持。
  - ZONE_NORMAL：包含高于16MB且低于896MB的常规内存页框。
  - ZONE_HIGHMEM：包含从896MB开始的高端物理页框。内核不能直接访问这部分页框。在64位体系结构上，该区总是空的。

管理区描述符

```
struct zone {
    unsigned long  free_pages;  空闲页框数
    struct per_cpu_pageset  pageset[NR_CPUS];
    /*每CPU页高速缓存*/
    struct free_area  free_area[11];
    /*伙伴系统中的11个空闲页框链表*/
    struct list_head active_list; /*活动页框链表*/
    struct list_head inactive_list; /*非活动…*/
    struct page *zone_mem_map;/*指向首页框的*/
    …….};
```

8.2.4 分区页框分配器


管理区分配器
每CPU页框
高速缓存
伙伴系统
每CPU页框
高速缓存
伙伴系统
每CPU页框
高速缓存
伙伴系统
ZONE_DMA
内存管理区
ZONE_NORMAL
内存管理区
ZONE_HIGHMEM
内存管理区

分区页框分配器

- 负责处理对连续物理页框的分配请求。
- 管理区分配器负责搜索一个能满足动态内存请求的内存管理区。当空闲页框不足时，管理区分配器应当触发页框回收算法。
- 在每个管理区内的页框，除了一小部分页框被保留为每CPU页框高速缓存外（以满足本地CPU发出的对单个页框的请求），其它的由伙伴系统来管理。

伙伴系统

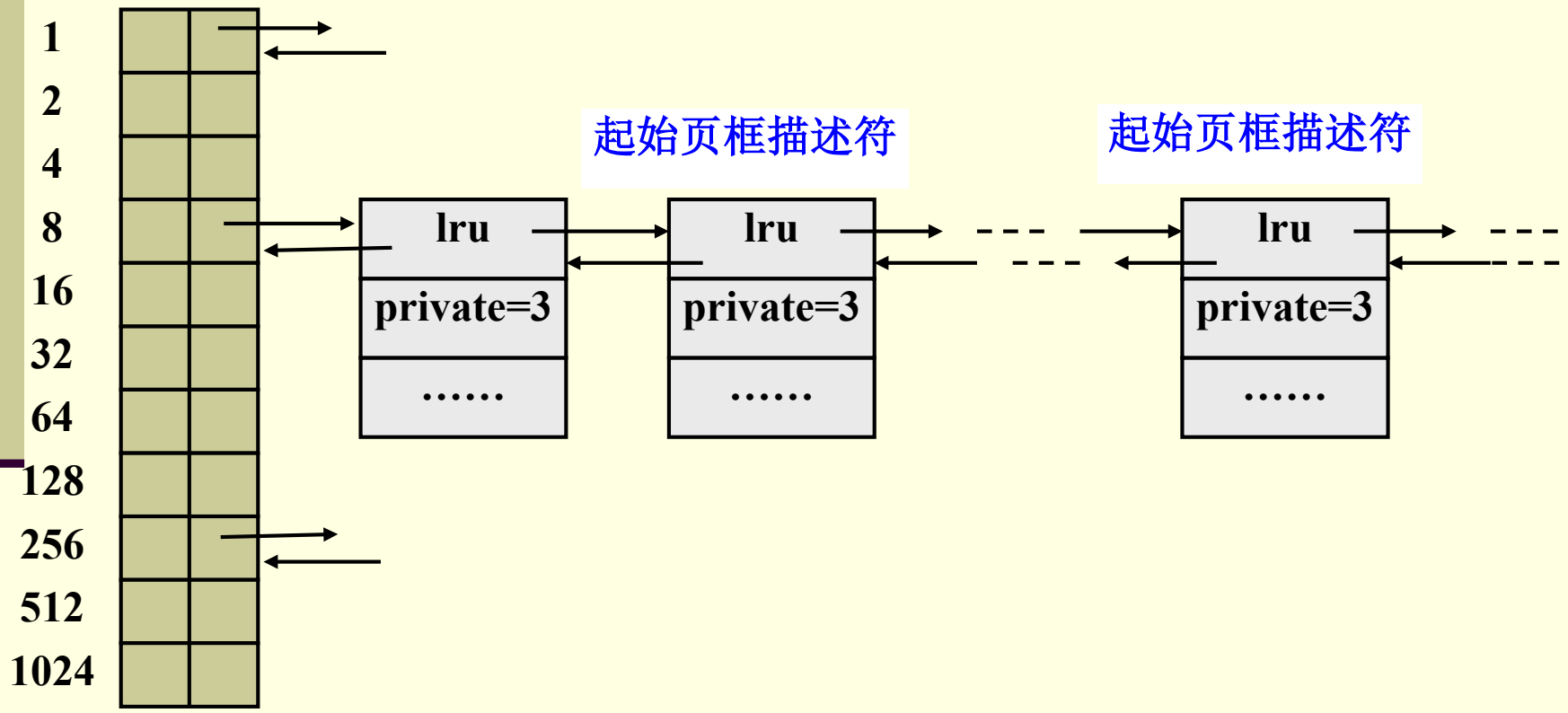
- 采用伙伴系统(buddy system)管理连续的空闲内存页框，以解决外碎片问题。
- 外碎片就是夹杂在已分配页框中间的那些连续的小的空闲页框。
- 伙伴算法把空闲页框组织成11个链表，分别链有大小为1，2，4，8，16，32，64，128，256，512和1024个连续页框的块。

伙伴系统

- 假设要请求一个具有8个连续页框的块，该算法先在8个连续页框块的链表中检查是否有一个空闲块。如果没有，就在16个连续页框块的链表中找。如果找到，就把这16个连续页框分成两等份，一份用来满足请求，另一份插入到具有8个连续页框块的链表中。如果在16个连续页框块的链表中没有找到空闲块，那么就在更大的块链表中查找。直到找到为止。

伙伴系统

管理区描述符的
free_area数组



伙伴系统

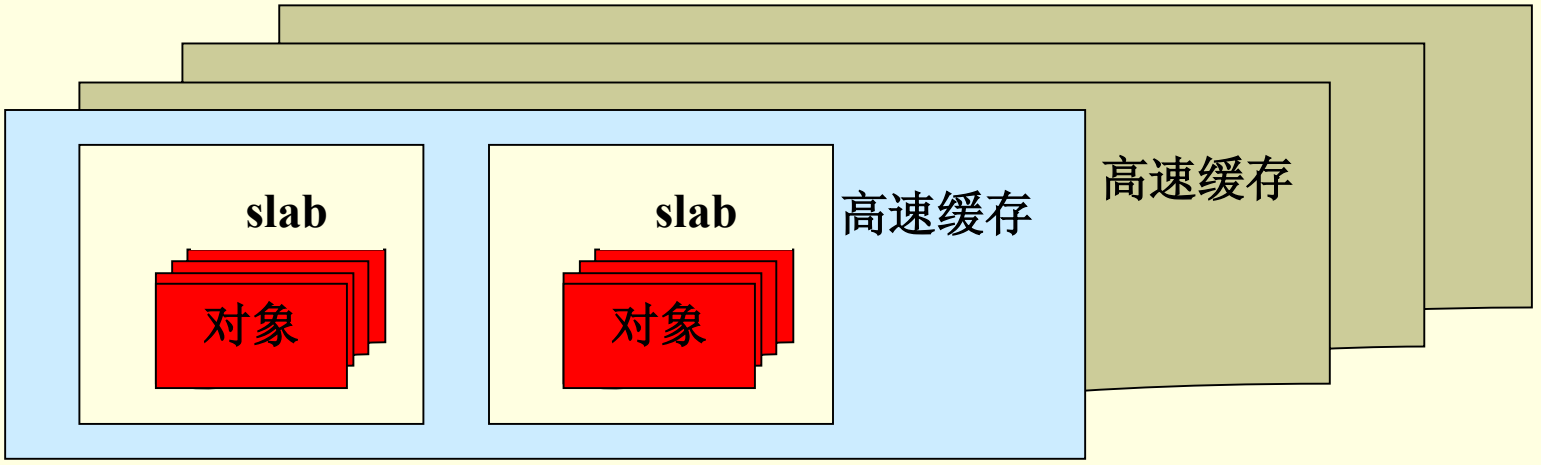
- 管理空闲页框的链表，包含着每个空闲页框块的起始页框描述符；指向链表中相邻元素的指针存放在页框描述符的lru字段中。
- 当页框不空闲时，页框描述符的lru字段用于构建页的最近最少使用双向链表。
- 用页框描述符的private字段存放页框块的幂指数order。

8.3 slab管理

- 伙伴系统算法以页框为单位，适合于对大块内存的分配请求。
- slab分配器用于为只有几十或几百个字节的小内存区分配内存。如，file对象。
- slab分配器把小内存区看作对象，slab分配器对不再引用的对象只是释放但内容保留，以后再请求新对象时，就可直接使用而不需要重新初始化。

slab分配器

- 从分区页框分配器获得几组连续空闲页框
- slab分配器为不同类型的对象生成不同的高速缓存，每个高速缓存存储相同类型的对象。高速缓存由一连串的slab构成，每个slab包含了若干个同类型的对象。



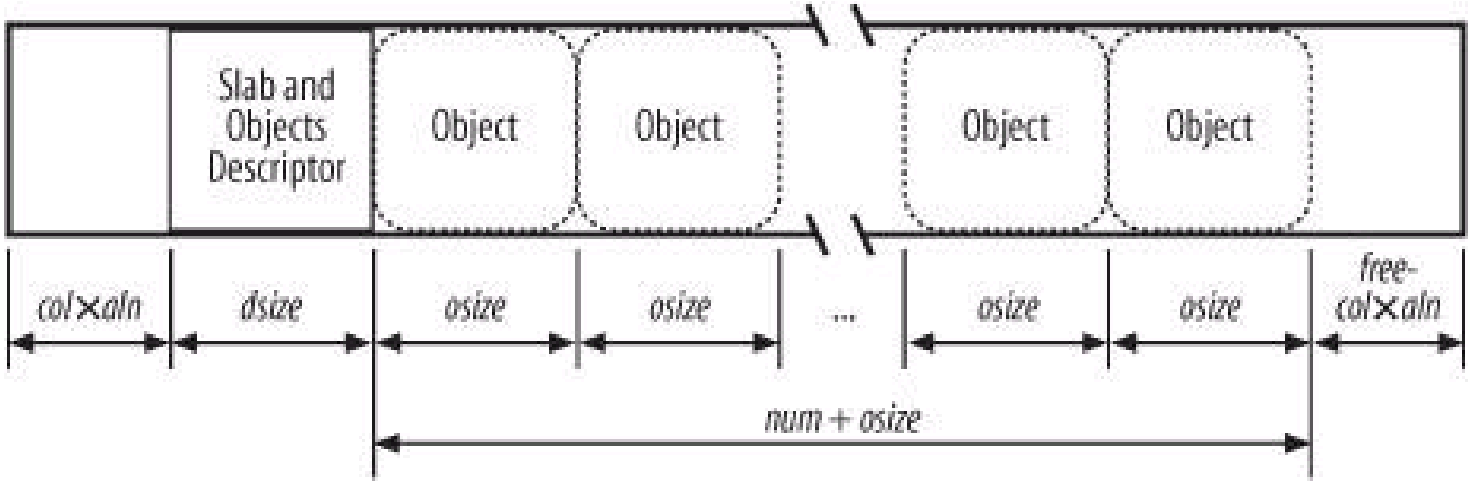
slab分配器

- 高速缓存有自己的描述符
- 高速缓存中的每个slab也有自己的描述符
- 对象描述符存放在一个数组中，位于相应的slab描述符之后。
- 将同一高速缓存里的多个slab进行分类。含有部分空闲对象的slab、不含有空闲对象的slab、只含有空闲对象的slab。

8.3.2 slab着色

- 硬件高速缓存处于CPU和RAM之间，一个硬件高速缓存由多个缓存行组成，一个缓存行有几十个字节。在高速缓存的不同slab内，具有相同偏移量的对象最终很可能映射到同一硬件高速缓存行。
- 着色就是利用空闲未用的字节数对slab进行着色，把slab中的一些空闲区域从末尾移到开头
- 着色后，具有不同颜色的slab把其第一个对象存放在不同偏移量处。可用的颜色数=（空闲未用的字节数free）/（某一给定正整数aln）。

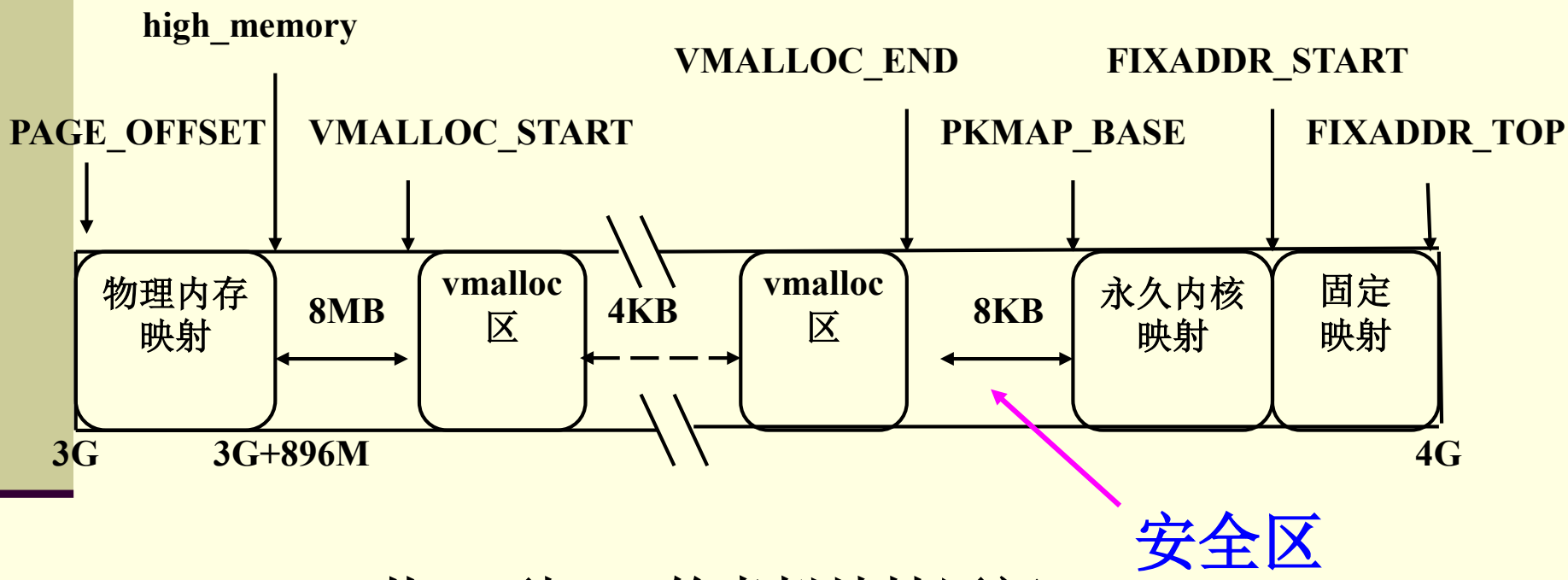
slab着色


Slab and Objects Descriptor
Object
Object
Object
Object
col×aln
dsize
osize
osize
...
osize
osize
free-col×aln
num + osize

8.4 高端内存区管理

- 没有虚地址的页框是不能被内核访问的。
- 用内核虚空间的最后128MB来映射超过896MB的高端内存页框。
- 这种映射是暂时的，否则只有128MB的高端内存页框能被访问到。
- 通过重复使用这部分虚地址，使得整个高端内存页框能够在不同的时间段内被访问。

内核虚空间



从3G到4GB的虚拟地址区间

永久内核映射

- 建立高端页框到内核虚空间的长期映射。
- 使用内核页表中的一个专门页表来实现这种映射。
- 当专门页表中没有空闲的页表项来记录映射的页框时，建立永久内核映射可能会阻塞当前进程。因此，永久内核映射不能用于中断处理程序和可延迟函数。

固定映射

- 又称临时内核映射。比永久内核映射的实现要简单，也不会阻塞当前进程，因此可以用在中断处理程序和可延迟函数的内部。
- 映射时，每个CPU分有13个窗口，可映射13个页框。从固定映射的末端开始向低地址方向依次分配窗口。
- 可以使用固定映射的虚地址来代替指针变量。就指针变量而言，固定映射的虚地址更有效。引用一个直接常量地址比引用一个指针变量要少一次内存访问。
- 某些硬件设备会使用固定映射来代替缓存。

非连续内存区

- 非连续内存区中包含多个子区域，用“安全区”隔开，每个子区域有自己的描述符。
- 不要求页框连续，所以子区域描述符需要一个页框描述符指针数组来记录所获得的页框情况。然后，把被映射页框的物理地址写进主内核页表中。
- 当核心态进程访问非连续内存区时，由于对应的进程页表项为空，所以发生缺页。若缺页处理程序发现所缺的页在主内核页表中，就把相应的页表项复制到当前进程页表项中。

8.5 地址转换

- 32位处理机普遍采用二级页表模式，为每个进程分配一个页目录表，页表一直推迟到访问页时才建立，以节约内存。
- 虚地址分成3个域：页目录索引（前10位）、页表索引（中10位）和页内偏移（后12位）。
- Linux系统的页目录项和页表项的数据结构相同。

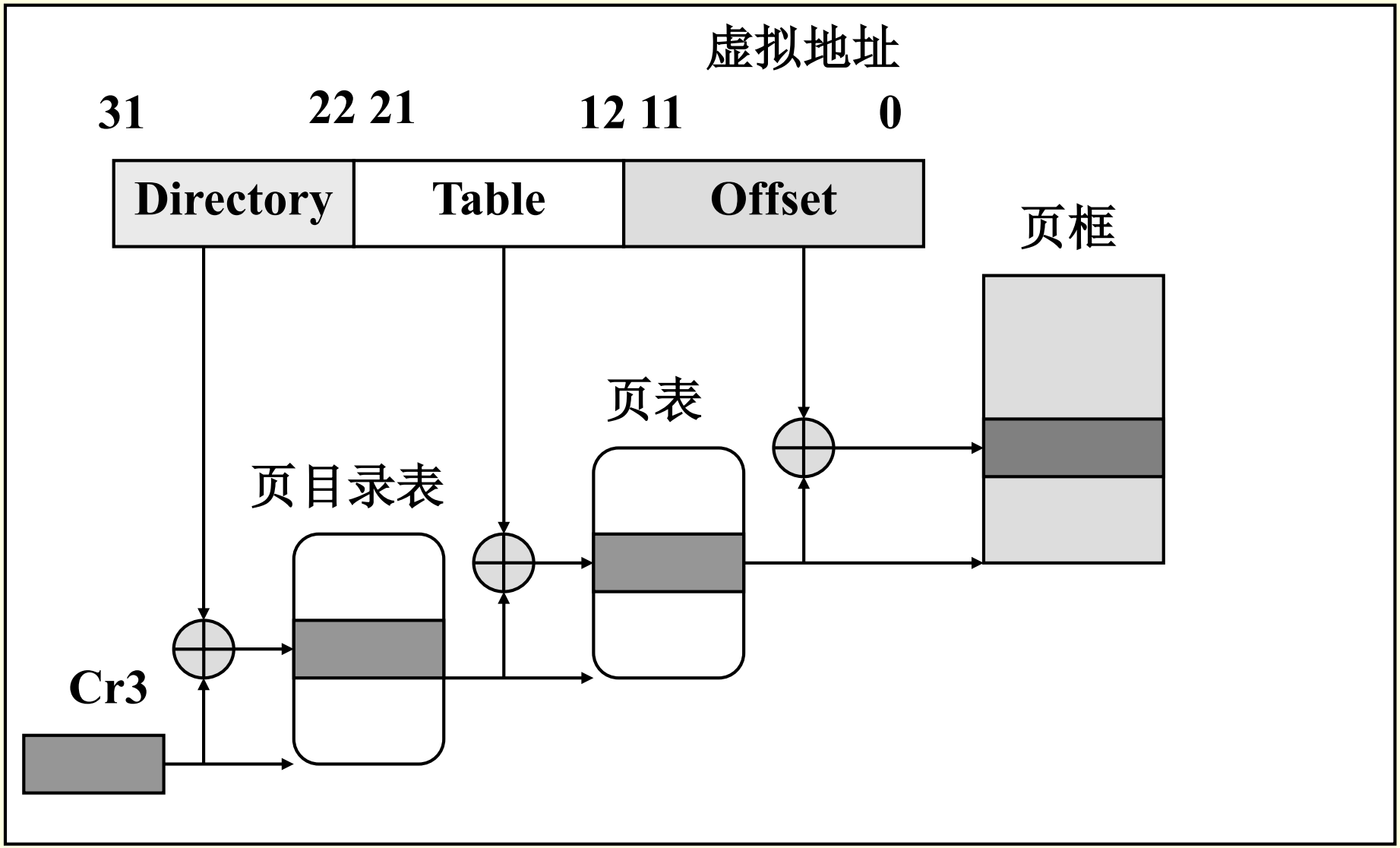

虚拟地址
31
22 21
12 11
0
Directory
Table
Offset
页框
页表
页目录表
Cr3

表8.4 页表项所包含的字段

页表项中的字段	说明
Present标志	为1，表示页（或页表）在内存；为0，则不在内存。
页框物理地址(20位)	页框大小为4096，占去12位。20+12=32
Accessed 标志	页框访问标志，为1表示访问过
Dirty标志	每当对一个页框进行写操作时就设置这个标志
Read/Write标志	存取权限。Read/Write或Read
User/Supervisor标志	访问页或页表时所需的特权级
PCD和PWT标志	设置PCD标志表示禁用硬件高速缓存，设置PWT表示写直通
Page Size 标志	页目录项的页大小标志。置1，页目录项使用2MB/4MB的页框
Global标志	页表项使用。在Cr4寄存器的PGE标志置位时才起作用

页表项的标志

- 写直通策略：又叫通写策略，控制器总是既写硬件高速缓存行也写RAM。
- 回写策略：只更新硬件高速缓存行，不改变RAM的内容；当回写结束以后，最终更新RAM。
- Linux清除了所有页目录项和页表项中的PCD和PWT标志，因此，对所有的页框都启用高速缓存，对于写操作总是采用回写策略。

8.6 请求调页与缺页异常处理

- 请求调页机制是把页框的分配一直推迟到进程要访问的页不在RAM中时引起一个缺页异常，才将所需的页调入内存。
- 请求调页增加了系统中的空闲页框的平均数。

所缺页的存放处

- 该页从未被进程访问过，且没有相应的内存映射。
- 该页属于非线性内存映射文件。非线性内存映射的是文件数据的随机页。给定文件的所有非线性映射虚拟内存区域描述符都存放在一个双向链表中。
- 该页已被进程访问过，但其内容被临时保存到磁盘交换区上。
- 该页在非活动页框链表中。
- 该页正在由其它进程进行I/O传输过程中。

所缺页的处理

- 该页从未被进程访问过，从指定文件中调页；
- 该页属于非线性内存映射文件。从页表项的高位中取出所请求文件页的索引，然后，从磁盘读入页并更新页表项。
- 该页保存到磁盘交换区上。从交换区调入；
- 该页在非活动页链表中，摘下直接使用。
- 该页正在由其它进程进行I/O传输过程中，睡眠等待传输完成。

8.7 盘交换区空间管理

- 盘交换区用来存放从内存暂时换出的数据页
- 每个盘交换区都由一组4KB的页槽组成。
- 盘交换区的第一个页槽用来存放该交换区的有关信息，有相应的描述符。
- 存放在磁盘分区中的交换区只有一个子区，存放在普通文件中的交换区可能有多个子区，原因是磁盘上的文件不要求连续存放。
- 内核尽力把换出的页存放在相邻的页槽中，减少访问交换区时磁盘的寻道时间。