# linux内存管理子系统

## 地址类型

物理地址：物理地址是指出现在CPU地址总线上的寻址物理内存的地址信号，是地址变换的最终结果。

逻辑地址：程序代码经过编译后在汇编程序中使用的地址。

线性地址：又称虚拟地址。

## 地址转换



CPU要将一个逻辑地址转换为物理地址，需要两步：首先CPU利用段式内存管理单元，将逻辑地址转换成线程地址，再利用页式内存管理单元，把线性地址最终转换为物理地址。

## 段式管理（16位CPU）

为了能够访问1M的内存空间，，CPU就就采用了内存分段的管理模式，并在CPU内部加入了段寄存器。16位CPU把1M内存空间分为若干个逻辑段，每个逻辑段的要求如下：

1、逻辑段的起始地址地址(段地址)必须必须是是16的的倍数，即最后4个二进制位必须全为0。

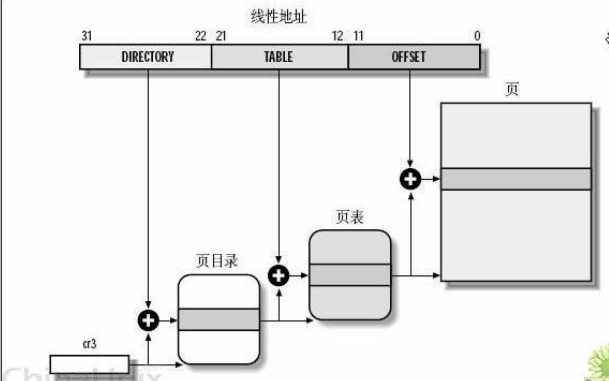
2、逻辑段的最大容量为为64K。

段地址是针对内存的分段而言的,将每一段的段首地址定义为段地址.段地址的存在是由系统的分段存储决定的,通过段地址和[偏移地址](http://baike.baidu.com/item/%E5%81%8F%E7%A7%BB%E5%9C%B0%E5%9D%80)就能对数据进行寻访。[偏移地址](http://baike.baidu.com/item/%E5%81%8F%E7%A7%BB%E5%9C%B0%E5%9D%80)(SA)是指段内相对于段起始地址的偏移值，例如一个[存储器](http://baike.baidu.com/item/%E5%AD%98%E5%82%A8%E5%99%A8)的大小是1KB,可以把它分为4段，第一段的地址范围就是0—255，第二段的地址范围就是256-511，以此类推。这些段内的[偏移地址](http://baike.baidu.com/item/%E5%81%8F%E7%A7%BB%E5%9C%B0%E5%9D%80)就是在0-255的范围内的。

而计算它们的物理地址只需要把段地址左移4位，再加上[偏移地址](http://baike.baidu.com/item/%E5%81%8F%E7%A7%BB%E5%9C%B0%E5%9D%80)就可以了。

## 分页管理

将程序的逻辑地址空间划分为固定大小的页(page)，而物理内存划分为同样大小的页框(pageframe)。程序加载时，可将任意一页放人内存中任意一个页框，这些页框不必连续，从而实现了离散分配。该方法需要CPU的硬件支持，来实现逻辑地址和物理地址之间的映射。在页式存储管理方式中地址结构由两部构成，前一部分是页号，后一部分为页内地址w（位移量），如下图所示



1、分页单元中，页目录的地址放在在CPU的cr3寄寄存器中，是进行地址转换的开始点。

2、每一个进程，都有其独立的虚拟地址空间，运行一个进程，首先需要将它的页目录地址放到到cr3寄存器中，将其他进程的保存下来。

3、每一个32位的线性地址被划分为三部份：页目录索引目录索引(10位)：页表索引(10位)：偏移(12位)

依据以下步骤进行地址转换：

1、装入进程的页目录地址（操作系统在调度进程时，把这个地址装入CR3）

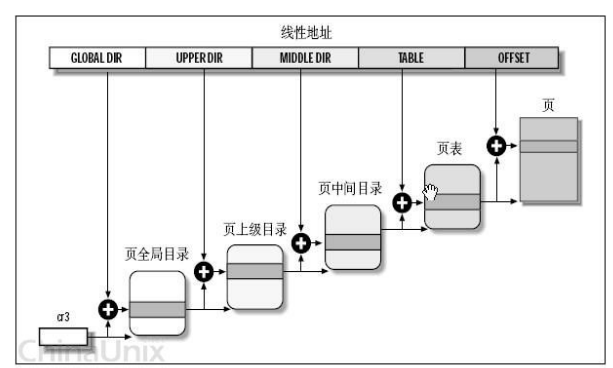
2、根据线性地址前十位，在页目录中，找到对应的索引项，页目录中的项是一个页表的地址

3、根据线性地址的中间十位，在页表中找到页的起始地址

4、将页的起始地址与线性地址的最后12位相加，得到物理地址。

Linux内核的设计并没有全部采用Intel所提供的段机制，仅仅是有限度地使用了分段机制。这不仅简化了Linux内核的设计，而且为把Linux移植到其他平台创造了条件，因为很多RISC处处理器并不支持段机制。由此可以得出，每个段的逻辑地址空间范围为0-4GB。。因为每个段的基地址为0，因此，逻辑地址与线性地址保持一致（即逻辑地址的偏移量字段的值与线性地址的值总是相同的），在Linux中所提到的逻辑地址和线性地址（虚拟地址），可以认为是一致的。看来，Linux巧妙地把段机制给绕过去了，而完全利用了分页机制。

Linux2.6.29内核为每种CPU提供统一的界面，采用了四级页管理架构，来兼容二级、三级、四级管理架构的的CPU。



## 虚拟内存

Linux操作系统采用虚拟内存管理技术，使得每个进程都有独立的进程地址空间，该空间是大小为3G，用户看到和接触的都是虚拟地址，无法看到实际的物理地址。利用这种虚拟地址不但能起到保护操作系统的作用，而且更重要的是用户程序可使用比实际物理内存更大的地址空间。Linux将4G的的虚拟地址空间划分为两个部分——用户空间与内核空间。用户空间从从0到0xbfffffff，内核空间从从3G到4G。用户进程通常情况下只能访问用户空间的虚拟地址，不能访问内核空间。例外情况是用户进程通过系统调用访问内核空间。

实际的物理内存只有当进程真的去访问新获取的虚拟地址时，才会由“请页机制”产生“缺页”异常，从而进入分配实际页框的程序。该异常是虚拟内存机制赖以存在的基本保证——它会告诉内核去为进程分配物理页，并建立对应的页表，这之后虚拟地址才实实在在地映射到了物理地址上。

### 内核内存分配

在应用程序中，常使用malloc函函数进行动态内存分配，而在Linux内核中，常使用kmalloc来动态分配内存。

kmalloc原原型是型是:

#include<linux/slab.h>

void\*kmalloc(size\_tsize,intflags)

参数：

size:要分要分配的内存大小。

flags:分配标志,它控制kmalloc的的行为。最常用的标志是GFP\_KERNEL，，它的意思是该内存分配是由运行在内核态的进程调用的。也就是说，调用它的函数属于某个

进程的，当空闲内存太少时，，kmalloc函数会使当前进程进入睡眠，等待空闲页的出现。

### 按页分配

如果模块需要分配大块的内存，那使用面向页的分配技术会更好

get\_zeroed\_page(unsignedintflags)

返回指向新页面的指针针,并将并将页面清零

\_\_get\_free\_page(unsignedintflags)

和get\_free\_page类类似，但不清零页面。

\_\_get\_free\_pages(unsignedintflags,unsignedintorder)

分配若干个连续的页面，返回指向该内存区域的指针，但也不清零这段内存区域。

### 释放

当程序用完这些页页,可以可以使用下列函数之一

来释放它们们:

voidfree\_page(unsigned long addr)

voidfree\_pages(unsignedlongaddr,unsigned long order)

\*\*如如果释放的和先前分配数目不等的页面，会导致

系统错误错误\*\*

### 内存使用流程图

