内核操作系统复习概念

**逻辑地址：**

每一个逻辑地址都由一个段或偏移量组成，偏移量指从开始的地方到实际地址的距离。

**线性地址（虚拟地址）：**

32&64位的无符号整数来表示表达地址，例如线性地址常用16进制数字表示地址范围0x00000000到0xffffffff

**物理地址：**

从电脑上微处理器对的地址引脚发送到内存总线上的电信号

内存的控制单元MMU通过分段单元的硬件把一个逻辑地址转换为虚拟地址，然后分页单元的硬件把虚拟地址转换成物理地址

**段是什么：**

段描述了一块有限的内存区域，区域的起始位置存在专门的[寄存器](https://so.csdn.net/so/search?q=%E5%AF%84%E5%AD%98%E5%99%A8&spm=1001.2101.3001.7020" \t "/home/saltfish/文档\\x/_blank)（段寄存器）中。

**段是逻辑概念**，是虚拟[地址空间](https://so.csdn.net/so/search?q=%E5%9C%B0%E5%9D%80%E7%A9%BA%E9%97%B4&spm=1001.2101.3001.7020" \t "/home/saltfish/文档\\x/_blank)的基本单位，大小不固定，取决于我们当前运行的程序，**把虚拟地址转换为线性地址**  
段的三个方面来描述：**基地址，界限，保护属性**

**硬件中的分段**

处理器有两种不同的方式进行地址转换，这两种方式分别是实模式和保护模式，

实模式主要维持早期的模型兼容

重要的是保护模式

**逻辑地址的组成：**

是由一个段标识符和一个指定端内相对地址的偏移量组成的

**段标识符：**是一个16位的字段又称段选择符

**偏移量**：是一个32位长的字段

**如何找到段选择符：**

**处理器提供了段寄存器**，段寄存器唯一目的是存放段选择符（cs,ss,ds,es,fs,gs）但程序可以把同一个段寄存器用于不同的目的，所以就先将值保存在内存中用完之后再恢复。

**有专门的用途的寄存器：**

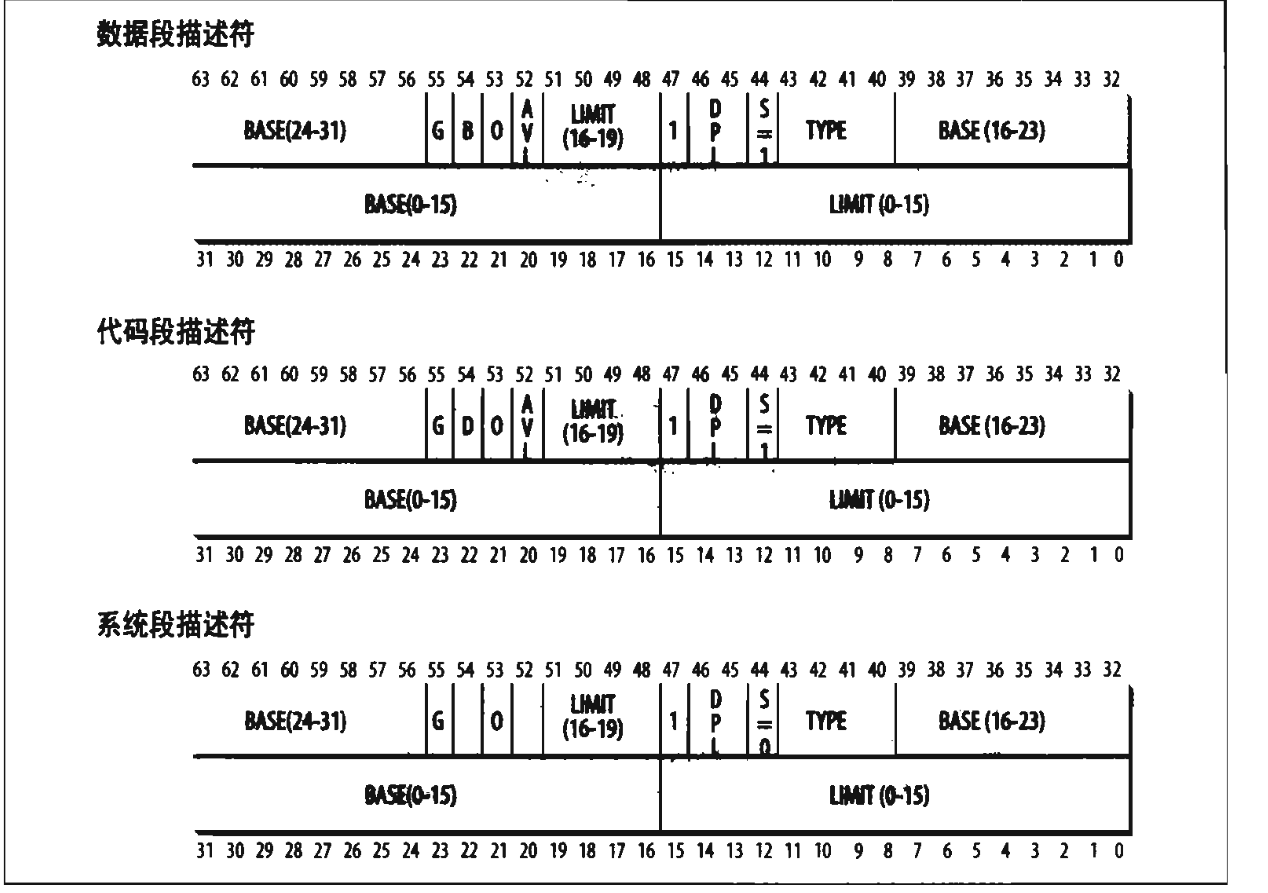
cs 代码段寄存器，包含程序指令的段，同时含有一个两位的字段指名cpu当前特权级0最高3最低用来分辨内核态和用户态

ss 栈段寄存器，指向包含当前程序栈的段

ds 数据段寄存器指向静态数据和全局数据段

计算机中每一个段都有一个段描述符，所有的段描述符放在全局描述符表GDT或者局部描述符表LDT

段描述符结构如下



**分页：**

分页是为了把线性地址转换成物理地址

线性地址被分为固定长度的组，就称为页，页内部连续的线性地址被映射到连续的物理地址中。

分页单元把所有的内存RAM分成了固定长度页框（物理页）每一个页框包含一个页，页框和页的长度一致。

分页单元处理4kb大小的页

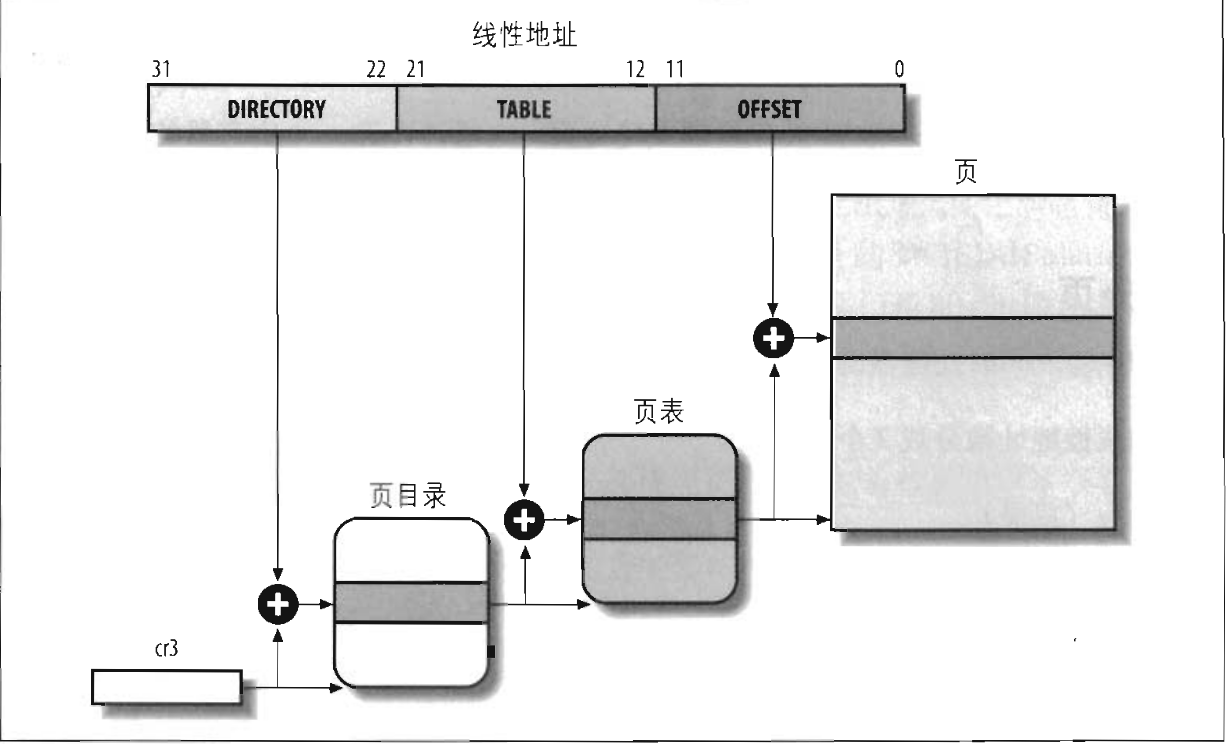
32位的虚拟地址被分成三个部分

directory（目录）10位 高

Table（页表）10位 |

offset（偏移量）12位 低

cr寄存器系列负责转换成物理地址和管理当前页目录的物理地址，虚拟地址内的页目录决定了目录项，目录项指向合适的页表，页表字段决定页表中的表项，表项中有页所在的页框物理地址。

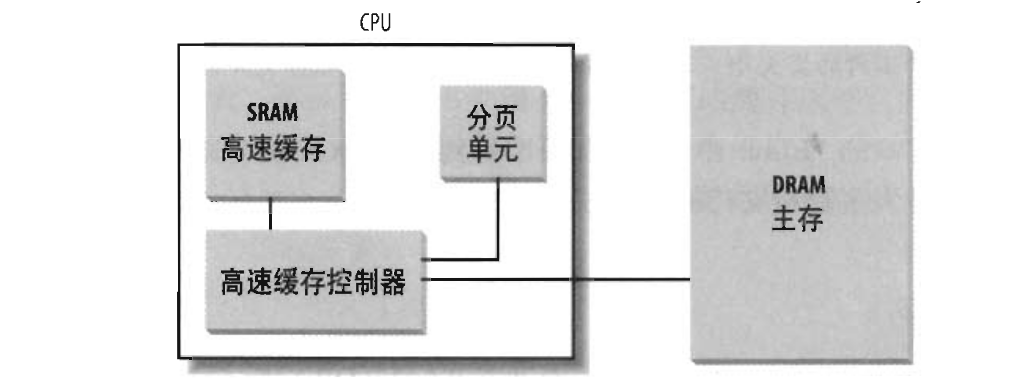


因为偏移量的字段决定页框内的相对位置，又是12位长，所以每一页就是4K

**硬件高速缓存**

动态的·RAM芯片存取时间要比时钟周期慢个几百倍，相当于操作RAM存放结果这样的指令cpu会等待很长时间，所以才有了硬件高速缓存

基于局部行原理得知最近常用的代码其实十分有意义所以才加上了这个缓存。



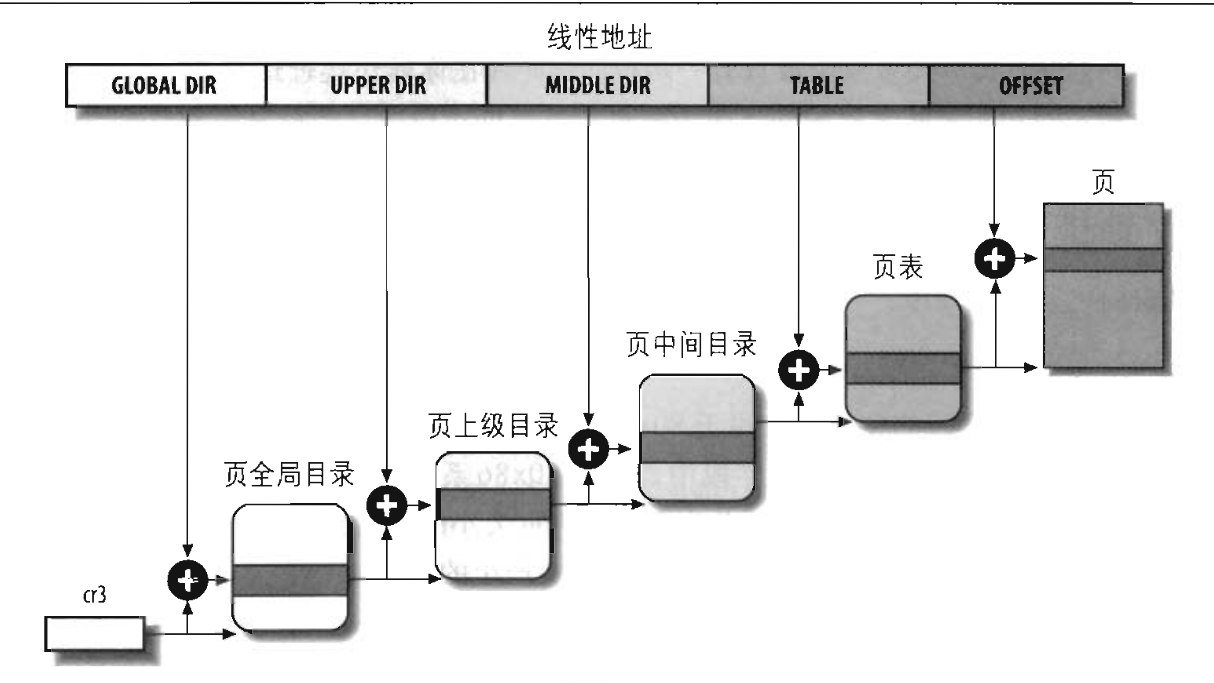
**Linux的分页是四级分页：**

页全局目录

页上级目录

页中间目录

页表



Linux把cr3控制寄存器的内容巴奥存在一个执行进程的描述符pid中把下一个要执行进程的描述符的值装进cr3里然后才完成了处理任务

linux虚拟地址的字段，由以下宏简化了页表处理

PAGE\_SHIFT 指定偏移量字段位数

PMD\_SHIFT 指定偏移量的和页表字段的总位数

PUD\_SHIFT 确定上级目录的映射区域大小

PGDIR\_SHIFT 确定全局目录映射的区域大小

**物理内存布局：**

在初始化的阶段，内核必须要划定一个物理地址映射来作为内核可用和不可用

内核将页框变为保留：

在不可用的的物理地址范围的页框，

含有内核代码和已初始化的数据结构的页框

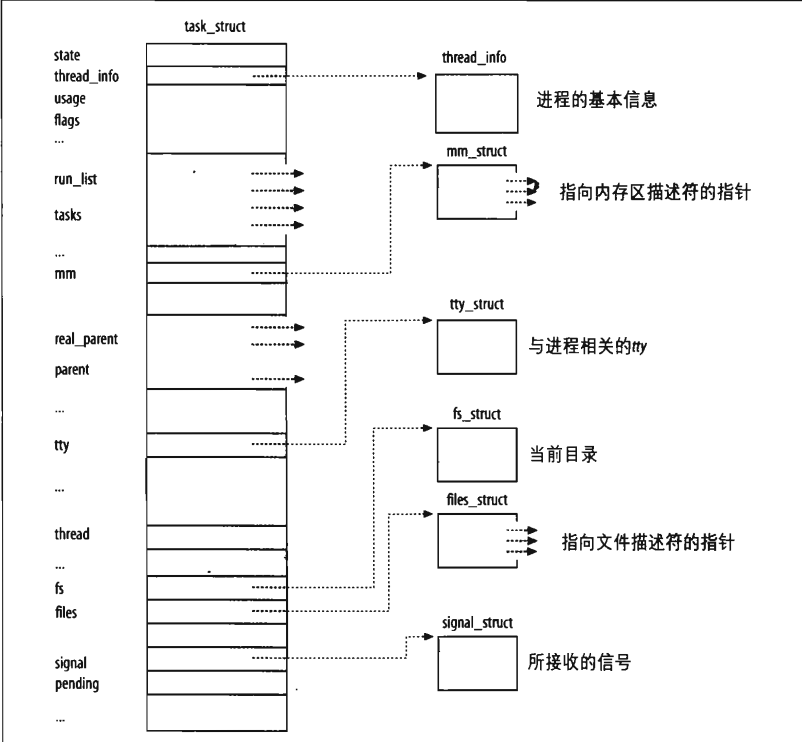
**进程**

**进程页表：**

进程的虚拟地址空间分成两部分，0x00000000~0xbfffffff无论进程运行在用户态还是内核态都可以寻址，0xc0000000~0xffffffff只有内核态进程才能寻址

**进程描述符：**

就是task\_struct类型的结构，他的字段包含了一个进程相关的所有信息，这里的结构略微复杂下图只列出了基本的东西



**进程状态**

**可运行：准备执行中或在CPU上执行**

**可中断：进程被挂起或者睡眠，需要被某个信号唤醒**

**不可中断的等待状态：就是强行传递给这个睡着的进程唤醒信号也不能改变它的状态**

**暂停状态：进程的执行被暂停**

**跟踪状态：当一个进程被另一个进程监控**

**僵死状态：俗称zombie进程的执行被终止，但是父进程还没有发布wait4或者waitpid的返回死亡进程的信息，在内核发布wait系统调用前内核不能丢地死亡的进程描述符的数据。**

**僵尸撤销状态：最终状态，就是上面的父进程发出了wait4或者waitpid的系统调用，此进程由系统删除**

**进程链表：**

早期版本的linux把所有可运行进程都放在同一个叫做运行队列的链表中

**进程切换：**

为了控制进程的执行内核必须挂起正在cpu上运行的进程，并且恢复以前的挂起进程的执行。。。额。。就是异步就叫做进程切换

了解进程切换之前先了解几个概念

1. 硬件上下文：所有的进程可以拥有自己的地址空间，所有的进程必须共享cpu寄存器,因此在恢复一个进程的执行之前，内核必须确保每个寄存器挂起时候进程的值，而进程恢复执行前必须装入寄存器的数据叫做硬件上下文。
2. 任务状态段：用来存放上面的硬件上下文，尽管Linux不适用硬件上下文切换，但还是强制CPU创建任务状态段,因为CPU从用户态切换到内核态的时候，就是从状态段中获取内核态的堆栈地址；并且当用户态试图通过输入输出访问I/O端口的时候CPU需要存放在任务状态段落的I/O许可权 位图，检查是否有权限访问这个端口。
3. thread字段：这个很熟悉，在每次进程切换时被替换进程的硬件上下文必须保存在别处。内核不能设计像因特尔呢样把它保存在任务状态断种，因为linux每个处理器不是为每个进程使用任务状态段。

**执行进程切换的步骤**

1. 切换页全局目录安装一个新的地址空间
2. 切换内核态堆栈和硬件上下文，因为硬件上下文提供了内核执行新锦成所需要的所有信息，包括CPU寄存器的信息。

**clone() fork vfork系统调用**

轻量级进程由名为clone的函数创建

clone是c语言库中定义的负责实现对系统调用sys\_clone，把fn指针放在子进程战队的某个位置当封装完成后去fn（一个新进程执行的函数）

fork：早期的fork是由clone实现的，根vfork一样

copy\_process：创建进程描述付以及子进程执行所需要的所有其他数据结构

do\_fork：负责处理clone、fork、vfork的系统调用(p135)

**内核线程：**

创建一个内核线程：kernel\_thread函数来创建一个新的内核线程

**撤销进程：**

就是终止，一般都是用exit函数，该函数释放c函数库的所有分配资源执行编程者所注册的每个函数，内核可以有选择的钱够整个线程组死掉

**线程终止：**

exit\_group终止整个线程组exit只是终止某一个线程

do\_group\_exit函数杀死所有current线程组的所有线程