**[任务和特权级保护](http://blog.csdn.net/longintchar/article/details/51295755)**

# 1.LDT（局部描述符表）

在之前的学习中，不管是内核程序还是用户程序，我们都是把段描述符放在GDT中。但是，为了有效实施任务间的隔离，处理器建议每个任务都应该有自己的描述符表，称为局部描述符表LDT（Local Descriptor Table），并且把专属于这个任务的那些段描述符放到LDT中。

和GDT一样，LDT也是用来存放段描述符的。不同之处在于，LDT只属于某个任务。或者说，每个任务都有自己的LDT，每个任务私有的段，都应当在LDT中进行描述。

**需要注意的是：LDT的0号槽位，是有效的，可以使用的。**

GDT是全局性的，为所有的任务服务，是它们所共有的，我们只需要一个GDT就够了。为了追踪GDT，访问它内部的描述符，处理器使用了GDTR寄存器。   
和GDT不同，LDT的数量不止一个，具体有多少，根据任务的数量而定。为了追踪和访问这些LDT，处理器使用了LDTR（局部描述符表寄存器）。在一个多任务的系统中，会有很多任务轮流执行，正在执行的那个任务，称为当前任务（Current Task）。因为LDTR只有一个，所以，它只用于指向当前任务的LDT。每当发生任务切换时，LDTR的内容被更新，以指向新任务的LDT。

当向段寄存器加载段选择子的时候，段选择子的TI(Bit2)位是表指示器（Table Indicator）。   
TI=0：处理器从GDT中加载描述符   
TI=1：处理器从LDT中加载描述符

因为描述符索引占用了13个比特，所以每个LDT最多能容纳的描述符个数是8192（2的13次方），也就是说每个LDT只能定义8192个段；又因为每个段描述符占用8个字节，所以LDT的最大长度是64KB（2的16次方）。

# 2.TSS（任务状态段）

在一个多任务环境中，当任务发生切换时，必须保存现场（比如通用寄存器，段寄存器，栈指针等）。为了保存被切换任务的状态，并且在下次执行它时恢复现场，每个任务都应当有一片内存区域，专门用于保存现场信息，这就是任务状态段（Task State Segment）。

TSS的最小尺寸是104（0x68=104）字节。   
和LDT类似，处理器用任务寄存器TR（Task Register）指向当前任务的TSS。和GDTR、LDTR一样，TR在处理器中也只有一个。当任务发生切换的时候，TR的内容会跟着指向新任务的TSS。这个过程是这样的：首先，处理器将要挂起的任务的现场信息保存到TR指向的TSS；然后，使TR指向新任务的TSS，并从这个TSS中恢复现场。

# 3.全局空间和局部空间

每个任务实际上包括两个部分：全部部分和私有部分。全局部分是所有任务共有的，含有操作系统的数据、库程序、系统调用等；私有部分是每个任务自己的数据和代码，与任务要实现的功能有关，彼此并不相同。

从内存的角度来看，所谓的全局部分和私有部分，其实是地址空间的划分，即全局地址空间（简称全局空间）和局部地址空间（简称局部空间）。   
对地址空间的访问离不开分段机制，全局地址空间用GDT来指定，局部地址空间由每个任务私有的LDT来指定。

从程序员的角度看，任务的全局空间包含了操作系统的段，是由别人编写的，但是他可以调用这些段的代码，或者获取这些段中的数据；任务的局部空间的内容是由程序员自己编写的。通常，任务在自己的局部空间运行，当它需要操作系统提供的服务时，转入全局空间执行。

# 4.特权级保护概述

在分段机制的基础上，处理器引入了特权级的概念，并由固件负责实施特权级保护。   
特权级（Privilege Level），是存在于描述符及其选择子中的一个数值。当这些描述符或者选择子所指向的对象要进行某种操作，或者被别的对象访问时，该数值用于控制它们所能进行的操作，或者限制它们的可访问性。

Intel处理器可以识别4个特权级别，分别是0~3，数值越小特权级越高。   
如下图所示（图片来自维基百科）：这是Intel处理器所提供的4级环状结构。

通过，操作系统是为所有程序服务的，可靠性最高，而且必须对软硬件有完全的控制权，所以它的主体部分必须拥有特权级0，处于整个环形结构的中心。因此，操作系统的主体部分通常被称作内核（Kernel、Core）。

特权级1和2通常赋予那些可靠性不如内核的系统服务程序，比较典型的就是设备驱动程序。不过，在很多流行的操作系统中，驱动程序也是0特权级。

应用程序的可靠性被视为是最低的，而且通常不需要直接访问硬件和一些敏感的系统资源，通过调用设备驱动程序和操作系统例程就能完成绝大多数工作，所以赋予它们最低的特权级别3.

# 5.CPL，DPL，RPL

想搞清楚段级保护，必须要弄懂这三个概念。

## 5.1.CPL

CPL：当前特权级（Current Privilege Level），存在于CS寄存器的低两位。   
当处理器正在一个代码段中取指令和执行时，这个代码段所在的特权级叫做当前特权级。正在执行的这个代码段，其选择子位于段寄存器CS中，CS中的低两位就是当前特权级的数值。   
一般来说，操作系统的代码正在执行时，CPL就等于0；

相反，普通的应用程序则工作在特权级3上。应用程序的加载和执行，是由操作系统主导的，操作系统一定会将其放在特权级3上（具体的做法，我们会慢慢学到）。当应用程序开始执行时，CPL自然会是3.

需要注意的是，不能僵化地看待任务和任务的特权级别。当任务在自己的局部空间执行时，CPL等于3；当它通过调用系统服务，进入操作系统内核，在全局空间执行时，CPL就变成了0.

## 5.2.DPL

DPL：描述符特权级（Descriptor Privilege Level），存在于段描述符中的DPL字段。   
DPL是每个描述符都有的字段，故又称描述符特权级。描述符总是指向它所描述的目标对象，代表着该对象。因此，DPL实际上是目标对象的特权级。

## 5.3.RPL

RPL：请求特权级（Requested Privilege Level），存在于段选择子的低两位。   
要想将控制从一个代码段转移到另一个代码段，通常是使用jmp或者call指令，并在指令中提供目标代码段的选择子和偏移；为了访问内存中的数据，也必须先将段选择子加载到段寄存器，比如DS、ES、FS、GS中。不管是实施控制转移还是访问数据段，这都可以看成是一个请求，请求者提供一个段选择子，请求访问指定的段。从这个意义上来说，RPL也就是指请求者的特权级别。

也许你会疑惑：有CPL和DPL进行判断不就可以了吗？为什么还需要一个RPL呢？   
因为当低特权级的应用程序使用call far指令通过调用门将控制转移到较高特权级的非一致代码段（例如操作系统提供的例程，假设此代码段的DPL=0）时，会改变当前的特权级，而在目标代码段的特权级上执行，对于本例来说CPL的数值就会变成操作系统例程段的DPL的数值，即0。如果没有RPL，那么此时CPL权限是最高的，也就可以去访问任何数据，这就不安全了。所以引入RPL，让它代表访问权限，因此在检查CPL的同时，也会检查RPL.一般来说如果RPL的数值比CPL大(权限比CPL的低)，那么RPL会起决定性作用。

# 6.IO特权级

在处理器的标志寄存器EFLAGS中，位12、13是IOPL位，也就是输入/输出特权级（I/O Privilege Level），它代表着当前任务的I/O特权级别。

如果CPL在数值上小于等于IOPL，那么所有的I/O操作都是允许的，针对任何硬件端口的访问都可以通过。

相反，如果CPL的数值大于IOPL，也并不意味着所有的硬件端口都对当前任务关上了大门。事实上，处理器的意思是总体上不允许，但个别端口除外。至于是哪些个别端口，要找到当前任务的TSS，并检索I/O许可位串（具体细节我们以后会说）。

只有当CPL=0时，程序才可以使用POPF或IRET指令修改这个字段。

# 7. 加载用户程序与创建用户任务

## 7.1 任务控制块（Task Control Block，TCB）

### 7.1.1 TCB的格式

加载程序并创建一个任务，需要用到很多数据，比如程序大小、加载位置等等。内核应当为每一个任务创建一个内存区域，来记录任务的信息和状态，这个内存区域就称为任务控制块（Task Control Block，TCB）。   
需要说明的是：TCB不是处理器的要求，而是我们为了自己方便而发明的。

# 8. 如何转到用户程序执行

任务寄存器TR总是指向当前任务的TSS，而LDTR寄存器也总是指向当前任务的LDT。TSS是任务的主要标志，因此要使TR寄存器指向当前任务；而使用LDTR的原因是可以在任务执行期间加速对段的访问。

在多任务环境中，从旧任务切换的新任务的时候，TR和LDTR寄存器的值都会更新，以指向新任务。但是，目前我们只有一个任务，而且是特权级为3的任务，我们遇到的问题可以表述为：如何从任务的全局空间（处于特权级0）转移到它自己的局部空间（处于特权级3）？

答案是分为两步：   
1. 确立身份，使TR和LDTR寄存器指向这个任务；   
2. 假装从调用门返回；