

实验四 中断机制编程 实验报告

数据科学与计算机学院 计算机科学与技术 2016 级

王凯祺 16337233

2018 年 4 月 8 日

1 实验目的

- 学习中断机制知识，掌握中断处理程序设计的要求
- 掌握设计时钟中断处理程序的方法
- 掌握设计键盘中断处理程序的方法
- 掌握设计系统调用的方法

2 实验要求

- 操作系统工作期间，利用时钟中断，在屏幕最右端的 25×3 的区域上弹球。字母 A 从该区域左上角向下 45 度射出，遇到屏幕边缘反弹，如此类推，还可加上变色闪耀等效果。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 34 号、35 号、36 号和 37 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 34、int 35、int 36 和 int 37 产生中断调用你这 4 个服务程序。
- 扩充系统调用，实现三项以上新的功能，并编写一个测试所有系统调用功能的用户程序。

3 实验步骤

在本实验中，我使用 NASM 设计时钟中断处理程序，用 MASM + TCC 设计键盘处理程序及系统调用处理程序。

3.1 设计时钟中断处理程序

这个程序我使用了弹球的代码，删去循环，并加上中断返回后得到的。

做这个程序的时候我遇到了一个问题：修改时钟中断（08h）的中断向量表后，时钟中断程序能正常运行，但键入字符无响应。由于键入字符我使用的是 16h 中断，我怀疑是 16h 中断出问题了，

于是去群里问了一下其他同学有没有类似的问题，结果真有。同学说加了 pusha popa 就可以正常运行，但不知道为什么。

为什么它加上这两句就正常了，而不加不行？我分析了一下，pusha popa 是将当前所有寄存器压入栈的指令，以及将栈顶元素出栈恢复到所有寄存器中的指令。中断发生时，CPU 寄存器状态以及标志位应由中断服务程序负责保存，直到中断服务程序运行结束前进行恢复。若不保存寄存器状态，则中断返回后的寄存器状态不为中断调用前的寄存器状态，就会导致程序出错。

在本例中，受影响最大的寄存器应是 ds 寄存器。数据段寄存器被修改，中断返回后内核不能访问数据段，造成键入无响应。

以下是时钟中断处理程序的源代码 (NASM 格式):

```
1 ; 程序源代码 (stone.asm)
2 ; 本程序在文本方式显示器上从左边射出一个*号,以45度向右下运动,撞到边框后反射,如此类推.
3 ; 凌应标 2014/3
4 ; 王凯祺 2018/3
5 ; NASM汇编格式
6     delay    equ 4                ; 计时器延迟计数,用于控制画框的速度
7     sx       equ 0
8     sy       equ 77
9     wx       equ 25
10    wy       equ 3
11    org 100h                    ; 程序加载到100h,可用于生成COM
12
13 start:
14     push ds
15     push es
16     mov ax, 940h
17     mov ds, ax
18     mov es, ax
19     mov ax, 0B800h              ; 文本窗口显存起始地址
20     mov gs, ax                  ; GS = B800h
21 loop1:
22     dec word[count]             ; 递减计数变量
23     jnz end                     ; >0: 跳转;
24     mov word[count], delay
25
26     xor dx, dx
27     mov word ax, [t]
28     xor bx, bx
29     mov word bx, wx
30     add bx, bx
31     sub bx, 2
32     div bx
33     cmp dx, wx
34     jb xok
35     sub bx, dx
36     mov dx, bx
37 xok:
38     add dx, sx
```

```

39     mov word [x], dx
40
41     xor dx, dx
42     mov word ax, [t]
43     xor bx, bx
44     mov word bx, wy
45     add bx, bx
46     sub bx, 2
47     div bx
48     cmp dx, wy
49     jb yok
50     sub bx, dx
51     mov dx, bx
52 yok:
53     add dx, sy
54     mov word [y], dx
55
56 show:
57     xor dx, dx
58     mov word ax, [t]
59     mov bx, 15
60     div bx
61     mov cx, dx
62     add cx, 1
63     xor ax, ax           ; 计算显存地址
64     mov ax, word[x]
65     mov bx, 80
66     mul bx
67     add ax, word[y]
68     mov bx, 2
69     mul bx
70     mov bx, ax
71     mov ah, cl           ; 0000: 黑底、1111: 亮白字 (默认值为07h)
72     mov al, byte[char]   ; AL = 显示字符值 (默认值为20h=空格符)
73     mov [gs:bx], ax      ; 显示字符的ASCII码值
74     inc word [t]
75
76 end:
77     pop es
78     pop ds
79     mov al, 20h          ; AL = EOI
80     out 20h, al          ; 发送EOI到主8529A
81     out 0A0h, al         ; 发送EOI到从8529A
82     iret                 ; 从中断返回
83
84 datadef:
85     count dw delay
86     t      dw 0
87     x      dw 0
88     y      dw 0

```

```
89
90 char db 'A'
```

3.2 设计键盘中断处理程序

键盘中断也有一个大坑！修改 09h 中断后，只要敲击键盘就会在屏幕上随机的一个位置显示两个 OUCH!（键盘按下和键盘弹起），但是敲击字符没有回显，也就是 16h 中断坏掉了。

很快我就在网上查到了解释，由于 16h 中断会调用原 09h 中断，故修改 09h 中断会使原 16h 中断失效。

老师给出的一个方案是：在加载用户程序前将 09h 中断向量进行备份，然后修改 09h 中断向量；用户程序结束后，还原原 09h 中断向量。这要求用户程序不能有键盘输入（不能调用 16h 中断），所以我还修改了用户程序，从按任意键退出程序改为一定时间之后自动退出程序。

自从学会链接 TCC，写起程序来我都不想用汇编了。我喜欢写一个 loader 然后直接跳到 C 程序里执行。以下是键盘中断处理程序的源代码（TASM + TCC 格式）：

```
1 ; i9loader.asm
2 extrn _int_9_main:near ;声明一个c程序函数int_9_main
3
4 .8086
5 _TEXT segment byte public 'CODE'
6 DGROUP group _TEXT,_DATA,_BSS
7     assume cs:_TEXT
8 org 0100h
9 start:
10     push ax
11     push bx
12     push cx
13     push dx
14     push ds
15     push es
16     mov ax, 0980h
17     ;mov ax, cs
18     mov ds, ax ; DS = CS
19     mov es, ax ; ES = CS
20     call near ptr _int_9_main
21
22     in al, 61h ; get value of keyboard control lines
23     mov ah, al ; save it
24     or al, 80h ; set the 'enable kbd' bit
25     out 61h, al ; and write it out the control port
26     xchg ah, al ; fetch the origin control port value
27     out 61h, al ; and write it back
28     ; http://webpages.charter.net/danrollins/techhelp/0106.HTM
29
30     mov al, 20h
31     out 20h, al
32
33     pop es
```

```

34     pop ds
35     pop dx
36     pop cx
37     pop bx
38     pop ax
39     iret
40
41 _TEXT ends
42 ;*****DATA segment*****
43 _DATA segment word public 'DATA'
44 _DATA ends
45 ;*****BSS segment*****
46 _BSS     segment word public 'BSS'
47 _BSS ends
48 ;*****end of file*****
49 end start

```

```

1 // i9main.c
2 int tx = 15, ty = 24;
3
4 void yan_ge_putchar(char c, int ttx, int tty) {
5     char x, y;
6     int pos;
7
8     /* store origin position */
9     asm mov ah, 03h
10    asm mov bx, 0
11    asm int 10h
12    asm mov x, dh
13    asm mov y, dl
14
15    /* set position */
16    asm mov ah, 02h
17    asm mov bh, 0
18    asm mov dh, ttx
19    asm mov dl, tty
20    asm int 10h
21
22    /* write letter */
23    asm mov ah, 0ah
24    asm mov al, c
25    asm mov bh, 0
26    asm mov cx, 1
27    asm int 10h
28
29    /* restore origin position */
30    asm mov ah, 02h
31    asm mov bh, 0
32    asm mov dh, x
33    asm mov dl, y

```

```

34     asm int 10h
35 }
36
37 int_9_main(){
38     char x;
39     asm in al, 60h
40     asm mov x, al          // x 中表示输入的字符（在本例中未用到）
41     tx = (tx * 83 + 79) % 23; // 显示 ouch! 的 x 坐标，伪随机数
42     ty = (ty * 83 + 79) % 71; // 显示 ouch! 的 y 坐标，伪随机数
43     yan_ge_putchar('o', tx, ty+0);
44     yan_ge_putchar('u', tx, ty+1);
45     yan_ge_putchar('c', tx, ty+2);
46     yan_ge_putchar('h', tx, ty+3);
47     yan_ge_putchar('!', tx, ty+4);
48 }

```

3.3 设计系统调用测试程序

在写系统调用处理程序之前，先写一个短小精悍的测试程序来测试系统调用是否正确。
 以下是系统调用测试程序的源代码（NASM 格式）：

```

1  org 100h
2
3  mov ax, 900h
4  mov ds, ax
5  mov es, ax
6
7  int 34h
8  int 35h
9  int 36h
10 int 37h
11 ret

```

3.4 设计系统调用处理程序

会处理上面两个中断之后，这个中断就简单了。可我还是遇到了坑：

我将该系统调用处理程序放在 19 号扇区。当我试图从磁盘中读取第 19 号扇区、加载到内存，并运行测试程序时，始终无法成功调用该中断。

本来我怀疑程序放错内存位置，导致被 BIOS 的某些程序覆盖掉了。可是我换了好多个位置，都是一样的错误，无法调用该中断。

我开始把目光投向磁盘，果然不一会儿我就发现了一篇文章：《软盘结构（磁头号 and 起始扇区的计算方法）》（<https://blog.csdn.net/littlehedgehog/article/details/2147361>），上面明确记载：扇区号编号从 1 - 18。如果要表示 19 号扇区，应用“1 磁头 0 柱面 1 起始扇区”来表示。

以下是系统调用处理程序的源代码（TASM + TCC 格式）：

```

1  extrn _int_34_main:near ; 声明一个c程序函数int_34_main
2
3  .8086

```

```

4  _TEXT segment byte public 'CODE'
5  DGROUP group _TEXT,_DATA,_BSS
6      assume cs:_TEXT
7  org 0100h
8  start:
9      push ax
10     push bx
11     push cx
12     push dx
13     push ds
14     push es
15     mov ax, 0d00h
16     ;mov ax, cs
17     mov ds, ax          ; DS = CS
18     mov es, ax          ; ES = CS
19     call near ptr _int_34_main
20
21     mov al, 20h
22     out 20h, al
23
24     pop es
25     pop ds
26     pop dx
27     pop cx
28     pop bx
29     pop ax
30     iret
31
32 _TEXT ends
33 ;*****DATA segment*****
34 _DATA segment word public 'DATA'
35 _DATA ends
36 ;*****BSS segment*****
37 _BSS segment word public 'BSS'
38 _BSS ends
39 ;*****end of file*****
40 end start

```

```

1  void yan_ge_putchar(char c, int ttx, int tty) {
2      char x, y;
3      int pos;
4
5      /* store origin position */
6      asm mov ah, 03h
7      asm mov bx, 0
8      asm int 10h
9      asm mov x, dh
10     asm mov y, dl
11
12     /* set position */

```

```

13     asm mov ah, 02h
14     asm mov bh, 0
15     asm mov dh, ttx
16     asm mov dl, tty
17     asm int 10h
18
19     /* write letter */
20     asm mov ah, 0ah
21     asm mov al, c
22     asm mov bh, 0
23     asm mov cx, 1
24     asm int 10h
25
26     /* restore origin position */
27     asm mov ah, 02h
28     asm mov bh, 0
29     asm mov dh, x
30     asm mov dl, y
31     asm int 10h
32 }
33
34 void yan_ge_puts(const char* s, int ttx, int tty) {
35     int i = 0;
36     for (; s[i] != '\000'; ++i) {
37         yan_ge_putchar(s[i], ttx, tty + i);
38     }
39 }
40
41 int_34_main(){
42     yan_ge_puts("I_am_int_34h", 6, 10);
43     yan_ge_puts("16337233_Wangkaiqi", 7, 10);
44 }

```

3.5 加载中断及中断向量表

加载中断和修改中断向量表需考虑以上提到的两种情况：

- 09h 中断的向量表在修改前需备份
- 磁盘扇区号需转换为（磁头，柱面，起始扇区）的格式

在操作系统内核上加上一个 C 模块，模块的功能是将程序从第 diskAddr 个扇区读取到 memSeg:memAddr 位置，并修改中断向量 intVec 指向 memSeg:memAddr 位置。

```

1 int cs9, ip9;
2 unsigned char zhumian, citou, qishishanqu;
3
4 void load_int_2(int diskAddr, int memSeg, int memAddr, int intVec) {
5     int oldintVec;
6     unsigned char id;

```

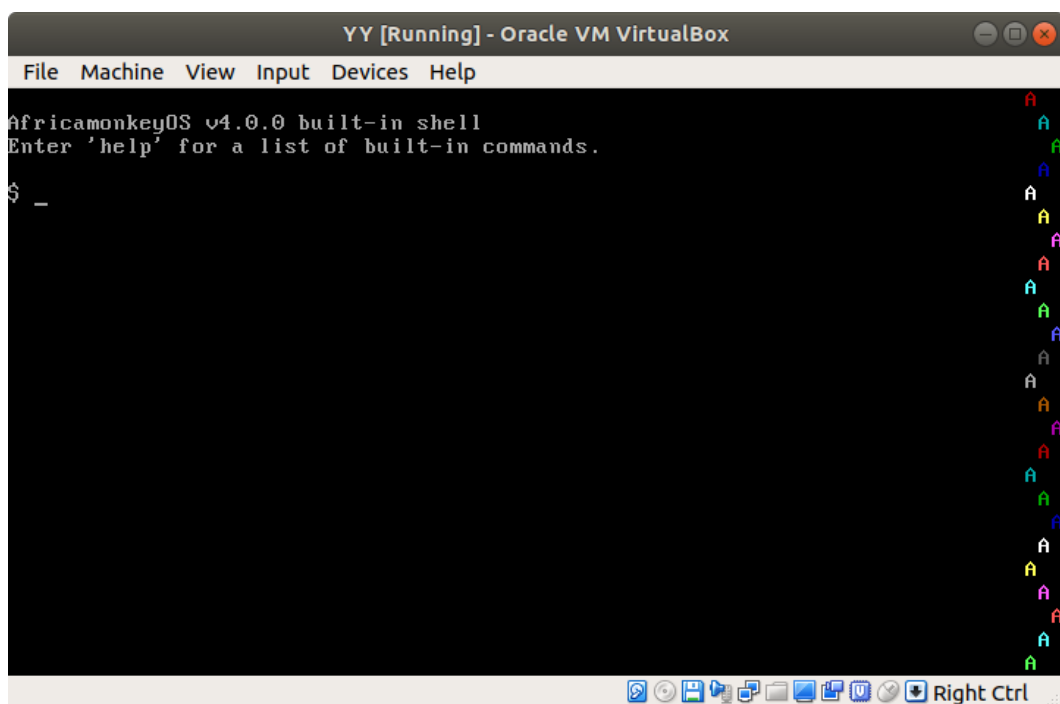


```

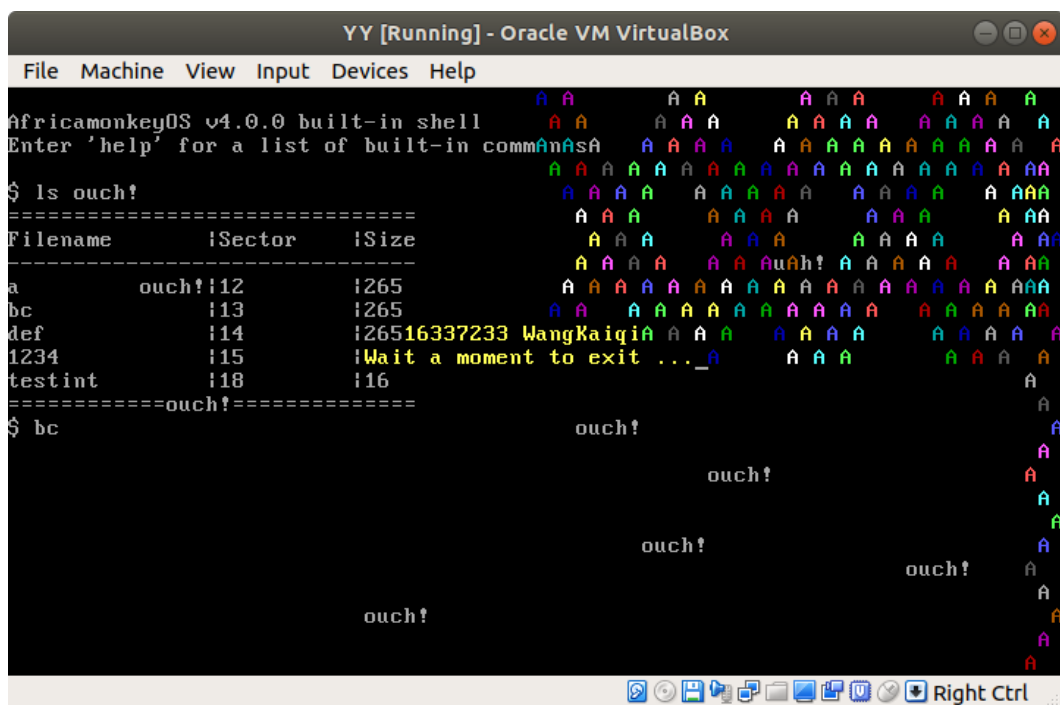
7      id = diskAddr;
8      id = id - 1;
9      zhumian = id / 18;
10     citou = zhumian % 2;
11     zhumian = zhumian / 2;
12     qishishanqu = id % 18 + 1;
13
14     oldintVec = intVec;
15     asm push es
16     asm mov ax, memSeg
17     asm mov es, ax
18     asm mov bx, memAddr
19     asm mov cl, qishishanqu
20     asm mov ah, 2
21     asm mov al, 1
22     asm mov dl, 0
23     asm mov dh, citou
24     asm mov ch, zhumian
25     asm int 13h
26     asm pop es
27
28     asm push ds
29     asm mov ax, 0h
30     asm mov ds, ax
31     intVec = intVec * 4;
32     asm mov bx, intVec
33     if (oldintVec == 9) {
34         asm mov ax, [bx]
35         asm mov ip9, ax
36     }
37     asm mov ax, memAddr
38     asm mov [bx], ax
39
40     intVec = intVec + 2;
41     asm mov bx, intVec
42     if (oldintVec == 9) {
43         asm mov ax, [bx]
44         asm mov cs9, ax
45     }
46     asm mov ax, 0
47     asm mov [bx], ax
48     asm pop ds
49 }

```

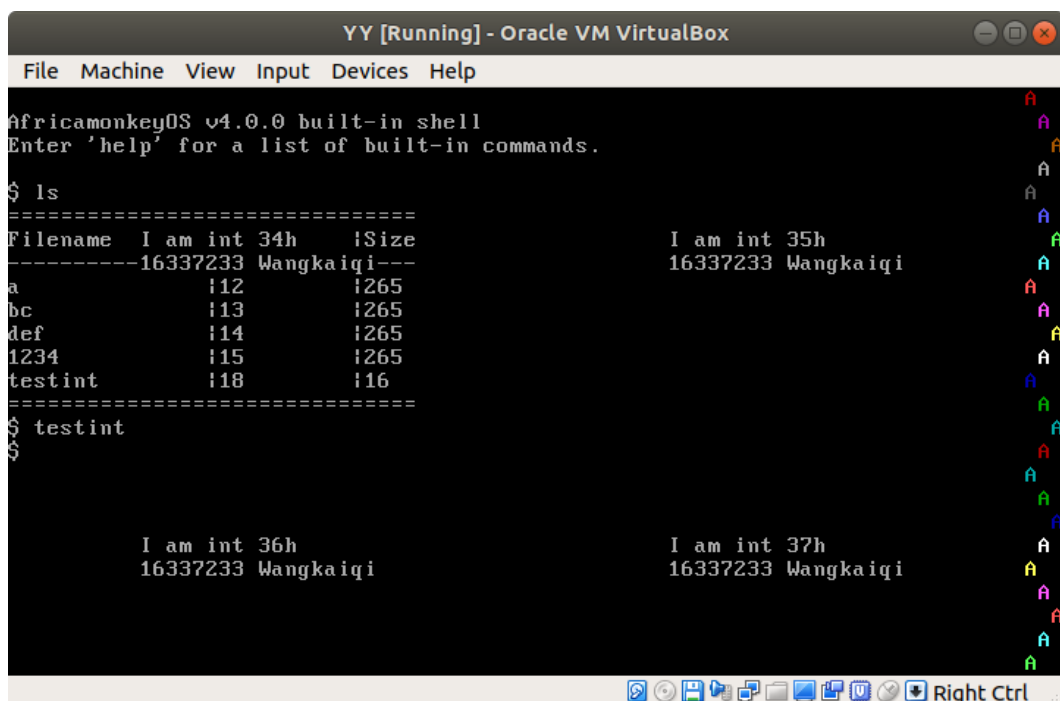
4 实验结果



上图展示的是时钟中断功能。在屏幕右侧会有'A' 字符不断反弹。



上图展示的是键盘中断功能。输入前 4 个用户程序（a, bc, def, 1234）即可响应键盘中断。键盘按下和弹起时，会在屏幕上随机一个位置显示 ouch! 。



```
YY [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

AfricamonkeyOS v4.0.0 built-in shell
Enter 'help' for a list of built-in commands.

$ ls
=====
Filename I am int 34h  |Size      I am int 35h
-----16337233 Wangkaiqi--
a         |12        |265
bc        |13        |265
def       |14        |265
1234      |15        |265
testint   |18        |16
=====
$ testint
$

I am int 36h      I am int 37h
16337233 Wangkaiqi 16337233 Wangkaiqi
```

上图展示的是系统调用功能。输入第 5 个程序（testint）即可依次调用 int 34h, 35h, 36h, 37h 。每个程序分别在对应 1/4 位置显示相关信息。

5 实验总结

这次实验让我深入理解了中断服务程序的工作原理。中断响应后，先到内存指定位置找到中断向量表，然后跳转到中断服务程序。中断服务程序需先保存寄存器。中断服务程序完成后，需先还原寄存器，然后调用中断返回指令。所以在做实验的时候，就需要修改操作系统内核，使操作系统内核修改中断向量表，才能实现自定义中断服务程序。