# 内核中的内存申请:kmalloc、vmalloc、kzalloc、kcalloc、get\_free\_pages

在内核模块中申请分配内存需要使用内核中的专用API:kmalloc、vmalloc、kzalloc、kcalloc、get\_free\_pages;当然,设备驱动程序也不例外;

对于提供了MMU功能的处理器而言,Linux提供了复杂的内存管理系统,使得进程所能访问到的地址空间可以达到4GB;而这4GB的空间又被划分为两个部分:0GB~3GB(PAGE\_OFFSET,x86中的值是0xC0000000)的区域被用作进程的用户空间,3GB~4GB的区域被用作内核空间;

在内核空间中,从3GB到vmalloc\_start之间的这段地址区域作为物理内存映射区使用,该段映射区域内包含了内核镜像、物理页框表mem\_map等等,比如,我们使用的系统物理内存为160MB,那么,3GB~3GB+vmalloc\_start之间的区域就应该是映射的物理内存;在物理内存映射区域之后,就是虚拟内存vmalloc区域;对于160MB的系统而言,vmalloc\_start的位置就应该在3GB+160MB位置附近(在物理内存映射区与vmalloc\_start位置之间还存在一个8M的gap来防止越界),vmalloc\_end的位置接近4GB的位置(系统会在最后的位置处保留一片128KB大小的区域专用于页面映射);

## 一、kmalloc

#include <linux/slab.h>

static inline void \*kmalloc(size\_t size, gfp\_t flags);

参数:size:指定要分配的块的大小,单位是字节;flags:指定分配内存时的控制方式;

该函数用于在内核空间中分配内存使用,它的返回速度快(除非被阻塞),并且对其分配的内存不进行任何初始化(清零)操作,分配的内存区域仍然保留有他原有的内容;

kmalloc申请得到的是物理内存,位于物理内存映射区,而且在物理地址上是连续的;但是kmalloc返回的内存地址却是虚拟地址(线性地址),返回的这个虚拟地址(线性地址)与真实的物理地址之间仅仅相差一个固定的偏移值;因此,kmalloc申请得到的物理内存块的首地址与其返回的虚拟地址之间存在着比较简单的转换关系;通过内核提供的函数virt\_to\_phys()可以实现该虚拟地址到真实的内核物理地址之间的转换:

#define \_\_pa(x) ((unsigned long)(x)-PAGE\_OFFSET)

static inline unsigned long virt\_to\_phys(volatile void\* address)

{

return \_\_pa(address);

}

参数address是kmalloc返回的一个虚拟地址;该转换过程就是虚拟地址减去3GB(PAGE\_OFFSET=0xC0000000);

一般情况下,PAGE\_OFFSET=3\*1024\*1024\*1024=0xC0000000(3G);

与之对应的函数就是phys\_to\_virt()用于把内核物理地址转换为虚拟地址:

#define \_\_va(x) ((void \*)((unsigned long)(x)+PAGE\_OFFSET))

static inline void \* phys\_to\_virt(unsigned long address)

{

return \_\_va(address);

}

这两个函数都定义在include/asm-i386/io.h中;

kmalloc()函数用于小块内存的申请,最小可以申请的内存是32字节或64字节,最大可以申请的内存是128KB-16,其中,被减掉的16个字节用于存储页描述符结构;这些都依赖于体系架构所使用的页面大小;kmalloc申请的内存在物理地址上是连续的,这对于要进行DMA传输的设备来说,是非常重要的;

kmalloc()的内存分配是基于slab机制实现的,slab机制是为分配小内存而提供的一种高效的机制;但是slab机制也不是独立的,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,系统先使用页分配器分配以页为最小单位的连续物理地址,然后,kmalloc()再在这个基础上根据调用者的需要进行切分的;另外,slab机制分配的内存在物理地址和虚拟地址(线性地址/逻辑地址)上都是连续的;

对于kmalloc()申请的内存,需要使用kfree()函数来释放;

备注:kmalloc是基于slab机制实现的;

## 二、get\_free\_pages

#include <asm/pages.h>

fastcall unsigned long \_\_get\_free\_pages(gfp\_t gfp\_mask, unsigned int order)

{

struct page \* page;

page = alloc\_pages(gfp\_mask, order);

if (!page)

return 0;

return (unsigned long) page\_address(page);

}

参数gfp\_mask用于指定申请内存时的控制方式,order用于指定申请的页数;它申请的内存位于(PAGE\_OFFSET,HIGH\_MEMORY)之间;

\_\_get\_free\_pages()函数是页面分配器提供给调用者的最底层的内存分配函数,它申请的内存也是连续的物理内存,同样位于物理内存映射区;它是基于buddy机制实现的;在使用buddy机制实现的物理内存管理系统中,最小的分配粒度(单位)也是以页为单位的;在\_\_get\_free\_pages()内部通过调用alloc\_pages()来分配物理内存页;

\_\_get\_free\_page()函数分配的是连续的物理内存,处理的是连续的物理地址,但是返回的也是虚拟地址(线性地址);如果想要得到正确的物理地址,也需要使用virt\_to\_phys()可进行转换;

对于\_\_get\_free\_pages()函数申请的内存,需要使用\_\_free\_pages()函数来释放;

备注:\_\_get\_free\_pages是基于buddy机制实现的;

## 三、vmalloc

#include <linux/vmalloc.h>

void\* vmalloc(unsigned long size)

{

return \_\_vmalloc(size, GFP\_KERNEL | \_\_GFP\_HIGHMEM, PAGE\_KERNEL);

}

void\* \_\_vmalloc(unsigned long size, gfp\_t gfp\_mask, pgprot\_t prot)

{

return \_\_vmalloc\_node(size, gfp\_mask, prot, -1);

}

void\* \_\_vmalloc\_node(unsigned long size, gfp\_t gfp\_mask, pgprot\_t prot, int node)

{

struct vm\_struct \*area;

size = PAGE\_ALIGN(size);

if(!size || (size >> PAGE\_SHIFT) > num\_physpages)

return NULL;

area = get\_vm\_area\_node(size, VM\_ALLOC, node);

if(!area)

return NULL;

return \_\_vmalloc\_area\_node(area, gfp\_mask, prot, node);

}

void\* \_\_vmalloc\_area\_node(struct vm\_struct\* area, gfp\_t gfp\_mask, pgprot\_t prot, int node);

void\* \_\_vmalloc\_area(struct vm\_struct\* area, gfp\_t gfp\_mask, pgprot\_t prot)

{

return \_\_vmalloc\_area\_node(area, gfp\_mask, prot, -1);

}

vmalloc()函数也是用于申请内存的,但是它申请的内存是位于vmalloc\_start到vmalloc\_end之间的虚拟内存;它申请的内存在虚拟地址(线性地址/逻辑地址)上是连续的,但是并不要求在物理地址上连续,并且返回的地址与物理地址之间没有简单的转换关系;

vmalloc()函数适用于大块内存的申请环境中;但是它申请的内存不能直接用于DMA传输;因为DMA传输需要使用物理地址连续的内存块;

对于vmalloc()申请的内存,需要使用vfree()函数来释放;

备注:vmalloc是基于slab机制实现的;

## 四、比较

1).kmalloc/\_\_get\_free\_pages申请的内存块都在物理内存映射区,即在(PAGE\_OFFSET,HIGH\_MEMORY)之间,处理的都是物理地址,且保证在物理地址空间上是连续的;二者返回的都是虚拟地址,如果需要得到正确的物理地址,需要使用virt\_to\_phys()进行转换;但是,kmalloc和vmalloc都是以字节为单位进行申请,而\_\_get\_free\_pages()则是以页为单位进行申请;

2).vmalloc函数申请的内存块位于虚拟内存映射区,即在(VMALLOC\_START,VMALLOC\_END)之间,处理的都是虚拟内存,且保证在虚拟地址空间上是连续的,但是在物理地址空间上不要求连续;一般作为交换区、模块的内存使用;

3).kmalloc和vmalloc都是基于slab机制实现的,但是kmalloc的速度比vmalloc的速度快;\_\_get\_free\_pages是基于buddy机制实现的,速度也较快;

4).kmalloc用于小块内存的申请,通常,一次所能申请的内存块的大小在(32/64字节,128KB-16)之间;而vmalloc可以用于分配大块内存的场合;

5).kmalloc申请的内存块在物理地址空间上是连续的,所以它申请的内存块可以直接用于DMA传输;vmalloc申请的内存块在虚拟地址空间上连续,但是在物理地址空间上不要求连续,所以它申请的内存块不能直接用于DMA传输;

6).kmalloc申请的内存块用kfree释放;vmalloc申请的内存块用vfree释放;\_\_get\_free\_pages申请的内存页用\_\_free\_pages释放;

7).kmalloc申请得到的地址称为内核逻辑地址,vmalloc申请得到的地址称为内核虚拟地址;

## 五、其它函数

1).static inline void \*kzalloc(size\_t size, gfp\_t flags);

该函数比kmalloc多了一个功能,就是会把申请得到的内存块初始化为0;

2).static inline void\* kcalloc(size\_t n, size\_t size, gfp\_t flags)

{

if(n != 0 && size > ULONG\_MAX / n)

return NULL;

return kzalloc(n \* size, flags);

}

该函数用于申请一个数组的内存空间,并把申请得到的内存都初始化为0;

## 六、GFP标记

kmalloc、kzalloc、kcalloc、vmalloc、get\_free\_pages函数在调用时都有一个gfp\_t类型的控制标记flags;这个标记用于控制申请内存时的内存分配控制方式; #include <linux/gfp.h>

GFP的标记有两种:带双下划线前缀的和不带双下划线前缀的;

不带双下划线前缀的GFP标志:

GFP\_ATOMIC:用于在中断上下文和进程上下文之外的其它代码中分配内存;从不睡眠;

GFP\_KERNEL:内核正常分配内存;可能睡眠;

GFP\_USER :用于为用户空间页分配内存;可能睡眠;

GFP\_HIGHUSER:如同GFP\_USER,但它是从高端内存中申请;

GFP\_NOIO和GFP\_NOFS:功能如同GFP\_KERNEL,但是它俩增加限制到内核能做的来满足请求;GFP\_NOFS分配不允许进行任何文件系统调用,而GFP\_NOIO分配根本不允许进行任何IO初始化;它俩主要用于文件系统和虚拟内存代码,那里允许一个分配睡眠,但是递归的文件系统调用会是个坏主意;

带有双下划线前缀的GFP标志:

\_\_GFP\_DMA:这个标志要求分配的内存在能够进行DMA的内存区;平台依赖的;

\_\_GFP\_HIGHMEM:这个标志指示分配的内存可以位于高端内存区;平台依赖的;

\_\_GFP\_COLD:正常地,内存分配器尽力返回"缓冲热"的页---可能在处理器缓冲中找到的页;相反,这个标志请求一个"冷"页---在一段时间内没被使用的页;它对分配页做DMA读是很有用的,此时在处理器缓冲中出现是没用的;

\_\_GFP\_NOWARN:这个标志用于分配内存时阻止内核发出警告,当一个分配请求无法满足时;

\_\_GFP\_HIGH:这个标志标识了一个高优先级请求,它被允许来消耗甚至被内核保留给紧急状况的最后的内存页;

\_\_GFP\_REPEAT:分配器的动作;当分配器有困难满足一个分配请求时,通过重复尝试的方式来"尽力尝试",但是分配操作仍然有可能失败;

\_\_GFP\_NOFAIL:分配器的动作;当分配器有困难满足一个分配请求时,这个标志告诉分配器不要失败,尽最大努力来满足分配请求;

\_\_GFP\_NORETRY:分配器的动作;当分配器有困难满足一个分配请求时,这个标志告诉分配器立即放弃,不再做任何尝试;

通常,一个或多个带双下划线前缀的标记相或,即可得到对应的不带双下划线前缀的标记;

最常用的标记就是GFP\_KERNEL,它的意思就是当前的这个分配代表运行在内核空间的进程而进行的;换句话说,这意味着调用函数是代表一个进程在执行一个系统调用;使用GFP\_KERNEL标记,就意味着kmalloc能够使当前进程在少内存的情况下通过睡眠来等待一个内存页;因此,一个使用GFP\_KERNEL的函数必须是可重入的,且不能在原子上下文中运行;当当前进程睡眠,内核采取正确的动作来定位一些空闲的内存页,或者通过刷新缓存到磁盘或者交换出去一个用户进程的内存页;

如果一个内存分配动作发生在中断处理或内核定时器的上下文中时,当前进程就不能被设置为睡眠,也就不能再使用GFP\_KERNEL标志了,此时应该使用GFP\_ATOMIC标志来代替;正常地,内核试图保持一些空闲页以便来满足原子的分配;当使用GFP\_ATOMIC标志时,kmalloc标志能够使用甚至最后一个空闲页;如果这最后一个空闲页不存在,那分配就会失败;