**Linux 高端内存**

高端内存是指物理地址大于 896M 的内存。  
  
对于这样的内存，无法在“内核直接映射空间”进行映射。  
  
为什么？  
因为“内核直接映射空间”最多只能从 3G 到 4G，只能直接映射 1G 物理内存，对于大于 1G 的物理内存，无能为力。  
  
实际上，“内核直接映射空间”也达不到 1G， 还得留点线性空间给“内核动态映射空间” 呢。  
因此，Linux 规定“内核直接映射空间” 最多映射 896M 物理内存。  
  
对 于高端内存，可以通过 alloc\_page() 或者其它函数获得对应的 page，但是要想访问实际物理内存，还得把 page 转为线性地址才行（为什么？想想 MMU 是如何访问物理内存的），也就是说，我们需要为高端内存对应的 page 找一个线性空间，这个过程称为高端内存映射。  
  
高端内存映射有三种方式：  
  
1、映射到“内核动态映射空间”  
这种方式很简单，因为通过 vmalloc() ，在”内核动态映射空间“申请内存的时候，就可能从高端内存获得页面（参看 vmalloc 的实现），因此说高端内存有可能映射到”内核动态映射空间“ 中。  
  
2、永久内核映射  
如果是通过 alloc\_page() 获得了高端内存对应的 page，如何给它找个线性空间？  
内核专门为此留出一块线性空间，从 PKMAP\_BASE 到 FIXADDR\_START ，用于映射高端内存。在 2.4 内核上，这个地址范围是 4G-8M 到 4G-4M 之间。这个空间起叫“内核永久映射空间”或者“永久内核映射空间”  
  
这个空间和其它空间使用同样的页目录表，对于内核来说，就是 swapper\_pg\_dir，对普通进程来说，通过 CR3 寄存器指向。  
  
通常情况下，这个空间是 4M 大小，因此仅仅需要一个页表即可，内核通过来 pkmap\_page\_table 寻找这个页表。  
  
通过 kmap()， 可以把一个 page 映射到这个空间来  
  
由于这个空间是 4M 大小，最多能同时映射 1024 个 page。因此，对于不使用的的 page，及应该时从这个空间释放掉（也就是解除映射关系），通过 kunmap() ，可以把一个 page 对应的线性地址从这个空间释放出来。  
  
  
3、临时映射

内核在 FIXADDR\_START 到 FIXADDR\_TOP 之间保留了一些线性空间用于特殊需求。这个空间称为“固定映射空间”

在这个空间中，有一部分用于高端内存的临时映射。

这块空间具有如下特点：

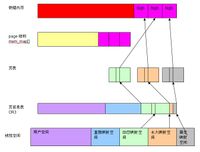
1、 每个 CPU 占用一块空间

2、 在每个 CPU 占用的那块空间中，又分为多个小空间，每个小空间大小是 1 个 page，每个小空间用于一个目的，这些目的定义在kmap\_types.h 中的 km\_type 中。

当要进行一次临时映射的时候，需要指定映射的目的，根据映射目的，可以找到对应的小空间，然后把这个空间的地址作为映射地址。这意味着一次临时映射会导致以前的映射被覆盖。

通过 kmap\_atomic() 可实现临时映射。

下图简单简单表达如何对高端内存进行映射

[](http://hiphotos.baidu.com/liu_bin0101/pic/item/e78de6a854ad9d92ca130ce7.jpg)

－－－－－－－－－－－－－－－－－－－－－－－－－

高端内存含义为：线性地址空间 PAGE\_OFFSET + 896M 至4G的最后128M线性地址 <==映射==> 896M以上的物理页框，非直接映射。有3种方法：非连续内存区映射，永久内核映射，临时内核映射（固定映射）  
   从 PAGE\_OFFSET开始的线性地址区域为：  
   PAGE\_OFFSET(3G)|物理内存映射 --8M-- vmalloc区 --4K-- vmalloc区 --8K-- 永久内核映射--临时内核映射（固定映射）|4G  
  
1. 非连续区映射  
  
1.1 每个非连续内存区都对应一个类型为 vm\_struct的描述符，通过next字段，这些描述符被插入到一个vmlist链表中。  
  
1.2 三种非连续区的类型：  
   VM\_ALLOC   -- 物理内存(调用alloc\_page)和线性地址同时申请，物理内存是 \_\_GFP\_HIGHMEM类型（分配顺序是HIGH, NORMAL, DMA ）（可见vmalloc不仅仅可以映射\_\_GFP\_HIGHMEM页框，它的主要目的是为了将零散的，不连续的页框拼凑成连续的内核逻辑地址空间...）  
   VM\_MAP     -- 仅申请线性区，物理内存另外申请，是VM\_ALLOC的简化版  
   VM\_IOREMAP -- 仅申请线性区，物理内存另外申请（这里的物理内存一般都是高端内存，大于896M的内存）  
  
2. 永久内核映射  
  
2.1 永久内存映射允许建立长期映射。使用主内核页表中swapper\_pg\_dir的一个专门页表。  
    pkmap\_page\_table: 专门的页表。页表表项数由LAST\_PKMAP(512或1024）产生。  
    page\_address\_htable: 存放地址的  
    pkmap\_count: 包含LAST\_PKMAP个计数器的数组。  
    PKMAP\_BASE: 页表线性地址从PKMAP\_BASE开始。  
  
2.2 如果LAST\_PKMAP个项都用完，则把当前进程置为 TASK\_UNINTERRUPTIBLE，并调用schedule()  
  
3. 临时内存映射  
  
3.1 可以用在中断处理函数和可延迟函数的内部，从不阻塞。因为临时内存映射是固定内存映射的一部分，一个地址固定给一个内核成分使用。  
  
3.2 每个CPU都有自己的一个13个窗口（一个线性地址及页表项）的集合。  
enum km\_type {  
    KM\_BOUNCE\_READ,  
    KM\_SKB\_SUNRPC\_DATA,  
    KM\_SKB\_DATA\_SOFTIRQ,  
    KM\_USER0,  
    KM\_USER1,  
    KM\_BIO\_SRC\_IRQ,  
    KM\_BIO\_DST\_IRQ,  
    KM\_PTE0,  
    KM\_PTE1,  
    KM\_IRQ0,  
    KM\_IRQ1,  
    KM\_SOFTIRQ0,  
    KM\_SOFTIRQ1,  
    KM\_TYPE\_NR  
};  
  
所有固定映射的固定线性地址  
enum fixed\_addresses {  
    FIX\_HOLE,  
    FIX\_VSYSCALL,  
        ....  
#ifdef CONFIG\_HIGHMEM  
    FIX\_KMAP\_BEGIN,    /\* reserved pte's for temporary kernel mappings \*/  
    FIX\_KMAP\_END = FIX\_KMAP\_BEGIN+(KM\_TYPE\_NR\*NR\_CPUS)-1,  
#endif  
        .......  
    \_\_end\_of\_permanent\_fixed\_addresses,  
    /\* temporary boot-time mappings, used before ioremap() is functional \*/  
#define NR\_FIX\_BTMAPS    16  
    FIX\_BTMAP\_END = \_\_end\_of\_permanent\_fixed\_addresses,  
    FIX\_BTMAP\_BEGIN = FIX\_BTMAP\_END + NR\_FIX\_BTMAPS - 1,  
    FIX\_WP\_TEST,  
    \_\_end\_of\_fixed\_addresses  
};  
  
3.3 注意 fixed\_addresses 的地址从上至下是倒着的，FIX\_HOLE的地址等于 0xfffff000，是一个洞  
#define \_\_fix\_to\_virt(x)    (FIXADDR\_TOP - ((x) << PAGE\_SHIFT))  
#define \_\_FIXADDR\_TOP    0xfffff000

－－－－－－－－－－－－－－－－－－－－－－－－－

VMALLOC\_RESERVE和896M  
  
LINUX 内核虚拟地址空间到物理地址空间一般是固定连续影射的。  
  
假定机器内存为512M，  
从3G开始，到3G + 512M 为连续固定影射区。zone\_dma, zone\_normal为这个区域的。固定影射的VADDR可以直接使用（get a free page, then use pfn\_to\_virt()等宏定义转换得到vaddr）或用kmalloc等分配. 这样的vaddr的物理页是连续的。得到的地址也一定在固定影射区域内。  
  
如果内存紧张，连续区域无法满足，调用vmalloc分配是必须的，因为它可以将物理不连续的空间组合后分配，所以更能满足分配要求。vmalloc可以映射高端页框，也可以映射底端页框。vmalloc的作用只是为了提供逻辑上连续的地址。。。  
  
但vmalloc分配的vaddr一定不能与固定影射区域的vaddr重合。因为vaddr到物理页的影射同时只能唯一。所以vmalloc得到的 vaddr要在3G + 512m 以上才可以。也就是从VMALLOC\_START开始分配。 VMALLOC\_START比连续固定影射区大最大vaddr地址还多8-16M（2\*VMALLOC\_OFFSET)--有个鬼公式在  
  
#define VMALLOC\_OFFSET   8\*1024  
#define VMALLOC\_START   (high\_memory - 2\*VMALLOC\_OFFSET) & ~(VMALLOC\_OFFSET-1)  
  
high\_memory 就是固定影射区域最高处。  
  
空开8-16M做什么？ 为了捕获越界的mm\_fault.  
同样，vmalloc每次得到的VADDR空间中间要留一个PAGE的空（空洞），目的和上面的空开一样。你vmalloc(100)2次，得到的2个地址中间相距8K。  
如果连续分配无空洞，那么比如  
p1=vmalloc(4096)；  
p2=vmalloc(4096)；  
如果p1使用越界到p2中了，也不会mm\_falut. 那不容易debug.  
  
下面说明VMALLOC\_RESERVE和896M的问题。  
  
上面假设机器物理512M的case. 如果机器有1G物理内存如何是好？那vmalloc()的vaddr是不是要在3G + 1G + 8M 空洞以上分配？超过寻址空间了吗。  
这时，4G 下面保留的VMALLOC\_RESERVE 128m 就派上用场了。  
也就是说如果物理内存超过896M, high\_memory也只能在3G + 896地方。可寻址空间最高处要保留VMALLOC\_RESREVE 128M给vmalloc用。  
  
所以这128M的VADDR空间是为了vmalloc在物理超过了896M时候使用。如果物理仅仅有512M, 一般使用不到。因为VMALLOC\_START很低了。如果vmalloc太多了才会用到。  
  
high\_memory在arch/i386/kernel, mm的初始化中设置。根据物理内存大小和VMALLOC\_RESERVE得到数值.  
  
所以说那128M的内核线性地址仅仅是为了影射1G以上的物理内存的不对的。如果物理内存2G,1G以下的vmalloc也用那空间影射。总之，内核的高端线性地址是为了访问内核固定映射以外的内存资源  
  
看vmalloc分配的东西可以用

show\_vmalloc()  
{  
struct vm\_struct \*\*p, \*tmp;  
  
for(p = &vmlist; (tmp = \*p); p = &tmp->next) {  
   printk("%p %p %d\n", tmp, tmp->addr, tmp->size  
  
}  
}

用户空间当然可以使用高端内存，而且是正常的使用，内核在分配那些不经常使用的内存时，都用高端内存空间（如果有），所谓不经常使用是相对来说的，比如内核的一些数据结构就属于经常使用的，而用户的一些数据就属于不经常使用的。  
  
用户在启动一个应用程序时，是需要内存的，而每个应用程序都有3G的线性地址，给这些地址映射页表时就可以直接使用高端内存。  
  
而且还要纠正一点的是：那128M线性地址不仅仅是用在这些地方的，如果你要加载一个设备，而这个设备需要映射其内存到内核中，它也需要使用这段线性地址空间来完成，否则内核就不能访问设备上的内存空间了。  
  
总之，内核的高端线性地址是为了访问内核固定映射以外的内存资源  
  
实际上高端内存是针对内核一段特殊的线性空间提出的概念，和实际的物理内存是两码事。进程在使用内存时，触发缺页异常，具体将哪些物理页映射给用户进程是内核考虑的事情。在用户空间中没有高端内存这个概念。