# 关于存储管理中的一些概念

**前言**

在编写程序的时候，在学习操作系统以及编写驱动的时候，尤其是在Linux内核空间中编程的时候，经常会被一些与存储相关的概念所困扰，而这也经常是我们程序出现错误概率很大的一个原因(指针相关的错误)。

我们经常遇到的问题，例如：什么是页？什么是段？什么是扇区？什么是块？什么是簇？什么是磁道？什么是物理地址？什么是线性地址？什么是虚拟地址？什么是逻辑地址？它们之间究竟有什么关系？……这些问题，这里暂时归结为存储管理中涉及到的问题，而存储管理，又可分为内存管理，外存管理。本文通过对外存、内存的管理的简单叙述，尝试达到理清对这些概念以及它们之间的关系的简单理解。更为具体的内容，可以参见列出的参考资料，或者其他更好的资料。

**主要内容：**

**一、非易失存储（例如磁盘）的管理**

**二、易失性存储(例如内存)的管理**

**三、其他**

**一、非易失存储（例如磁盘）的管理**

**===========================================**

**1、关于磁盘物理结构与寻址**

关键之处在于关于磁盘物理结构，首先理解了磁盘的柱面（磁道），扇区，以及盘面之后，再知道如下信息，就掌握了关于磁盘寻址基本的知识。

传统的大致情况就是，一个盘面上面有多个磁道（每个磁道就是一圈，这些磁道组成盘面上的同心圆），一个磁道上面有多个扇区（就是一个同心圆上面的一个弧线部分，每个扇区实际物理大小和硬件相关，但是内核内部默认和驱动交互时候采用扇区是512字节的逻辑扇区，所以物理扇区一定是512字节的整数倍），而多个盘片上的同一位置的磁道组成的圆柱就是柱面。综上，寻址磁盘，可以通过“(盘片，磁道，扇区)”达到目的，而这样的磁盘的大小为：盘片数\*每盘片上的磁道数\*每磁道上的扇区数\*每扇区的字节数。

另外，磁盘的分区是以磁道为边界的，所以如果只有2个磁道，因此最多只能创建2个分区。

传统的磁盘使用8个位表示盘面数、6个位表示每磁道扇区数、10个位表示磁道数，因此盘面、每磁道扇区、磁道的最大数值分别为255、63和1023。这也是传说中启动操作系统时的1024柱面(磁道)和硬盘容量8G限制的根源。

现代磁盘采用线性寻址方式突破了这一限制，从本质上说，如果你的机器还没生锈，那么你的硬盘无论是内部结构还是访问方式都与常识中的盘面、每磁道扇区、磁道无关。但为了与原先的理解兼容，对于现代磁盘，我们在访问时还是假设它具有传统的结构。目前比较通用的假设是：所有磁盘具有最大数目的(也就是恒定的)盘面和每磁道扇区数，而磁盘大小与磁道数与成正比。

因此，对于一块80G的硬盘，根据假设，这块磁盘的盘面和每磁道扇区数肯定是255和63，磁道数为：80\*1024\*1024\*1024/512(字节每扇区)/255(盘面数)/63(每磁道扇区数)=10043(小数部分看作不完整的磁道被丢弃)。 假设写磁盘驱动程序中我们指定了磁盘大小为16M，共包含16\*1024\*1024/512=32768个扇区。假设这块磁盘具有最大盘面和每磁道扇区数后，那么它的磁道数就是：32768/255/63=2。

**2、关于扇区(sector)、块(block)和簇(cluster)**

扇区是硬件的磁盘的最小存储单位，而块是文件系统中数据存储的最小单元。这里，文件系统是用来规范数据文件在磁盘上以什么方式进行存储的，以便操作系统可以通过文件系统中定义好的规范，访问到磁盘上的文件。

一个磁盘扇区一般512个字节（现在有4K的了）, 磁盘块应该是类似FAT的簇大小的概念，是操作系统中分配磁盘容量的最小单位了，一般是512B\*2^n。扇区是硬件上的单位，块一般是针对上层的，块一般要比扇区大。有些地方的说法，块和扇区都无什么区别了，关心逻辑和物理的就行了。也就是说，设备驱动的相关结构中，有两个地方，一个表示物理的扇区，一个表示逻辑扇区；物理的扇区就是实际物理扇区的大小，为512字节的整数倍；而逻辑扇区，就是512字节，操作系统认为所有扇区就是512字节，使用统一的逻辑扇区大小做为操作单位，和驱动进行交互，简化了写驱动的繁琐；而具体内部是如何转化两者之间关系的，写驱动的时候不用关心，我们只要告诉物理扇区大小，逻辑扇区大小，然后在驱动里面使用逻辑扇区（512字节）就行了。

而对于簇，在fat文件系统中，fat上面簇是多个磁道，当然不同的文件系统有所不同。

**二、易失性存储(例如内存)的管理**

**===========================================**

**1、关于实模式和保护模式**

说到内存管理，就不能不提到实模式和保护模式。处理器的两种工作方式：保护模式和实模式。早期的dos就是运行在实模式下，而现在的windows则运行在保护模式下。实模式使用的逻辑地址直接转换成物理地址，只能访问1M多一点的内存空间，在拥有32根地址线的cpu中访问1M以上的空间则变得很困难。为了满足计算机对资源（存储资源和cpu资源等等）的管理，由此产生了新的管理方式--保护模式。

80386及以上的处理器功能要大大超过其先前的处理器，但只有在保护模式下，处理器才能发挥作用。在保护模式下，全部32根地址线有效，可寻址4G的物理地址空间；扩充的存储分段机制和可选的存储器分页机制，不仅为存储器共享和保护提供了硬件支持，而且为实现虚拟存储器提供了硬件支持；支持多任务；4个特权级和完善的特权级检查机制，实现了数据的安全和保密。计算机启动后首先进入的就是实模式，通过设置相应的寄存器才能进入保护模式。

**2、关于页(page)和段(segment)**

和扇区和块等一般是针对于非易失存储介质（如磁盘）不同，段和页的概念一般是对于易失性存储而言的。也就是主要体现在内存访问方式上的存储方式。从逻辑地址到线性地址的转换由80386分段机制管理，分页机制是在段机制之后进行的，它进一步将线性地址转换为物理地址。

(1)段是信息的逻辑单位。

分段的作业地址空间是二维的，程序员在标识一个地址时，既需给出段名，又需给出段内地址。细言之：段式 分段由用户设计划分，每段对应一个相应的的程序模块，有完整的逻辑意义。分段的目的是为了能更好的满足用户的需要。

(2)页是信息的物理单位。

分页的作业地址空间是维一的，即单一的线性空间，程序员只须利用一个记忆符，即可表示一地址。分页的目的是为实现离散分配方式，以消减内存的外零头，提高内存的利用率；或者说，分页仅仅是由于系统管理的需要，而不是用户的需要。

(3)页和段的大小。

页的大小固定且由系统确定，把逻辑地址划分为页号和页内地址两部分，是由机器硬件实现的，因而一个系统只能有一种大小的页面。段的长度却不固定，决定于用户所编写的程序，通常由编辑程序在对源程序进行编辑时，根据信息的性质来划分。

**3、分页机制和分段机制**

处理器在得到逻辑地址后首先通过分段机制转换为线性地址，线性地址再通过分页机制转换为物理地址最后读取数据。分段机制是必须的，分页机制是可选的，当不使用分页的时候线性地址将直接映射为物理地址，设立分页机制的目的主要是为了实现虚拟存储。

**4、分段机制和逻辑地址**

分段机制中，将逻辑地址转换成线性地址的细节就省略了，总的思想就是首先通过段选择子在描述符表中找到相应段的描述符，根据描述符中的段基址首先确定段的位置，再通过OFFSET加上段基址计算出线性地址。进一步解释，一个任务会涉及多个段(代码段，数据段……)，每个段需要一个描述符来描述，为了便于组织管理，80386及以后处理器把描述符组织成表，即描述符表。逻辑地址结构形式一般为:"seg:offset"形式，用描述表中记录的段基址加上逻辑地址“sel:offset“中的“offset“部分，即转换成线性地址。

**5、分页机制和线性地址**

通过分段机制，将逻辑地址转换成的线性地址，简单的说就是0000000h~ffffffffh(即0~4G)的线性结构，是32个bite位能表示的一段连续的地址，但它是一个概念上的、抽象的地址，并不存在在现实之中。线性地址地址主要是为分页机制而产生的。

分页机制是在段机制之后进行的，它进一步将线性地址转换为物理地址。80386使用4K字节大小的页，且每页的起始地址都被4K整除；因此，80386把4GB字节的“线性地址”空间划分为1M个页面，采用了两级页表结构进行转换。具体转换过程也省略了，大致如下：

(1)第一级表称为页目录，存储在一个4K字节的页中，每个表项为4个字节，线性地址最高的10位（22-31）对第一级表进行索引，索引得到的表项内容定位了二级表中的一个表的地址（即下级页表所在的内存块号）。

(2)第二级表称为页表，存储在一个4K字节页中，包含了1K字节的表项，线性地址的中间10位（12-21）位对二级页表进行索引，索引得到的表项包含了一个页的物理地址。

(3)页物理地址的高20位与线性地址的低12位形成最后的物理地址。

**6、Linux内核中的内存管理**

在Linux内核中，内存分为内核空间和用户空间。

(1)用户空间

在Linux中，每个用户进程都可以访问4GB的线性虚拟内存空间。其中从0到3GB的虚存地址是用户空间，用户进程可以直接访问。

(2)内核空间

从3GB到4GB的虚存地址为内核态空间，存放供内核访问的代码和数据，用户态进程不能访问。所有进程从3GB到4GB的虚拟空间都是一样的，linux以此方式让内核态进程共享代码段和数据段。

(3)内核虚拟地址，用户空间虚拟地址

这里的虚拟地址，实际上就是分页机制用来转化成物理地址的线性地址。讲述这里的时候，使用下面三个地址描述内存：

(a)物理地址(phyaddr):对应真实的内存.

(b)内核虚拟地址(kervir):内核的虚地址空间(3g-4g)，例如\_get\_free\_pages等就是从这里分配。

(c)用户虚拟地址(usrvir):用户的虚拟空间地址(0g-3g).例如malloc等返回的就是这里的地址。

(4)内存地址空间之间的转换：

phyaddr<->kervir:有类似\_pa, \_va这样的宏。

kervir->usrvir:有类似remap\_pfn\_range这样的函数，一般在驱动里面调用，返回内核地址给用户空间。

phyaddr->usrvir:知道phyaddr的基地址，与usrvir的基地址，然后计算偏移量即可。

(5)关于分配内存：

(a)内核空间内存分配

\_get\_free\_pages:连续物理地址，且连续最大页为2^PAGE\_SHIFT\*2^MAXORDER，宏可以配置。

kmalloc:连续物理地址，不过分配的空间太小了，只有128k,也有一个可以配置的宏。

vmalloc:分配的地址空间物理上不连续。

想要知道更具体的信息，内核源代码中的kmalloc.h/c, kmalloc\_size.h/c, slab.h/c等文件会有助于了解。

(b)用户空间内存分配

malloc:从用户堆中动态分配内存。

**三、其他**

**===========================================**

这里，是一些补充性的内容。

**1、块设备的bio**

编写块设备驱动，最终用户请求的数据（读或者写）都会通过bio这个结构反应出来，也就是说，bio代表一次请求。这里就用到了page。对于page，一般各种cpu操作page的最小单位是4k，当然有的设成8k等，但是最小是4k。

当块设备请求到来的时候，会为用户请求数据分配一块虚拟地址，存放在请求结构(request)中的bio结构中，而bio结构中的bi\_io\_vec数组存放实际的数据。数组元素为：

struct bio\_vec

{

struct page\* bv\_page;

unsigned int bv\_len;

unsigned int bv\_offset;

}

实际上，分配给用户请求数据的虚拟地址不一定以page进行对齐，所以要对其align，如下：

|--###|#####|#####|##---|

这里，分配了4个页给用户请求数据，这四个页都存放在一个bio\_vec中的bv\_page列表中。而由于需要align，所以'#'中的才是实际>的数据，而'-'的可能是别人的或者没有用的数据等。这里，bv\_offset就是第一个bv\_page中第一个'#'中的偏移，而bv\_len就是从第1个'#'到最后一个'#'的长度。这一点要注意，不要从bv\_page开始的页对应的虚拟地址访问page。

获取一个page对应的起始地址方式是使用page\_address宏，这样返回page的起始地址，再加上bv\_offset就得到整个bio结构中数据的起始地址了。获取bio数据对应的虚拟地址的函数的实现就是如下：

//include/linux/bio.h

static inline void \*bio\_data(struct bio \*bio)

{

if (bio->bi\_vcnt)

return page\_address(bio\_page(bio)) + bio\_offset(bio);

return NULL;

}

可知，通过bio\_data就可以获得bio数据的虚拟地址了，通过内核代码发现，这个虚拟地址只是bio数据的当前vec索引地址，而不一>定是整个的。

**2、通过io映射将外设映射到内存空间**

我们使用ioremap来将外设的空间映射到内存空间，借以访问外设，而这里所映射得到的就是物理地址，物理地址是一个固定的常量，而不是我们以为的随意的一个地址。

以上，是本人根据理解，综合所参考的资料，书上所学，以及工作时候的时间，所做的总结。尽量只用文字的形式描述，只通过文本文件便可以学习。如其中有不准或者更好的建议，可以联系我，谢谢！

参考资料：

http://bbs.chinaunix.net/thread-2017377-2-1.html

http://bbs.chinaunix.net/thread-2083672-1-1.html#

http://www.cnblogs.com/c1230v/articles/1432525.html

http://hi.baidu.com/jingxshi/blog/item/7056f612fdcead58f919b865.html

*作者：QuietHeart*

*Email:quiet\_heart000@126.com*

*日期：2011年11月17日*