|  |  |  |  |
| --- | --- | --- | --- |
| 武汉大学国家网络安全学院教学实验报告 | | | |
| 课程名称 | 操作系统设计与实践 | 实验日期 |  |
| 实验名称 |  | 实验周次 |  |
| 姓名 | 学号 | 专业 | 班级 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| 1. 实验目的及实验内容   （本次实验所涉及并要求掌握的知识；实验内容；必要的原理分析） | | | |
| 1. 理解x86架构下的段式内存管理  2. 掌握实模式和保护模式下段式寻址的组织方  式、关键数据结构、代码组织方式  3. 掌握实模式与保护模式的切换  4. 掌握特权级的概念，以及不同特权之间的转  移  5. 了解调用门、任务门的基本概念 | | | |
| 1. 实验环境、实验步骤、实验结果   （本次实验所使用的器件、仪器设备等的情况；具体实验步骤；实验结果） | | | |
| 1. 认真阅读章节资料，掌握什么是保护模式，弄清关键数据结构：  GDT、descriptor、selector、GDTR， 及其之间关系，阅读  pm.inc文件中数据结构以及含义，写出对宏Descriptor的分析。  GDT（全局描述符表）全局存储各个段描述符。  Descriptor（段描述符）负责指向段基址，并描述段的相关属性和访问属性，如段界限、存在、特权级、描述符类型、界限粒度、访问方式等。  Selector（段选择子）：负责在GDT、LDT中，定位相关描述符的位置。  GDTR：全局描述符表寄存器，用于存储GDT的基址。  关系：在保护模式下，内存进行分段，程序索引一个内存地址的时候，用段寄存器中存储的段选择子，在GDT/LDT中索引这个选择子要访问的内存对应的段描述符，其中，GDT和LDT的位置分别由GDTR和LDTR寄存器给出。找到之后检查权限，然后使用段描述符中对对应内存段的基址与权限的描述，对对应内存进行访问。  对宏Descriptor的分析：    可以发现，这个宏是由三个参数组成：段基址（32位）、段界限（20位（作为参数传为双字，实际上只有20位））、属性（16位）。  %macro Descriptor 3：应该是定义这个宏，参数为3个。  dw %2 & 0FFFFh 段界限（参数2）的低16位跟FFFF掩码与，定义在最前面一个字。  dw %1 & 0FFFFh 段基址（参数1）的低16位于FFFF掩码，定义在第二个字。  db (%1 >> 16) & 0FFh 接下来是一个字节，用于连接16-23位的段基址。（注意字节序）  dw ((%2 >> 8) & 0F00h) | (%3 & 0F0FFh) 。%2 >> 8 就是将段界限右移1个字节，然后跟0F00与，那么就是取原来的第4个hex，也就是19-16位。 %3 & 0F0FF 的掩码用于防止跟界限冲突。  db (%1 >> 24) & 0FFh 取基址高24位放在这个描述符的最高字节。  具体图从PPT中截得：    2. 调试代码，/a/ 掌握从实模式到保护模式的基本方法，画出代码流程图，特别注意跳转问题，如果把跳转直接改成jmp  offset，而不用selector:offset形式，会是什么结果，反汇编  比较一下区别。  进入保护模式流程图：    关于jmp的跳转问题：  由于进入保护模式后，CPU认为段寄存器保存的都是段选择子，所以会用段寄存器中存储的东西对描述符表进行搜索，用查询到的描述符里保存的基址加上ip等偏移进行内存跳转访问。所以需要使用长跳转对cs赋值为段选择子，如果不赋值CS则为段内跳转，因为CS没改变所以这个选择符指向的base也是原来的地址。  调试与反汇编过程：  在调试器内反汇编。    被视为短转移所以里面存的偏移。可以发现目标地址0x32240是CS:IP为0x3224:0的地址：        执行后到达的是cs段的偏移0位置，。由于程序起始位置是0100h所以此处是未定义的。  若jmp为selector:offset  反汇编为：    是jmpf，即远跳转会改变cs和ip  比较二进制数可以发现，前者短跳转指令码为66 e9,操作数如图为偏移    后者指令码为66 ea 操作数为00000000，0008（注意字节序）    如图所示，保护模式启动，分段生效。    ndisasm的使用。  3. 调试代码，/b/，掌握GDT的构造与切换，从保护模式切换回实模式方法  GDT的构造：1.定义段描述符，给出每个描述符的基址初始值、界限和属性。2.计算GDT的长度和基址。3.在实模式时填入描述符对应代码段/数据段的基址，也就是定义段的操作。4.用lgdt加载GDT的基址，并切换cr0的保护模式位，（关掉中断，打开地址线），最后把cs置为保护模式代码段的选择子就可以切换到GDT。  从保护模式切换回实模式：  1.初始化时的准备工作：    将实模式的段地址填入返回的jmp行的cs。以及保存实模式堆栈指针。      调试代码如图，我们看一下LABEL\_GO\_BACK\_TO\_REAL的代码    ea 58 04 00 00，ea为跳转，则555+3为558，即填充00 00    填充后，此时如果执行这里的代码将会跳转到cs:ip=0x3224:0x0458  +3 则填充    这个CODE16段是在保护模式到实模式的中间态意义下执行的，所以要有描述符。将cs<<4 + LABEL\_SEG\_CODE16 意思是算出LABEL\_SEG\_CODE16的基址，+2行为填入BYTE2,BYTE3，然后右移16位意思是填入原来基址的BYTE2，填入描述符的BYTE4.最后+7则为填BYTE3到描述符的BYTE7.注意此处为什么会有4个字节，cs为2字节，但是偏移可能会很大，其宽度应该为4字节。  其余描述符的构造与这个类似。    此处Code32跳转到CODE16。    Code16将其余段寄存器赋值为Normal选择子  以es为例：  es赋值前为测试段选择子    赋值后为Normal选择子，基址为0，    其余类似。  然后cr0 & 11111110b 将PE位置0。即关闭保护模式。    注意此处偏移位数的变化。但是此处还没有jmp，所以现在的cs还是段选择符的形式。  然后jmp    现在cs就变回16位段寄存器的形式了。现在就已经是真正的回到实模式，需要处理一些善后工作。    然后就是设置其余段寄存器（跟普通实模式程序的开头一样），以及恢复原来的堆栈指针，和关闭地址线与开中断。  关闭地址线和开中断之所以在回到实模式之后再执行，是因为保护模式下，对IO的访问是有权限限制的，以及sti的执行也是限制权限的IO敏感指令，防止出现问题所以在实模式操作。    最后回到实模式的DOS。  为什么回到实模式之前要置Normal Selector 在这里把这段代码删掉看情况：    如图进入CODE16但是不设置段选择子。    执行完实模式初始化代码后如图，其实因为实模式没初始化fs和gs所以这俩还是选择子的形态，我们回到实模式之前并不能预知实模式程序会初始化哪些段寄存器，所以在CODE16段我们就要把所有segment段初始化为0。应该也是防止其他程序非法访问段吧。可能是安全性的考虑  4. 调试代码，/c/，掌握LDT切换  主要添加内容为LDT在GDT中的描述符以及LDT中的代码段CODE\_A的描述符    进入保护模式后，发现对LDT的代码    这个代码用lldt把LDT的基址放入ldtr中，然后跳转到CODEA选择子。    CODEA的选择子使用SA\_TIL （0x04）置TI为1表示该选择子索引的是LDT中的描述符，于是最后的jmpf把CodeA选择子装入cs。  CodeA的执行过程为  CPU从ldtr中寻找ldt的选择子，从gdtr中找到GDT的基地址，用ldt选择子在gdt中找ldt基地址，使用CodeA的cs选择子在LDT中索引到CodeA的描述符，然后使用该描述符访问到对应的地址    lldt前    lldt后，ldtr被赋值为LDT的选择子。    跳转的目标选择子为0x0004，其中4为TI位，所以进入后的cs是在LDT中找第0项描述符。  所以可以发现，LDT切换与GDT切换不同，LDTR中存储的是LDT的选择子，GDTR中存储的是GDT的基地址。类似的点是他们都要对cs寄存器赋值为对应选择子才能生效。  5. 调试代码，/d/掌握一致代码段、非一致代码段、数据段的权限访问规则，掌握CPL、DPL、RPL之间关系，以及段间切换的基本方法  一致性代码段可以由低特权级和相同特权级的代码跳转。  数据段总是非一致性的，只能由高特权级和相同特权级的代码访问。  非一致性代码段只能由相同特权级代码跳转。  在调用门的作用下：  CALL指令：非一致性代码段可以由特权级高于或等于门描述符特权级，但低于或等于目标代码段特权级的代码跳转  jmp指令：非一致性代码段可以由特权级高于或等于门描述符特权级，但等于目标代码段特权级的代码跳转。  也就是说，CALL可以实现当前特权级到高特权的转换，JMP只能同特权跳转。  CPL是CS与SS的低2位字段，说明了当前执行的代码的权限。  DPL是存储在段描述符中的，说明了该段的权限。  RPL是存储在段选择符中，由代码发出请求的时候用的。  关于CPL跟RPL怎么与DPL进行CHECK：  CPL和RPL取特权最低的，跟DPL进行CHECK，即书上写的CPL & RPL <= DPL（对于数据段）。  对于代码段，如手册上写的：    即自己的代码段CPL数值上大于等于DPL，而不用检查RPL。  非一致性代码段：    不使用调用门就必须CPL跟DPL相等。而且RPL必须数值上小于等于CPL。  那么举3个例子，分别是对一致性代码段、非一致性代码段的访问（数据段访问书上已给出）。  一致性代码段：可以直接由低特权级的代码向高特权级一致性代码段进行转移，该过程不改变CPL，意味着系统内某些运算库虽然存在系统内，但是可以以低特权状态运行。  非一致性代码段：除了不用调用门只能同特权转移之外，使用调用门的情况下可以由低特权级转为高特权级，但是要注意是CALL。  当特权级变化的时候，堆栈也需要从TSS中进行更换，具体看书上P58.  见书上的表：      调试d代码：  这个代码添加了一个非一致代码段的DESTCODE，以及存储指向它的描述符的选择子的门描述符，和这个门描述符的选择子。以及添加了相应的初始化描述符的逻辑。    可以发现，其使用相应的门转移到了对应的代码段，输出C表示这个代码段确实执行了。    但是此处没有特权级的变更，只是同特权级转移。  所以，从低特权级到高特权级可以使用CALL调用门（因为d的代码都是ring0，而高到低在e讲，所以此处先不实验）。那么高到低该使用什么，笔者根据riscv的惯例猜测使用某种返回。  6. 调试代码，/e/掌握利用调用门进行特权级变换的转移的基本方  法  pmtest5a.asm  这个代码添加了在ring3特权级下的代码段和堆栈段（为什么要有RING3的堆栈段，这在手册中写了，在ret的过程中会装入SS，SP，CS,IP）    然后就在CODE32中有把ring3的栈选择子，栈指针，cs和ip压栈，最后用一个retf进入ring3。    注意0x43和0x2b最低2位是11说明plv=3  ret前的sreg    ret后的sreg    注意到cs和ss已成功装载，其他选择子寄存器会放入空选择符，系统默认给的。  最后c一下    就在ring3输出完3之后一直循环停在这里了。  到此处就实现完高到低的转换，是内核态到用户态转换的关键过程。  pmtest5c.asm  我们想要在ring3中使用调用门转移到ring0的代码。  pmtest5b在5a的基础上把codedest的调用门权限改成ring3，让ring3能够call。注意此处RPL是3，Gate DPL是3，目标代码段的DPL是0。  pmtest5c还添加了一个TSS，用于转到ring0的时候切换堆栈。在retf到ring3之前ltr把TSS选择符加载到TR（任务寄存器）中。注意：此处retf是高到低，直接retf的机制切换堆栈，而ring3进去之后的call是由CPU去换栈，此时才是需要TSS的时候。而且ltr这个指令只有ring0才能用。  debug：  retf前：    retf后：    调用点：      此时call进去了，并且权限变成ring0。  栈也进行了更换。      跳回，0x2b也指示了回到ring3这个权限。同时ss也从栈中换回来。    看到了两个的执行结果。  pmtest5:  将调用局部任务放到Dest中，让程序从ring0->ring3->ring0之后，再从这个ring0调用taska显示L，然后跳转到中间商，切换回实模式，最后回到DOS中。    调用TASKA之前    跳转到中间状态之前。    这里之后就是正常的回到实模式的代码，我们c一下    正常退出到实模式。  所以得出一个简短的结论，特权级要升高则使用CALL调用门，降低则使用retf，其本质就是CS与SS段的检查和赋值。其他形式不会更改CPL。  问题与动手改：  1.见实验步骤1。pm.inc中一共有    宏，在定义描述符宏的时候，描述符LABEL：Descriptor 段基址，段界限，属性  门宏，则为门LABEL: Gate 目标选择子，偏移，Dcount，属性。  2.实模式到保护模式的关键步骤为，lgdt，置位cr0的PE位，关中断，打开A20地址线。关中断在书上写了，因为保护模式下的中断是基于IDT，中断描述符表，也就是说，系统用中断号，当做选择子在IDT中索引相应的中断处理程序的段描述符。包括硬件中断。所以跟实模式下的处理方式不同。A20地址线是因为DOS的实模式下最多索引20位地址，保护模式下可以32位地址，为了兼容所以设置了这个地址线，这个地址线关闭的时候最多索引20位，打开的时候索引20位以上。  3.高权限到低权限代码切换只能由retf进行，retf会把目前栈中的四个元素认为是ss、esp、cs、eip，那么retf的时候就会将自己的ss,esp,cs,eip都恢复为栈中的值。（可以这样想，我们在低特权级的时候要用高特权级的功能，那么call一个调用门过去，我的特权升高了（CPL下降了）用于执行高特权代码，然后我回到用户态那么我retf就需要根据栈中保存的我原来的代码cs去更改我的CPL，那么就是根据cs进行更改）所以CPU会这样设置。  call的话就刚好相反，使用调用门的方法，将当前CPL与RPL与DPL进行比较，CPL>=RPL，CPL>=DPL的时候跳转过去，并且把CPL置位为目标代码的DPL。  jmp对调用门来说是有限制的，也就是说CPL必须要跟目标的DPL相同，那么就不能做特权变换。  所以这三个应该是不能互换的。  4.  1. 自定义添加1个GDT代码段、1个LDT代码段，GDT段内要对一个内存  数据结构写入一段字符串，然后LDT段内代码段功能为读取并打印  该GDT的内容；  最后一句话可能是打印GDT代码段所修改的数据结构的内容吧。GDT感觉按照正常操作没法打印。  所以按照写入数据，然后读出并打印数据的理解，写了以下代码：  首先添加要复制的源数据段    以及复用了代码中原有的目标数据段，只不过增加了一个待填充条目    然后增加CodeCustom01的代码段用于写数据    这里使用rep movsb，因为固定长度移动  然后增加LDT描述符访问的CODE02用于显示对应数据段的内容    此处使用loop进行操作。  最后分别在GDT和LDT中增加这些段的描述符，并定义他们的选择子        然后给他们添加初始化代码：      最后执行，如图：    2. 自定义2个GDT代码段A、B，分属于不同特权级，功能自定义，要求  实现A-->B的跳转，以及B-->A的跳转。  这里实现俩代码段，一个是ring0的    会休眠10个nop，然后把颜色-1    一个是ring3的，不休眠，把颜色+1，那么就预期会有闪烁效果。  他们的门定义，描述符定义如下：      注意call权限切换是CPL置为描述符选择子  复用原代码的ring0堆栈和ring3堆栈以及TSS      初始化这两个段描述符    在这里先放一个1，用于准备显示闪烁效果。    ltr不知道为什么，不能放到内层，不然从0到1,1再call0就会发现到这条指令之后触发某种异常，可能与任务切换有关，目前进度还没看到那里。  闪烁效果见视频：  https://www.bilibili.com/video/BV1L2nZzfEq7/  不断闪烁其内在的原因就是ring0的代码1和ring3的代码2不断切换执行。  关于从ring3切换到ring0时，堆栈切换的问题，TSS会保存一个初始栈指针，每次切换到这个堆栈的时候，栈指针都从初始栈指针开始赋值。  后面测试了一下，发现执行两次ltr就会异常。TODO    应该是ltr之后会认为这是一个任务，然后把相应描述符的忙标志置为1，用于防止递归切换任务。  然后基于以上代码修改，我们修改了CODE02的特权级为RING02，而GATE的Selector的RPL和GATE的Descriptor的DPL为3，此时的过程为，ring0的CODE01，把2的ss，esp，cs，eip压栈，然后retf。进入ring2。然后ring2call一个gate，此时是2call3，也就是说CPL<=RPL,CPL<=DPL，符合调用门，且CPL>=DEST-DPL，符合权限提升的逻辑。所以程序同样能够运行，此时是0-2-0-2···的循环。  进一步强调retf的逻辑是把cs的RPL装入CPL（本质是栈替换到CS，CS本来就是选择子）  call的逻辑是靠自己的权限只要大于等于门和门选择子就可以到权限比自己高的地方。 | | | |
| 1. 实验分析与总结   （对实验过程、结果进行分析总结，提出实验的改进意见） | | | |
| 1.本次实验分析了80x86架构下保护模式的概念，学习了保护模式相关的数据结构，以及分段机制。了解了进入保护模式和退出保护模式的方法。并分析了代码中Descriptor宏。  2.调试了代码映像中a~e的代码，掌握了实模式到保护模式的方法与流程。并区分了call,jmp,retf的区别。掌握了GDT和LDT的构造与切换，并学习了从保护模式切换回实模式的方法。了解了特权级切换时的栈切换过程。初步学习了TSS，了解了任务门的部分内容。掌握了一致代码段、非一致代码段、数据段的权限访问规则，以及CPL、DPL、RPL的关系，学习了如何使用调用门。以及特权级升高和降低的方法。  改进意见：可能先掌握架构手册的全部内容再开始做实验会好一些吧。比如以上的ltr重复出现会发生异常的问题。  以及特权级切换我认为在课堂上把所有常用的例子举一下可能会好一些。 | | | |
| 1. 个人实验贡献与体会（每人各自撰写） | | | |
| 1. 林水利同学：完成了所有任务   个人体会：巩固了汇编知识，初步学习了保护模式的原理，分段的机制，保护模式相关的数据结构，栈切换过程，特权级切换过程，段间切换的过程，代码段转移的特权级check过程。学习了如何索引架构手册。   1. 同学：承担了……任务   个人体会：…… | | | |