|  |  |  |  |
| --- | --- | --- | --- |
| **武汉大学国家网络安全学院教学实验报告** | | | |
| 课程名称 | 操作系统及安全设计 | 实验日期 | 2019/11/8 |
| 实验名称 | 认识保护模式（三）：中断异常 | 实验序号 | 四 |
| 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） | | | |
| **实验目的：**  理解中断与异常机制的实现机理  对应章节：第三章3.4节，3.5节  **实验内容：**  1. 理解中断与异常的机制  2. 调试8259A的编程基本例程  3. 调试时钟中断例程  4. 建立IDT，实现一个自定义的中断，功能可自  定义，如特定键盘组合触发某个动作、电子  钟、自己游走的字符显示、蜂鸣器等  5. 了解IOPL的作用  **完成本次实验要思考的问题：**  1.什么是中断，什么是异常  2.8259A的工作原理是怎样的？  3.如何建立IDT，如何实现一个自定义的中  断  4.如何控制时钟中断  5.IOPL的作用与基本机理 | | | |
| 1. 实验环境及实验步骤   （本次实验所使用的器件、仪器设备等的情况；具体的实验步骤） | | | |
| **实验环境：**  VMwareWorkstationPro 15.5.0  Ubuntu 12.04.5 desktop i386 32位  bochs 2.6.9  **关键技术：**  中断与异常机制的实现  **实验步骤：**   1. 理解中断与异常的机制   中断和异常都是程序执行过程中的强制性转移，转移到相应的处理程序。不管中断还是异常，通俗来讲，都是软件或者硬件发生了某种情形而通知处理器的行为。  每一种中断（异常）都会对应一个中断向量号，而这个向量号通过IDT就与相应的中断处理程序对应起来（见图3.37）。  表3.8中断向量表给出了处理器可以处理的中断和异常列表，以及它们对应的向量号以及其他一些描述。     1. 中断   中断通常在程序执行时因为硬件而随机发生，它们通常用来处理处理器外部的事件，比如外围设备的请求。软件通过执行int n指令也 可以产生中断。   1. 异常   异常则通常在处理器执行指令过程中检测到错误时发生，比如遇到零除的情况。处理器检测的错误条件有很多，比如保护违例、页错误等。   1. 同步中断   同步中断是由cpu内部的电信号产生的中断，其特点为当前执行的指令结束后才转而产生中断，由于有cpu主动产生，其执行点必然是可控的。是当指令执行时由 CPU 控制单元产生，之所以称为同步，是因为只有在一条指令执行完毕后 CPU 才会发出中断，而不是发生在代码指令执行期间，比如系统调用。   1. 异步中断   异步中断是由cpu的外设产生的电信号引起的中断，其发生的时间点不可预期。其他硬件设备依照 CPU 时钟信号随机产生，即意味着中断能够在指令之间发生，例如键盘中断。   1. Fault，Trap，Abort   Fau l t是一种可被更正的异常，而且一旦被更正，程序可以不失连续性地继续执行。当一个fault发生时，处理器会把产生fault的指令之前的状态保存起来。异常处理程序的返回地址将会是产成fault的指令，而不是其后的那条指令。  Trap是一种在发生trap的指令执行之后立即被报告的异常，它也允许程序或任务不失连续性地继续执行。异常处理程序的返回地址将会是产成trap的指令之后的那条指令。  Abort是一种不总是报告精确异常发生位置的异常，它不允许程序或任务继续执行，而用来报告严重错误的。  可以称呼它们为错误、陷阱和终止   1. 保护模式的异常与中断实现机制   在保护模式下，中断机制发生了很大变化，原来的中断向量表已经被IDT所代替，实模式  下能用的BIOS中断在保护模式下已经不能用了。IDT也是个描述符表，叫做中断描述符表（Interrupt Descriptor Table）。IDT中的描述符可以是下面三种之一：  中断门描述符  陷阱门描述符  任务门描述符  IDT的作用是将每一个中断向量和一个描述符对应起来。从这个意义上说，IDT也是一种向量表，虽然它形式上跟实模式下的向量表非常不同。  中断向量到中断处理程序的对应过程    联系调用门我们知道，其实中断门和陷阱门的作用机理几乎是一样的，只不过使用调用门时使用call指令，而这里我们使用 int指令。  中断门和陷阱门的结构如图3.38所示。  对比调用门的结构我们知道，在中断门和陷阱门中BYTE4的低5位变成了保留位，而不再是Param Count。而且，表示TYPE的4位 也将变为0xE（中断门）或0xF（陷阱门）。当然，S位仍将是0。   1. 调试8259A的编程基本例程 2. 外部中断与8259A   外部中断，也就是由硬件产生的中断，另一种是由指令int n产生的中断。  指令int n产生中断时的情形如图所示，n即为向量号，它类似于调用门的使用。  外部中断分为不可屏蔽中断（NMI）和可屏蔽中断两种，分别由CPU的两根引脚NMI和INTR来接收，如图所示。    NMI不可屏蔽，因为它与IF是否被设置无关。NMI中断对应的中断向量号为2。  可屏蔽中断与CPU的关系是通过对可编程中断控制器8259A建立起来的。可以认为它是中断机制中所有外围设备的一个代理，这个代理不但可以根据优先级在同时发生中断的设备中选择应该处理的请求，而且可以通过对其寄存器的设置来屏蔽或打开相应的中断。  与CPU相连的是两片级联的8259A，每个8259A有8根中断信号线，于是两片级联总共可以挂接 15个不同的外部设备。  在BIOS初始化的时候，IRQ0～IRQ7被设置为对应向量号08h～0Fh，而通过中断向量表我们知道，在保护模式下向量号08h-0Fh已经被占用了，所以需要重新设置主从8259A。  通过向相应的端口写入特定的ICW（Initialization Command Word）来实现。主8259A对应的端口地址是20h和21h，从8259A对应的端口地址是A0h和A1h。  ICW共有4个，每一个都是具有特定格式的字节。初始化过程：  1. 往端口20h（主片）或A0h（从片）写入ICW1。  2. 往端口21h（主片）或A1h（从片）写入ICW2。  3. 往端口21h（主片）或A1h（从片）写入ICW3。  4. 往端口21h（主片）或A1h（从片）写入ICW4。  这4步的顺序是不能颠倒的。  我们现在来看一下4个如图3.40所示的ICW的格式。  我们看到，在写入ICW2时涉及与中断向量号的对应，这便是窍门所在了。     1. Pmtest9a   开启中断所要做的工作分设置8259A和建立IDT两大部分，以pmtest8.asm为基础对代码进行修改，形成pmtest9.asm。   1. 初始化8259A   函数Initial8259A  每一次I/O操作之后都调用了一个延迟函数io\_delay以等待操作的完成。      分别往主、从两个8259A各写入了4个ICW。在往主8259A写入ICW2时，我们看到IRQ0对应了中断向量号20h，于是IRQ0～IRQ7就对应中断向量20h～27h；类似地，IRQ8～IRQ15对应中断向量28h～2Fh。对照表3.8我们知道，20h～2Fh处于用户定义  中断的范围内。  在这段代码的后半部分，通过对端口21h和A1h的操作屏蔽了所有的外部中断，写入  OCW（Operation Control Word）。OCW共有3个，OCW1、OCW2和OCW3。我们只在两种情况下用到它  **:**  屏蔽或打开外部中断。  发送EOI给8259A以通知它中断处理结束。  若想屏蔽或打开外部中断，只需要往8259A写入OCW1.OCW1的格式如图所示。 若想屏蔽某一个中断，将对应位设成1。实际上，OCW1是被写入了中断屏蔽寄存器（IMR，全称Interrupt Mask Register）中，当一个中断到达，IMR会判断此中断是否应被丢弃。  当每一次中断处理结束，需要发送一个EOI给8259A， 以便继续接收中断。发送EOI是通过往端口20h或A0h写OCW2来实现的。OCW2的格式如图所示。  发送EOI给8259A可以由如下代码完成： 见（pmtest9，时钟中断）  mov al, 20h  out 20h或A0h, al    在相应的位置添加调用Init8259A的指令之后，对8259A的操作就结束了（见pmtest9b）   1. 建立IDT   IDT放进一个单独的段中    利用了NASM的%rep预处理指令，将每一个描述符都设置为指SelectorCode32:SpuriousHandler的中断门。  SpuriousHandler也很简单，在屏幕的右上角打印红色的字符“!”，然后  进入死循环。    加载IDT的代码    在执行lidt之前用cli指令清IF位，暂时不响应可屏蔽中断。     1. Pmtest9b   pmtest9a完成了保护模式下中断异常处理机制的初始化，但并没有利用中断来做任何事。下面就继续修改代码。  在[SECTION .s32]中添加代码    由于IDT中所有的描述符都指向SelectorCode32:SpuriousHandler处，所以，无论我们添加的代码调用几号中断，都应该在屏幕的右上角打印出红色的字符。  运行，屏幕右上角出现红色的“!”，并且程序进入死循环。     1. pmtest9c   修改IDT，把第80h号中断单独列出来，并新增加一个函数来处理这个中断：UserIntHandler。UserIntHandler与SpuriousHandler类似，只是在函数末尾通过iretd指令返回，而不是进入死循环         1. 调试时钟中断例程 2. 可屏蔽中断，设置8259A,IDT   打开时钟中断（IRQ0）  可屏蔽中断与NMI的区别在于是否受到IF位的影响，而8259A的中断屏蔽寄存器（IMR）也影响着中断是否会被响 应。所以，外部可屏蔽中断的发生就受到两个因素的影响，只有当IF位为1，并且IMR相应位为0时才会发生。  想打开时钟中断的话，一方面不仅要设计一个中断处理程序，另一方面还要设置IMR，并且设置IF位。设置IMR 可以通过写OCW2来完成，而设置IF可以通过指令sti来完成。    修改初始化8259A的代码，不再屏蔽IR(0)(主8259AIR0为时钟中断)。（11111110，第0位0，开启接收定时器中断）      修改IDT，20h为时钟中断程序     1. 时钟中断程序   把屏幕第0行、第 70列的字符增一，变成ASCII码表中位于它后面的字符。如果我们在调用80h号中断之后打开中断的话，由于第0行、第70列处已被写入字符I，所以第一次中断发生时那里会变成字符J，再一次中断则变成K，以后每发生一次时钟中断，字符就会变化一次，就会 看到不断变化中的字符。  发送EOI（设置IMR，接收下一个(定时器)中断）  iretd  当每一次中断处理结束，需要发送一个EOI给8259A，以便继续接收中断。发送EOI是通过往端口20h或A0h写OCW2来实现的。     1. 设置IF   调用80h号中断之后执行sti来打开中断（STI指令置IF为1，接收可屏蔽中断），时钟中断程序的效果就应该可以看到了。设置死循环原因是有一个问题：程序马上会继续执行，可能没等第一个中断发生程序已经执行完并退出了。     1. 时钟向量初始化   在IDT初始化的时候.20h中断设置为对应自己写的时钟中断程序。  在8529A初始化的时候（见pmtest9a的描述），设置IR(0)对应.20h中断。所以每次有外部时钟信号，都会调用时钟中断程序。   1. 效果        1. 建立IDT，实现一个自定义的中断，功能可自定义，如特定键盘组合触发某个动作、电子钟、自己游走的字符显示、蜂鸣器等   在pmtest9.asm的基础上修改，实现功能：  利用键盘中断，检测到键盘输入，就把上述第70列的字符+1.同时在第0行第72列显示键盘输入的字符。  修改IDT：定义21h中断对应键盘中断。（报告上面已经解释了，8259a的初始化设置使IRQ1对应21h）    修改8259a初始化函数，使得开启键盘中断  11111101b。0对应IRQ1键盘中断    编写KeyBoardHandler函数：    in al,60h 为从键盘缓冲区读取输入的字符。  保存为p.asm，编译为p.com  测试  注意，键盘中断包含按下与松开的扫描码和断码，所以按下按键并松开是两次中断。同时一直按住按键会一直有中断。  按下ctrl并松开：    观察到70列字符增加2.同时72列显示￥，查扩展ascii表  对应ascii码为157  157-80h=29  并不与ctrlascii对应。  查资料得知这样读取的al中存储的是扫描码  <https://eason.blog.csdn.net/article/details/7754097>  扫描码与ascii不同，故显示的不是键盘的按下的字符。  解决办法为用cmp  Cmp al, ‘r’   1. 了解IOPL的作用   IOLP是实现I/O保护的关键机制，位于eflag寄存器的12,13位。    指令in、ins、out、outs、cli、sti只有在CPL≤IOPL时才能执行。  这些指令被称为I/O敏感指令。如果低特权级的指令试图访问这些I/O敏感指令将会导致常规保护错误（#GP）。  可以改变IOPL的指令只有popf和iretd，但只有运行在ring0的程序才能将其改变。运行在低特权级下的程序无法改变IOPL，不  过，如果试图那样做的话并不会产生任何异常，只是IOPL不会改变，仍然保持原样。  指令popf同样可以用来改变IF（就好像执行了cli和sti）。然而，在这种情况下，popf也变成了I/O敏感指令。只有CPL≤IOPL  时，popf才可以成功将IF改变，否则IF将维持原值，不会产生任何异常。 | | | |
| 1. 实验过程分析   （实验分工，详细记录实验过程中发生的故障和问题，进行故障分析，说明故障排除的过程及方法。根据具体实验，记录、整理相应的数据表格等） | | | |
| **遇到的问题及解决：**   1. **不清楚 IDT中%rep 32的作用**     通过查资料得知这个预处理指令的作用类似于循环。然后在IDT的作用就是连续定义多各descriptor，相当于定义多个中断。比如上面的%rep 32，就把int 0 - int 31 全都对应SpuriousHandler。   1. **编写KeyBoardHandler遇到的问题**   键盘中断定义为    问题就在于原来系统中断16h是键盘，现在根据IDT变了,变为指向\_SpuriousHandler，不能这么用。改为    只能读一次键盘  问题在于    internal keyboard buffer full, ignoring scancode  键盘缓冲区满了。改为    60h为cpu对应键盘的输入缓冲端口  In al,60h就从键盘缓冲中读取1个byte，到al中。然后在第72列显示读取的字符。  按下时送入的为“扫描码”  注意按键按下有中断，持续按不松开会一直有中断送入，所以第70列的字符会一直改变  同时按键松开也有中断，参考  <https://blog.csdn.net/sxhelijian/article/details/72869738>  此时送入的为“断码”，为断码=扫描码+80h  所以松开按键，70列字符会再增加一次，同时72列显示刚刚按下的按键对应的断码   1. **第72列显示的输入字符与键盘按下的字符不同**   查资料得知al存扫描码，与ascii不同  <https://eason.blog.csdn.net/article/details/7754097>  **实验结果记录：**  pmtest9a    pmtest9b    pmtest9c    pmtest9      自己编写的实现键盘中断的p | | | |