操作系统
Operating System

北京理工大学计算机学院
马　锐
Email: mary@bit.edu.cn

版权声明

第8章　Linux存储器管理

第8章　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

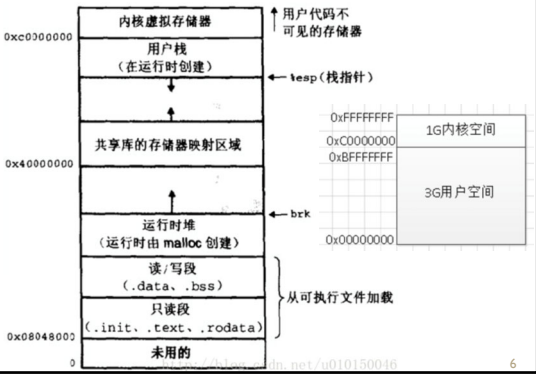
4

8.1　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)

5

8.1　虚拟地址空间布局(2)



6

8.1 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0x08048000~0xbfffffff
  - 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（其中0x0000000000000000~0x00007fffffffffff表示用户空间，0xFFFF800000000000~0xFFFFFFFFFFFFFFFF表示内核空间），40位表示物理地址

7

8.1 虚拟地址空间布局(4)

- 逻辑地址空间
- 线性地址（虚拟地址）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

8

8.1 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

9

8.2 进程地址空间管理

- 8.2.1 虚拟内存区域
- 8.2.2 进程地址空间管理
- 8.2.3 创建进程的地址空间
- 8.2.4 堆的管理

10

8.2.1 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

11

8.2.1 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
   unsigned long vm_flags; //虚拟内存区域的标识
   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
   ……
}
```

12

8.2.1 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）

13

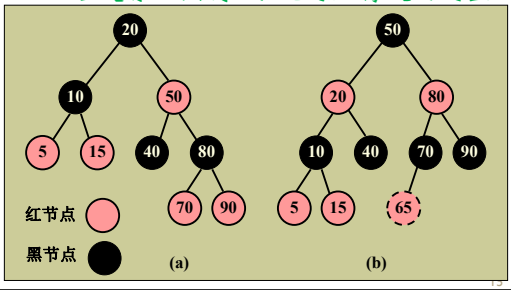
8.2.1 虚拟内存区域(4)

- 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)

14

8.2.1 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

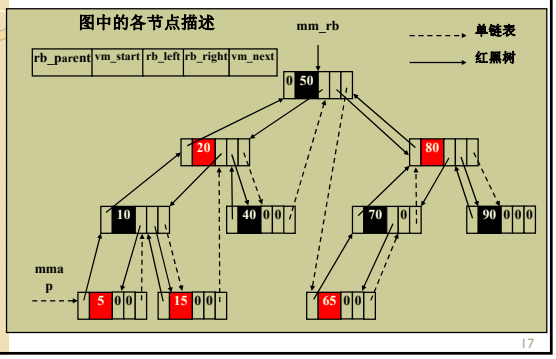


8.2.1 虚拟内存区域(6)

- 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

16

8.2.1 虚拟内存区域(7)



17

8.2.1 虚拟内存区域(8)

➢ 虚拟内存区域访问权限

- 每个虚拟内存区域由一组连续的页组成，具有相同的访问方式
- 在进程访问该区域的某个页时，才为其建立页表，并由虚拟内存区域描述符vm_area_struct中的vm_flags字段给出允许的访问方式的标识

18

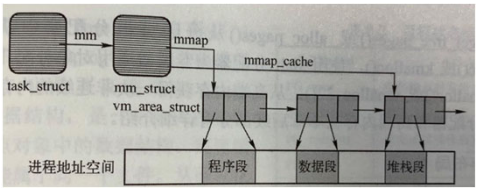
8.2.1 虚拟内存区域(9)

- 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);

19

8.2.2 进程地址空间管理(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



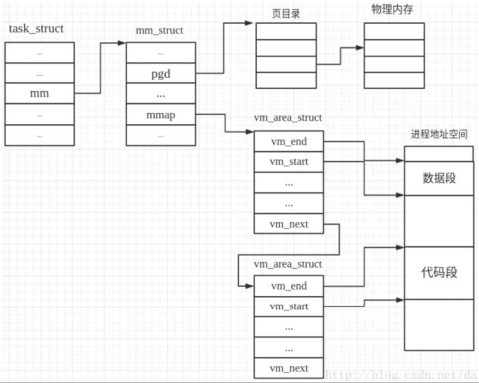
20

8.2.2 进程地址空间管理(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
   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
   ……
}
```

21

8.2.2 进程地址空间管理(3)



22

8.2.3 创建进程的地址空间

- 创建进程地址空间
  - 内核调用copy_mm()函数建立新进程的页表和虚拟内存描述符
  - 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
      - CLONE_VM：共享内存空间
- 释放进程地址空间
  - exit_mm()：释放虚拟内存描述符和所有相关数据结构

23

8.2.4 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

24

8.3 物理内存管理与分配(1)

- 8.3.1 物理内存布局
- 8.3.2 物理页框管理
- 8.3.3 物理内存分区
- 8.3.4 物理页框分配
- 8.3.5 伙伴系统
- 8.3.6 Slab分配

25

8.3 物理内存管理与分配(2)

- 当用户态进程请求动态内存时，系统并不立即为它分配物理内存，而是在进程的地址空间为它分配一个新的虚拟内存区域。当进程运行产生缺页中断时，系统通过缺页中断处理程序调用相应函数实现物理页框的分配。
  - _get_free_pages()/ alloc_pages()：从分区页框分配器中获得页框
  - kmem_cache_alloc()/kmalloc()：使用slab分配器为专用或通用对象分配几十或几百字节的内存块
  - vmalloc()/vmalloc_32()：从高端内存获得一块非连续的内存区

26

8.3.1 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

27

8.3.2 物理页框管理（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

28

8.3.2 物理页框管理（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，
                                 //用于高端物理页框
};
```

29

8.3.3 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
    - 内核通过把ZONE_DMA和ZONE_NORMAL线性映射到虚拟地址空间的第4个GB，实现对它们的直接访问
  - 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

30

8.3.3 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

31

8.3.3 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框)

32

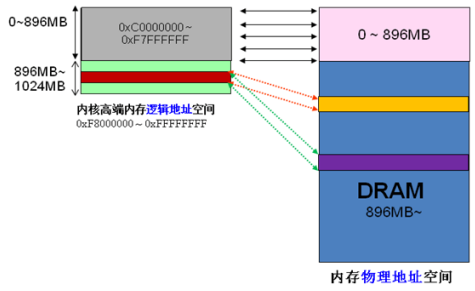
8.3.3 物理内存分区(4)

➢ 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，使得这段地址空间可以循环使用，访问所有物理内存

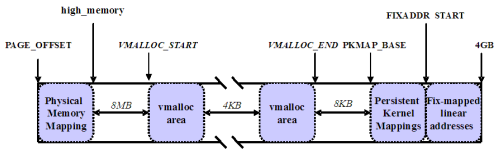
33

8.3.3 物理内存分区(5)



8.3.3 物理内存分区(6)

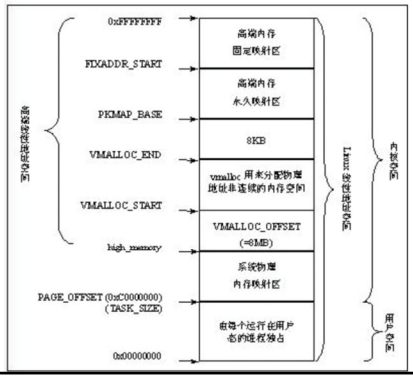
- 内核将高端内存划分为3部分：
  - VMALLOC_START~VMALLOC_END
  - KMAP_BASE~FIXADDR_START
  - FIXADDR_START~4G



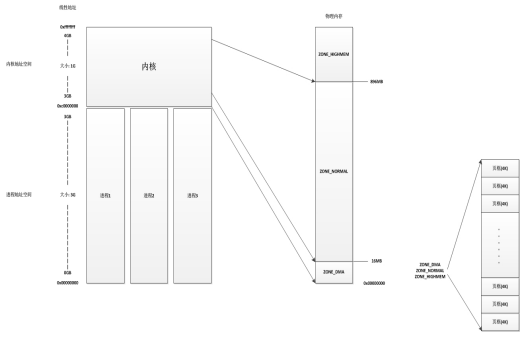
8.3.3 物理内存分区(7)

- 高端内存映射的3种方式
  - 非连续内存区映射
    - 映射到“内核动态映射空间”
    - 用4KB大小的安全区隔离成多个非连续的内存区
    - 利用vmalloc()分配
  - 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
  - 固定映射

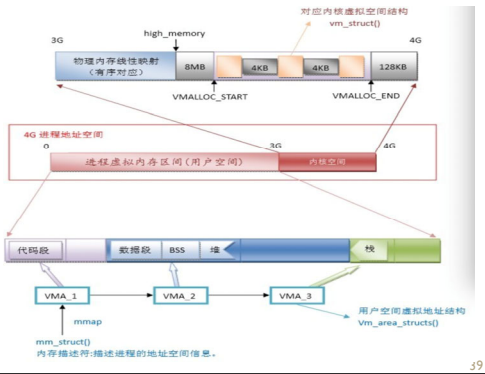
8.3.3 物理内存分区(8)



8.3.3 物理内存分区(9)



8.3.3 物理内存分区(10)



8.3.3 物理内存分区(11)

```
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
   ……
};
```

40

8.3.4 物理页框分配(1)

- 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
    - 系统中空闲页框不足时，触发页框回收算法

41

8.3.4 物理页框分配(2)

- 分区页框分配器的内部组织结构

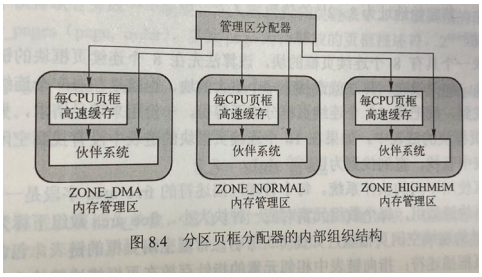


图 8.4 分区页框分配器的内部组织结构

42

8.3.5 伙伴系统(1)

- Buddy系统基本思想
  - 在现代OS中广泛用于分配连续物理内存块，可以高效、有效地管理内存，解决外部碎片问题
  - 基本思想
    - 将物理内存划分成连续的块，以块为基本单位进行分配
    - 不同块的大小可以不同，但每个块都由一个或多个连续的物理页组成，物理页的数量是2的整数次幂
    - 预设的最大值决定能够分配的连续物理内存区域的最大大小

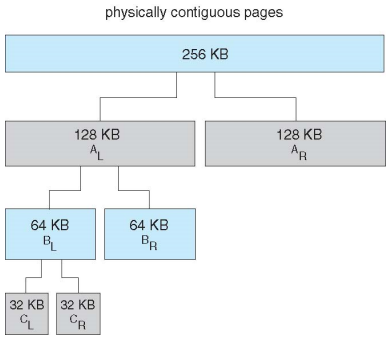
43

8.3.5 伙伴系统(2)

- 当一个请求需要分配m个物理页时，伙伴系统将寻找一个大小合适的块，该块包含2^n个物理页，且满足$2^{n-1}<m<=2^n$
- 处理分配请求时，大的块可以一分为二，形成两个伙伴块；分裂得到的块可以继续分裂，直至得到大小合适的块去满足相应的分配请求
- 在一个块被释放后，分配器会查找其伙伴块，若伙伴块也处于空闲状态，则将两个伙伴块合并（合二为一），形成一个大的空闲块，然后继续尝试向上合并

44

8.3.5 伙伴系统(3)



Buddy系统基本思想：基于伙伴块进行分裂与合并

45

8.3.5 伙伴系统(4)

- 应用示例
  - 利用空闲数组实现伙伴系统
  - 全局有一个有序数组，数组的每一项指向一条空闲链表，每条链表将其对应大小的空闲块连接起来（一条链表中的空闲块大小相同）
  - 当接收到分配请求时，伙伴分配器先计算出应该分配多大的空闲块，然后查找对应的空闲链表

46

8.3.5 伙伴系统(5)

- Eg1：请求分配15KB内存
- Eg2：请求分配8KB内存
- Eg2：请求分配4KB内存

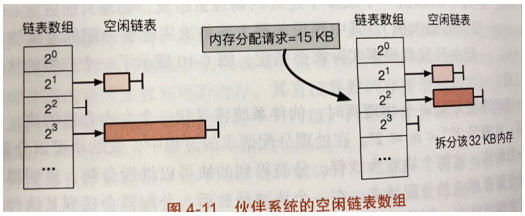


图 4-11 伙伴系统的空闲链表数组

47

8.3.5 伙伴系统(6)

- 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个连续空闲页框的块，每个块的第一个页框的物理地址是该块大小的整数倍
  - 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

48

8.3.5 伙伴系统(7)

- Linux的伙伴算法

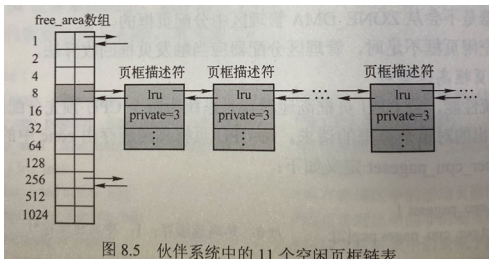


图 8.5 伙伴系统中的 11 个空闲页框链表

49

8.3.6 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
  - 内核通常需要分配的内存大小只有几十或几百字节，仍采用伙伴系统会产生严重的内部碎片
- Slab分配
  - 满足OS（频繁的）分配小对象的需求
  - slab依赖于伙伴系统进行物理页的分配，是由一个或多个物理上连续的页组成的空间
  - slab分配器将伙伴系统分配的大块内存进一步细分为小块内存进行管理，只分配固定大小的内存块

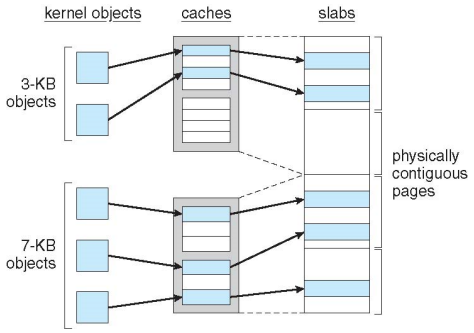
50

8.3.6 Slab分配(2)

- 分配方法
  - 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
  - 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

51

8.3.6 Slab分配(3)



slab分配

52

8.3.6 Slab分配(4)

- 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
  - 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

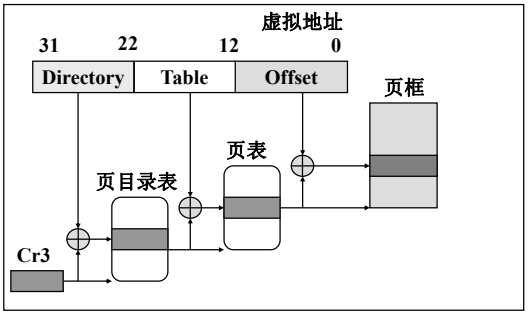
53

8.4 地址转换(1)

- 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
- 虚地址分为3个域：页目录索引（前10位）、页表索引（中10位）、页内偏移（后12位）

54

8.4 地址转换(2)



55

8.4 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

56

8.5 请求调页与缺页异常处理

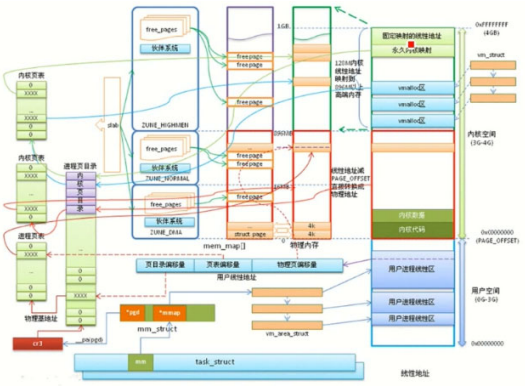
- 请求调页
  - 增加了系统中的空闲页框数
  - 页面置换策略是LFU（Least Frequently Used）
- 缺页调入
  - 该页从未被进程访问过，且没有相应的内存映射，从指定文件中调页
  - 该页属于非线性内存映射文件，从磁盘读入页
  - 该页已被进程访问过，但其内容被临时保存到磁盘交换区上，从交换区调入
  - 该页在非活动页框链表中，直接使用
  - 该页正在由其它进程进行I/O传输过程中，等待传输完成

57

8.6 盘交换区空间管理

- 每个盘交换区都由一组4KB的页槽组成，即由一组与页大小相等的块组成
- 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

58



59