

4.9 Linux存储器管理

- 4.9.1 虚拟地址空间布局
- 4.9.2 进程地址空间管理
- 4.9.3 物理内存管理与分配
- 4.9.4 地址转换
- 4.9.5 请求调页与缺页异常处理
- 4.9.6 盘交换区空间管理

158

4.9 Linux存储器管理

- 每个进程都有独立的虚拟地址空间，进程访问的虚拟地址并不是真正的物理地址
- 虚拟地址可通过每个进程上的页表(在每个进程的内核虚拟地址空间)与物理地址进行映射，获得真正物理地址
- 如果虚拟地址对应物理地址不在物理内存中，则产生缺页中断，真正分配物理地址，同时更新进程的页表；如果此时物理内存已耗尽，则根据内存替换算法淘汰部分页面至物理磁盘中

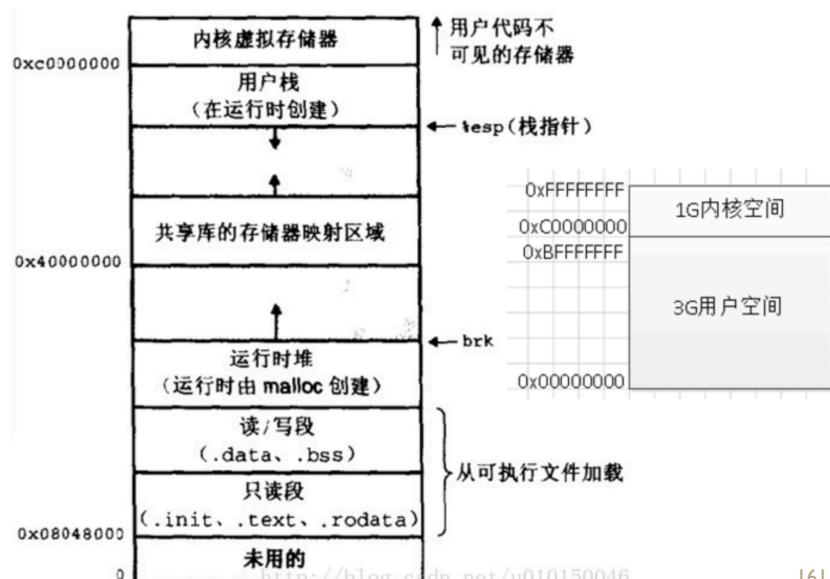
159

4.9.1 虚拟地址空间布局(1)

- Linux进程使用虚拟地址空间，由低地址到高地址分别为
 - 只读段：包括代码段、.rodata段(C常量字符串和#define定义的常量))
 - 数据段：保存全局变量、静态变量的空间
 - 堆：动态内存(malloc/new分配)
 - 文件映射区域：如动态库、共享内存等映射物理空间的内存，一般是mmap函数所分配的虚拟地址空间
 - 栈：用于维护函数调用的上下文空间，一般为8M
 - 内核虚拟空间：用户代码不可见的内存区域，由内核管理(页表就存放在内核虚拟空间)

160

4.9.1 虚拟地址空间布局(2)



161

4.9.1 虚拟地址空间布局(3)

- 32位系统，每个进程的地址空间为4GB
 - 每个进程的私有地址空间（用户空间）是前3G，即0x08048000~0xffffffff
 - 进程的公有地址空间（内核空间）是后1G，即0xc0000000~0xffffffff
- 64位系统，每个进程的地址空间为 2^{48}
 - 不需要 2^{64} 这么大的寻址空间
 - 一般使用48位来表示虚拟地址空间，40位表示物理地址

162

4.9.1 虚拟地址空间布局(4)

- 逻辑地址空间
- 线性地址空间
 - 从0x00000000到0xFFFFFFFF整个4GB虚拟存储空间
- 用户空间
 - 从0x00000000到0xBFFFFFFF共3GB的线性地址空间
 - 每个进程都有一个独立的3GB用户空间，用户空间由每个进程独有
 - 内核线程没有用户空间

163

4.9.1 虚拟地址空间布局(5)

➤ 内核空间

- 占用从0xC0000000到0xFFFFFFFF的1GB线性地址空间
- 内核线性地址空间由所有进程共享，但只有运行在内核态的进程才能访问，用户进程可以通过系统调用切换到内核态访问内核空间
- 进程运行在内核态时所产生的地址都属于内核空间

164

4.9.2 进程地址空间管理

➤ 4.9.2.1 进程地址空间结构

➤ 4.9.2.2 虚拟内存区域

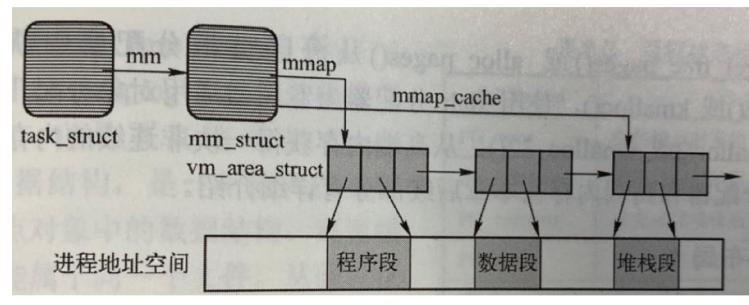
➤ 4.9.2.3 创建进程的地址空间

➤ 4.9.2.4 堆的管理

165

4.9.2.1 进程地址空间结构(1)

- 每个进程结构有一个mm_struct结构（虚拟内存描述符）管理该进程的地址空间
- 每个分配的内存块由一个vm_area_struct结构表示，所有的vm_area_struct结构构成一个单链表/红黑树，开始于mm_struct 中的mmap



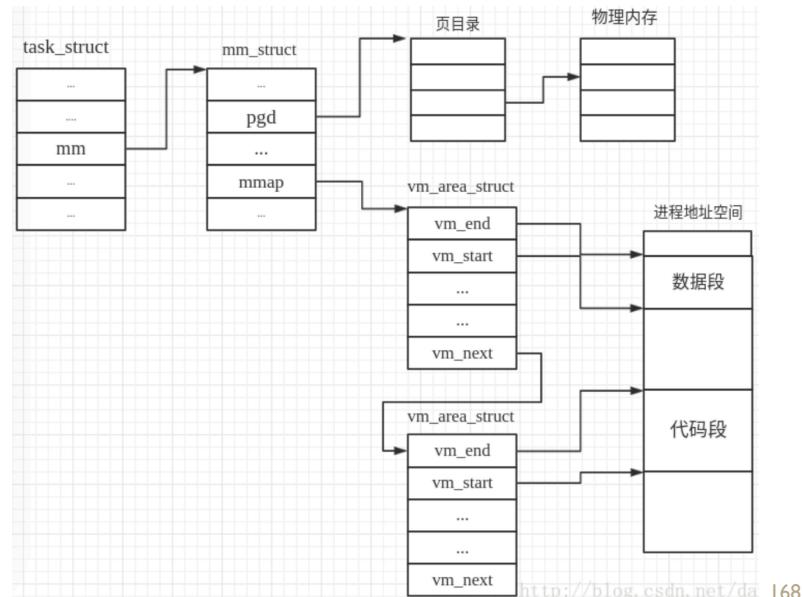
166

4.9.2.1 进程地址空间结构(2)

```
struct mm_struct {           //虚拟内存描述符
    struct vm_area_struct *mmap; //指向虚拟内存区域
                           //链表表头
    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
    int map_count;             //进程拥有的VMA个数
    struct list_head mmlist;   //内存描述符双向链表
    unsigned long start_code, end_code; //可执行代码
                           //占用的地址区间
    unsigned long start_brk, end_brk; //堆起始和结束地址
    .....
}
```

167

4.9.2.1 进程地址空间结构(3)



4.9.2.2 虚拟内存区域(1)

➤ 虚拟内存区域描述符vm_area_struct

- 进程地址空间是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
- 进程的虚拟内存空间会被分成不同的若干区域，每个区域由一个虚拟内存区域描述符描述
- 一个vm_area_struct(vma)就是一块连续的线性地址空间的抽象，它拥有自身的权限(可读，可写，可执行等等)
- vma是具有相同访问属性的虚存空间，该虚存空间的大小为物理内存页面的整数倍

169

4.9.2.2 虚拟内存区域(2)

```
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; //单链表  
    struct rb_node vm_rb; //红-黑树  
    struct file *vm_file; //映射文件时指向文件对象  
    .....  
}
```

170

4.9.2.2 虚拟内存区域(3)

➤ 虚拟内存区域的组织

- 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
- ◆ 默认情况下，一个进程最多可以拥有 65536 个不同的虚拟内存区域

- 红黑树

- ◆ 虚拟内存区域较多时，为提高效率使用红黑树管理虚拟内存区域 (linux 2.6)

171

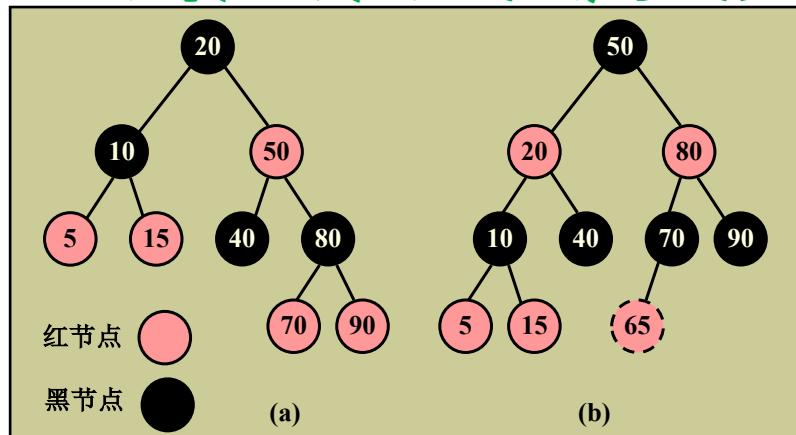
4.9.2.2 虚拟内存区域(4)

- 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径，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 $2 \log(n+1)$

| 72

4.9.2.2 虚拟内存区域(5)

- ◆ 新插入节点为叶子节点，着色为红
- 若违背红黑树4条规则，需进行调整



| 73

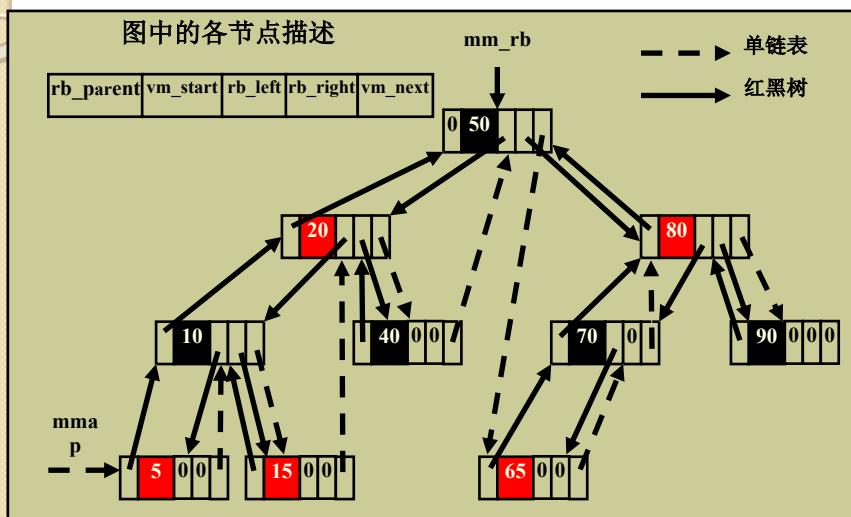
4.9.2.2 虚拟内存区域(6)

- 单链表+红黑树

- ◆ 当插入或删除一个虚拟内存区域时
 - 内核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
- ◆ 红黑树用来快速确定含有指定地址的虚拟内存区域
- ◆ 单链表用来扫描整个虚拟内存区域集合

| 74

4.9.2.2 虚拟内存区域(7)



| 75

4.9.2.2 虚拟内存区域(8)

- 虚拟内存区域访问权限
- 分配/释放虚拟内存区域
 - 进程创建或映射文件时，分配虚拟内存区域
 - ◆ `do_mmap(file,addr,len,long prot, flag,offset);`
 - 进程完成或取消文件映射时，释放虚拟内存区域
 - ◆ `do_munmap(mm,start,len);`

176

4.9.2.3 创建进程的地址空间

- 内核调用`copy_mm()`函数建立新进程的页表和内存描述符，以此创建其的地址空间
- 通常，每个进程都有自己独立的地址空间
 - `clone()`系统调用提供创建线程的功能
 - ◆ 通过传递一组标志，决定在父任务和子任务之间发生多少共享

标 志	含 义
<code>CLONE_FS</code>	共享文件系统信息
<code>CLONE_VM</code>	共享内存空间
<code>CLONE_SIGHAND</code>	共享信号处理程序
<code>CLONE_FILES</code>	共享打开文件集合

- `exit_mm()`函数释放进程的地址空间

177

4.9.2.4 堆的管理

- 每个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
- malloc/free

| 78

4.9.3 物理内存管理与分配

- 4.9.3.1 物理内存布局
- 4.9.3.2 物理内存分区
- 4.9.3.3 物理页框管理与分配
- 4.9.3.4 伙伴系统
- 4.9.3.5 slab分配

| 79

4.9.3.1 物理内存布局

- 页框大小为4KB
- 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
- Linux跳过RAM的第一个MB的空间，以避免将内核装入一组不连续的页框中
- Linux内核安装在RAM的从物理地址0x00100000开始的地方

| 80

4.9.3.2 物理内存分区(1)

- 内存空间的3个管理区
 - **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

| 81

4.9.3.2 物理内存分区(2)

- ZONE_DMA+ZONE_NORMAL 属于直接映射区
 - ◆ 虚拟地址 = 3G + 物理地址 或 物理地址 = 虚拟地址 - 3G，从该区域分配内存不会触发页表操作来建立映射关系
- ZONE_HIGHMEM 属于动态映射区
 - ◆ 128M 虚拟地址空间可以动态映射到 (X-896)M (其中 X 为物理内存大小) 的物理内存，从该区域分配内存需要更新页表来建立映射关系

182

4.9.3.2 物理内存分区(3)

- 直接映射区的作用
 - ◆ 是为了保证能够申请到物理地址上连续的内存区域
- 动态映射区
 - ◆ 会产生内存碎片，导致系统启动一段时间后，想要成功申请到大量的连续的物理内存，非常困难
 - ◆ 但是动态映射区带来了很高的灵活性(比如动态建立映射，缺页时才去加载物理页)

183

4.9.3.2 物理内存分区(4)

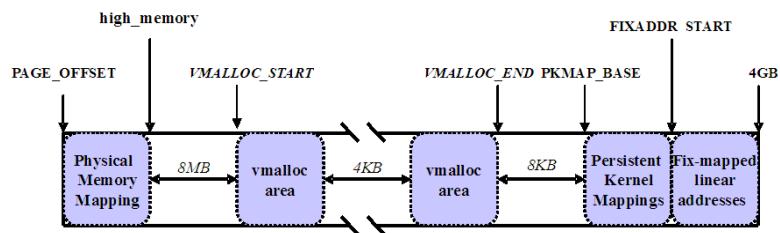
➤ 高端内存

- 借助128MB高端内存地址空间可以访问所有物理内存
- 基本思想
 - ◆ 借一段地址空间，建立临时地址映射，用完后释放，达到这段地址空间可以循环使用，访问所有物理内存

| 184

4.9.3.2 物理内存分区(5)

- 内核将高端内存划分为3部分：
 - ◆ VMALLOC_START~VMALLOC_END
 - ◆ KMAP_BASE~FIXADDR_START
 - ◆ FIXADDR_START~4G



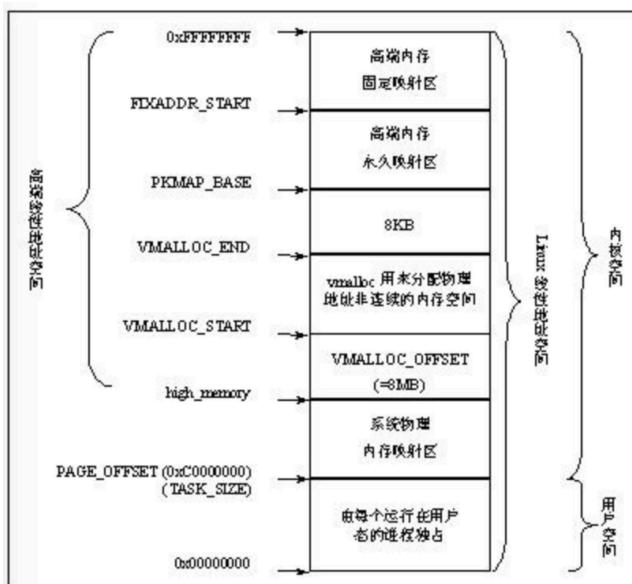
| 185

4.9.3.2 物理内存分区(6)

- 高端内存映射的3种方式
 - ◆ 映射到“内核动态映射空间”
 - `vmalloc()`, 给内核分配一个非连续内存区
 - ◆ 永久内核映射
 - 允许内核建立高端页框到内核地址空间的长期映射
 - 使用内核页表中的一个专门页表实现
 - ◆ 临时内核映射

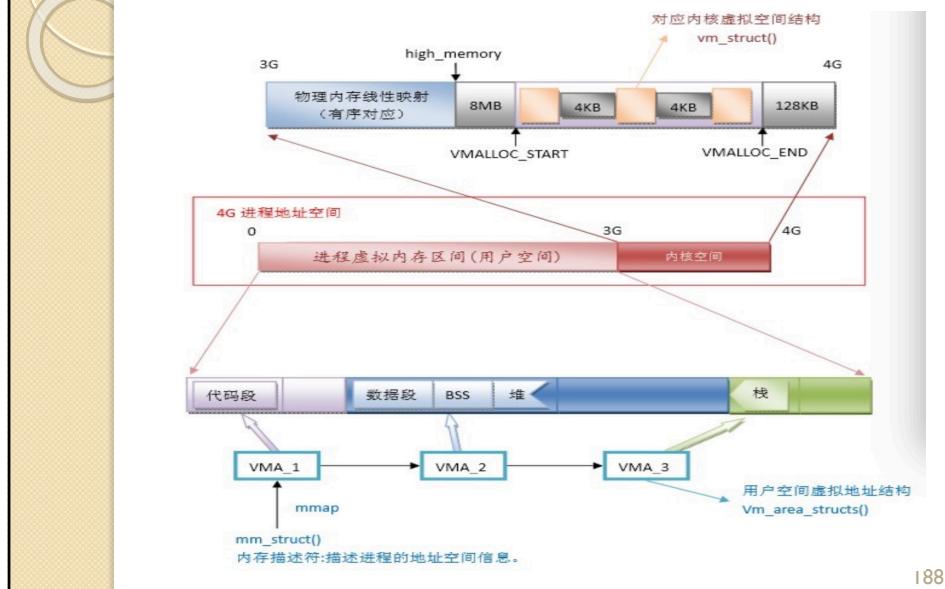
186

4.9.3.2 物理内存分区(7)

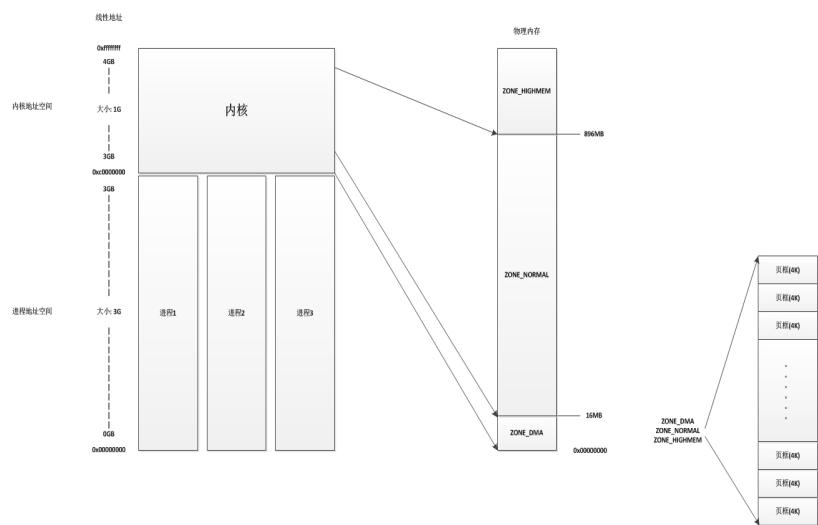


187

4.9.3.2 物理内存分区(8)



4.9.3.2 物理内存分区(9)



4.9.3.2 物理内存分区(10)

```
struct zone {  
    unsigned long free_pages;      //管理区中的空闲页框数  
    struct free_area free_area[MAX_ORDER]; //伙伴系统  
                                         //中的11个空闲页框链表  
    struct list_head active_list;   //活动页框列表  
    struct list_head inactive_list; //非活动页框列表  
    unsigned long nr_active;       //活动页框列表中页框数  
    unsigned long nr_inactive;     //非活动页框列表中页框数  
    spinlock_t lru_lock;           //自旋锁  
    struct page *zone_mem_map;    //指向管理区中  
                                 //首页框描述符的指针  
    .....  
};
```

190

4.9.3.3 物理页框管理与分配(1)

- 物理页框大小为4KB
- 内核记录每个物理页框的当前状态
 - 哪些页框属于进程
 - 哪些页框是内核代码或内核数据页
 - 哪些页框是空闲页框
 - 使用一个类型为struct page的页框描述符来记录页框的当前信息
 - 所有页框描述符存放在mem_map数组中

191

4.9.3.3 物理页框管理与分配(2)

```
struct page {          //页框描述符  
    unsigned long flags; //页框状态标志  
    atomic_t _count;    //页框的引用计数  
    atomic_t _mapcount; //页框对应的页表项数目  
    unsigned long private; //空闲时由伙伴系统使用  
    struct address_space *mapping; //用于页高速缓存  
    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/非活动页框链表  
    void *virtual; //页框所映射的内核虚地址  
};
```

192

4.9.3.3 物理页框管理与分配(3)

➤ 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
 - ◆ 负责接收动态内存的分配与释放请求
 - ◆ 保留的每CPU页框高速缓存
 - 为提高性能，系统为每个CPU预先分配的一些页框，以满足本地CPU发出的对单个页框的请求
 - ◆ 伙伴系统
 - 管理连续的空闲内存页框

193

4.9.3.4 伙伴系统(1)

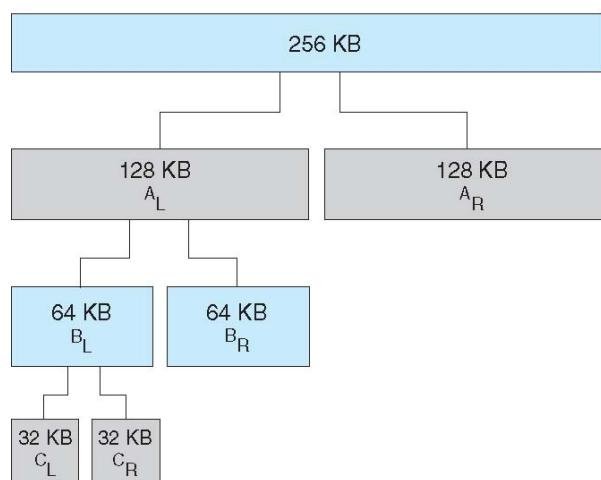
➤ Buddy 系统

- 从物理上连续的大小固定的内存上分配
- 内存按2的整数次幂大小进行分配
- 如果请求大小不是2的整数次幂，则调整到下一个更大的整数次幂
- 如果需要的空间比当前可用空间小，则将当前可用空间一分为二，直至满足需要空间的大小
- 如果需要的空间比当前可用空间大，则合并可用空间

| 94

4.9.3.4 伙伴系统(2)

physically contiguous pages



Buddy系统分配

| 95

4.9.3.4 伙伴系统(3)

➤ Linux的伙伴算法

- 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个连续空闲页框
- 请求一个具有8个连续页框的块
 - ◆ 先在8个连续页框块的链表中查找是否满足，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，就在更大的块链表中查找，直到找到为止

196

4.9.3.4 伙伴系统(4)

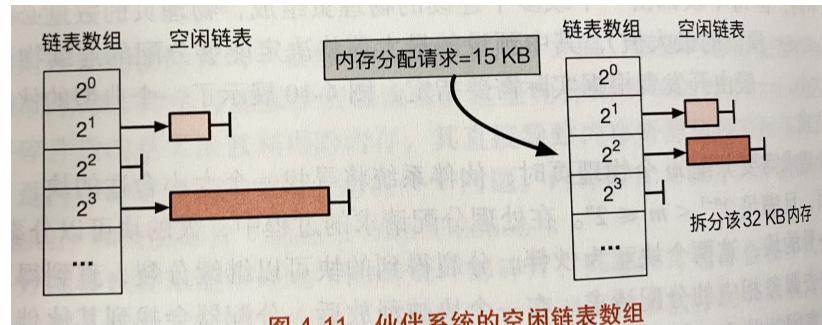


图 4-11 伙伴系统的空闲链表数组

197

4.9.3.5 slab分配(1)

➤ Buddy系统

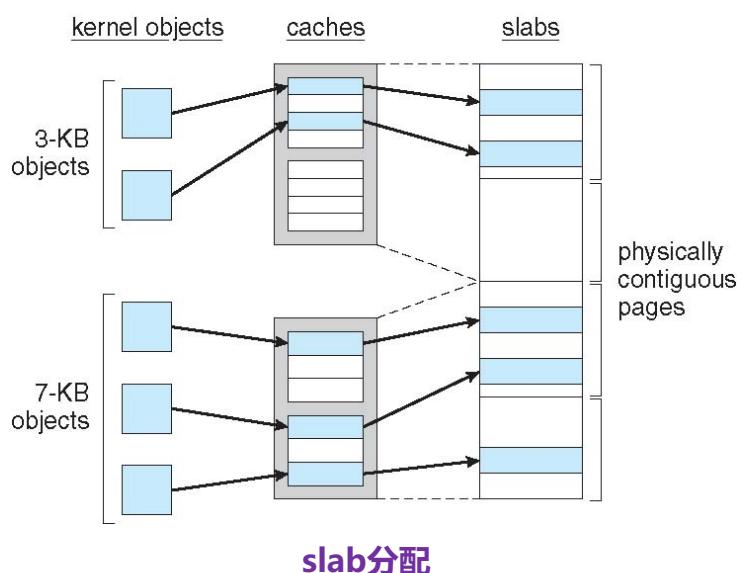
- 以页框为单位，适合于对大块内存的分配请求
- 调整到下一个2的整数次幂易产生碎片

➤ slab分配

- 为只有几十或几百个字节的小内存区分配内存
- slab是由一个或多个物理上连续的页组成的空间
- cache含有一个或多个slab
- 每个内核数据结构都有一个cache，每个cache含有内核数据结构的对象实例

198

4.9.3.5 slab分配(2)



slab分配

199

4.9.3.5 slab分配(3)

- 分配方法

- ◆ 创建cache时包括标记为空闲的若干对象，对象数量与slab大小相关
- ◆ 当需要内核数据结构对象时，可直接从cache获取，并将该对象标记为使用

- Linux的slab分配

- ◆ slab的状态

- 满：slab中的所有对象标记为使用
 - 空：slab中的所有对象标记为空闲
 - 部分：slab中的对象部分标记为空闲，部分标记为使用

200

4.9.3.5 slab分配(4)

- ◆ slab分配器首先从部分空闲的slab开始分配，如没有则从空的slab进行分配，如没有空slab则从连续物理块上分配新的slab，并把它赋给一个cache，然后再从新slab分配空间

- slab分配的优点

- 没有因碎片引起的内存浪费
 - 内存请求可以快速得到满足

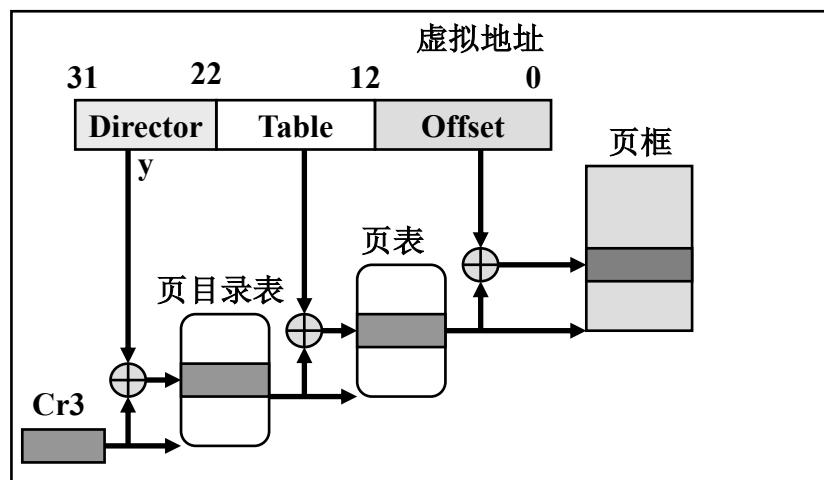
201

4.9.4 地址转换(1)

- 32位处理器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

202

4.9.4 地址转换(2)



203

4.9.4 地址转换(3)

➤ 页表项

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

204

4.9.5 请求调页与缺页异常处理

➤ 请求调页

- 增加了系统中的空闲页框数
- 页面置换策略是LFU (Least Frequently Used)

➤ 缺页调入

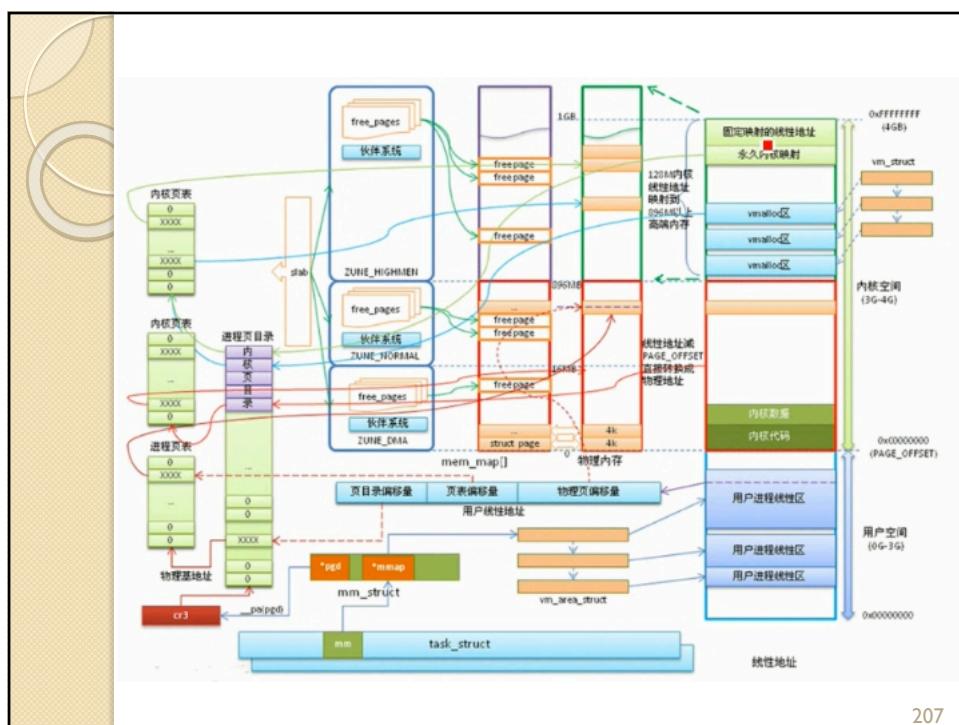
- 该页从未被进程访问过, 且没有相应的内存映射, 从指定文件中调页
- 该页属于非线性内存映射文件, 从磁盘读入页
- 该页已被进程访问过, 但其内容被临时保存到磁盘交换区上, 从交换区调入
- 该页在非活动页框链表中, 直接使用
- 该页正在由其它进程进行I/O传输过程中, 等待传输完成

205

4.9.6 盘交换区空间管理

- 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

206



207

4.10 Windows的存储器管理(1)

4.10.1 存储器管理的基本概念

4.10.2 Windows地址转换

4.10.3 页调度策略

208

4.10 Windows的存储器管理(2)

- 存储管理器是执行体的一个组件
- 提供基本服务
 - 存储器管理需要的系统服务
 - ◆ 分配、释放、保护虚存和物理主存，写时复制，虚拟页信息等
 - ◆ 以Win32 API或核心态设备驱动程序接口形式提供
 - 提供运行在核心态系统线程上的例程

209

4.10 Windows的存储器管理(3)

- 提供运行在核心态系统线程上的例程
 - ◆ 平衡工作集管理器
 - 维护空闲主存数量不低于某一界限并及时调整进程的工作集
 - ◆ 进程/堆栈交换器
 - 执行换入/换出操作
 - ◆ 更改页写入器
 - 将“脏”（被修改页）写回磁盘
 - ◆ 废弃段线程
 - 高速缓存和页文件的扩大与缩写
 - ◆ 零页线程
 - 维护系统有足够的零填充的空闲页存在

210

4.10.1 存储器管理的基本概念

4.10.1.1 进程地址空间的布局

4.10.1.2 进程私有空间的分配

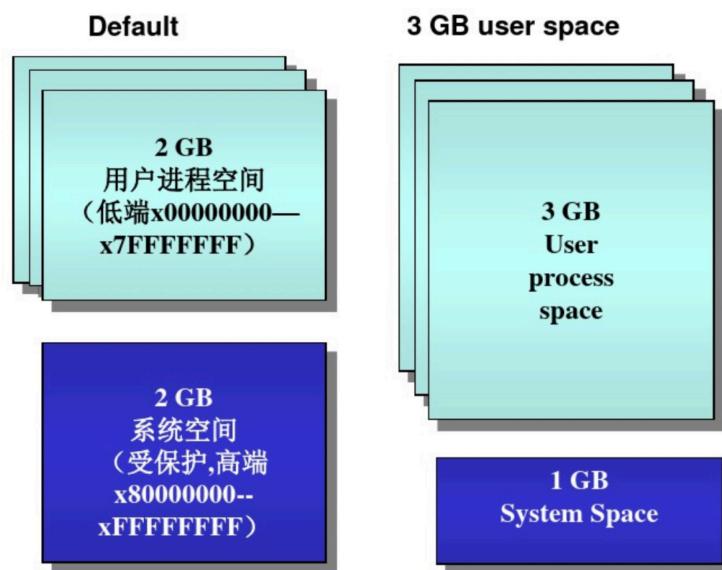
211

4.10.1.1 进程地址空间的布局(1)

- 在32位的地址空间上，允许每个用户进程占有4G的线性虚拟地址空间
 - 低2GB为进程的私有地址空间
 - ◆ 每个进程的私有代码和数据
 - 高2GB为进程公用的操作系统空间
 - ◆ 系统范围的代码和数据
 - NTOSKRNL.exe, HAL, 引导程序等
- Windows企业版有一个引导选项，允许用户拥有3GB的地址空间

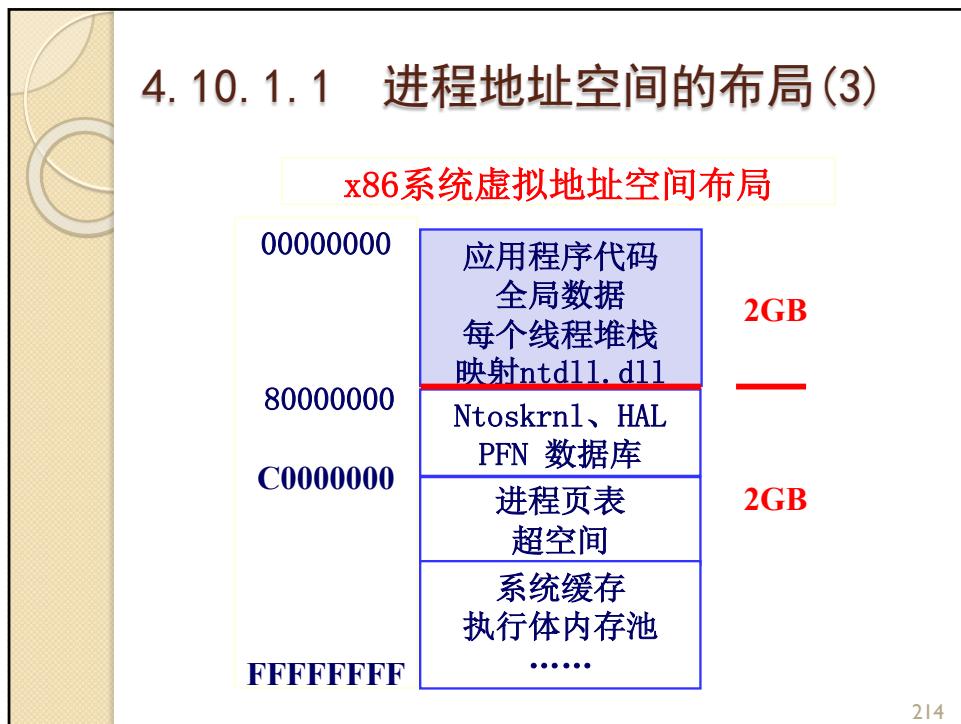
212

4.10.1.1 进程地址空间的布局(2)

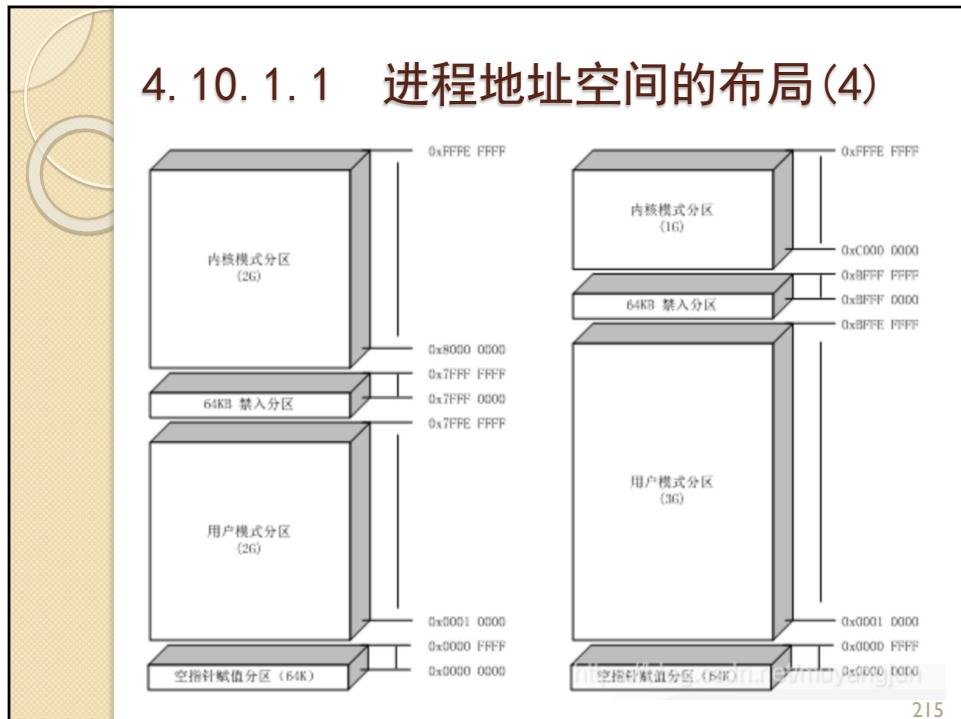


213

4.10.1.1 进程地址空间的布局(3)



4.10.1.1 进程地址空间的布局(4)

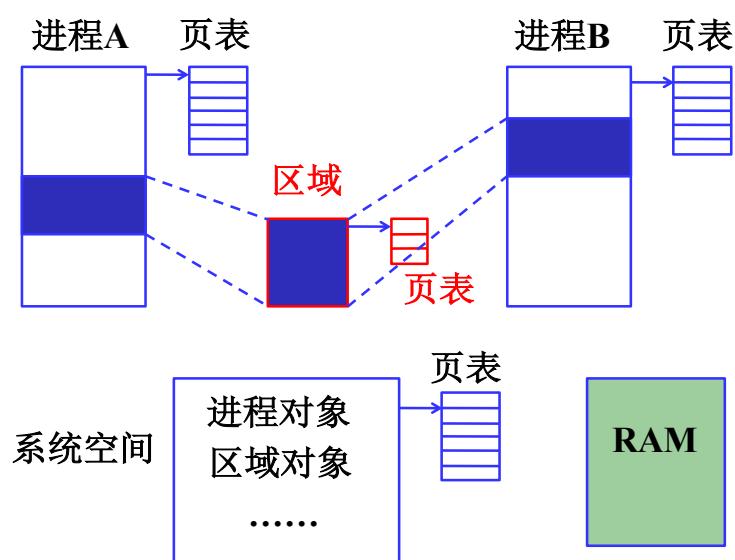


4.10.1.2 进程私有空间的分配(1)

- x86模型中，用全局描述符表(GDT)和局部描述符表(LDT)分别实现操作系统公共空间和进程私有虚拟地址空间的分配
- 进程私有空间分配
 - 一部分映射物理内存
 - ◆ 使用虚拟地址描述符VAD记录进程空间
 - 一部分映射硬盘上的交换文件
 - ◆ 多个进程共享存储区

216

4.10.1.2 进程私有空间的分配(2)



217

4.10.1.2 进程私有空间的分配(3)

➤ 虚拟地址描述符VAD

- 当进程要求分配一块连续虚存时，系统并不立即构造页表，而是为其建立一个VAD结构，记录该地址空间相关信息
 - 被分配的地址域的起始地址和结束地址
 - 该域是共享的还是私有的、该域的存取保护、是否可继承等
- 进程页表的构建一直推迟到访问页时才建立
- 一个进程的一组VAD结构构成一棵自平衡二叉树，便于快速查找

218

4.10.1.2 进程私有空间的分配(4)

虚拟地址描述符树

范围:20000000到2000FFFF
保护限制:读/写
继承: 有

VAD

范围:00002000到0000FFFF
保护限制:只读
继承: 有

范围:4E000000到4F000000
保护限制:写时复制
继承: 有

范围:32000000到330000FF
保护限制:只读
继承: 无

范围:7AAA0000到7AAA0OFF
保护限制:读/写
继承: 无

219

4.10.1.2 进程私有空间的分配(5)

➤ 区域对象

- 文件映射对象，可被多个进程共享的存储区
- 一个区域对象可被多个进程打开
- 可利用区域对象映射磁盘上的文件（包括页文件，可执行文件），访问这个文件就象访问内存中的一个大数组，而不需要读/写操作
 - ◆ 执行体利用区域对象将一个可执行的映像装入主存
 - ◆ 高速缓冲管理器使用区域对象访问一个被缓冲文件中的数据
 - ◆ 进程使用区域对象将一个大于进程地址空间的文件映射到进程整个或部分地址空间中

220

4.10.1.2 进程私有空间的分配(6)

• 区域对象的结构

◦ 对象头

◦ 对象体

- 最大尺寸：区域的最大长度；如果映射一个文件，则为文件大小；最大尺寸可达 2^{64} B
- 页保护方式：创建区域时，分配给该区域的所有页的保护方式
- 页文件（交换区）/映射文件：指出区域是否被创建为空，或是加载一个文件
- 基准的/非基准的：基准则要求共享该区域的所有进程在相同的虚拟地址空间出现
- 系统提供的对象服务
- 检索和更改区域对象体中属性

221

4.10.1.2 进程私有空间的分配(7)

➤ WIN32子系统实现文件映射的过程

- 用CreateFile()创建/打开一个被映射文件
- 用CreateFileMapping()创建一个与被映射文件大小相等的区域对象
- 用MapViewOfFile()将区域对象的一个视口映射到进程保留的某部分地址空间，之后进程就可以像访问主存一样访问文件
- 访问完成，用UnMapViewOfFile()解除被映射的视口

222

4.10.1.2 进程私有空间的分配(8)

➤ 虚拟存储空间

- 物理内存空间
- 页文件（交换区）
 - ◆ 作为主存补充的磁盘上的那部分空间
 - Eg. 计算机有128MB物理主存，同时在磁盘上有256MB的页文件，那么就认为计算机拥有384MB主存
 - Windows 支持多个页文件，当系统启动时，打开页文件，系统为每个页文件都维持一个打开的句柄
 - 一旦打开页文件，在系统运行期间不能删除

223

4.10.1.2 进程私有空间的分配(9)

➤ 虚存的分配

- 进程私有2G地址空间的地址域可能是
 - ◆ 空闲的：还没有被使用过
 - ◆ 被保留 reserved：已预留虚存，还没分配物理主存
 - ◆ 被提交 committed：已分配物理主存或交换区
- 主存分配原则：两阶段
 - ◆ 可以先保留地址域，后提交物理主存；也允许保留和提交同时实现

224

4.10.1.2 进程私有空间的分配(10)

● 保留地址空间

- ◆ 线程创建大的动态数据结构时，可以采用保留地址空间方法在虚地址空间预留一块区域，以防止进程的其他线程占用这段连续的虚拟地址空间，并用一个虚拟地址描述符记录
 - 试图访问保留的虚拟地址空间会造成访问冲突，由系统进行缺页处理

● 提交

- ◆ 在保留的地址空间中分配物理内存，并建立虚实映射

225

4.10.2 Windows地址转换

4.10.2.1 地址转换涉及的数据结构

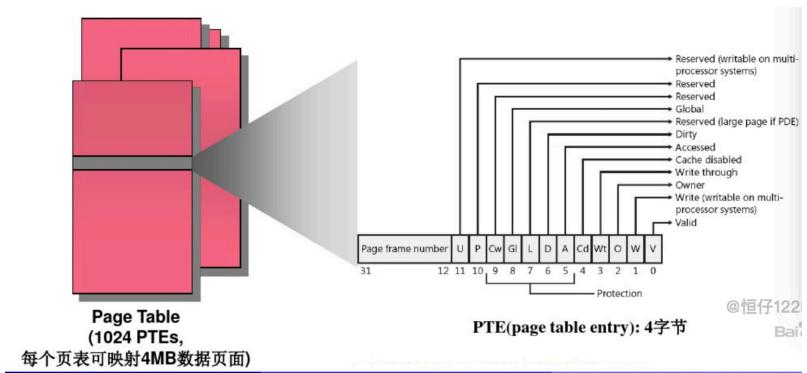
4.10.2.2 页错误处理

226

4.10.2.1 地址转换涉及的数据结构(1)

- 采用虚拟页式管理，页面大小默认为4KB
- 采用二级页表结构：页目录、页表

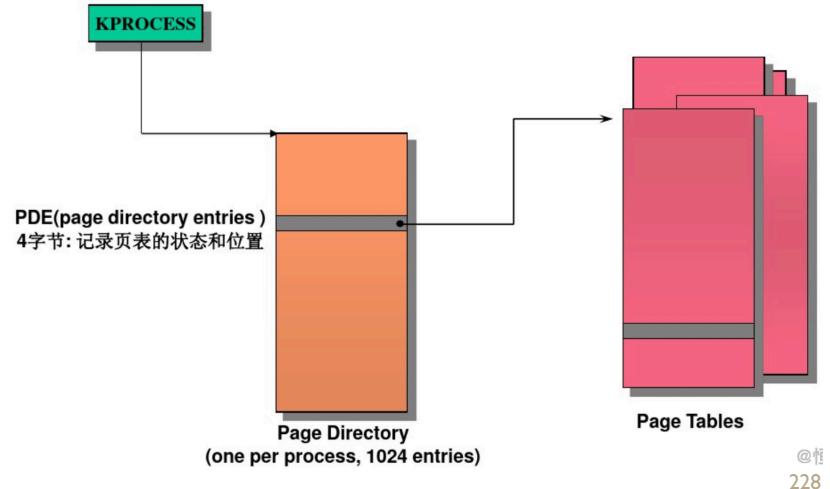
● 页表



227

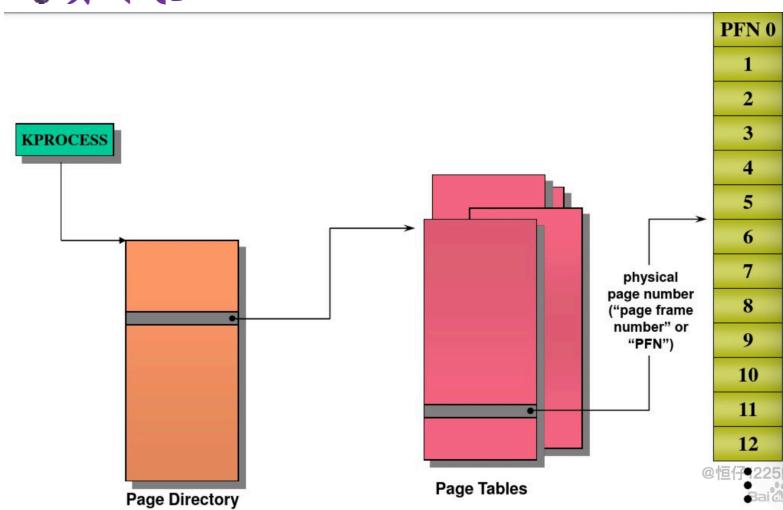
4.10.2.1 地址转换涉及的数据结构(2)

● 目录



4.10.2.1 地址转换涉及的数据结构(3)

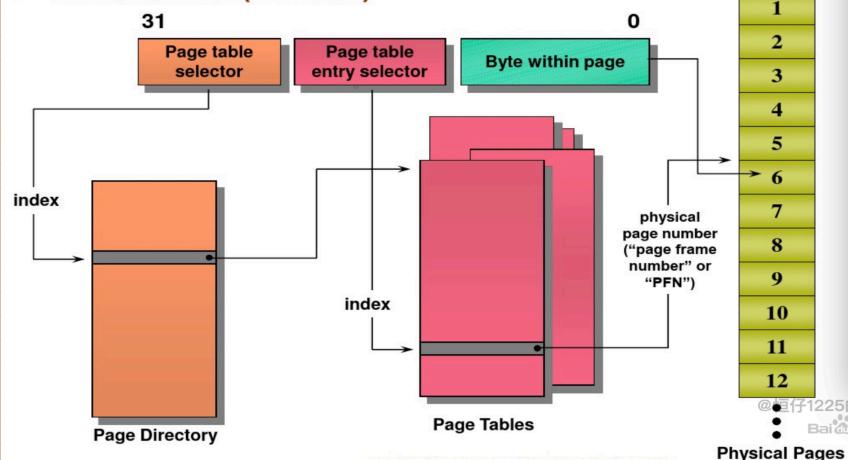
● 目录



4.10.2.1 地址转换涉及的数据结构(4)

➤ 虚拟地址变换过程

地址变换过程(x86系统)



4.10.2.1 地址转换涉及的数据结构(5)

➤ 页框数据库

- 页框数据库用于跟踪物理主存的使用
- 是一个数组，其索引号从0到主存的页框总数-1
- 内存页框有八种状态
 - 活动（有效）：是进程工作集的一部分
 - 转换：一个页框正处于I/O操作进行中
 - 备用：已不属于工作集，页表项仍然指向该页框，但被标记为正在转移的无效PTE

231

4.10.2.1 地址转换涉及的数据结构(4)

- ◆更改：已不属于工作集，修改后未写磁盘，页表项仍指向该页框，被标记为正在转移的无效PTE
- ◆更改不写入：更改但不写入磁盘
- ◆空闲：不属于任何一个工作集
- ◆零初始化：清零的空间页框
- ◆坏页框
- 页框链表
 - ◆为快速定位，形成6个页框链表

232

4.10.2.1 地址转换涉及的数据结构(5)

- 零初始化
- 空闲
- 备用
- 更改
- 更改不写入
- 坏的
- ◆活动（有效）页框由进程页表管理
- ◆转换页框不在链表中

233

4.10.2.1 地址转换涉及的数据结构(6)

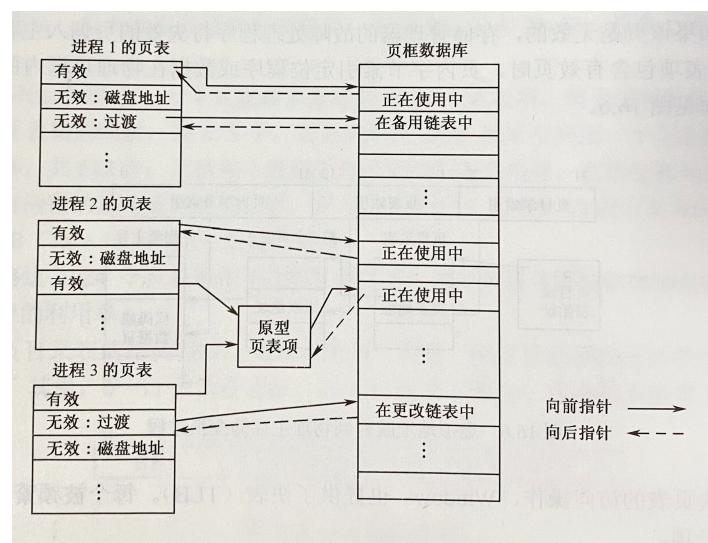
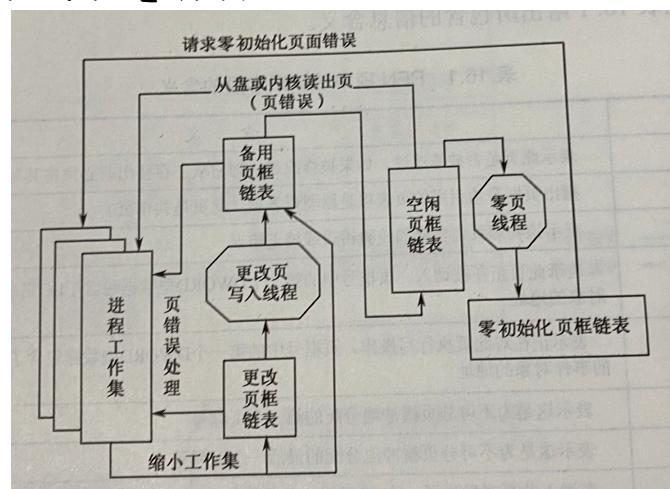


图 16.7 页表与页框数据库的关系

234

4.10.2.1 地址转换涉及的数据结构(7)

➤ 页框的状态转换



235

4.10.2.2 页错误处理(1)

➤ 无效页处理

- 当被访问的页无效时，产生无效页错误
- 处理
 - ◆ 访问一个未知页，其页表项为0或页表不存在（首次访问）
 - 系统为包含该地址的页创建一个页表
 - 从该进程的VAD树中查找包含该地址的VAD，填充页表项
 - ◆ 所访问的页不在主存，在外存页文件或映像文件中
 - 系统分配一个物理页框，将所需页从磁盘读出，并放入工作集中

236

4.10.2.2 页错误处理(2)

- ◆ 所访问的页在备用链表或更改链表
 - 将该页从指定链表中移出，放入进程或系统工作集
- ◆ 访问一个请求零初始化的页
 - 页调度器查看零页链表是否为空，是则从自由链表中取一页将其清零；若自由链表也为空，则从备用页链表中取一页将其清零，放入工作集中

237

4.10.2.2 页错误处理(3)

- ◆ 页访问违约，与访问权限不符
 - 对一个只读页执行写操作
 - 从用户态访问一个只能在核心态下访问的页
 - 对一个写保护页执行写操作
- ◆ 对一个写时复制页执行写操作，为进程进行页复制
- ◆ 在多处理机系统中，对一个有效但尚未执行写操作的页执行写操作，在页表项中将修改位置1

238

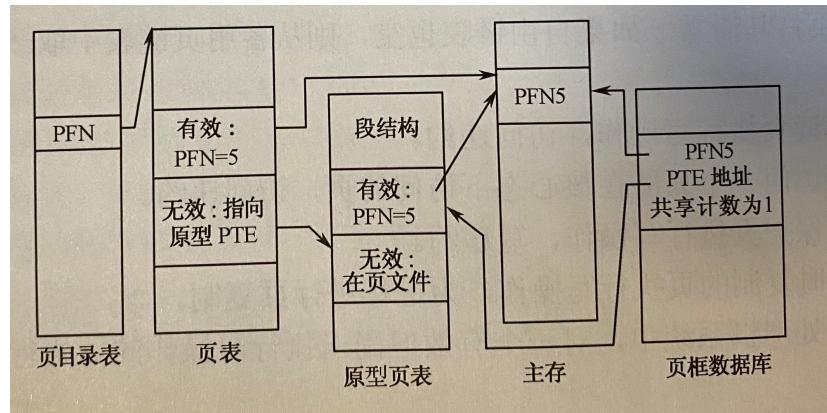
4.10.2.2 页错误处理(4)

➤ 原型页表项

- 当一个页框被两个或多个进程共享时，存储器管理器依靠一个称为“原型页表”(Prototype PTE)的结构记录这些被共享的页框
- 引入
 - ◆ 尽可能减少对各进程页表的影响
- 原型页表项位于页表和页框数据库之间
- 区域对象有原型页表
 - ◆ 当进程访问区域对象中的页时，利用原型页表填写进程页表

239

4.10.2.2 页错误处理(4)



240

4.10.3 页调度策略(1)

- 采用请求调页和集群方式
 - 产生缺页中断时，根据程序局部性原理，将所缺的页及其前后的一些页装入主存
- 置页策略
 - 产生缺页中断时，存储区管理器必须确定将调入的虚拟页放在物理内存的具体位置
- 置换策略
 - 在多处理器系统中，采用局部先进先出置换策略
 - 在单处理器系统中，更接近于最近最久未使用策略

241

4.10.3 页调度策略(2)

➤ 进程工作集

- 进程正在使用的物理页面的集合
- 系统根据内存情况允许进程工作集规模增大或缩小

➤ 系统工作集

- 为可换页的系统代码和数据分配一定数量的页框

➤ 平衡工作集管理器

- 是一个系统线程，用来调整进程和系统工作集

242

本章要点(1)

- 逻辑地址、物理地址与地址映射
- 静态重定位与动态重定位
- 存储器保护
- 交换与覆盖
- 动态分区存储管理算法
- 分页管理方式
- 分段管理方式
- 段页式管理方式

243

本章要点(2)

- 请求分页管理方式
- 最佳置换算法
- 先进先出置换算法
- 最近最久未用置换算法
- 时钟置换算法
- 程序局部性原理
- 工作集
- Linux存储器管理
- Windows存储器管理

244