0.进入保护模式

0.1.控制寄存器



1.保护模式的内存段保护

GDT: Global Description Table，全局描述符

LDT: Local Description Table,局部描述符

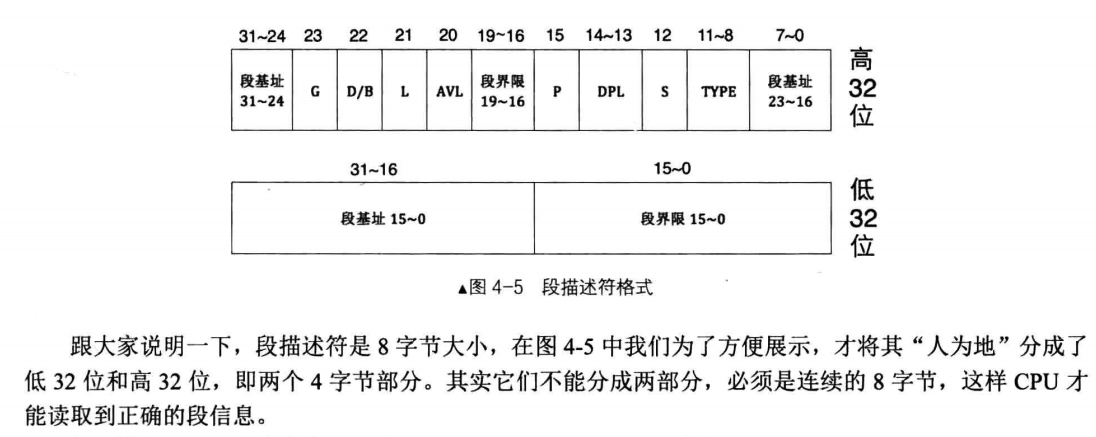
1.1.向段寄存器加载选择子时的保护：越界检测（GDT或LDT的界限地址）

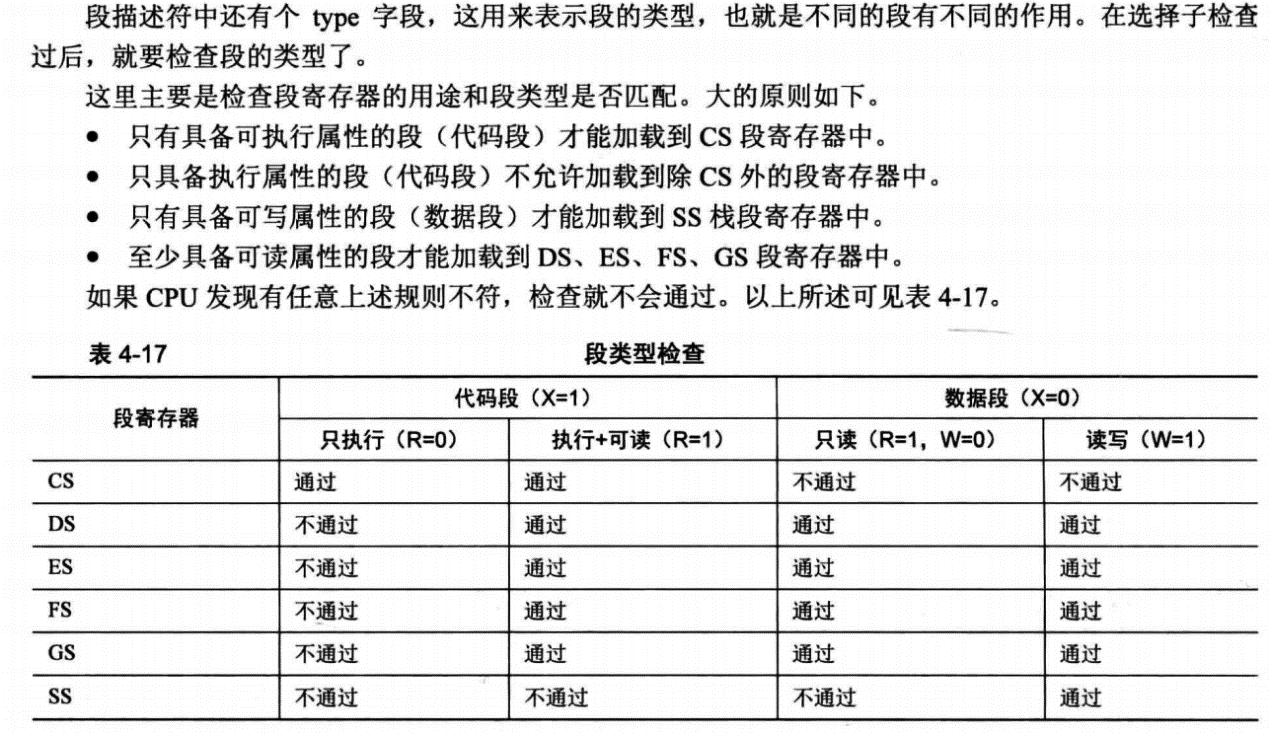
选择子：根据选择子的值验证段描述符是否超越界限

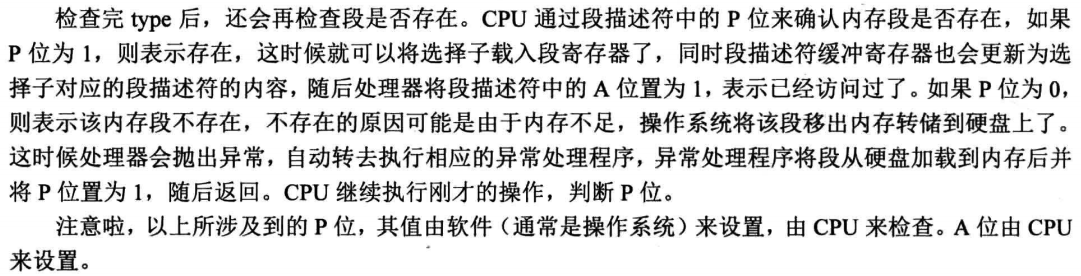
高13位：段描述符的索引值

第0-1位：RPL，段访问请求权限

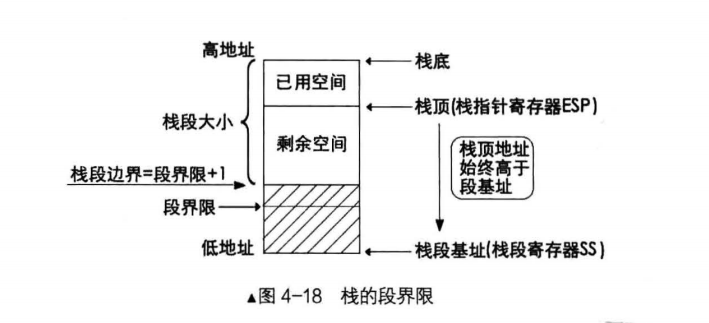
第2位：TI，检查TI值判断：GDT or LDT







1.2.栈段的保护



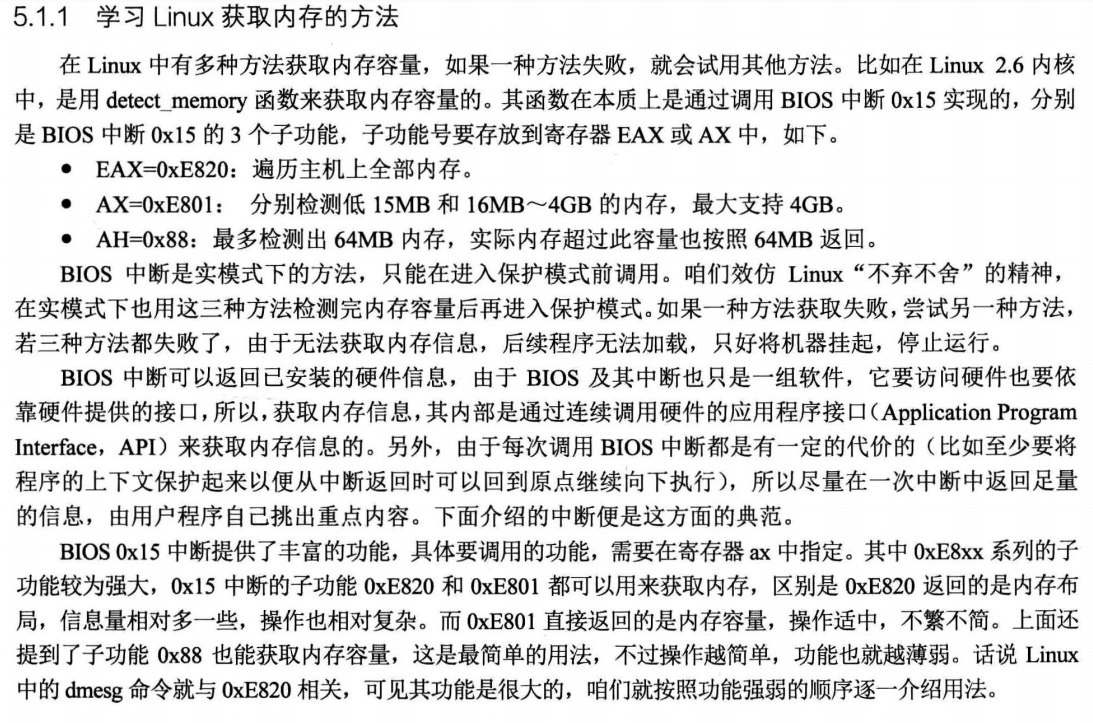
2.保护模式进阶

2.1.学习Linux获取内存的方法

a.利用BIOS中断0x15子功能0xe820获取内存

b.利用BIOS中断0x15子功能0xe801获取内存

c. 利用BIOS中断0x15子功能0x88获取内存

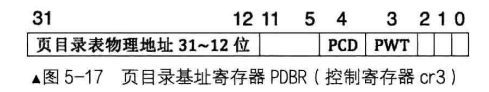


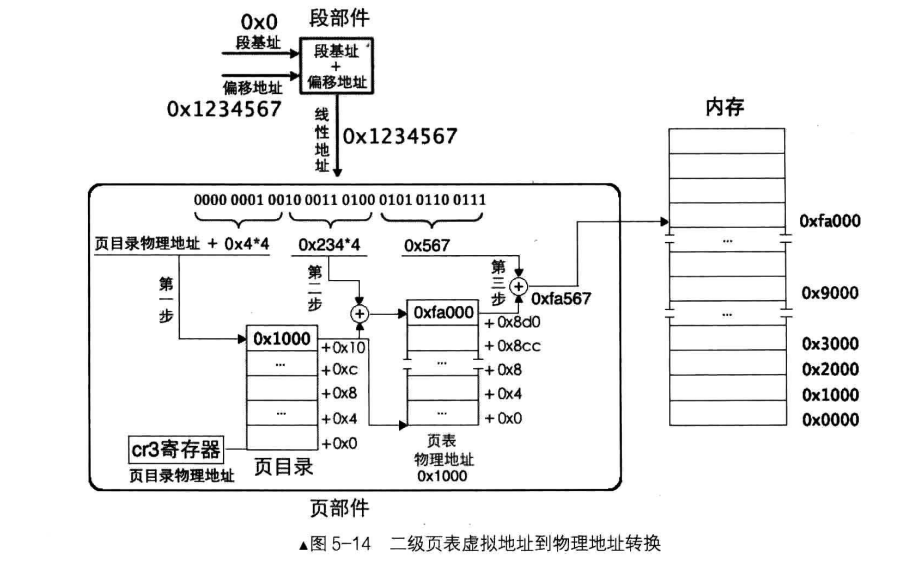
内存分页机制：用于进程的内存空间分配

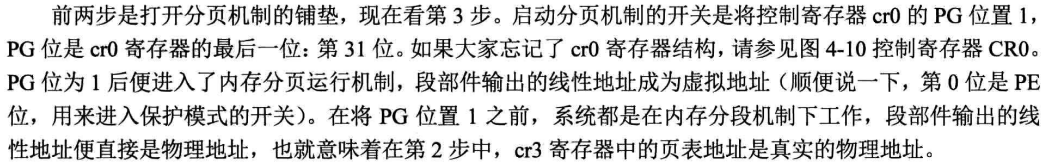
二级页表(1024\*4字节) 一级页表(1024\*4字节) 物理页（4K字节）

虚拟地址：高10位 中间10位 低12位

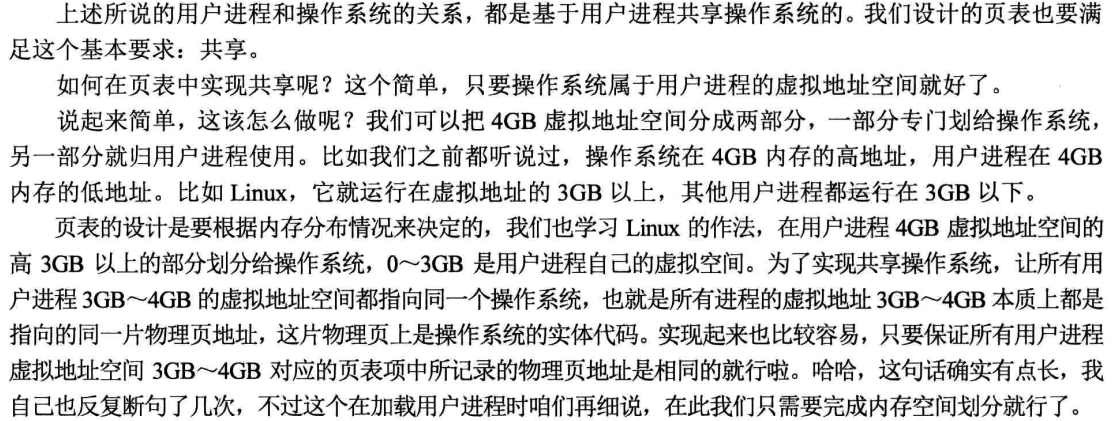


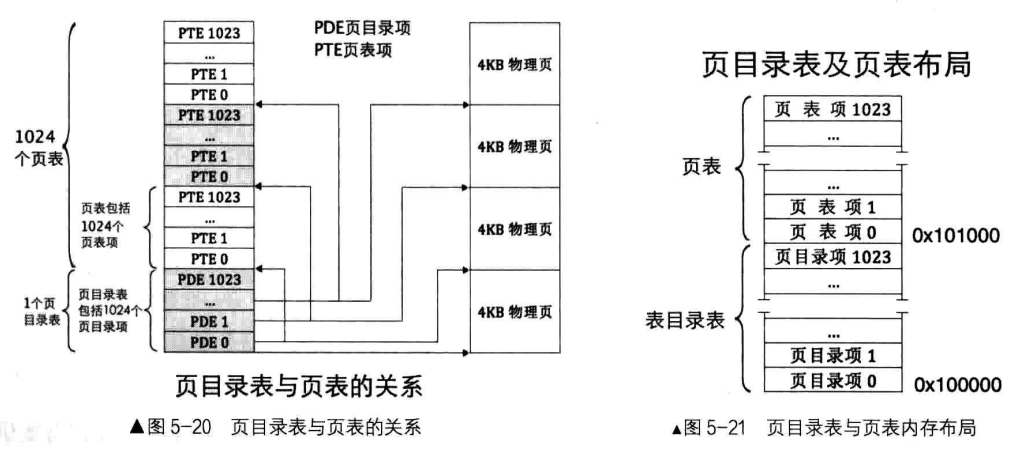


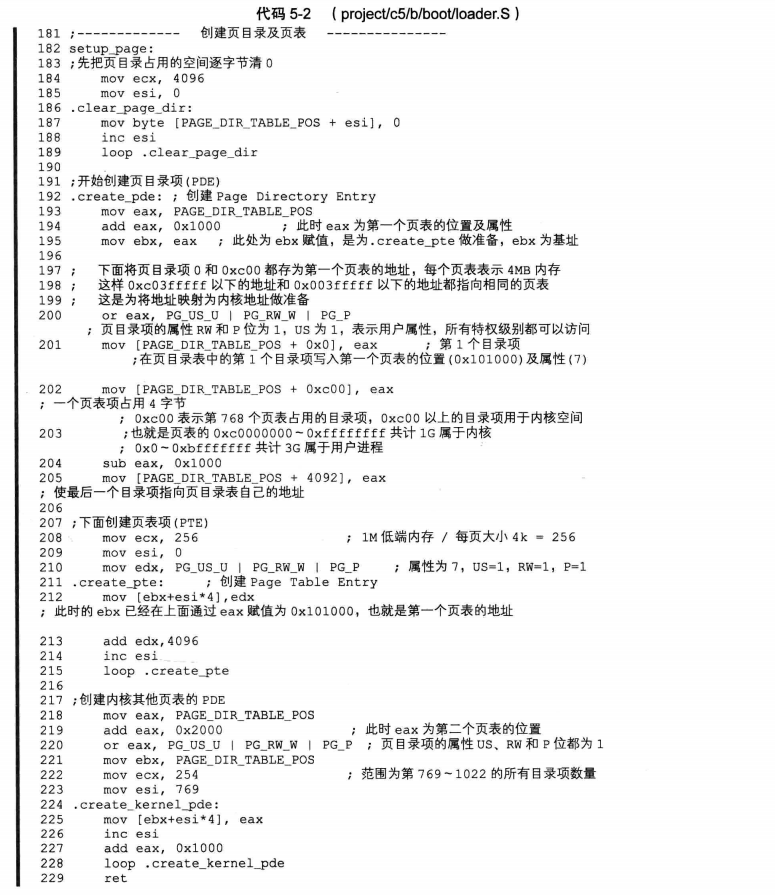




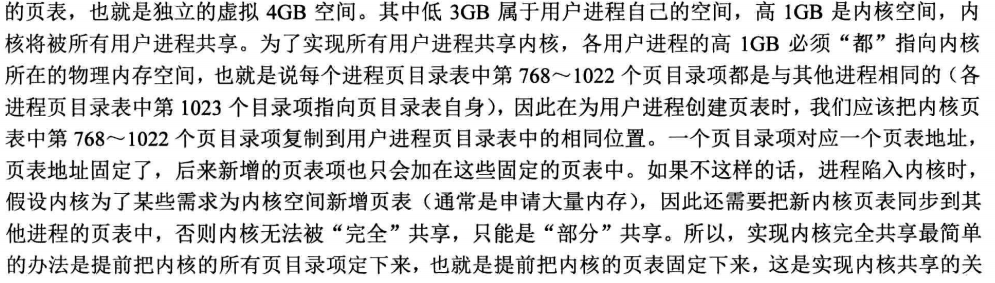
共享页表







共享内核



块表TLB（Translation Lookaside Buffer）

由于对页表需要多次访存，访存时间过长，所以根据局部性原理使用缓存将最近访问的虚拟地址到物理地址的映射。

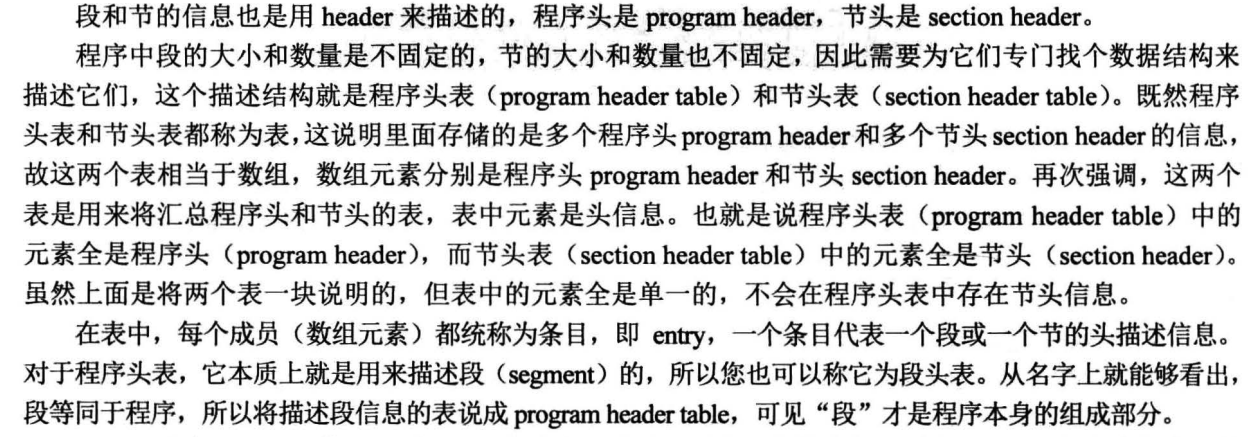
内核

ELF格式的二进制文件

Windows的可执行文件格式是PE（Portable Executable），后缀名是EXE。

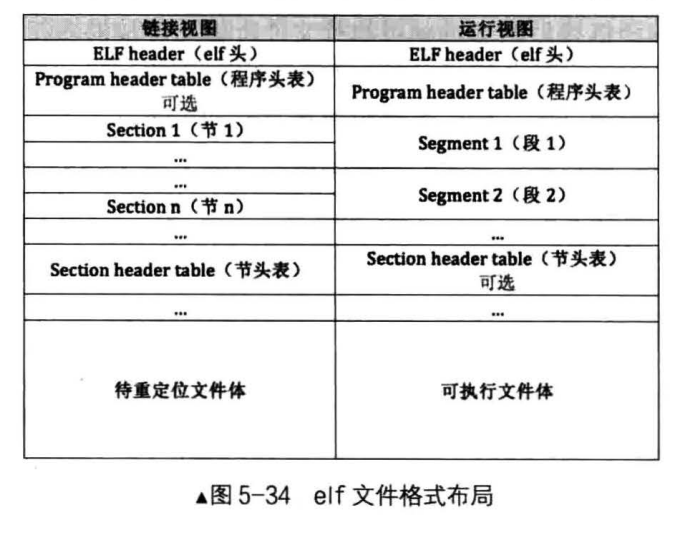
Linux的可执行文件格式是ELF（Executable and Linkable Format），可执行链接格式

程序最终的部分是段（segment）和节（section），他们是真正的程序体



ELF header是个用来描述各种“头”的“头”，程序头表和节头表中的元素也是程序头和节头。

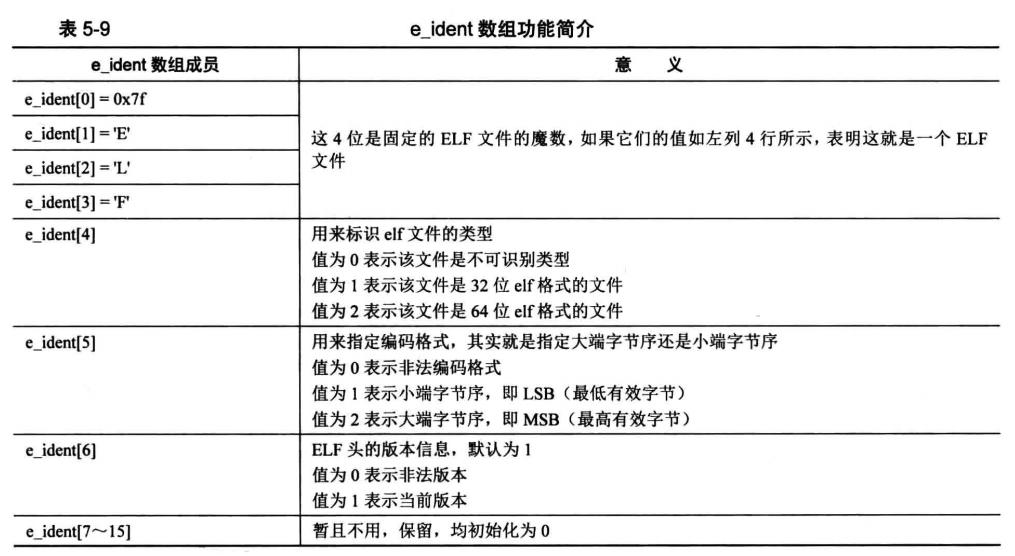
ELF文件格式分为文件头和文件体



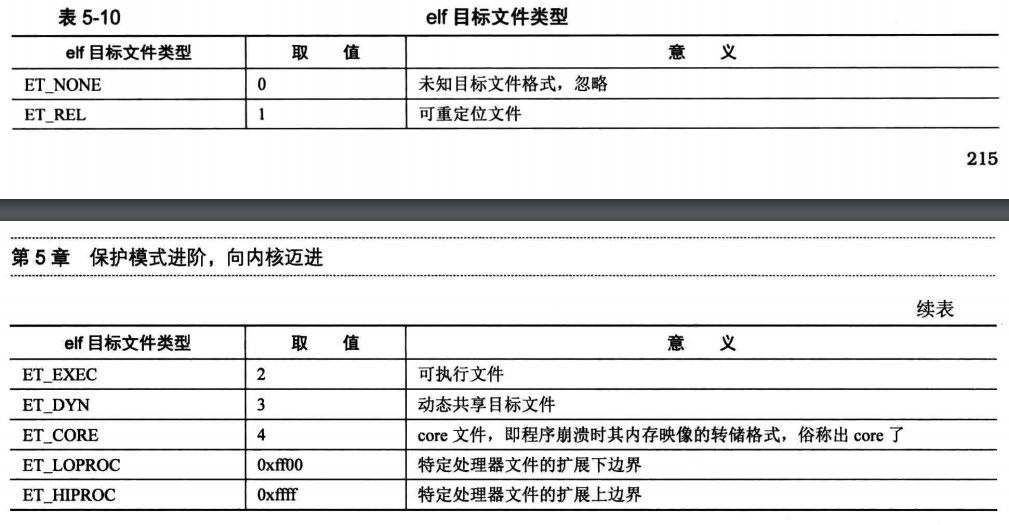




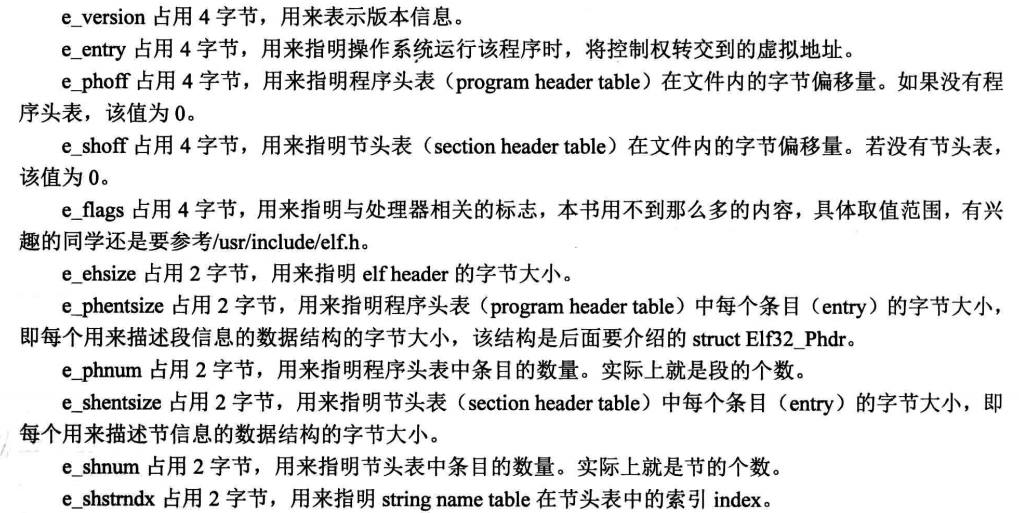
E\_ident是16字节大小的数组，用来表示elf字符等信息，开头的4给字节是固定不变的，是elf文件的魔数，他们分别是0x7f，以及字符串ELF的sac码：0x45,0x4c,0x46。



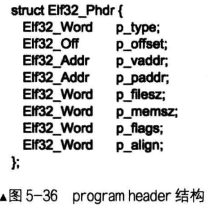
E\_type占用2字节，是用来指定elf目标文件的类型

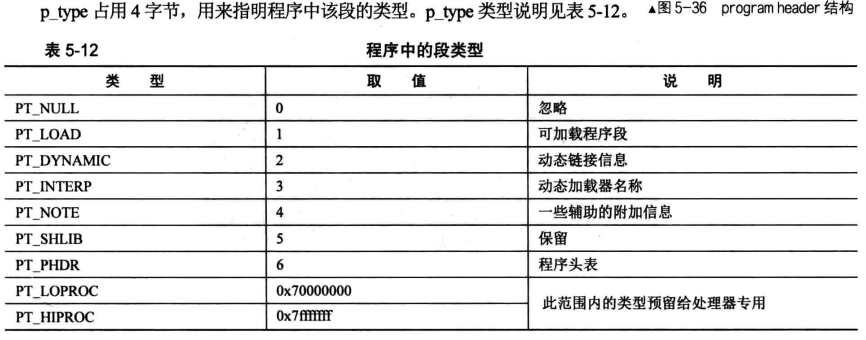






程序投标的条目的数据结构，主要用来描述各个段（数据段或代码段等）的信息用的



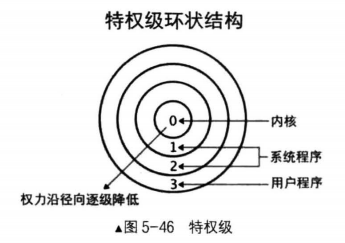
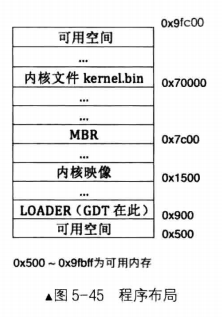


64位机 & 64位gcc生成i386可执行文件：gcc -c -m32 -o kernel/main.o kernel/main.c

Ld kernel/main.o -Ttext 0xc0001500 -e main -m elf\_i386 -o kernel/kernel.bin

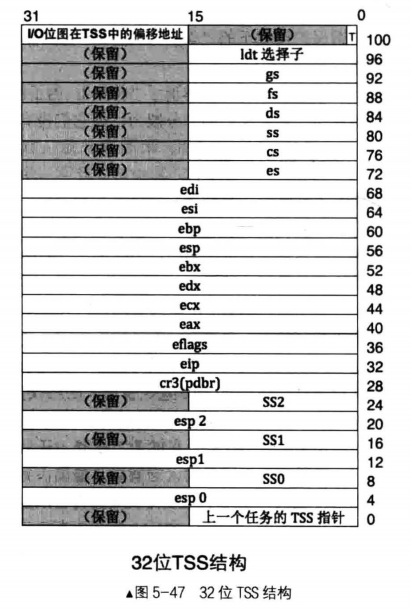
将内核载入内存位置





Task State Segment(TSS，意为任务状态段)，可用于特权级，但它的功能还包括许多内容。它是处理器在硬件上原生支持多任务的一种实现方式。

TSS是每个任务都有的结构，用于任务表示。



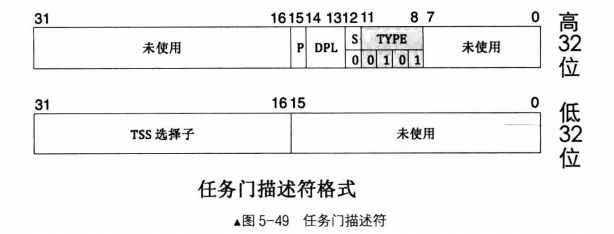
计算机特权级别的标签体现在DPL、CPL和RPL。CPL（Current Privilege Level,当前特权级）,它表示处理器正在执行的代码的特权级别。转移后的目标代码段的DPL是将来处理器的当前特权级CPL

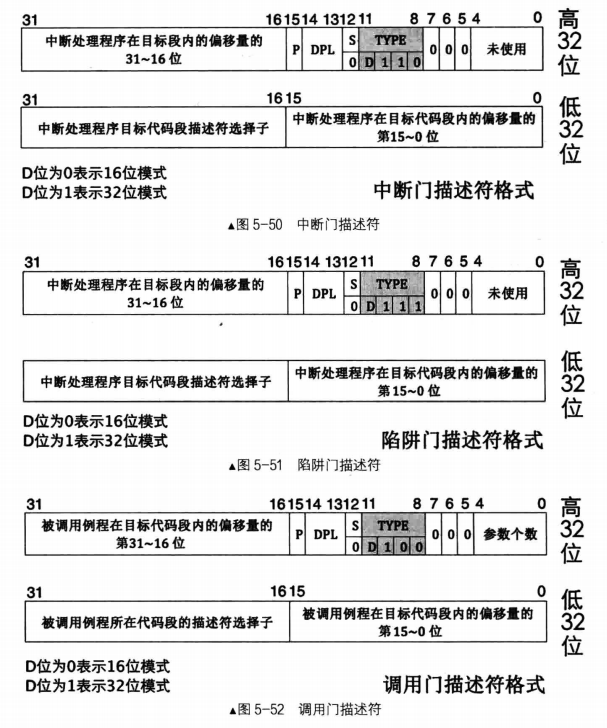
RPL（请求特权级）的产生主要是为了解决系统调用时的“越权”问题，系统调用的实现方式中，以调用门和中断门最为合适。

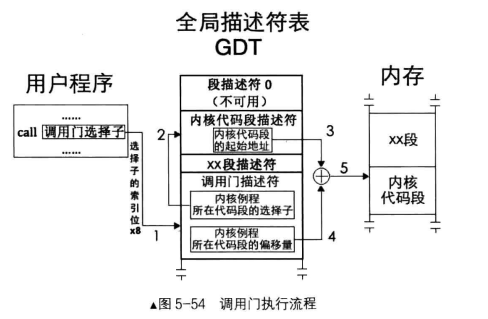
处理器只有通过“门结构”才能由低特权级别转移到高特权级别，就是这样设计处理器的。

门结构就是记录一段程序起始地址的描述符。

调用门描述符是位于GDT或LDT中的，所以要调用它就要通过门描述符选择子。





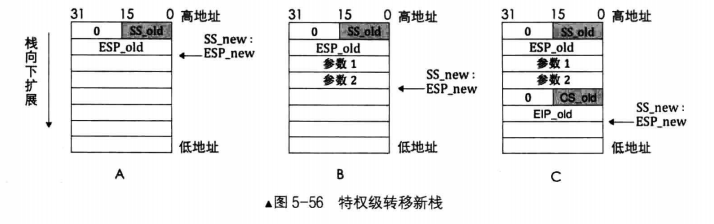


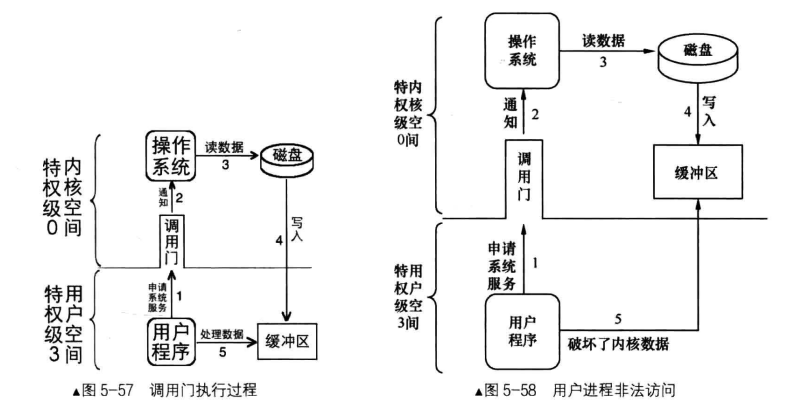
不同特权级下处理器用不同的栈。

调用门的过程保护：

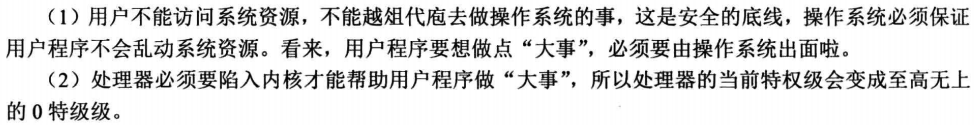
调用门涉及两个特权级：1）转移前的低特权级，是程序调用“调用门”时的CPL

2）转移后的目标特权级，是由门描述符中选择子对应的目标代码段的DPL决定





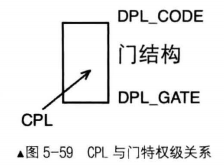
用户进程就算由特权级0，也不应该获取内核数据。用户进程如果提交的数据源参数是内核数据段选择子就会使内核完全保护，仅仅依靠CPL和DPL不能剞劂相关问题。

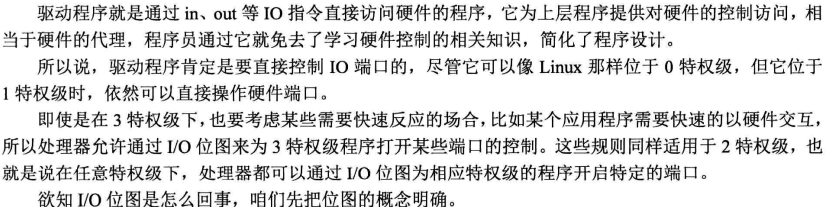


所以由了RPL（Request Privilege Level,请求特权级）用以揭示请求者的身份，它的引入的目的是避免低特权级的程序访问高特权级的资源。

特权级检查：CPU检查数值上CPL<=DPL && RPL <= DPL是否成立

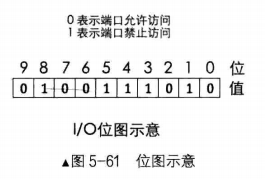
32位保护模式下对内存的访问要通过段描述符，段描述符中有DPL。

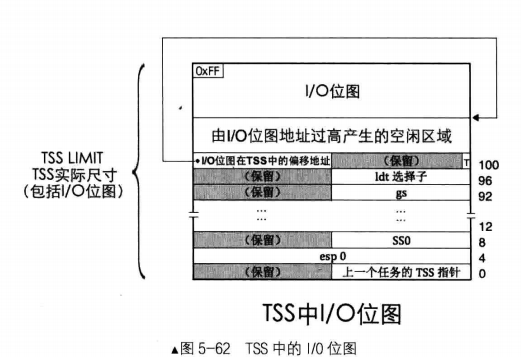




IO特权指令上，采用局部放权，用于处理器提速，不用再通过内核访问相关资源

I/O位图（映射），Intel处理器最大支持65536个端口，允许任务通过I/O位图来开启特定端口





函数调用约定（calling conventions）:参数的传递方式，参数的传递顺序，使调用者保护寄存器环境还是被调用者保存，保存哪些寄存器

每个进程都有自己的栈，就是每个内存自己的专用内存空间，可以用于保存参数

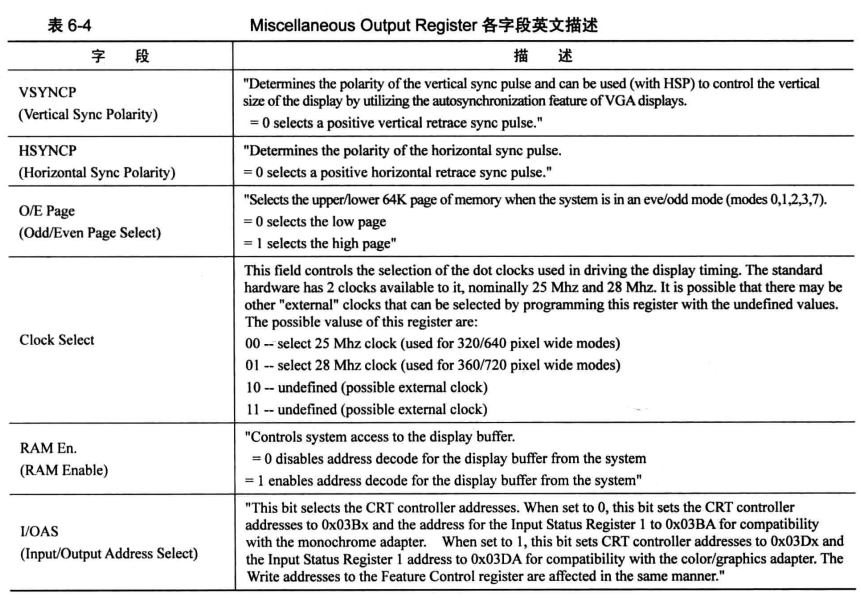


系统调用类似调用BIOS的中断，但是入口只有0x80中断

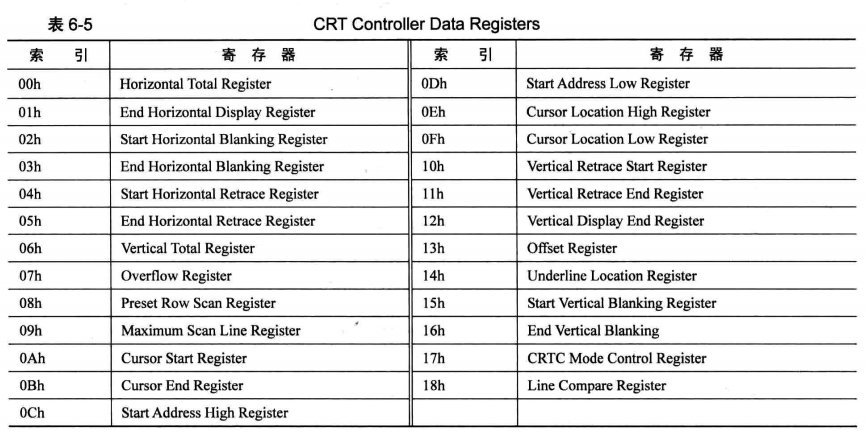
显卡的端口控制

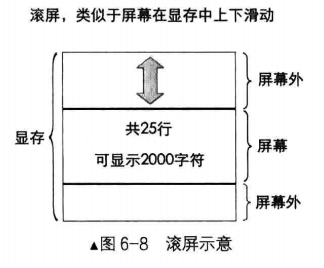






本系统只用到CRT Controller Register分组中的寄存器





nasm -I boot/include -o boot/mbr.bin boot/mbr.S

nasm -I boot/include -o boot/loader.bin boot/loader.S

nasm -f elf -o lib/kernel/print.o lib/kernel/print.S

gcc -I lib/kernel -c -m32 -o kernel/main.o kernel/main.c

ld -Ttext 0xc0001500 -e main -m elf\_i386 -o kernel.bin kernel/main.o lib/kernel/print.o

dd if=boot/mbr.bin of=/~/Desktop/Bochs/hd60M.img bs=512 count=1 conv=notrunc

dd if=boot/loader.bin of=/~/Desktop/Bochs/hd60M.img bs=512 count=3 seek=2 conv=notrunc

dd if=kernel.bin of=/~/Desktop/Bochs/hd60M.img bs=512 count=200 seek=9 conv=notrunc

gcc -m32 -I lib/kernel -c -o build/timer.o device/timer.c

gcc -m32 -I lib/kernel/ -I lib/ -I kernel/ -c -fno-builtin -o build/main.o kernel/main.c

gcc -m32 -I lib/kernel/ -I lib/ -I kernel/ -c -fno-builtin -o build/init.o kernel/init.c

gcc -m32 -I lib/kernel/ -I lib/ -I kernel/ -c -fno-builtin -o build/interrupt.o kernel/interrupt.c

gcc -m32 -I lib/kernel/ -I lib/ -I kernel/ -c -fno-builtin -o build/main.o kernel/main.c

nasm -f elf -o build/print.o lib/kernel/print.S

nasm -f elf -o build/kernel.o kernel/kernel.S

gcc -m32 -I lib/kernel/ -I lib/ -I kernel/ -c -fno-builtin -o build/interrupt.o kernel/interrupt.c

gcc -m32 -I lib/kernel/ -I lib -I kernel/ -c -fno-builtin -o build/init.o kernel/init.c

ld -Ttext 0xc0001500 -e main -m elf\_i386 -o build/kernel.bin build/main.o build/init.o build/interrupt.o build/print.o build/kernel.o build/timer.o

dd if=build/kernel.bin of=/~/Desktop/Bochs/hd60M.img bs=512 count=200 seek=9 conv=notrunc

内联汇编

一般的C语言编译是windows dos，但是unix支持AT&T的内联汇编，允许在C代码中直接嵌入汇编