**1.伙伴系统算法**

内核应该为分配一组连续的页框而建立一种健壮、高效的分配策略。为此，必须解决著名的内存，也就是所谓的外锁片问题（external fragmentation）。频繁的请求和释放不同大小的一组连续页框，必然导致在已分配的块内分散了许多小块的空闲页框。由此带来的问题时，即使有足够的空闲页框可以满足请求，但要分配一个大块的连续页框无法满足。

从本质上来说，避免外碎片的方法有两种：

（1）利用分页单元把一组非连续的空闲页框映射到连续的线性地址空间；

（2）开发一中适当的技术来记录现存的空闲连续页框快的情况，以尽量满足对小块的请求而分割大的空闲块。

 Linux内核中引入了伙伴系统算法(buddy system)。把所有的空闲[页框](http://baike.baidu.com/view/1838099.htm)分组为11个块链表，每个块链表分别包含大小为1，2，4，8，16，32，64，128，256，512和1024个连续页框的页框块。最大可以申请1024个连续[页框](http://baike.baidu.com/view/1838099.htm)，对应4MB大小的连续内存。每个[页框](http://baike.baidu.com/view/1838099.htm)块的第一个页框的物理地址是该块大小的整数倍。例如，大小为16个页框的块，其起始地址是16\*2^12（2^12=4096）的整数倍。

假设要申请一个256个[页框](http://baike.baidu.com/view/1838099.htm" \t "_blank)的块，先从256个页框的链表中查找空闲块，如果没有，就去512个[页框](http://baike.baidu.com/view/1838099.htm" \t "_blank)的链表中找，找到了则将页框块分为2个256个页框的块，一个分配给应用，另外一个移到256个页框的链表中。如果512个页框的链表中仍没有空闲块，继续向1024个页框的链表查找。如果1024块存在，则将其中的256页框作为请求返回，剩余的768分成256块和512块分别插到相应的链表中。如果仍然没有，则返回错误。

[页框](http://baike.baidu.com/view/1838099.htm)块在释放时，会主动大小为相同的一个空闲伙伴块合成为2倍大小的单独块较大的页框块。两个块称为伙伴需要满足一下条件：

（1）两个块具有相同的大小

（2）它们的物理地址是连连续的。

（3）第一块的第一个页框的物理地址是2\*b\*2^12的倍数。

**2.数据结构**

包含一个11元素、元素类型为free\_area的一个数组，每个元素对应一块大小。

free\_area每个元素中有一个free\_list，表示双向循环链表的头，这个双向循环链表集中了大小为2^k页的空闲块对应的页描述符。该链表包含每个空闲页框块（大小为2^k）的起始页框的页描述符。指向链表中相邻元素的指针存放在页描述符的lru字段中。

 free\_area每个元素还包含一个nr\_free字段，它指定了大小为2^k的页框块个数。当然，如果没有大小为2^k的空闲页框块，则nr\_free等于0且free\_list为空。

 一个空闲块的第一个页的描述符的private字段存放了块的order，也就是k。正式由于这个字段，当页框被释放时，内核可以确定这个块的伙伴是否也空闲。如果是的话就可以把两个块合成一个2^(i+1)的块。

**3.实现**

**（1）分配块**

\_\_rmqueue()用来在管理区找到一个空闲块。需要两个参数：管理区描述符的地址和order。

         struct free\_area \*area;

         unsigned int current\_order;

         for(current\_order=order;current\_order<11;++current\_order){

         area=zone->free\_area+current\_order;//到相应的数组中

         if(!list\_empty(&area->free\_list)) goto block\_found;//进入相应的链表,将相应的page(描述符)从freelist中去掉

          }

         block\_found:

         page=list\_entry(area->free\_list.next,struct page,lru);//找到链表的第一个节点

         list\_del(&page-lru);

         ClearPagePrivate(page);

         page->private=0;

         area->nr\_free--;//相应区域的空闲页框块减少

         zone->free\_pages -= 1UL<<order;//管理区内的页框

如果从curr\_order链表中找到的块大于order，就执行一个while循环将剩余的块插到相应的free\_list中去。例如申请256，找到1024，则把剩下的块插到256和512的free\_list中去。

          size = 1<< current\_order;

          while(curr\_order>order){

area--;

                curr\_order--;

                size>>=1;

                buddy=page+size;

                list\_add(&buddy\_lru,&area\_free\_list);

                area->nr\_free++;

                buddy->private=current\_order;

          setPagePrivate(buddy);

          }

**（2）释放块**

\_\_free\_pages\_bulk函数按照伙伴系统的策略释放页框。三个参数：page（被释放块的第一个页框描述符地址），zone（管理区描述符地址），order（块大小的对数）。

          struct page\*base = zone->zone\_mem\_map;

          unsigned long buddy\_idx,page\_idx=page-base;

          struct page\* buddy,\*coalesced;

          int order\_size=1<<order;

          while(order<10){

                  buddy\_idx=page\_idx^(1<<order);//得到伙伴块的索引

                  buddy=base+buddy\_idx;

                  if(!page\_is\_buddy(buddy,order))break;//判断符不符合buddy的条件

                  list\_del(&buddy->lru);//满足从链表中删除去合成新的页框块

                  zone->free\_area[order].nr\_free--;

             ClearPagePrivate(page);

            page->private=0;

            page\_idx &= buddy\_idx;

            order++;

       }

      //合成

      coalesced = base+page\_idx；

      coalesced ->private=order;

      SetPagePrivate(coalesced );

       list\_add(&coalesced->lru,&zone->free\_area[order].free\_list);

       zone->free\_area[order].nr\_free++;