linux内存管理之kmallocLinux认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/644/2021_2022_linux_E5_86_85_E5_AD_c103_644864.htm 在设备驱动程序中动态开辟内存，不是用malloc，而是kmalloc，或者用get_free_pages直接申请页。释放内存用的是kfree，或free_pages. 对于提供了MMU（存储管理器，辅助操作系统进行内存管理，提供虚实地址转换等硬件支持）的处理器而言，Linux提供了复杂的存储管理系统，使得进程所能访问的内存达到4GB。 进程的4GB内存空间被人为的分为两个部分--用户空间与内核空间。用户空间地址分布从0到3GB(PAGE_OFFSET，在0x86中它等于0xC0000000)，3GB到4GB为内核空间。 内核空间中，从3G到vmalloc_start这段地址是物理内存映射区域（该区域中包含了内核镜像、物理页框表mem_map等等），比如我们使 用的VMware虚拟系统内存是160M，那么3G～3G 160M这片内存就应该映射物理内存。在物理内存映射区之后，就是vmalloc区域。对于 160M的系统而言，vmalloc_start位置应在3G 160M附近（在物理内存映射区与vmalloc_start期间还存在一个8M的gap 来防止跃界），vmalloc_end的位置接近4G(最后位置系统会保留一片128k大小的区域用于专用页面映射) kmalloc和get_free_page申请的内存位于物理内存映射区域，而且在物理上也是连续的，它们与真实的物理地址只有一个固定的偏移，因此存在较简单的转换关系，virt_to_phys()可以实现内核虚拟地址转化为物理地址： #define __pa(x) ((unsigned long)(x)-PAGE_OFFSET) extern inline unsigned long virt_to_phys(volatile void * address) { return __pa(address). } 上面

转换过程是将虚拟地址减去3G（PAGE_OFFSET=0XC000000）。 与之对应的函数为phys_to_virt()，将内核物理地址转化为虚拟地址： #define __va(x) ((void *)((unsigned long)(x) PAGE_OFFSET)) extern inline void * phys_to_virt(unsigned long address) { return __va(address). } virt_to_phys()和phys_to_virt()都定义在include\asm-i386\io.h中。

---------- 1、kmalloc() 分配连续的物理地址，用于小内存分配。 2、__get_free_page() 分配连续的物理地址，用于整页分配。 至于为什么说以上函数分配的是连续的物理地址和返回的到底是物理地址还是虚拟地址，下面的记录会做出解释。 kmalloc() 函数本身是基于 slab 实现的。slab 是为分配小内存提供的一种高效机制。但 slab 这种分配机制又不是独立的，它本身也是在页分配器的基础上来划分更细粒度的内存供调用者使用。也就是说系统先用页分配器分配以页为最小单位的连续物理地址，然后 kmalloc() 再在这上面根据调用者的需要进行切分。 关于以上论述，我们可以查看 kmalloc() 的实现，kmalloc()函数的实现是在 __do_kmalloc() 中，可以看到在 __do_kmalloc()代码里最终调用了 __cache_alloc() 来分配一个 slab，其实 kmem_cache_alloc() 等函数的实现也是调用了这个函数来分配新的 slab。我们按照 __cache_alloc()函数的调用路径一直跟踪下去会发现在 cache_grow() 函数中使用了 kmem_getpages()函数来分配一个物理页面，kmem_getpages() 函数中调用的alloc_pages_node() 最终是使用 __alloc_pages() 来返回一个struct page 结构，而这个结构正是系统用来描述物理页面的。这样也就证实了上面所说的，slab 是在物理页面基

础上实现的。kmalloc() 分配的是物理地址。