# 第一章 绪论

内核线程是一个能被**独立调度**的执行环境

系统的根本样子和能力是由内核决定

“并发”是代表几个应用程序能够同时处于活动状态并竞争各种资源

“独立”是代表每个应用程序能执行自己的任务，无需考虑其他用户的应用程序在干什么

用户程序的非特权模式和内核的特权模式在Unix下被称为用户态和内核态

进程可以理解为一个程序的实例，也就是说一个正在运行的程序，线程是在一个进程内的可以多个线程

操作系统中有一个名叫调度程序的程序，这个程序决定了哪个进程可以执行哪个不可以

但是有一些操作系统是只允许有非抢占式进程，也就说只有进程自愿放弃CPU的时候，程序调度才会被调用

多用户系统中的进程必须是抢占式的，操作系统记录下每个进程占有的CPU时间，并周期性的激活调度程序

微内核操作系统比单块内核（Unix）更加充分地利用了RAM，因为暂且不需要执行的进程可以直接被调出或者撤销

即使模块依赖于某些特殊的硬件特点，但它不依赖于某个固定的硬件平台

模块代码一旦被连接到内核，其作用与静态链接的内核的目标代码完全等价

如果路径名的第一个字符是斜杠“/”那么这个路径就是绝对路径，它的起点是跟目录

如果第一项是目录名或文件名，那么这个路径就是相对路径，它的起点是进程当前的目录

包含在目录中的文件名就是一个文件的硬链接，简称连接，在同一目录或不同的目录中，同一文件可以有几个连接，因此对应不同文件名

软连接的路径名可以指向位于任意一个文件系统的任意文件或目录，甚至可以指向一个不存在的文件

除了设备文件和特殊文件系统文件外，每个文件都是由字符序列组成，文件内容不包含任何控制信息，每个文件都有自己的**索引节点**，文件系统用索引节点来标识文件

文件访问权限的组合用九种不同的二进制来标记，还有三种附加标记

当文件由一个进程创建时，文件拥有者的ID就是该进程的UID，而其用户组ID可以是进程创建者的ID，也可以是父目录的ID，这取决于父目录sgid的值

文件系统是硬盘分区物理组织的用户级视图

每个实际的文件操作必须在内核态下进行

文件描述符表示进程与打开文件之间的交互

同一打开文件对象也许由同一个进程中的几个文件描述符标识

文件系统给每个文件分配一个单独的打开文件对象以及但速度的文件描述符

对普通Unix文件，可以顺序地访问，也可以随机地访问，而对设备文件和命名管道文件通常只能顺序地访问

顺序访问是文件的默认访问方式

Unix内核提供了应用程序可以运行的执行环境

标准的Unix内核只用到了内核态和用户态

当一个程序在用户态下执行时，它不能直接访问内核数据结构或内核的程序，当在内核态是这些限制将不再有效

每种CPU两个状态的切换都提供了特殊的指令

一个程序在执行时一般情况下都是在用户态下执行，只有需要内核所提供的服务时才会切换到内核态，当内核满足了用户程序的请求后，它让程序又回到用户态

内核本身不是一个进程，而是进程的管理者

在单处理器系统中，任何时候只有一个进程在运行，它要么处于内核态要么处于用户态

当一个进程切换到内核态的时候调度程序就会开启定时器，到定时器中断的时候调度程序就会把这个进程切换到用户态

每个中断信号都是由内核中的中断处理程序来处理的

每个进程都会有一个进程描述符，这个描述符包含有关进程当前状态的信息

因为程序计数器中所存的值指向下一条将要执行的指令，所以进程从它停止的地方恢复执行

所有的Unix内核都是可重入的，也就是说可以有若干个进程可以同时在内核态下执行

内核控制路径表示内核处理系统调用、异常或中断所执行的指令序列

每个进程都运行在它私有的地址空间，但有时进程之间也共享部分地址空间

每个内核控制路径都引用它自己的私有内核栈

进程间也能共享部分地址空间，以实现一种进程间通信

临界区：当进程进入临界区之后该进程就必须执行完，它是不能被打断的区域（也就是一段代码）

信号量：也就是一个可以代表这个进程现在的状况的一个量

自旋锁：跟信号量差不多，当一个进程发现锁被另外一个进程锁住时，那么这个进程就会不停地死循环直到锁打开为止

自旋锁在单处理器环境下是无效的

死锁：也就是说当出现了非常复杂的情况的时候就会出现死锁，比如说进程a可以访问c，进程b可以访问d，但是现在进程却要访问d，进程b要访问c，这时候就会形成死锁，死锁会导致很严重的后果，很可能使系统崩溃

所以现在的系统通过按照规定的顺序请求信号量来避免死锁

内核通常要求使用物理内存的连续区域，因此，即使有足够大的内存时，如果它不是连续的大块内存的话，那就会产生内存请求失败

KMA----内核内存分配器，试图满足所有对内存的请求，它由所有的子系统调用包括中断处理程序

Linux在KMA上使用的slab算法

内存分配情况

实际上我们使用的malloc函数并不是马上就给我们分配了一个空间，首先在堆内开辟了一个框架，然后等我们读取的时候会产生一个异常，因为没有对应的内存空间，这时候异常处理函数就会给它分配一个空闲的内存空间

磁盘的读取速度远没有RAM快，所以当进程在执行的时候会首先检查数据是否在RAM中

Sync()是把脏的缓冲区( 缓冲区内容和磁盘块的内容不一样 )周期性地放到磁盘中

# 第二章 内存寻址

内存地址分三种地址

逻辑地址 线性地址 物理地址

物理地址只用于内存级寻址

MMU（内存控制单元）中有两个硬件电路，首先分段单元会把逻辑地址转换成为线性地址，然后分页单元会把线性地址转换成为物理地址

在硬件电路中有一种内存仲裁器的一个东西，就是用来判断内存的访问顺序的

一个逻辑地址由两个部分组成：一个段标识符和一个指定段内相对地址偏移量

段标识符是一个16位的字段（成为段选择符），偏移量是32位的

处理器里面有段寄存器，段寄存器的唯一目的就是存放段选择符

段寄存器中

cs寄存器指向当前程序指令段

ss寄存器指向当前程序的栈段

ds寄存器指向静态数据段和全局数据段

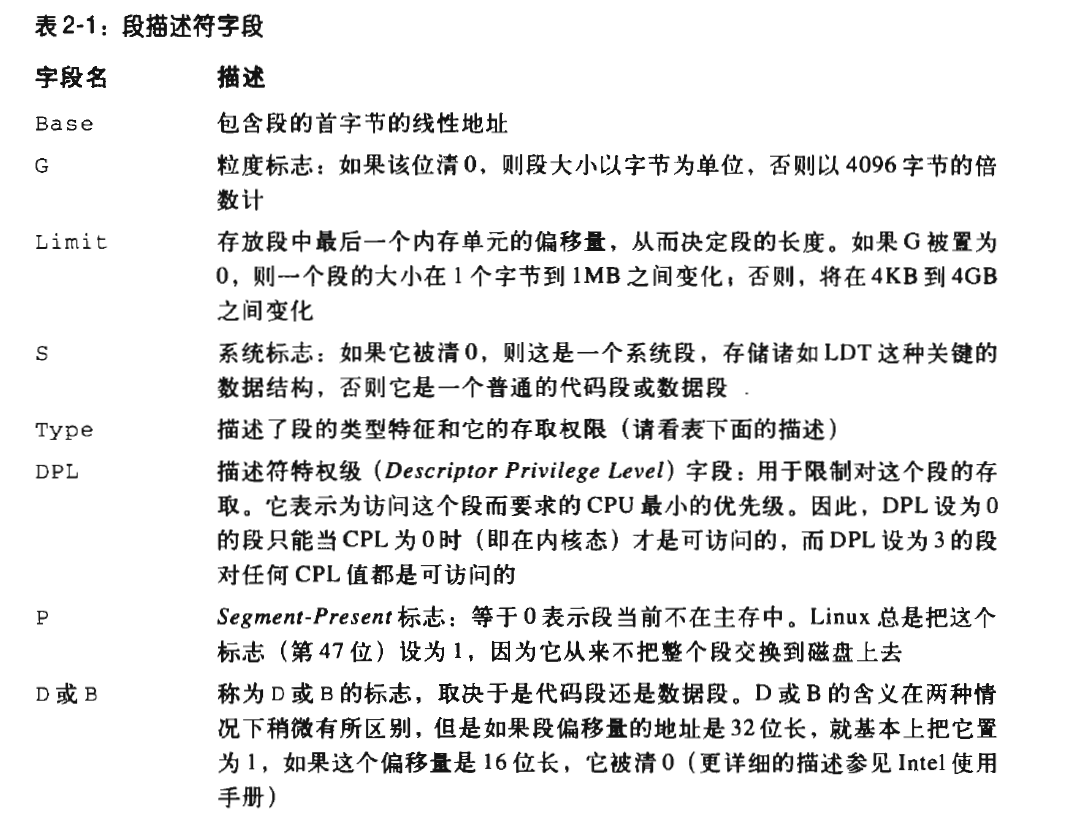
其它寄存器都可以用在一般用途上

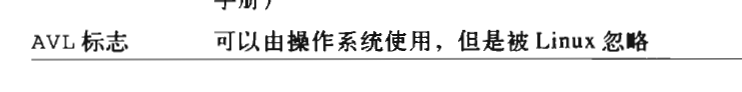
Cs寄存器的数据分成两个段用来指明CPU现在的特权级，一共就0到3级，0代表最高级3级代表最低级，但是Linux就用0级和3级，也就是内核态和用户态

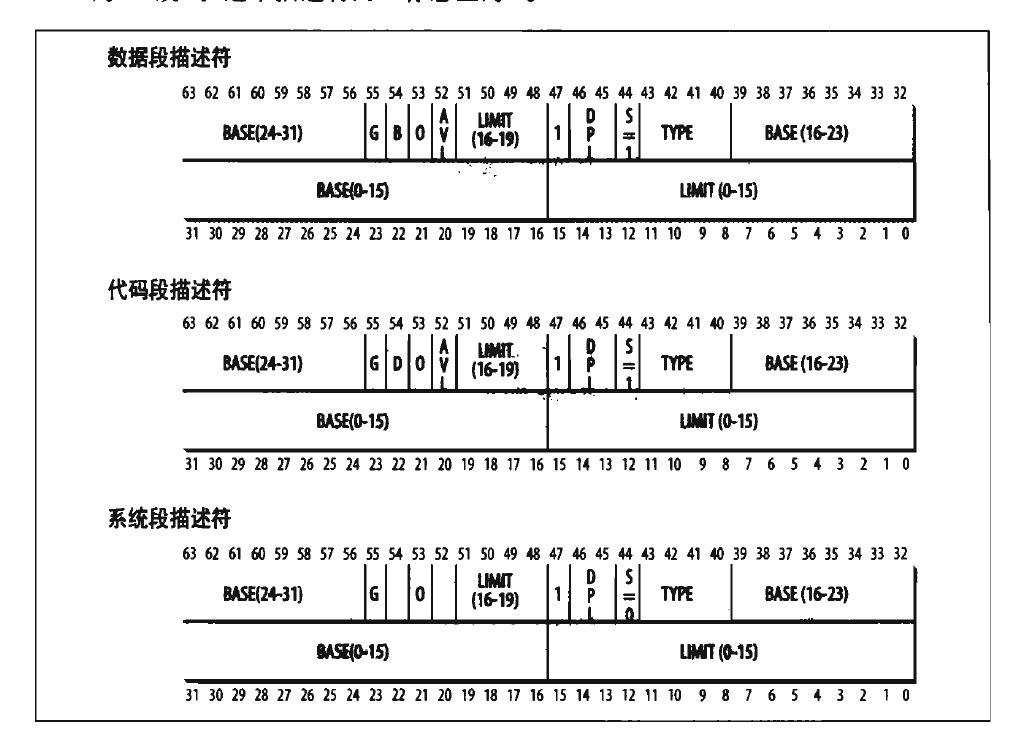
每个段由一个段描述符来表示，一个段描述符有8字节的长度

全局描述符表——GDT，局部描述符表——LDT

段描述符字段分一下几个块







栈段是通过一般数据段来实现的

任务状态段描述符（TSSD）只能放在GDT中

硬件中的逻辑地址是这么转换到线性地址的

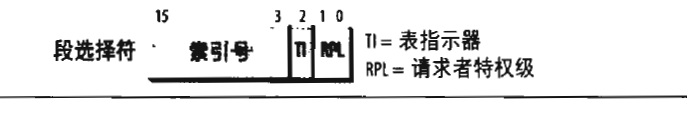
首先逻辑地址分两个部分，一个是段选择符和偏移量，然后每个段都会有一个段描述符

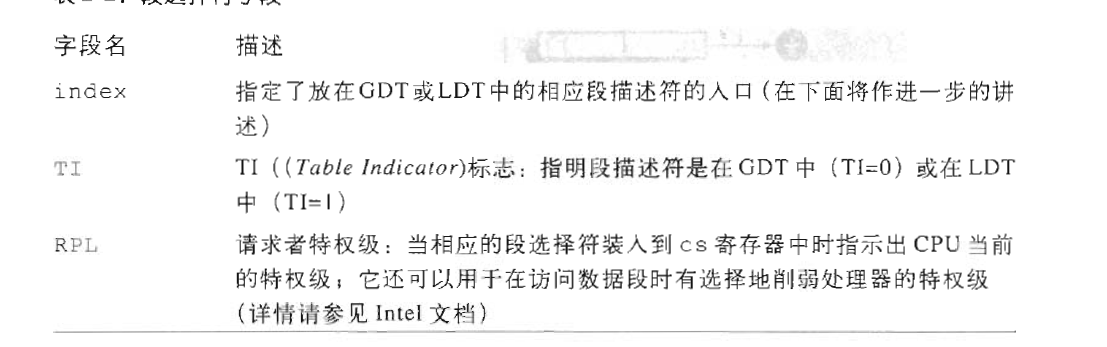
首先段选择符是一个16位长度的描述符，偏移量是32位的，而段描述符是64位的

一个段描述符是放在GDT或者是LDT中，而GDT和LDT分别放在gdtr和ldtr寄存器中

所以我们如果需要找段描述符需要找到GDT或者LDT

我们再看段选择符，段选择符是放在段寄存器中的，我们看一下段选择符的构造





在80x86处理器中是有一种非编程寄存器的（提供6个），它可以直接存放段描述符，也就是说每当一个段选择符被装入段寄存器中的时候段描述符就会被内存装入这个非编程寄存器中去这样CPU就不用访问GDT或者LDT就可以转换逻辑地址了

只有段寄存器改变的时候才需要访问GDT

我们来分析一下分段单元是怎么来把逻辑地址转换成线性地址的

首先需要从段选择符中读出TI位来判断是读取GDT还是LDT，读出GDT后，我们再看选择符的index也就是索引地址是哪个，通过这个索引地址来从GDT中读取出段描述符，需要注意的是index不能直接使用，需要左移3位也就是乘以8才可以使用，需要变成13位的数据才能够寻址

然后我们用取出的段描述符中的Base字段加上之前的偏移量就可以得出线性地址了