操作系统原理实验报告

实验四：中断处理与系统调用

数据科学与计算机学院　17大数据与人工智能

17341015　陈鸿峥

一、 实验目的

- 学习中断中断机制知识，掌握中断处理程序设计的要求
- 学习通过汇编程序实现时钟中断处理

二、 实验要求

- 操作系统工作期间，利用时钟中断，在屏幕24行79列位置轮流显示‘—’、‘/’和‘\’，适当控制显示速度，以方便观察效果。
- 编写键盘中断响应程序，原有的你设计的用户程序运行时，键盘事件会做出有事反应：当键盘有按键时，屏幕适当位置显示“OUCH! OUCH!”。
- 在内核中，对33号、34号、35号和36号中断编写中断服务程序，分别在屏幕1/4区域内显示一些个性化信息。再编写一个汇编语言的程序，作为用户程序，利用`int 33`、`int 34`、`int 35`和`int 36`产生中断调用你这4个服务程序。

三、 实验环境

具体环境选择原因已在实验一报告中说明。

- Windows 10系统 + Ubuntu 18.04(LTS)子系统
- gcc 7.3.0 + nasm 2.13.02 + GNU ld (Binutils) 2.3.0
- GNU Make 4.1
- Oracle VM VirtualBox 5.2.8
- Bochs 2.6.9
- Sublime Text 3

虚拟机配置：内存4M，无硬盘，1.44M虚拟软盘引导。

四、 实验方案

本次实验的关键在于写中断向量表。中断向量表存放在物理地址`0x0000`到`0x03ff`，大小为1KB(1024B)。每个中断在中断向量表中占輾轂，高位为中断处理程序的段地址，低位为中断处理程序的偏移量。且每个中断对应着一个中断标号，计算标号为i的中断入口地址可以通过$4 \times i$得到。

而写向量表的操作，我已在实验二中实现宏汇编代码，因此本次实验复用就比较简单。

1. 时钟中断

需要通过可编程中断控制器(Programmable Interrupt Controller, PIC)（见下表），实现中断响应。

PIC 1	硬件中断	PIC 2	硬件中断
0	时钟	8	实时钟
1	键盘	9	通用IO
2	PIC 2	10	通用IO
3	COM 2	11	通用IO
4	COM 1	12	通用IO
5	LPT 2	13	协处理器
6	软盘	14	IDE总线
7	LPT 1	15	IDE总线

具体到时钟中断，即对8号中断进行编程。将0x08放入0x20的位置，处理时钟中断函数的入口放入0x22。注意时钟中断最后要告诉硬件端口已经处理完中断，并正常返回，见下面程序。

```
push eax
mov al, 20h
out 20h, al
out 0A0h, al
pop eax
iret
```

实现无敌风火轮则是通过一个计数器count递减，如果count递减为0，则显示字符char，并将char的值修改为下一符号（|变为/，/变为\，\变为/）；否则通过上面代码正常返回。

2. 键盘中断

在实验二中，我已经实现int 20和int 21的软件中断，用于用户键入Ctrl+C实现返回内核。

而本次实现要实现OUCH!OUCH!效果，只需在原int 20中断中判断到按键就显示字符，然后再判断是否Ctrl+C决定是否返回内核。

3. 其他软件中断

实现了int 33h、int 34、int 35和int 36中断，分别调用我原来的四个用户程序。如下程序所示，其中loadPrg为我自己编写的宏汇编指令。

```
INT33H:
    loadPrg 1
    iret
```

用户程序5实现了这几个中断的连续调用。

4. 其他新特性

- 完善Shell交互界面，如实现**退格键**输入，实现**自动滚屏**而不是清屏等
- 添加`clr`清屏指令
- 添加**画框**的用户程序

五、 实验结果

无敌风火轮‘—’、‘/’和‘\’，见图1右下角所示。注意这里没有办法截取动画过程，故这里只给出了代表性的三帧作为样例。

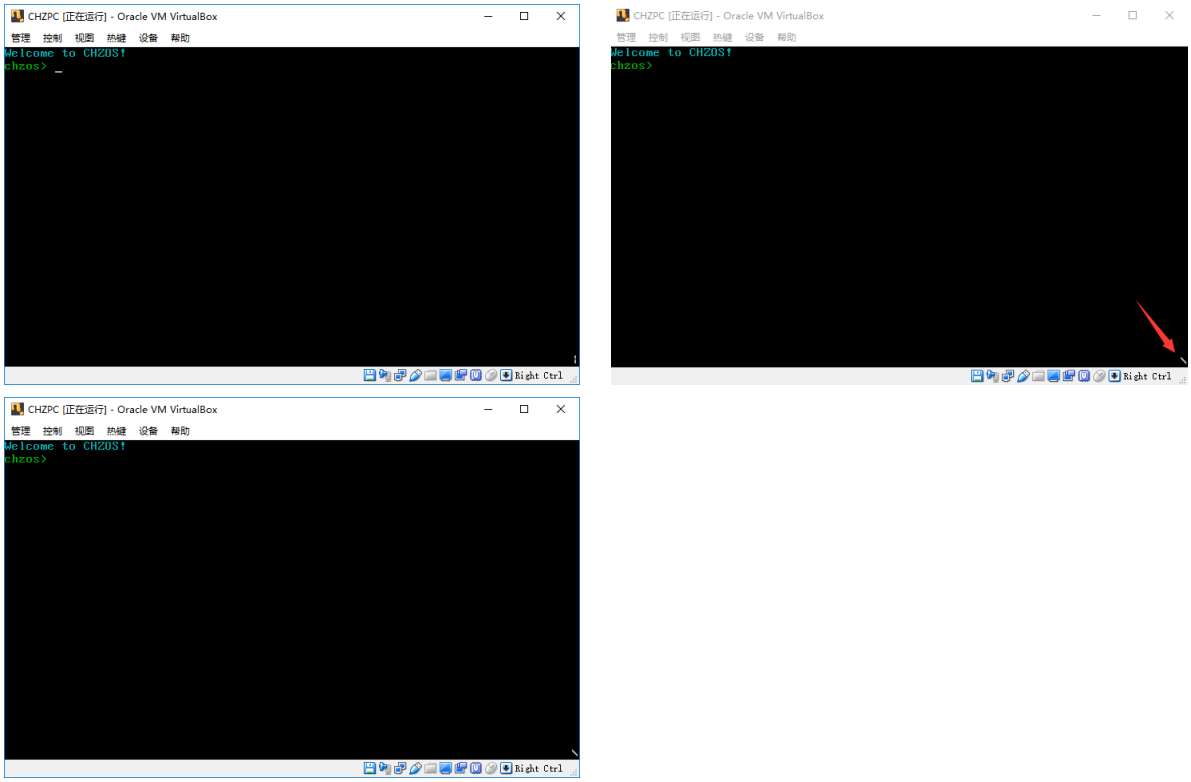

图 1: 无敌风火轮

所有用户程序执行过程中，碰到按键会在屏幕中间红色高亮显示“OUCH!OUCH!”，如图2所示。

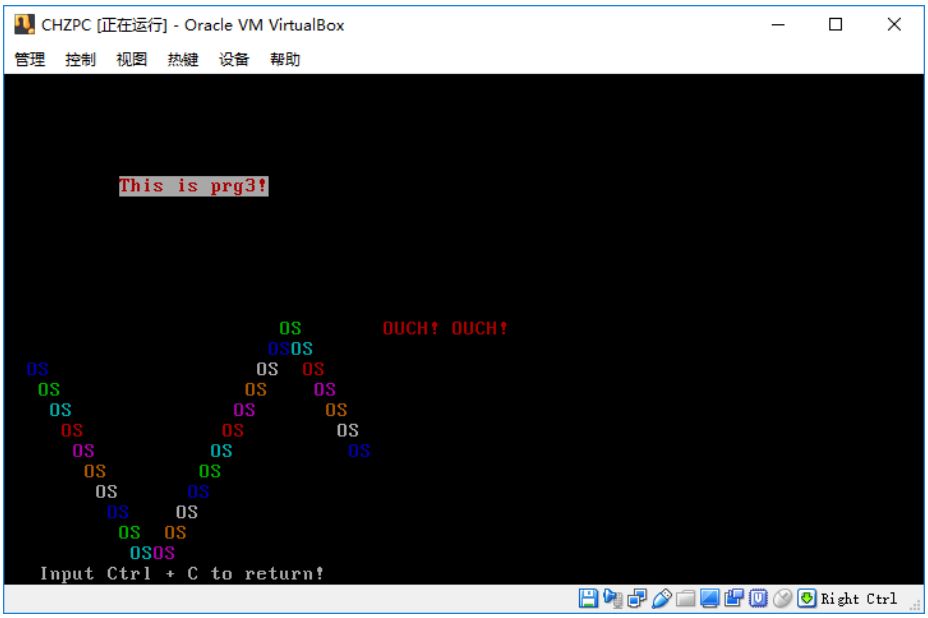

图 2: 键盘中断显示

连续依次调用`int 33h`、`int 34`、`int 35`和`int 36`中断，这里为了显示方便，没有添加清屏指令，结果如图3所示。注意这几条指令都是顺序执行的，即中断调用可**正常返回父程序**，并进入下一行执行，所以才有办法保持原程序产生的结果不发生改变。

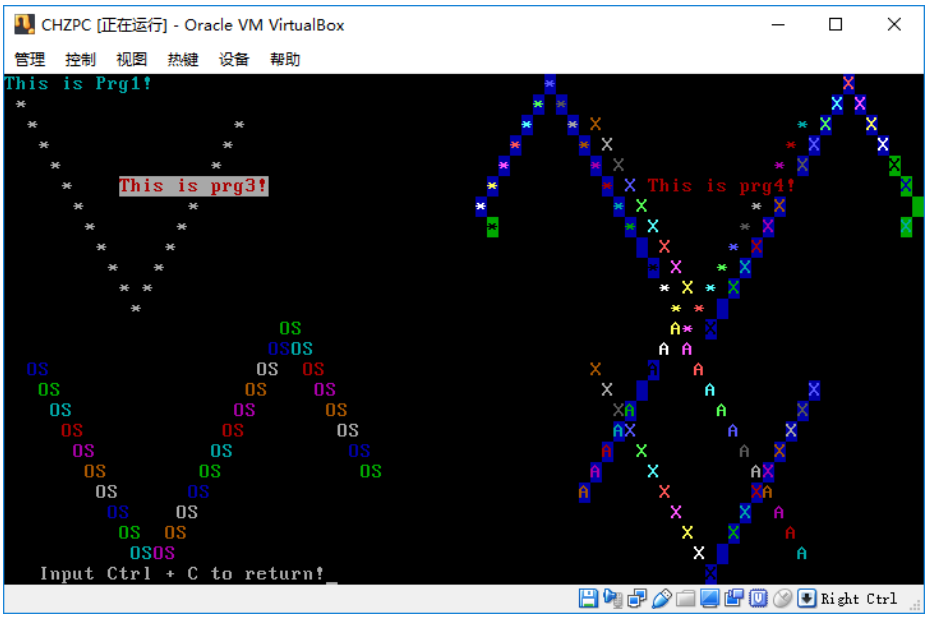

图 3: 连续中断调用

自动滚屏如图4所示。可以见到最上面上一次`help`的输出结果被截断了。

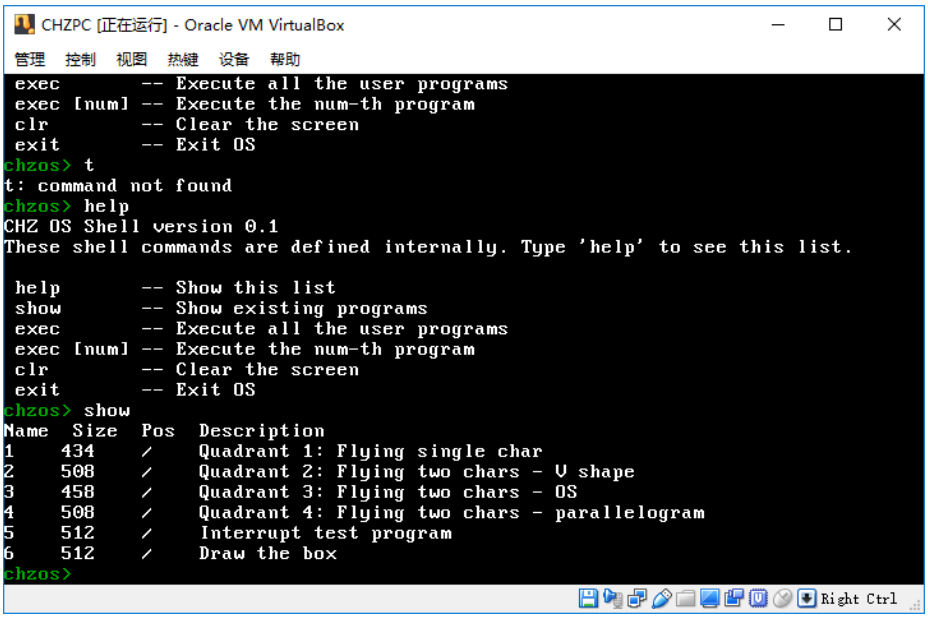

图 4: 自动滚屏

由于无敌风火轮已经占用了时钟中断，故自动画框变为用户程序，效果如图5所示。

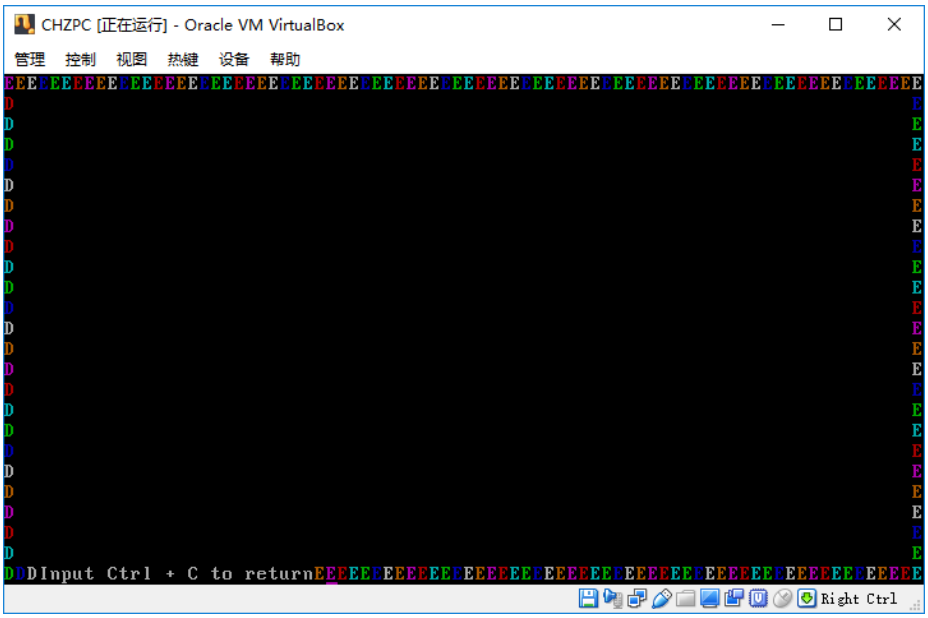

图 5: 自动画框

六、 实验总结

由于实验二我已经自行实现了中断，因此宏汇编代码可以复用，本次实验就比较简单。

但还是遇到了很多问题，原来我写用户程序时，都没有将其当成一个函数，即最后都是陷入死循环`jmp $`，而不是返回原处。如果用户程序嵌套调用，或者直接在用户程序中调用中

断，那很可能回不到原来的断点。这对于正常的用户程序运行显然是不行的。因而我认真学习了C编译出来的汇编代码，发现它每一个编译出来的函数体都一定会有ret函数，主函数main也是，这导致外部程序可以很方便地call。因此，我认为我自己写的用户程序也应该是这样，于是便在我的每一个用户程序的主函数中添加ret指令。

但这里很重要一点在于不能破坏堆栈，一旦在用户程序执行中堆栈被破坏，ret指令就不能正确找到返回地址，因而跳转到非法位置。这一点我也单步调试了很长时间，最终发现不管哪一个段寄存器被修改了（因为在显存显示时需要设置段地址，故段寄存器会改变），原来的堆栈就会被修改。因此，在每次显示字符之前需要将段寄存器es、gs进栈，显示完后再将段寄存器出栈，这样就可以保证堆栈不会被破坏，最终ret可以正常返回父用户程序。

另一个麻烦的点在于，时钟中断总是会和其他中断冲突（主要是读入字符的int 16中断）。查了很多资料，设置好中断处理规则（关中断、中断处理、开中断），倒腾了很久，最终才终于实现两者都能工作。

总的来说，本次实验还是比较轻松愉悦的。

七、 参考资料

1. 李忠，王晓波，余洁，《x86汇编语言-从实模式到保护模式》，电子工业出版社，2013
2. 周杰英，张萍，郭雪梅，黄方军，《微机原理、汇编语言与接口技术》，人民邮电出版社，2011
3. Little OS Book, https://littleosbook.github.io/book.pdf
4. Interrupts, https://wiki.osdev.org/Interrupts
5. 8259 PIC, https://wiki.osdev.org/PIC#Programming_the_PIC_chips
6. 函数调用规则，https://www.cs.princeton.edu/courses/archive/spr11/cos217/lectures/15AssemblyFunctions.pdf

附录 A. 程序清单

由于程序太多，请直接见压缩文件。

附录 B. 附件文件说明

序号	文件	描述
1	`bootloader.asm`	主引导程序
2	`os.asm`	内核汇编部分
3	`kernel.c`	内核C部分
4	`Makefile`	编译指令文件
5	`link.ld`	链接文件
6	`bochsrc.bxrc`	bochs调试文件
7	`mydisk.img`	核心虚拟软盘
8∼13	`prgX.asm`	用户程序
14∼18	`/include`	内核头文件