14章任务和特权级保护

1140310606 张茗帅

这一章主要进行的仍然是用户程序的加载、重定位和运行，特别之处在于涉及到了特权级的转换和任务，包括LDT表的设置，TSS，TCB的设置以及相关描述符的创建和添加。特权级很重要的有三个值：CPL（当前特权级）、DPL（描述符特权级，每个描述符中都有）、RPL（请求特权级，段选择子中）。

操作系统的特权级为0，用户程序的特权级为3。总结有一下几个原则：

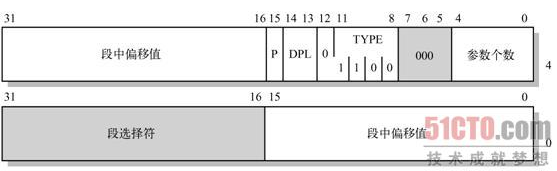
1. 高特权级的程序可以访问低特权级的数据段，反之不可。
2. 一般来说，控制转移只能发生在两个特权级相同的代码段之间。
3. 如果高特权级的代码段定义为依从（段描述符type字段中的C=1），可以从特权级比他低的程序调用并进入。前提是当前特权级（CPL）必须低于（数值上是大于）或等于目标代码段描述符的DPL。
4. 在任何时候，不允许将控制从较高的特权级转移到较低的特权级。（返回除外）

另外，一个很有用的概念：调用门，转移控制。Jmp far将控制转移到比当前特权级高的代码段，不改变CPL；call far则会改变cpl到目标代码段的特权级别。

**①设置RPL的必要性。**

一般情况下，RPL=CPL（另外，堆栈段的特权级别永远等于CPL），但是RPL在有些时候发挥着不可替代的作用。

按书上的话来说，RPL确保特权代码不会替代应用程序访问一个段，除非应用程序自己拥有访问那个段的权限。举个例子，当一个用户程序（特权级为3，开始时CPL=RPL=3）通过call far转移控制到了操作系统，此时CPL=0,RPL=3。当转移控制之后，一旦出于某种不可控的原因，代码请求访问DPL=0的段中的数据，如果没有RPL的限制，按照CPL=0=DPL来考量这个访问就是被允许的，这就可能造成用户程序通过转移控制获得了最高访问权限，肆无忌惮的访问所有数据的情况。而有了RPL的限制之后，RPL=3<DPL，不能完成这次访问，达到了识别访问请求到底是从哪个程序发出的作用，毕竟RPL叫做请求特权级。



调用门描述符

**②调用门的细节：**

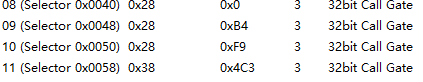
通过调用门调用例程时，不使用指令中给出的偏移量，type字段1100说明这个描述符是调用门描述符，p有效位一般=1。最低的5个参数用于记录栈切换时，栈中需要复制的参数的个数，最多31个。（补充：栈的特权级跟CPL时刻一致，目的是防止栈空间不足和交叉引用，一个任务栈的数量跟它的特权级有关：特权级+1，为了满足控制转移到高特权级时使用。）

一个程序能访问调用门前提是当前特权级CPL和请求特权级RPL高于或等于该调用门特权级DPL，并且当前特权级低于或者等于目标代码段描述符DPL。

调用门安装：代码中完成的是将内核SALT表例程地址转换成调用门。

循环调用make\_gate\_descriptor和set\_up\_gdt\_descriptor函数进行调用门描述符构造并增加到gdt表中，后者的功能和13章中增加内核描述符方法一样，而make\_gate\_descriptor也只是使用了32位偏移地址和代码段选择子根据门属性进行了一系列转化，形成门描述符。

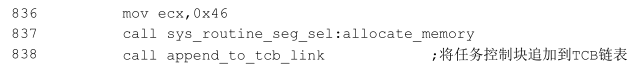
全部循环结束后，GDT表中增加了4项调用门描述符。



对门进行测试的过程中，不再使用salt表中给出的偏移量，因为门描述符中已经设置好了。

**③创建一个TCB（任务控制块），并追加到TCB链表中**

因为声明第一TCB块是在内核数据段中，只知道偏移地址，所以在函数append\_to\_tcb\_link中使用内核数据段遍历tcb链表。



838行结束后，函数返回值ecx为TCB首地址，0x100000。

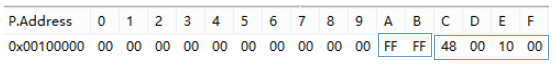
IMG_256

**④加载用户程序**

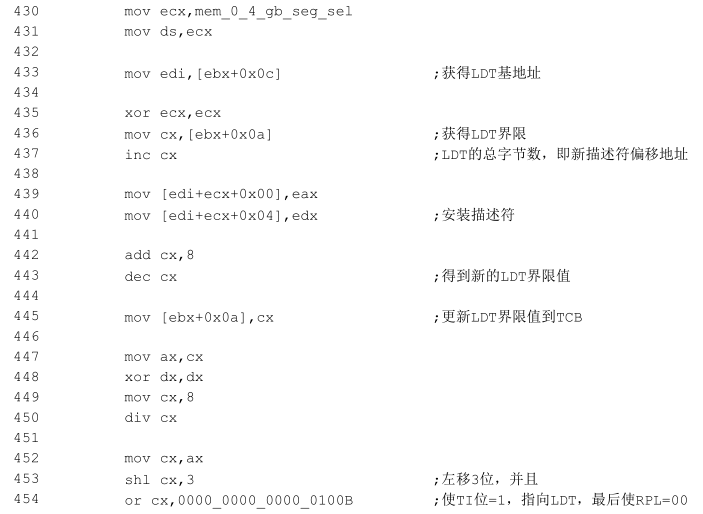
与13章不同的是，增加了TCB起始地址参数，而且和用户程序第一个扇区的位置参与一起，用栈进行传递，如图0x100000为TCB起始地址，可以从SS:[EBP+11\*4]处得到，0x32是第一扇区，可以从SS:[EBP+12\*4]处得到。



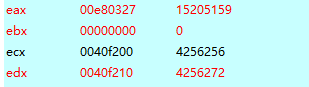
创建LDT表之后，将LDT基地址0x00100048和界限0xffff存入TCB：



添加LDT表中的描述符，首先建立头部段描述符，由函数fill\_descriptor\_in\_ldt接收参数要安装的描述符和TCB基地址，其中要安装的描述符通过sys\_routine\_seg\_sel:make\_seg\_descriptor计算得到。通过内核数据段找到TCB，在TCB中得到LDT表的基地址，进行描述符装载和段界限计算，452行还计算了该描述符的选择子，将TI位置1，表示查询LDT。

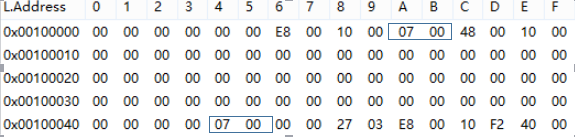


得到的头部数据段描述符(如图)：edx：eax=0x40f210:0xe80327



创建完成头部描述符加载后，直接添加到LDT表中，LDT表的基地址和界限都存放在TCB中，添加之后要将界限更新，并将头部描述符选择子回写到TCB和头部。

TCB如图：

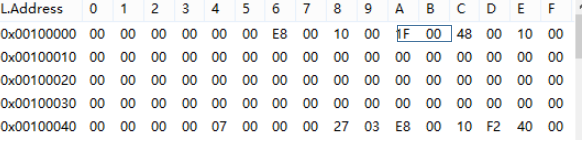


LDT如图：

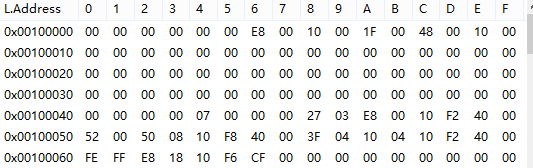


其他描述符的创建和添加过程于此类似，选择子不登记到TCB只登记到头部。

所有描述符加载结束后，TCB如图：



LDT表如图：



**⑤对用户程序SALT表进行重定位**

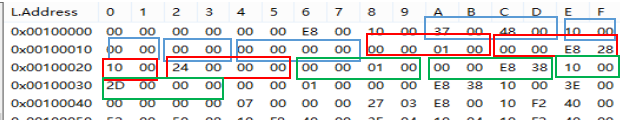
与13章重定位有两个重要的不同：

1. 由于LDT表还没有生效，所以在访问用户程序头部的时候不能使用已经创建的头部描述符，要使用内核数据段来访问U-SALT表，进行逐一对比。
2. 由于内核的SALT表中的公共例程地址是特权级为0下的调用门选择子和段内偏移，605、606行将选择子中特权级部分更改为3。



用户程序是特权级为3，要为它创建特权级为0、1、2的栈。

由于还没有设置TSS故先将申请好的栈的信息和选择子暂存在TCB中。创建过程大致相同，只不过特权级设置不同。全部创建完成后，TCB如图（蓝色框0级栈的长度、栈基地址、选择子、初始ESP，红色1级，绿色2级）：

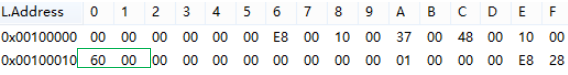


安装LDT描述符到GDT表中：

使用函数make\_seg\_descriptor创建LDT描述符，从TCB中获取LDT基地址和界限，LDT描述符特权级为0，TI位为0。LDT选择子登记在TCB中。GDT表如下图：



TCB如下图：

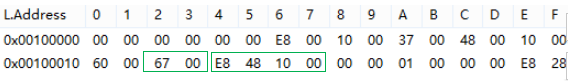


创建TSS：

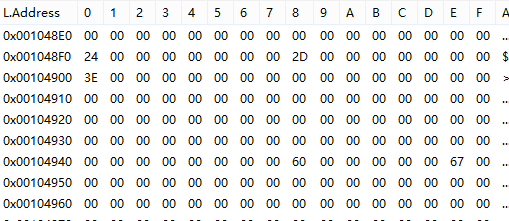
调用函数sys\_routine\_seg\_sel:allocate\_memory,得到TSS基地址：



将TSS的界限0x67和基地址保存在TCB中。

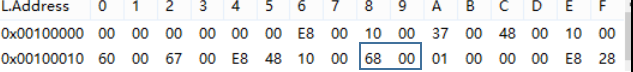


并将TCB中的相应数值对应的复制到TSS中，第100字节的第一位设置为0，T=0，说明不可调试。



之后，创建TSS描述符，添加到GDT表中，描述符特权级为0。



同时TSS选择子也须登记在TCB中。

带参数的返回指令：ret 8

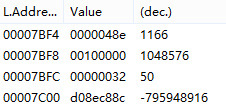
Esp寄存器指向调用过程前的栈的位置，进行栈平衡，返回前的栈为：



Popad指令，弹出通用寄存器的值：



POPad弹出之后，栈：

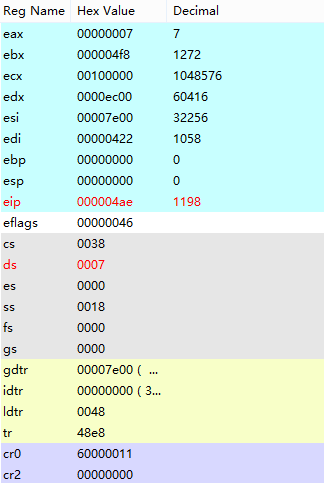


参数8表示要弹出2个双字。返回后栈：

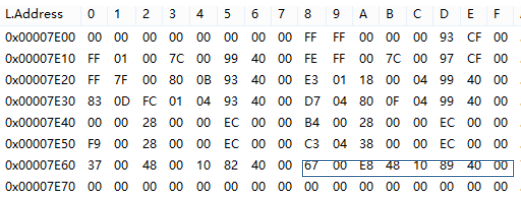


**⑥用户程序的执行，特权级转移。**

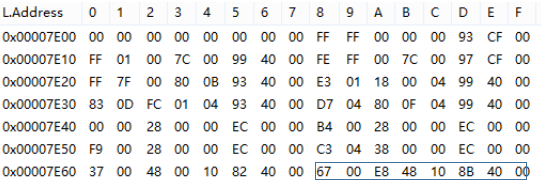
首先将控制转移到用户程序，采用模拟从调用门返回的方式。先从TCB中装载LDTR和TR寄存器。



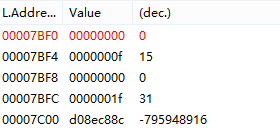
装载后将tss描述符B位置1，表示忙。



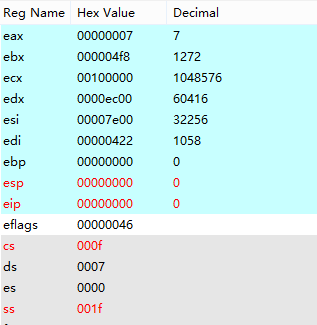
变化后的GDT，0x7e7d位置，89->8b,即10001001->10001011



之后根据call far调用门的特性，在栈中压入模拟调用门返回的值：



Retf指令后，栈切换到用户级，cs段发生改变：

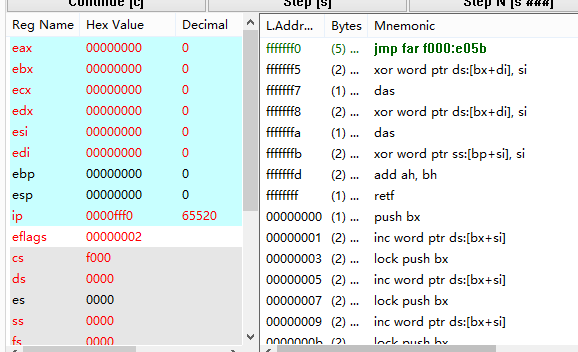


我们已经来到用户程序：

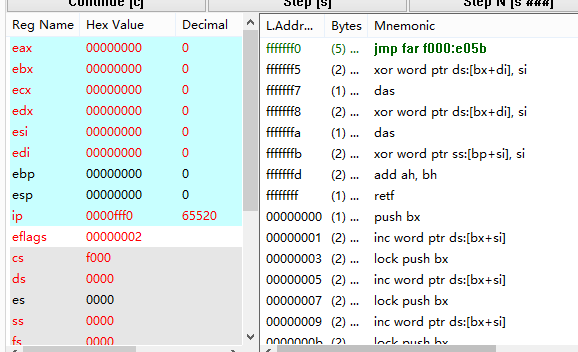


**⑦从用户程序回到内核程序。**

以jmp far指令通过调用门回到内核程序，CPL=3，要使用core\_data\_seg\_sel，虽然请求特权级RPL=0，但是目标代码特权级DPL为0，CPL>DPL，故引发了异常中断。



寄存器的值变化：



结束