

《操作系统实验》

实验报告

(实验四五)

学 院 名 称 : 数据科学与计算机学院

专业 (班级) : 16 计科 2 班

学 生 姓 名 : 杨志成

学 号 : 16337281

时 间 : 2018 年 3 月 17 日

目录

一，实验目的-----	3
二，实验要求-----	3
三，实验方案-----	4
(1) 基础原理-----	4
(2) 实验工具与环境-----	7
(3) 程序流程-----	8
(4) 程序模块功能-----	9
(5) 代码文档组成-----	9
(6) 实现效果-----	10
四，实验过程及实验结果-----	12
五，实验总结-----	15

成绩：

实验五：中断机制编程技术

一 实验目的

- 1 在内核实现多进程的三状态模，理解简单进程的构造方法和时间片轮转调度过程。
- 2 解释多进程的控制台命令，建立相应进程并能启动执行。
- 3 至少一个进程可用于测试前一版本的系统调用，搭建完整的操作系统框架，为后续实验项目打下坚实基础。

二 实验四/五的目的和要求(合并)：

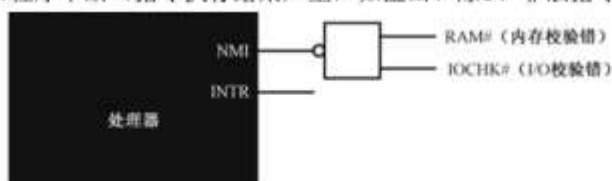
- 1 掌握 pc 微机的实模式硬件中断系统原理和中断服务程序设计方法，实现对时钟、键盘/鼠标等硬件中断的简单服务处理程序编程和调试，让你的原型操作系统在运行以前已有的用户程序时，能对异步事件正确捕捉和响应。
- 2 掌握操作系统的系统调用原理，实现原型操作系统中的系统调用框架，提供若干简单功能的系统调用。
- 3 学习掌握 c 语言库的设计方法，为自己的原型操作系统配套一个 c 程序开发环境，实现用自建的 c 语言开发简单的输入/输出的用户程序，展示封装的系统调用。

三 实验方案

(1) 基础原理

什么是中断？

- 中断(interrupt)是指对处理器正常处理过程的打断。中断与异常一样，都是在程序执行过程中的强制性转移，转移到相应的处理程序。
 - 硬中断（外部中断）——由外部（主要是外设[即I/O设备]）的请求引起的中断
 - 时钟中断（计时器产生，等间隔执行特定功能）
 - I/O中断（I/O控制器产生，通知操作完成或错误条件）
 - 硬件故障中断（故障产生，如掉电或内存奇偶校验错误）
 - 软中断（内部中断）——由指令的执行引起的中断
 - 中断指令（软中断`int n`、溢出中断`into`、中断返回`iret`、单步中断`TF=1`）
 - 异常/程序中断（指令执行结果产生，如溢出、除0、非法指令、越界）



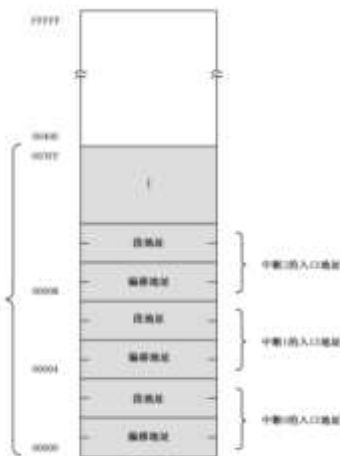
- x86 PC机的中断系统的功能强大、结构简单、使用灵活。采用32位的中断向量（中断处理程序的映射地址），可处理256种不同类型的中断。
- x86处理器有两条外部中断请求线
 - NMI（Non Maskable Interrupt，不可屏蔽中断）
 - INTR（Interrupt Request，中断请求[可屏蔽中断]）
- CPU是否响应在INTR线上出现的中断请求，取决于标志寄存器FLAGS中的IF标志位的状态值是否为1。可用机器指令STI/CLI置IF标志位为1/0来开/关中断。
- 在系统复位后，会置IF=0（中断响应被关闭）。在任意一中断被响应后，也会置IF=0（关中断）。若想允许中断嵌套，必须在中断处理程序中，用STI指令来打开中断
- 在NMI线上的中断请求，不受标志位IF的影响。CPU在执行完当前指令后，会立即响应。不可屏蔽中断的优先级要高于可屏蔽中断的。
- 保护断点的现场
 - 要将标志寄存器FLAGS压栈，然后清除它的IF位和TF位
 - 再将当前的代码段寄存器CS和指令指针寄存器IP压栈
- 执行中断处理程序
 - 由于处理器已经拿到了中断号，它将该号码乘以4（毕竟每个中断在中断向量表中占4字节），就得到了该中断入口点在中断向量表中的偏移地址
 - 从表中依次取出中断程序的偏移地址和段地址，并分别传送到IP和CS，自然地，处理器就开始执行中断处理程序了。
 - 注意，由于IF标志被清除，在中断处理过程中，处理器将不再响应硬件中断。如果希望更高优先级的中断嵌套，可以在编写中断处理程序时，适时用sti指令开放中断。
- 返回到断点接着执行
 - 所有中断处理程序的最后一条指令必须是中断返回指令iret。这将导致处理器依次从堆栈中弹出（恢复）IP、CS和FLAGS的原始内容，于是转到主程序接着执行。

中断的处理过程：

- 保护断点的现场
 - 要将标志寄存器FLAGS压栈，然后清除它的IF位和TF位
 - 再将当前的代码段寄存器CS和指令指针寄存器IP压栈
- 执行中断处理程序
 - 由于处理器已经拿到了中断号，它将该号码乘以4（毕竟每个中断在中断向量表中占4字节），就得到了该中断入口点在中断向量表中的偏移地址
 - 从表中依次取出中断程序的偏移地址和段地址，并分别传送到IP和CS，自然地，处理器就开始执行中断处理程序了。
 - 注意，由于IF标志被清除，在中断处理过程中，处理器将不再响应硬件中断。如果希望更高优先级的中断嵌套，可以在编写中断处理程序时，适时用sti指令开放中断。
- 返回到断点接着执行
 - 所有中断处理程序的最后一条指令必须是中断返回指令iret。这将导致处理器依次从堆栈中弹出（恢复）IP、CS和FLAGS的原始内容，于是转到主程序接着执行。

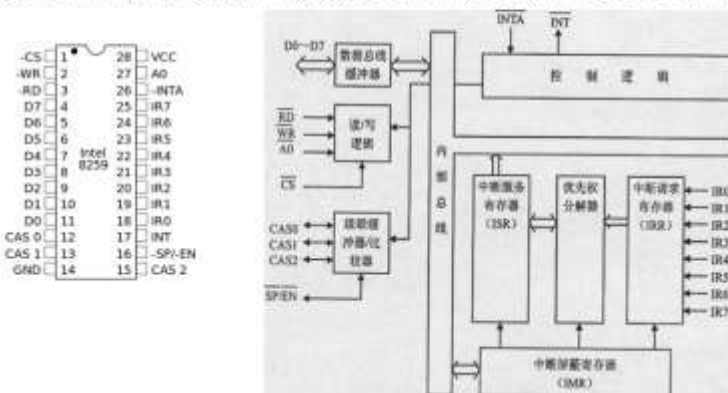
中断向量表在内存中的位置：

- x86计算机在启动时会自动进入实模式状态
 - 系统的BIOS初始化8259A的各中断线的类型（参见前图），
 - 在内存的低位区（地址为0~1023[3FFH]，1KB）创建含256个中断向量的表IVT（每个向量[地址]占4个字节，格式为：16位段值:16位偏移值）。
- 当系统进入保护模式
 - IVT（Interrupt Vector Table，中断向量表）会失效
 - 需改用IDT（Interrupt Descriptor Table，中断描述表），必须自己编程来定义8259A的各个软中断类型号和对应的处理程序。



8259A 对终端的控制作用：

- 8259A是一种PIC（Programmable Interrupt Controller，可编程中断控制器）
- 8259是Intel于1976年作为8位处理器8085支持芯片的一部分引进的，8259A作为ISA总线的中断控制器被包含在1981年推出的最初IBM PC机（采用8088 CPU）中，1984年推出的PC/AT机（采用80286 CPU）中添加了第二个8259A芯片，现代PC的两个级联8259A芯片被集成在主板上的南桥中。



- 在有多中断源的系统中，接受外部的中断请求
- 进行判断，选中当前优先级最高的中断请求
- 将此请求送到CPU的INTR端
- 当CPU响应中断并进入中断子程序的处理过程后，中断控制器仍负责对外部中断请求的管理。
- 在一个8259A芯片中，有如下三个内部寄存器：
 - IMR (Interrupt Mask Register, 中断屏蔽寄存器) ——用作过滤被屏蔽的中断
 - IRR (Interrupt Request Register, 中断请求寄存器) ——用作暂时放置未被进一步处理的中断
 - ISR (In-Service Register, 在使用中断) ——当一个中断正在被CPU处理时，此中断被放置在ISR中。
- 8259A还有一个单元叫做优先级分解器 (Priority Resolver)，当多个中断同时发生时，优先级分解器根据它们的优先级，将高优先级者优先传递给CPU。

软中断实现系统调用：

- BIOS调用。
 - 其实它与内核子程序软中断调用方式原理是一样的
 - 每一种服务由一个子程序实现，指定一个中断号对应这个服务，入口地址放在中断向量表中，中断号固定并且公布给用户，用户编程时才可以中断调用，参数传递可以使用栈、内在单元或寄存器。
- 系统调用
 - 因为操作系统要提供的服务更多，服务子程序数量太多，但中断向量有限，因此，实际做法是专门指定一个中断号对应服务处理程序总入口，然后再将服务程序所有服务用功能号区分，并作为一个参数从用户中传递过来，服务程序再进行分支，进入相应的功能实现子程序。
 - 这种方案至少要求向用户公开一个中断号和参数表，即所谓的系统调用手册，供用户使用。
 - 如果用户所用的开发语言是汇编语言，可以直接使用软中断调用
 - 如果使用高级语言，则要用库过程封装调用的参数传递和软中断等指令汇编代码
 - 规定系统调用服务的中断号是21h。

(2) 实验工具环境

实验支撑环境

硬件：个人计算机

主机操作系统：Windows/Linux/Mac OS/其它

虚拟机软件：VMware/VirtualPC/Bochs/其它

PC 虚拟机裸机/DOS 虚拟机/其它

实验开发工具

汇编语言工具：x86 汇编语言

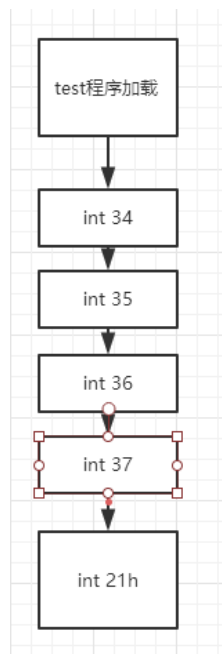
高级语言工具：标准 c 语言

磁盘映像文件浏览编辑工具

调试工具：Bochs

(3) 程序流程：

测试程序流程图



- (4) 代码模块组成：
修改中断向量表模块：

```
56  _change_int08h:
57      push bx
58      push es
59      push ax
60      cli
61
62      mov bx, 0
63      mov es, bx
64      mov ax, [es:8*4]
65      mov [old_08H], ax
66      mov ax, [es:8*4+2]
67      mov [old_08H+2], ax
68      mov word[es:8*4], INT_08H
69      mov word[es:8*4+2], 0
70
71      sti
72      pop ax
73      pop es
74      pop bx
75      pop ecx
76      jmp cx
```

一开始写这个模块时，我的思路是按照老师的思路重修 INT 08H 但随后我发现，我们重写的 INT 08H 会修改时钟当前的计数‘滴答’值，经过网上查阅资料我得知，int 08H 有一部分和系统时钟有关，因此如果重写 INT 08H，我们的系统时钟有可能改变。在实验三中，我利用 gettimeofday 函数，通过获取当前的系统时钟实现了随机数程序，但在本次实验中，如果我毫无顾忌的直接重写 INT 08H，会导致系统时钟被不断重新设置，随机数程序就会变成彻彻底底的伪随机。为了修改这个 BUG，我采取了‘重载’ bios 中断的方式：将原终端地址保存到一个内存中，等运行完我的程序后，再将它调到原中断程序中执行。

修改键盘响应中断模块和上述模块类似

修改 int 34—37 中断模块：

```
102  _change_int34to37:
103      push bx
104      push es
105      push ax
106      push ds
107      cli
108
109      mov bx, 0
110      mov es, bx
111      mov word[es:34*4], INT_34
112      mov bx, cs
113      mov word[es:34*4+2], bx
114      mov word[es:35*4], INT_35
115      mov bx, cs
116      mov word[es:35*4+2], bx
117      mov word[es:36*4], INT_36
118      mov bx, cs
119      mov word[es:36*4+2], bx
120      mov word[es:37*4], INT_37
121      mov bx, cs
122      mov word[es:37*4+2], bx
```


修改系统中断模块：(与上述模块相似)

```
49     mov word[es:33*4],INT_21H
50     mov bx,cs
51     mov word[es:33*4+2],bx
52
```

新的中断响应：

时钟中断：

```
89     mov ax, 0
90     mov ds, ax
91     push ax
92
93     call _int_08h_override
94     pushf
95     call far [old_08H]
```

Int_08h_override 是我实现的 c 语言函数

```
62 void int_08h_override()
63 {
64     num++;
65     if(num == 60) num = 0;
66     int tmp = getgb();
67     int ox = tmp/256%256;
68     int oy = tmp%256;
69     setgb(24,76);
70     printint(num);
71     setgb(24,78);
72     output(time_c[num%4]);
73     setgb(ox,oy);
74 }
```

键盘响应中断：

```
38     mov ax, 0
39     mov ds, ax
40     push ax
41
42     call _int_09h_override
43
44     pushf
45     call far [old_09H]
```

Int_09h_override 是我实现的 c 语言函数

```
49 void int_09h_override()
50 {
51     ouch_color++;
52     if( ouch_color >= 20 ) ouch_color = 2
53     int tmp = getgb();
54     int ox = tmp/256%256;
55     int oy = tmp%256;
56     setgb(23,76);
57     prints_color("OUCH",ouch_color/2);
58     setgb(ox,oy);
59 }
```

重写 INT 34, 35, 36, 37 :

```
133 INT_34:
134     push ax
135     push bx
136     push cx
137     push dx
138     push ds
139     push es
140     cli
141
142     mov bx, 0
143     mov ds, bx
144     push bx
145     call _int_34_override
146
147     sti
148     pop es
149     pop ds
150     pop dx
151     pop cx
152     pop bx
153     pop ax
154     iret
```

Int_34_override 是我写的 c 语言程序

```
96 void int_34_override()
97 {
98     // clear(1,1,14,40); //clear the area
99     setgb(1,0);
100     printstr("INT 34: press 'a' to stop:");CR();
101     int pos[2] = {3,7}; //x,y
102     char info = 'V'; //display info
103     int dir[2] = {1,1}; //initial the direction
104     char if_start;
105     int color = 1;
106     int tmp = getgb();
107     int ox = tmp/256*256;
108     int oy = tmp%256;
109
110     while(if_start != 'a')
111     {
112         color = (color+1)%9;
113         //delay
114         for(int delay = 500000 ; delay > 0 ; delay--)
115             for(int ddelay = 50 ; ddelay > 0 ; ddelay--);
116         if_start = isinput();
117
118         pos[0] += dir[0]; pos[1] += dir[1];
119         //judging the boundary when the ball touch it
120         if(pos[0] == 14) dir[0] = -1;
121         if(pos[0] == 1) dir[0] = 1;
122         if(pos[1] == 40) dir[1] = -1;
123         if(pos[1] == 1) dir[1] = 1;
124         //out put the ball on scan
125         setgb(pos[0],pos[1]);
126         output_color(info,color);
127         setgb(ox,oy);
128     }
129     setgb(ox,oy);
130 }
```

(34, 35, 36, 37 中断类似)

这段 c 语言程序通过调用我自己实现的 stdio 库，实现了在屏幕四分之一区域控制字符的弹跳，33 号中断控制左上方屏幕，34 号中断控制右上方屏幕，35 号中断控制左下方屏幕，36 号中断控制右下方屏幕。分别通过输入字符 ‘a’ ， ‘b’ ， ‘c’ ， ‘d’ 来结束程序的弹跳。

新的系统中断：

```
62 = INT_21H:
63     push ax
64     push bx
65     push cx
66     push dx
67     push ds
68     push es
69     cli
70
71     mov bx, 0
72     mov ds, bx
73
74     push bx
75 = AH0:
76     cmp ah,0
77     jnz AH1
78     call _int_21h0_override
79     jmp END
80 = AH1:
81     cmp ah,1
82     jnz AH2
83     call _int_21h1_override
84     jmp END
85 = AH2:
86     cmp ah,2
87     jnz DEFAULT_21h
88     call _int_21h2_override
89     jmp END
90 = DEFAULT_21h:
91     call _int_21h_default
92 = END:
93     sti
94     pop es
```

我们通过比较 ah 中的值来进行判断，将程序跳到哪一个我们实现的 c 语言程序中执行。其实此处有很多中方法来进行比较与跳转。我这里使用的时 CMP 指令，在汇编语言中不断比较再进行跳转。

这里给出我实现的 INT 21H 终端号参数表：

参数 AH	功能
0	获取并打印当前时间（年月日小时分钟）
1	输入字符串并打印之
2	在频幕上输出“I love OS”
其他	在屏幕上提示系统调用错误信息

系统功能调用中的 c 语言函数:

```
25 void int_21h0_override()  
26 {  
27     clear(); setgb(0,0);  
28     int date = getdate();  
29  
30     int tmp_m = date/256;  
31     int month = tmp_m>>4;  
32     month = month*10+(tmp_m&=0x000F);  
33  
34     int tmp_d = date%256;  
35     int day = tmp_d>>4;  
36     day = day*10+(tmp_d&=0x000F);  
37  
38     int now = getnow();  
39  
40     int tmp_h = now/256;  
41     int hour = tmp_h>>4;  
42     hour = hour*10+(tmp_h&=0x000F);  
43  
44     int tmp_n = date%256;  
45     int minute = tmp_n>>4;  
46     minute = minute*10+(tmp_n&=0x000F);  
47  
48     printstr("INT 21h,ah=0");CR();  
49     prints_color("now time: ",0x0E);  
50     printstr("2018/");printint(month);output('/');printint(day);  
51     printstr(" ");  
52     printint(hour);output(':');printint(minute);  
53     CR();printstr("press any key to continue");  
54     input();  
55 }
```

该段程序通过我实现在 stdio 中的函数来进行输出时间的操作
这里先介绍一下 INT 1aH:

(3). 功能02H

功能描述: 读取时间

入口参数: AH = 02H

出口参数: CH = BCD码格式的小时

CL = BCD码格式的分

DH = BCD码格式的秒

DL = 00H——标准时间, 否则, 夏令时

CF = 0——时钟在走, 否则, 时钟停止

(5). 功能04H

功能描述: 读取日期

入口参数: AH = 04H

出口参数: CH = BCD码格式的世纪

CL = BCD码格式的年

DH = BCD码格式的月

DL = BCD码格式的日

CF = 0——时钟在走, 否则, 时钟停止

接下来我将 BCD 码转为十进制的操作:

```
int month = tmp_m>>4;  
month = month*10+(tmp_m&=0x000F);
```

该操作确保了将一个 int 类型的 BCD 码转化为十进制, 并储存在 int 中

```

66 void int_21h1_override()
67 {
68     clear();
69     setgb(0,0);
70     printstr("INT 21h , ah=1:");CR();
71     prints_color("input the string:",0x0E);
72     char str[100];
73     scanf(str);
74     prints_color("output the string:",0x0E);
75     prints_color(str,0x0E);CR();
76     printstr("press any key to continue");
77     input();
78 }
79 void int_21h2_override()
80 {
81     clear();
82     setgb(0,0);
83     printstr("INT 21h , ah=2:");
84     prints_color("I love OS !!!",0x0E);CR();
85     printstr("press any key to continue");
86     input();
87 }
88 void int_21h_default()
89 {
90     clear();
91     setgb(0,0);
92     prints_color("INT 21h,wrong ah you use, there is only 0,1,2 provided:",0x0E);CR();
93     printstr("press any key to continue");
94     input();
95 }

```

这三个函数分别对应着 ah = 0, 1, 2 的情况

中断响应测试程序：

```

15 int 34
16 int 35
17 int 36
18 int 37
19 mov ah,0
20 int 21h
21 mov ah,1
22 int 21h
23 mov ah,2
24 int 21h
25 mov ah,3
26 int 21h

```

该模块思路比较简单，明了，保护完段寄存器之后，直接测试即可

(5) 代码文档组成

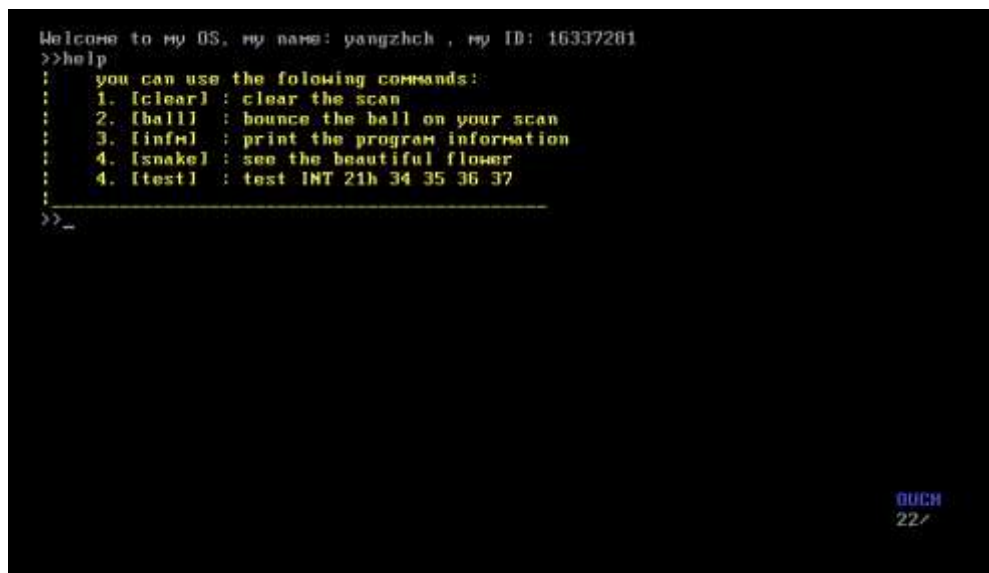
ball.c	2018/4/12 21:58	C Source File
IO.asm	2018/4/12 21:58	Assembler Source
kernal.c	2018/4/12 21:58	C Source File
leader.asm	2018/4/12 21:58	Assembler Source
myos.bat	2018/4/12 21:58	Windows 批处理...
newint.asm	2018/4/12 21:58	Assembler Source
snake.c	2018/4/12 21:58	C Source File
stdio.c	2018/4/12 21:58	C Source File
test.c	2018/4/12 21:58	C Source File

Kernal.c	c 语言内核
Stdio.c	实现的 c 语言库
IO.asm	汇编语言中的 IO 接口
Leader.asm	引导程序
Snake.c	贪吃蛇游戏
Ball.c	小球弹跳程序
Test.c	测试中断程序
Newint.asm	中断重写的 asm 文件
Myos.bat	批处理文件

(6) 实现效果:

此为刚进入虚拟机时，可以看见右下角的‘摩天轮’，计数器，以及键盘输出响应的 ouch
在本次实验中，我的键盘输出响应变化是变色，即每按一次键盘变依次 ouch 颜色

```
Welcome to my OS, my name: yangzhch, my ID: 16337281
>>help
: you can use the following commands:
: 1. [clear] : clear the scan
: 2. [ball] : bounce the ball on your scan
: 3. [info] : print the program information
: 4. [snake] : see the beautiful flower
: 4. [test] : test INT 21h 34 35 36 37
:
>>_
```

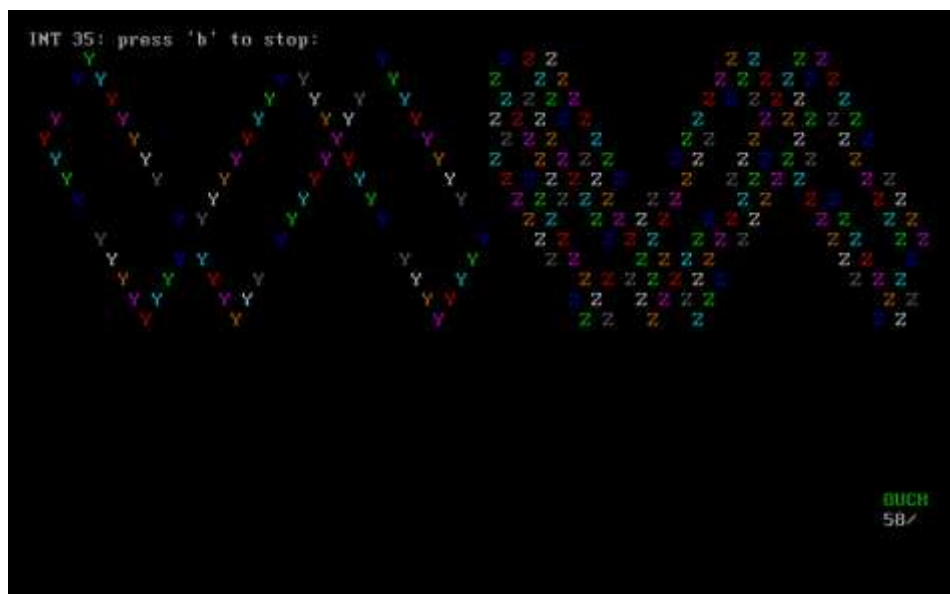


运行 test 程序效果：此为
INT 34



按下 a

INT 35:



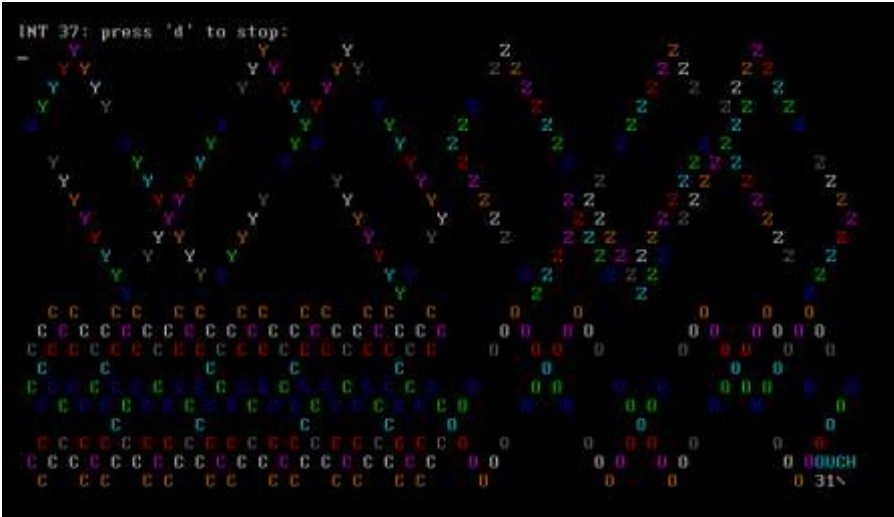
按下 b

INT 36:



按下 c

INT 37



按下 d

INT 21h (ah = 0)

```
INT 21h,ah=0
now time: 2018/4/12 22:12
press any key to continue_
```

OUCH
45-

如图可见时间显示是正确的

INT 21H(ah = 1):

```
INT 21h , ah=1:
input the string:yangzhicheng
output the string:yangzhicheng
press any key to continue_
```

OUCH
17-

INT 21H(ah = 1)



此外，在我设计的系统调用终端中，若错误使用端口号，我会进行提醒：



此外也可以看见我的时钟和键盘中断在用户程序里也能运行：



五 实验过程及其总结

在本次实验中，虽然更改终端的代码量并不大，但是 debug 却不一定是一件容易的事情，因为 BIOS 终端对我们而言更多的像是一个黑箱，我们不知道黑箱里究竟发生了什么。

接下来列举一下我在本次实验中遇到的问题，以及我自己的思考：

(1) 为什么跳入原中断底之前，要使用 pushf ？

```
44      pushf
45      call far [old_09H]
```

这里就涉及到了 int 指令与 iret 指令

CPU 执行 int n 指令，相当于引发一个 n 号中断的中断过程，执行过程如下：

- 1) 取中断类型码 n；
- 2) 标志寄存器入栈，IF=0，TF=0；
- 3) CS、IP 入栈
- 4) (IP) = (n*4)，(CS)=(n*4+2)

从此处转去执行 n 号中断的中断处理程序。

而 iret 指令相当于此指令的逆过程，由于在此处我们是 call 到了 [old_09h] 的旧中断地址，而 call 指令的执行过程中，并没有 pushf 的操作，因此为了中断程序能够正常返回，我们需要加上一条 pushf 指令，将标志寄存器压栈。

(2) 如何进行保护断点的操作？

这个问题并没有困扰我很久，我们只需要将段寄存器 ds，es，以及某些通用寄存器压栈即可，当然也不排除某些特殊的情况，导致我们需要将 bp，sp 等偏移寄存器压栈。

(3) 如何获取时间？

个人认为在本次实验中，获取时间这个操作是我的一个小创新点，我通过调用 BIOS 的 int 1Ah 准确获取了系统时间，在这个过程中我运用了 BCD 转十进制的知识，技术细节如下：

```
int month = tmp_m>>4;
month = month*10+(tmp_m&=0x000F);
```

因为 BCD 每 4 位代表一个十进制位，因此通过将 tmp 右移四位来实现然后与 0x000F 进行与运算来获得正确的十进制数。

然后将其打印在屏幕上。

```
INT 21h,ah=0
now time: 2018/4/12 22:12
press any key to continue
```

其实，在本次实验中，我可以将时钟中断（INT 08H）修改一下，让他不断的在屏幕右下角刷新显示当前的系统时钟，这是轻而易举的事情，但是老师本

次的实验并没有这个要求，我就把这个功能放到了系统调用中实现，若老师认为这个策略可行，我会在下次的实验中把该功能放到时钟中断上。

(4) 一些低级错误

在此次中断编写过程中，我曾犯下了一个低级错误：把 INT 33 和 INT 33h 搞混淆了，虽然我修改对了中断向量表，但是响应中断时，用的时 INT 33H，将十六进制和十进制搞混淆是一个十分低级的错误，下次一定不会再犯。

(5) 时钟中断和键盘响应中断修改后，是否需要还原？

在我的这个操作系统里，是不需要还原的，我在微信群看到老师和同学们讨论时，认为跳入到用户程序前，应该将中断向量表还原，我觉得没有这个必要，让 ouch 和时钟计数相应应在特定的位置即可。而且我的时钟中断和键盘响应中断进入用户程序之后依然正常运行，没有任何异处。

(6) 系统中断和 34 到 37 中断写在哪个文件？

本次实验中我写在了一个新的文件，要测试系统终端时，直接让内核加载这个程序即可。但是我的 08 和 09 中断（时钟中断和键盘响应中断）直接写在了 c 语言函数库中，这样，这两个终端就可以从头到尾一直存在；且可在用户程序中一直正常运行。

六 实验感想

本次实验不难，但还是花费了我的一些时间。

中断机制是我们在学写操作系统中必不可少的一个知识点。他是我们进行多进程操作（而状态进程模型）的基础。也是我们想要掌握操作系统所必经的路。

这么多次操作系统实验以来，我总结了不少经验，以后的操作系统实验想必依然不会顺畅，但我仍会保持我的好奇心，迎难而上。

争取要在这个学期，写出一个令老师也令我自己满意的操作系统。