

实验报告——实验五

姓名：任文頔

学号：14322181

院系：数据科学与计算机专业

专业、年级：14级计算机科学与技术

指导教师：凌应标

【实验题目】

中断机制编程技术

【实验目的】

- 1、学习中断机制知识，掌握中断处理程序设计的要求
- 2、设计一个汇编程序，实现时钟中断处理程序
- 3、扩展MyOS4，增加时钟中断服务，利用时钟中断实现与时间有关的操作
- 4、实现简单的系统调用

【实验要求】

1. 操作系统工作期间，利用时钟中断，在屏幕最边缘处动态画框，第一次用字母A，第二次画用字母B，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。
2. 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!"。
3. 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。
4. 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。

【实验方案】

1 硬件或虚拟机配置方法

系统环境：Linux Ubuntu 14.04

虚拟机配置方法：在Linux下VMware Player是收费软件，因此选择免费的VirtualBox软件。

配置方法：操作系统选择其他，选择从软盘启动，添加自己的软盘，虚拟机名称为14322181renwendi

2 软件工具与作用

- (1) 汇编语言编译器NASM：针对Intel x86架构的汇编与反汇编程序
- (2) C语言编译器GCC：由 GNU 开发的编程语言编译器。
- (3) 编辑器Vim：功能强大、高度可定制的文本编辑器。
- (4) 虚拟机软件Bochs：自己编写的操作系统的测试环境
- (5) 软盘创建工具:dd 软盘写入工具:Linux 自带挂载命令

3 方案的思想

由于实验四的fat12文件系统的内核较为不完善，因此本实验选择在实验三的基础上进行改进和添加功能。

对系统时钟08h号编写时钟中断，在内核中，对33号、34号、35号和36号中断编写中断服务程序，手动修改中断向量表。在系统初始化时，设置时钟和自定义中断向量。

对09h号键盘中断进行修改,在用户程序执行完毕之后再把 09h中断向量修改回去。

4 相关知识原理

1、中断技术

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，包括

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）

2、PC中断的处理过程

(1) 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

(2) 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。

(3) 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。

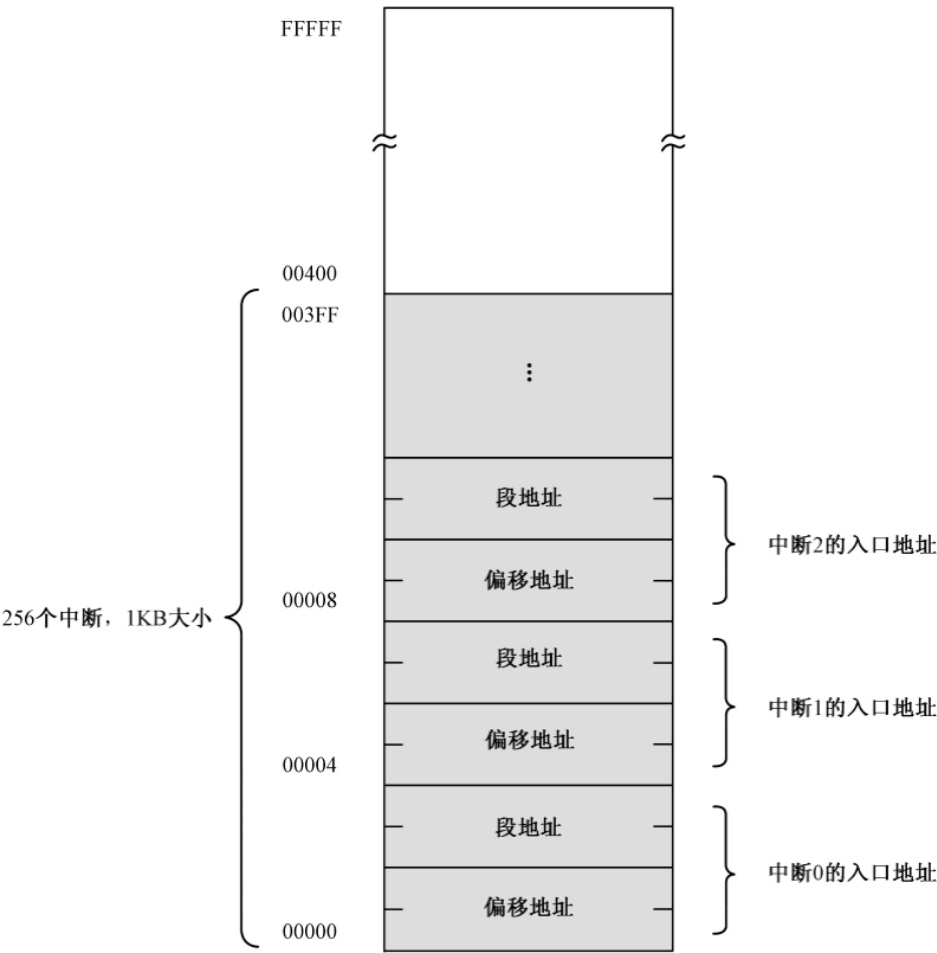
3、中断向量

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。



4、8259A主要功能

8259A是一种PIC（Programmable Interrupt Controller，可编程中断控制器）。

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。

每个可编程中断控制器8259A都有两个I/O端口主8259A所对应的端口地址为20h和21h，从8259A所对应的端口地址为A0h和A1h。程序员可以通过in/out指令读写这些端口，来操作这两个中断控制器。

5、时钟中断编程

连接在主8259A的0号引脚上。

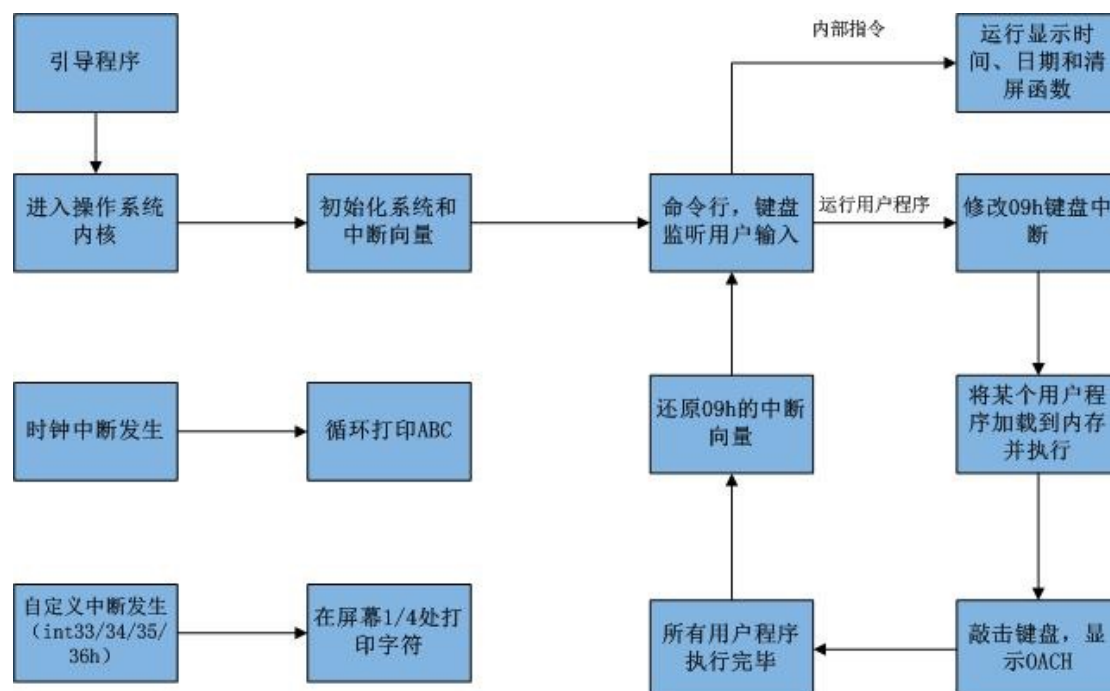
系统时钟中断号为08h。

中断向量：地址为32、33、34、35共4字节

“高高低低”原则：

32和33保存IP，34和35保存CS。

5 程序流程



6 程序关键模块

(1) 引导程序boot.asm

与实验三相同

(2) 内核C语言部分os.c

主要实现命令行的控制

A. 命令行部分的代码:

```
_ 1 [bits 16] //实模式
2 __asm__(".code16gcc"); //将汇编嵌入GCC
... (省略部分为extern声明引用别的文件 (os.asm, oslib.asm, osclib.c) 中的全局变量)
29 char key[64];
30 char screen_sc_T = 1;
31 //=====MAIN=====
32 void main(){
33     __asm__("movw $0,%ax");      //系统初始化, 内联汇编
34     __asm__("movw %ax, %ss");
35     screen_init();
36     interrupt_init();
37     print_welcome_str();
38     print_message();
39     print_cmd();      //命令行提示语ren@ren-Inspiron:~ position
40     while(1){
41         char length = listen_key();
42
43         unsigned short int now_row = get_mouse_pos()/256;
44         if( now_row >23){          // 0~24 屏幕最多显示24行
45             while( screen_sc_T--){
46                 scroll_screen();
47                 cmd_scroll_up();
48             }
49         }
50
51         cmd_scroll();              //将命令行提示语移动到下一行
52         print_cmd();
```

```
53 }  
54 }
```

与之前不同的是尝试了gcc内联汇编，同时通过interrupt_init()函数对中断向量进行初始化。

B. 监听用户输入并显示部分代码:

```
59 char temp;  
60 char i=0;  
61  
62 while(( temp=input_char())!=0x0d ){  
63     //当输入不是回车符时  
64     if( temp == '\b'){          // 若输入是退格符  
65         unsigned short int now_pos = get_mouse_pos();  
66         if( (now_pos&0x00ff) <21){ //命令行提示语长度为21，不能删除  
67             continue;  
68         }  
69         printT('\b');  
70         printT(' ');  
71         printT('\b');  
72         i--;  
73         continue;  
74     }  
75  
76     key[i] = temp;      //将每次输入的字符存储在key[]字符数组中  
77     //char key[64]  
78     if(i<63)  
79         i++;  
80     printT( temp);  
81     printTimeClock();   //时钟中断发生  
82 }  
83 key[i] = '\0';
```

将字符显示并存储，为了之后的判断

C. 字符串比较并调用相应程序

其余time, date, clear 和 run（批处理用户程序）的部分与实验三相同，增加部分为调用自定义中断

```
96  if( strcmp( key, "date\0")){
97      date();
98      return i;
99  }
100 //仅以date为例
101 //-----run programs command-----
102  if( synCheck( key, "run\0")){
103      run( key);
104      return i;
105  }
106 //批处理用户程序（1234共4个）
107  if( synCheck( key, "int\0")){
108      if( strcmp( key, "int 33h")){
109          __asm__( "int $0x33");
110          return i;
111      }
112      if( strcmp( key, "int 34h")){
113          __asm__( "int $0x34");
114          return i;
115      }
116      if( strcmp( key, "int 35h")){
117          __asm__( "int $0x35");
118          return i;
119      }
120      if( strcmp( key, "int 36h")){
121          __asm__( "int $0x36");
122          return i;
123      }
124  }
```

使用内联汇编，用int指令调用操作系统提供服务的子程序，系统服务的子程序在内核。

(3) 内核汇编语言部分os.asm

A. 声明的全局变量

```

1 [bits 16]
2 global print_welcome_str,screen_init, print_message
3 global input_char,printT
4 global set_mouse_pos,get_mouse_pos
5 global print_cmd,scroll_screen,cmd_scroll
6 global cmd_position:data,cmd_scroll_up
7 global init_cmd_position
8 global printTimeClock
9 global interrupt_init,insert_interrupt_vector
10 global interrupt_num:data,interrupt_vector_offset:data
11
12 extern main
13 jmp main //运行命令行主程序

```

B. 设置中断向量

```

156 interrupt_init:
157     mov ax,cs
158     mov ds,ax
159
160     ;-----time interrupt -----
161     mov ax,0x1c
162     mov [ interrupt_num], ax
163     mov ax, printTimeClock
164     mov [ interrupt_vector_offset],ax
165     call insert_interrupt_vector
166
167     ;-----int 33-----
168     mov ax,0x33
169     mov [ interrupt_num], ax
170     mov ax, process_int33
171     mov [ interrupt_vector_offset],ax
172     call insert_interrupt_vector
173
174     ;-----int 34-----
175     mov ax,0x34

```



```

176  mov [ interrupt_num], ax
177  mov ax, process_int34
178  mov [ interrupt_vector_offset],ax
179  call insert_interrupt_vector
180
181  ;-----int 35-----
182  mov ax,0x35
183  mov [ interrupt_num], ax
184  mov ax, process_int35
185  mov [ interrupt_vector_offset],ax
186  call insert_interrupt_vector
187
188  ;-----int 36-----
189  mov ax,0x36
190  mov [ interrupt_num], ax
191  mov ax, process_int36
192  mov [ interrupt_vector_offset],ax
193  call insert_interrupt_vector

```

获取中断向量号和中断向量的偏移地址，分别存储在interrupt_num和interrupt_vector_offset中，而后调用insert_interrupt_vector函数，修改中断向量表

195 insert_interrupt_vector:

```

196  mov ax,0
197  mov es,ax
198  mov bx,[ interrupt_num]
199  shl bx,2      ;interrupt num * 4 = entry, 位置=中断号X4, 4字节CS-IP
200  mov ax,cs
201  shl eax,8
202  mov ax,[ interrupt_vector_offset]
203  mov [es:bx], eax
204  ret

```

C.时钟中断循环打印ABC

207 printTimeClock:

```

208      mov ax,cs
209      mov ds,ax

```

```

210    mov ax,0xb800
211    mov es,ax
212
213    mov dl,30
214    mov al,[mouse]
215    cmp al,dl
216    jne next_print_c
217    mov byte [mouse], 0
218    jmp cotinue_timeclock
219
220    next_print_c:
221    mov eax, timeclockstring
222    mov ebx,0
223    mov bl,[mouse]      ;ebx is added sum
224    add eax,ebx
225    mov al,[ eax]
226
227    inc ebx
228    mov [mouse],bl
229
230    mov bx,3998D
231    mov [es:bx],al
232
233    cotinue_timeclock:
234    iret
235//-----用到的变量-----
458    mouse db 0
459    timeclockstring db 'AAAAAAAAAABBBBBBBBBBCCCCCCCCCCCC'

```

D.对33号、34号、35号和36号中断编写的中断服务程序

下面仅以33号中断为例，其余3个基本一样，只是显示位置和字符有差别

```

236 process_int33:

```

```

237    mov ax,0xb800
238    mov es,ax
239

```

```

240  mov cx,220D    ;stop
241  mov bx,180D    ;stat
242
243  ;h0 direction
244  loop_int33_h0:
245      mov byte [es:bx],'A'
246      inc bx
247      mov byte [es:bx],78D;font color
248      inc bx
249      cmp bx,cx
250      jle loop_int33_h0
251
252  ;h1 direction
253  mov cx,1820D    ;stop
254  mov bx,220D    ;stat
255
256  loop_int33_h1:
257      mov byte [es:bx],'B'
258      inc bx
259      mov byte [es:bx],78D;font color
260      add bx, 159D
261      cmp bx,cx
262      jle loop_int33_h1
263
264  ;h2 direction
265  mov cx,1780D    ;end
266  mov bx,1820D    ;strat
267  loop_int33_h2:
268      mov byte [es:bx],'C'
269      dec bx
270      mov byte [es:bx],78D
271      dec bx
272      cmp bx,cx
273      ja loop_int33_h2
274
275  ;h3 direction

```

```

276  mov cx,1780D
277  mov bx,180D
278  loop_int33_h3:
279      mov byte[es:bx],'D'
280      inc bx
281      mov byte[es:bx],78D
282      add bx,159D
283      cmp bx,cx
284      jle loop_int33_h3
285
286  iret

```

在屏幕左上1/4处显示一个长方形框，四个边分别由ABCD组成。

使用iret中断返回指令，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。

其余三个中断显示的位置和对应的字符分别为：

34h：右上1/4处，“+ - * %”

35h：左下1/4处，“a b c d”

36h：右下1/4处，“1 2 3 4”

（4）内核C语言库文件osclib.c

考虑到内核主要控制模块为os.c文件，为了不让主模块显得过于庞大繁杂，将字符串的相关操作（字符串的比较、复制、获取长度等）、时间的显示、日期的显示和批处理的实现放在该辅助C语言文件中。类似于C语言的string标准库。与实验三一样。

批处理的实现：

```

158 void run( char *str){
159
160     while( *str != '\0'){
161         if('0'<*str && *str<'6'){
162             load_user( *str-'0'+14);    //in oslib.asm
163             run_user();
164             init_cmd_position();
165             screen_init();
166             print_welcome_str();
167             print_message();
168             print_cmd();

```

```

169
170     }else{
171         run_error();
172     }
173     str++;
174 }
175 }

```

(5) 内核汇编库文件oslib.asm

提供无法在c语言实现而需要在汇编实现的库文件，主要就是清屏函数。

此外，提供加载到内存并运行用户程序的具体实现，包括09h键盘中断的修改。

A. 清屏

```

13 clear:      ;clear the screen
14  push ax
15  mov ax,0x0000
16  mov [ cmd_position], ax
17  pop ax
18  call screen_init
19 ret

```

B. 运行用户程序

```

45 load_user:
46
47  mov ax,ds
48  mov es,ax
49
50  mov dx,bp
51  mov bp,sp
52  mov cx,[bp+4];扇区号参数
53  mov bp,dx
54
55  xor ax,ax
56  mov es,ax
57  mov bx,1000h ;加载到1000h内存位置
58  mov ax,0201h

```

```

59  mov dx,0
60  int 13h
61  ret
63
64  run_user:
65  push ecx
66  push ax
67  call screen_init
68
69  ;-----重新设置09h键盘中断-----
70  cli
71  mov ax,0
72  mov es,ax
73  mov ecx,[ es:36]      ;backup
74  mov [ press],ecx
75
76  mov ax,0x09
77  mov [ interrupt_num], ax
78  mov ax, key_detect
79  mov [ interrupt_vector_offset],ax
80  call insert_interrupt_vector
81  mov ax,0
82  mov [i],ax
83  sti
84  ;-----运行用户程序-----
85  call 0x1000
86
87  ;-----恢复09h键盘中断-----
88  mov ax,0
89  mov es,ax
90  mov ecx,[press]
91  mov [ es:36],ecx
92
93  pop ax
94  pop ecx

```

95 ret

修改键盘中断的中断向量的时候,要考虑用户程序执行完了再把 09h中断向量修改回去。所以要几个变量来保存之前09h指向的地址。此外要通过in/out指令读写8259A的端口,来操作中断控制器。显示OUCH的函数如下:

97 key_detect:

```
98  cli
99  push ax
100 push bx
101  mov ax,0xb800
102  mov es,ax
103
104  in al,60h
105  in al,61h
106  or al,80h
107  out 61h,al
108  mov al,61h
109  out 20h,al
110
111  mov bl,0
112  mov bh,[j]
113  cmp bl,bh
114  je change1
115
116  mov bh,0
117  mov [j],bh
118  mov byte [es:00],'O'
119  mov byte [es:02],'U'
120  mov byte [es:04],'C'
121  mov byte [es:06],'H'
122
123  jmp next_de
124
125  change1:
126  mov bh,1
127  mov [j],bh
```

```

128
129  mov byte [es:00],''
130  mov byte [es:02],''
131  mov byte [es:04],''
132  mov byte [es:06],''
133
134  next_de:
135
136  mov bl,[i]
137  inc bl
138  mov [i],bl
139  mov bh,8      ;press key 8 times
140  cmp bh,bl
141  jle closesti
142  jmp niret
143  closesti:
144      mov cl,'A'
145
146  niret:
147
148  pop bx
149  pop ax
150  sti
151  fret
155 i db 0
156 j db 0

```

执行用户程序的时候，用户按下键盘的时候，按下和抬起均会触发09h中断，因此要用一个变量j来区分是按下还是抬起（0/1表示）。i表示09中断被触发几次，达到4*2=8则返回。

(6) Makefile文件

与实验三一致，只是此次生成的软盘改名叫test.img

(7) 用户程序use1/2/3/4.asm

前三个用户程序与之前的实验相同，在屏幕1/4处显示四行字符串，但是字符串内容有所改变，最后提示用户按键4次，显示4次OCUH后退出，因此退出方式也有相应改变，之后在遇到的问题并解决的模块进行说明。

将第4个用户程序改成了执行自定义的int33、int34、int35、int36中断。

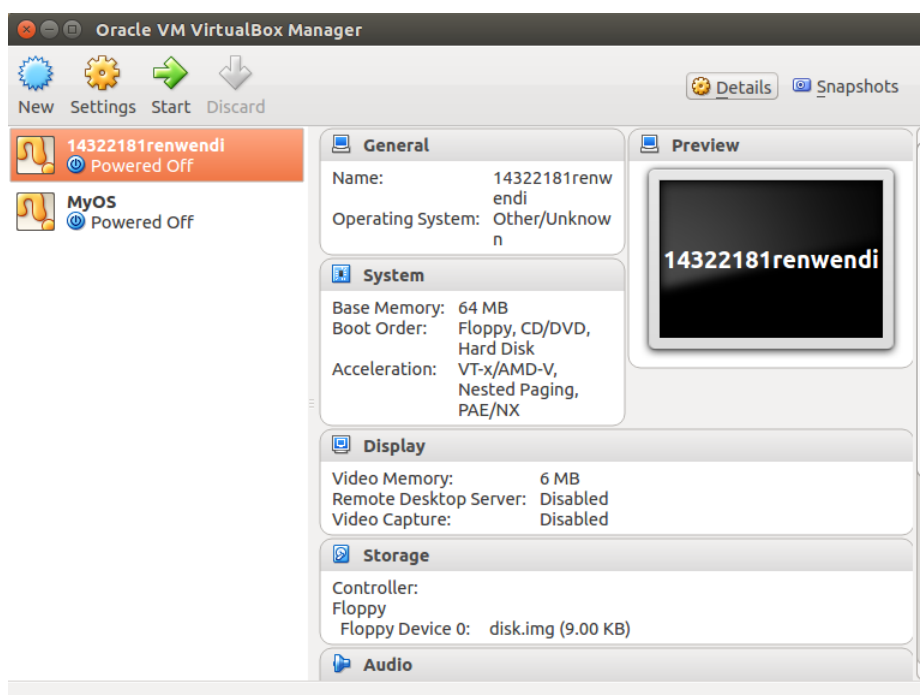
use4.asm代码如下：

```
1 org 0x1000
2
3 int 33h
4 int 34h
5 int 35h
6 int 36h
7
8
9 ;LISTEN_EXIT----
10 listen:
11     mov ch,'A'
12     cmp ch,cl
13 jne listen
14
15 ret
```

【实验过程】

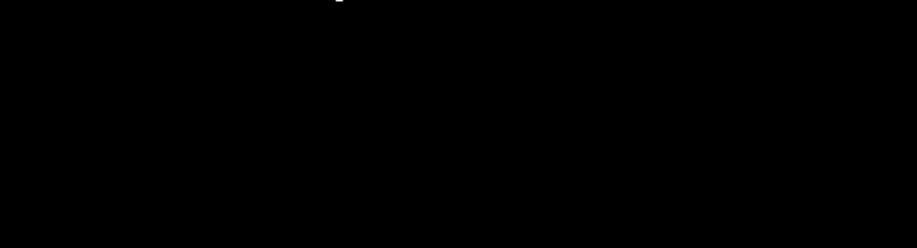
1 主要工具安装使用过程及截图结果

虚拟机配置：虚拟机名称为14322181renwendi



2 程序过程中的操作步骤

(1) 尽管最后决定在原型三的基础上进行修改，但是还是先在原型四的fat12文件系统中运行了老师所给的参考代码，结果如下：

[illegible]

MyOS [Running] - Oracle VM VirtualBox

MyOS (C) 2016 Ren

Welcome! You can enter h for help

Ren:/\$tc2

N

(2) 用vim编写代码，在编写好Makefile文件之后，直接用make指令生成软盘、编译链接、将各个文件存储在相应扇区（dd命令实现）。

```
4 test.img: boot.bin os.bin use1.com use2.com use3.com use4.com
5 dd if=/dev/zero of=test.img count=2880 bs=512
6 dd if=boot.bin of=test.img
7 dd if=os.bin of=test.img seek=1
8 dd if=use1.com of=test.img bs=512 seek=14
9 dd if=use2.com of=test.img bs=512 seek=15
10 dd if=use3.com of=test.img bs=512 seek=16
11 dd if=use4.com of=test.img bs=512 seek=17
```

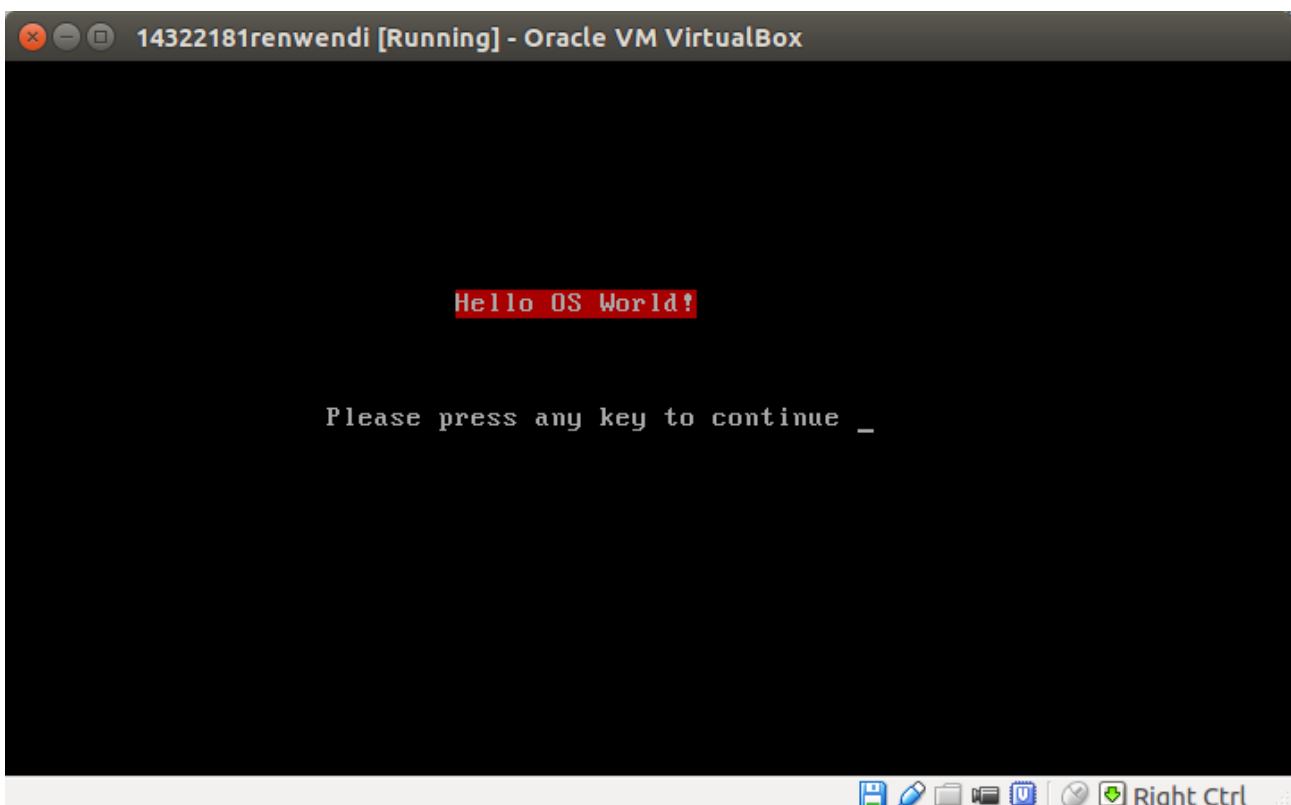
软盘第1个扇区存储操作系统引导程序

软盘第2~14扇区存储操作系统内核

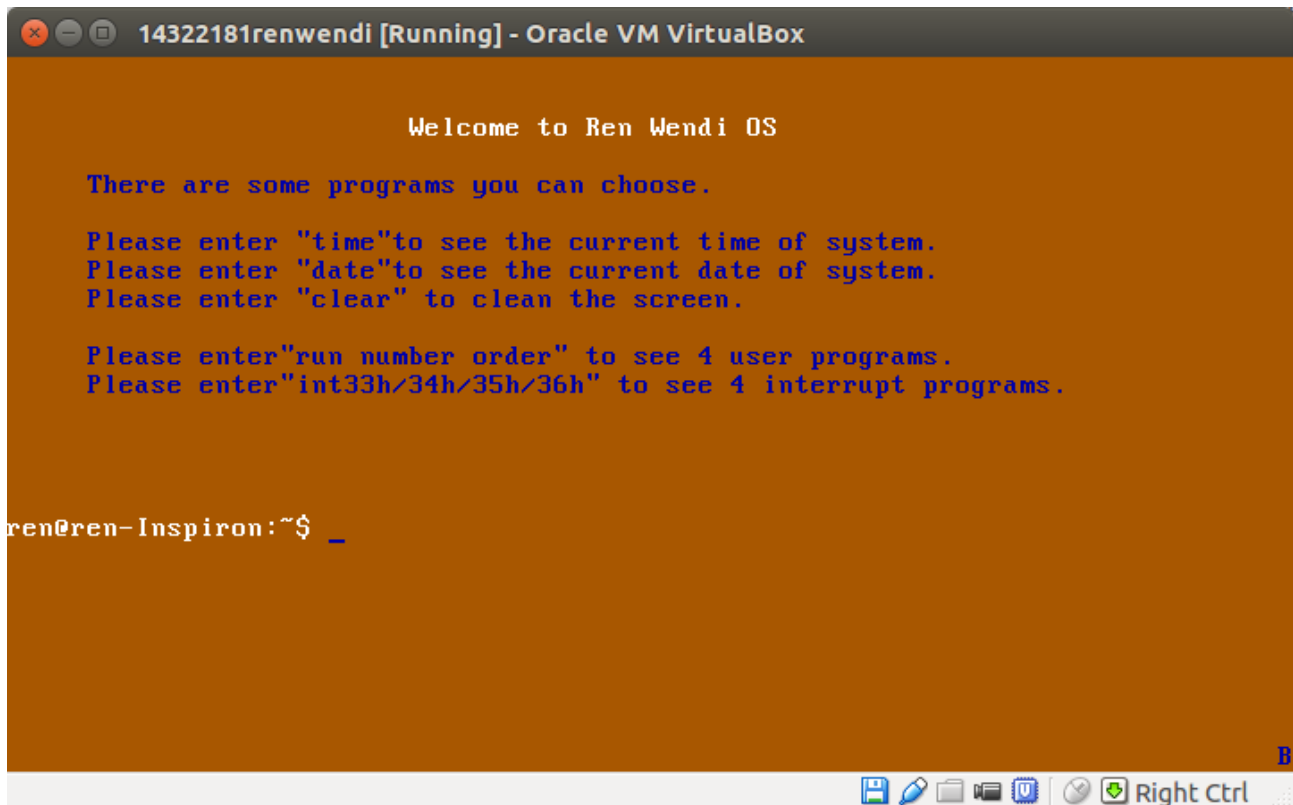
软盘第15~18扇区分别存储四个用户程序

3 输出说明

(1) 引导程序欢迎界面



(2) 内核主界面 [注：右下角循环显示字符ABC]



The screenshot shows a VirtualBox window titled "14322181renwendi [Running] - Oracle VM VirtualBox". The main display area has an orange background with white text. The text reads: "Welcome to Ren Wendi OS", "There are some programs you can choose.", "Please enter 'time' to see the current time of system.", "Please enter 'date' to see the current date of system.", "Please enter 'clear' to clean the screen.", "Please enter 'run number order' to see 4 user programs.", "Please enter 'int33h/34h/35h/36h' to see 4 interrupt programs.", At the bottom left, a command prompt shows "ren@ren-Inspiron:~\$ _". At the bottom right, there is a small blue character "B". The bottom of the window features a toolbar with icons for save, edit, folder, printer, USB, network, and a "Right Ctrl" button.

```
14322181renwendi [Running] - Oracle VM VirtualBox

Welcome to Ren Wendi OS

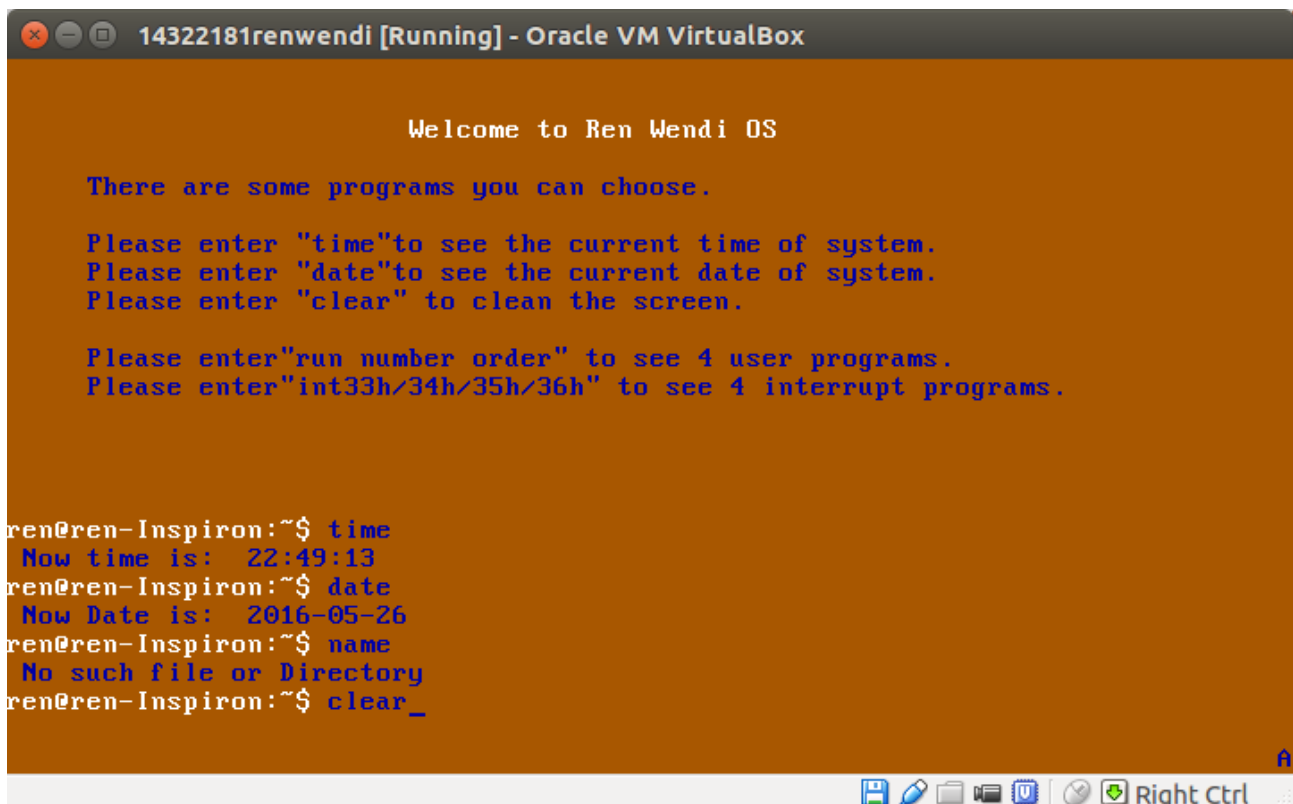
There are some programs you can choose.

Please enter "time" to see the current time of system.
Please enter "date" to see the current date of system.
Please enter "clear" to clean the screen.

Please enter "run number order" to see 4 user programs.
Please enter "int33h/34h/35h/36h" to see 4 interrupt programs.

ren@ren-Inspiron:~$ _
```

(3) 显示日期、时间、错误提示界面



This screenshot shows the same VirtualBox window as the previous one, but with user input and output. The command prompt now shows: "ren@ren-Inspiron:~\$ time", "Now time is: 22:49:13", "ren@ren-Inspiron:~\$ date", "Now Date is: 2016-05-26", "ren@ren-Inspiron:~\$ name", "No such file or Directory", "ren@ren-Inspiron:~\$ clear_". The blue character at the bottom right has changed to "A". The toolbar at the bottom remains the same.

```
14322181renwendi [Running] - Oracle VM VirtualBox

Welcome to Ren Wendi OS

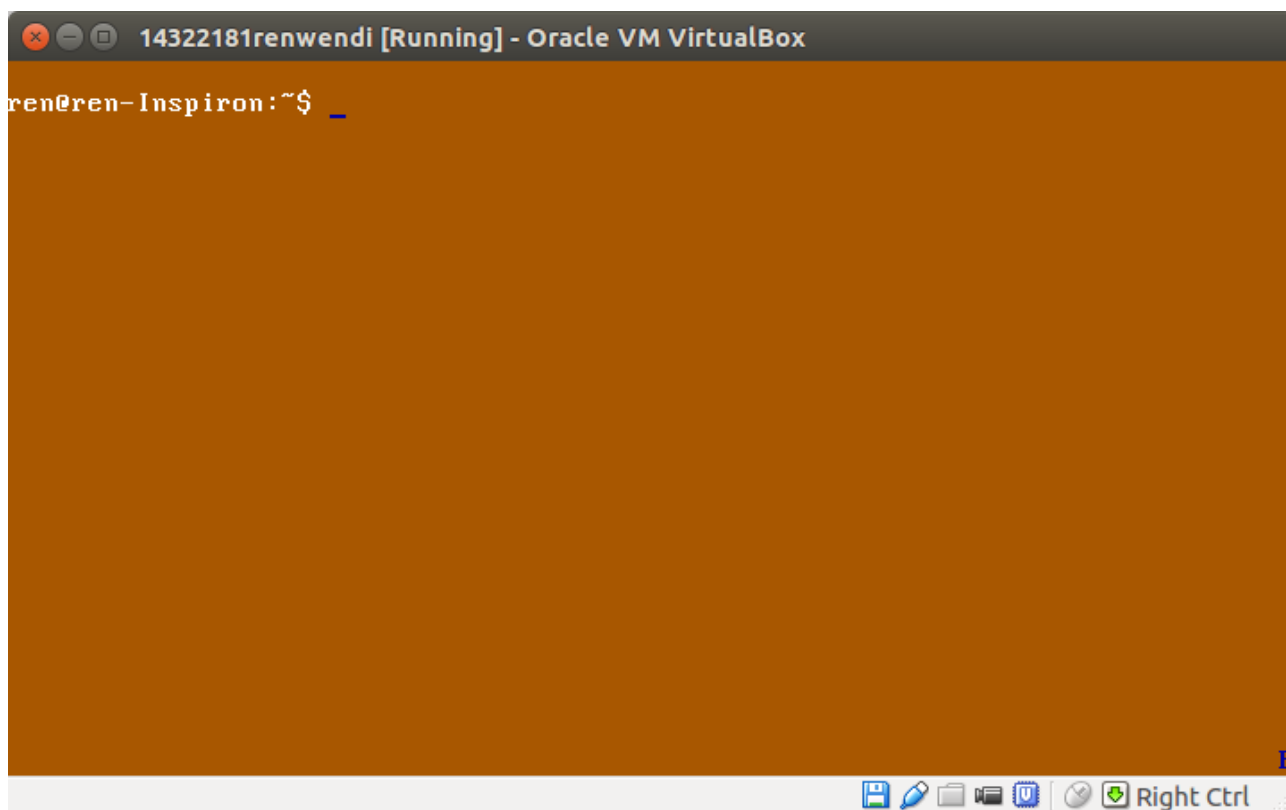
There are some programs you can choose.

Please enter "time" to see the current time of system.
Please enter "date" to see the current date of system.
Please enter "clear" to clean the screen.

Please enter "run number order" to see 4 user programs.
Please enter "int33h/34h/35h/36h" to see 4 interrupt programs.

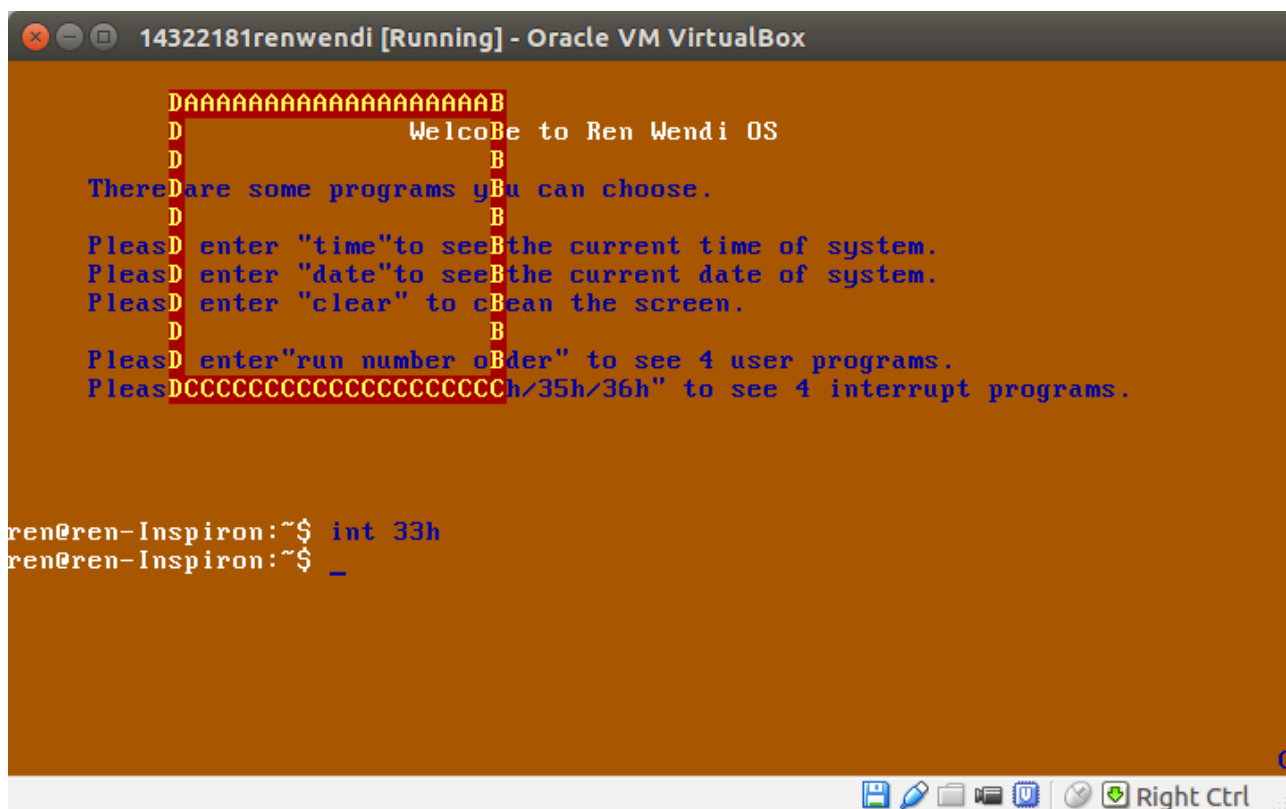
ren@ren-Inspiron:~$ time
Now time is: 22:49:13
ren@ren-Inspiron:~$ date
Now Date is: 2016-05-26
ren@ren-Inspiron:~$ name
No such file or Directory
ren@ren-Inspiron:~$ clear_
```

(4) 清屏之后



(5) 调用33/34/35/36号中断

分别输入int33h/int34h/int35h/int36h

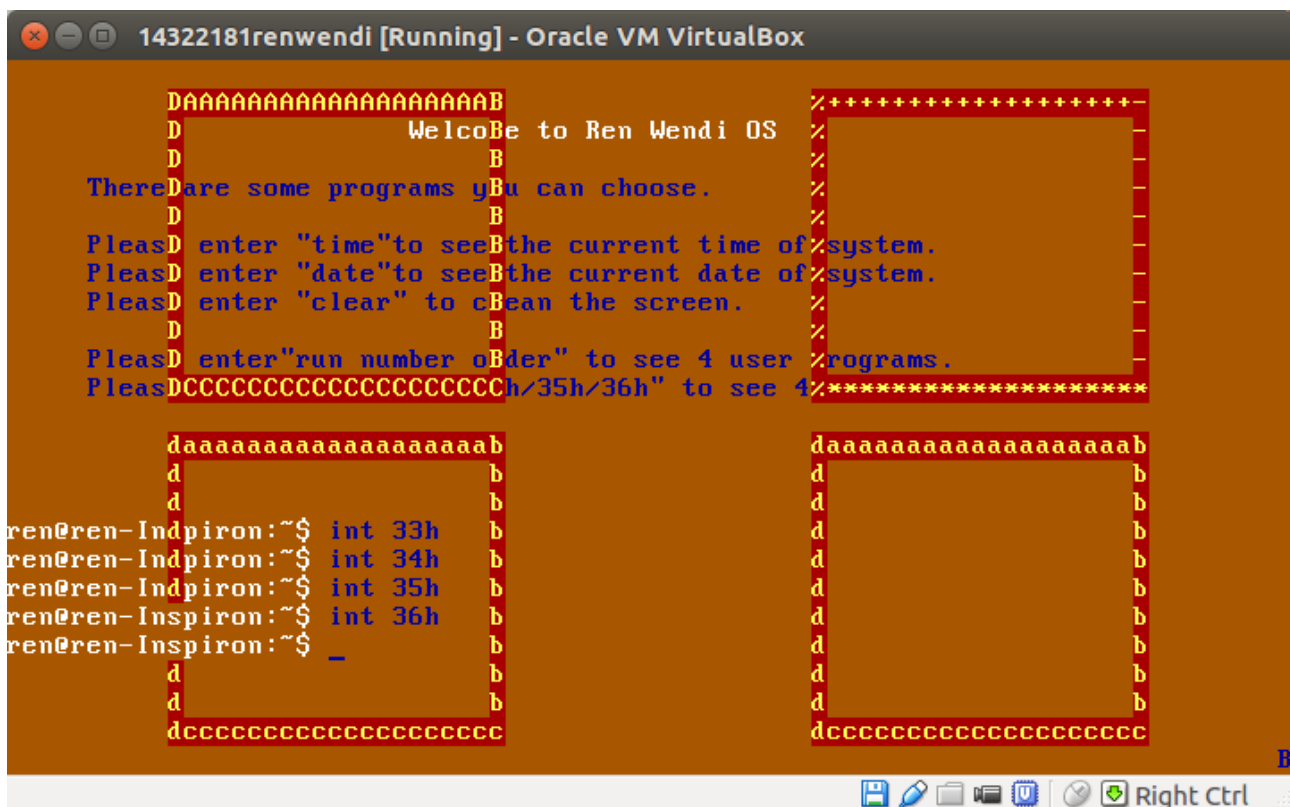


```
14322181renwendi [Running] - Oracle VM VirtualBox
DAAAAAAAAAAAAAAAAAAAAAB
D      WelcoBe to Ren Wendi OS
D      B
ThereDare some programs yBu can choose.
D      B
PleasD enter "time"to seeBthe current time ofxsystem.
PleasD enter "date"to seeBthe current date ofxsystem.
PleasD enter "clear" to cBean the screen.
D      B
PleasD enter"run number oBder" to see 4 user xprograms.
PleasDCCCCCCCCCCCCCCCCCCCCh/35h/36h" to see 4x*****

ren@ren-Inspiron:~$ int 33h
ren@ren-Inspiron:~$ int 34h
ren@ren-Inspiron:~$ _
```

```
14322181renwendi [Running] - Oracle VM VirtualBox
DAAAAAAAAAAAAAAAAAAAAAB
D      WelcoBe to Ren Wendi OS
D      B
ThereDare some programs yBu can choose.
D      B
PleasD enter "time"to seeBthe current time ofxsystem.
PleasD enter "date"to seeBthe current date ofxsystem.
PleasD enter "clear" to cBean the screen.
D      B
PleasD enter"run number oBder" to see 4 user xprograms.
PleasDCCCCCCCCCCCCCCCCCCCCh/35h/36h" to see 4x*****

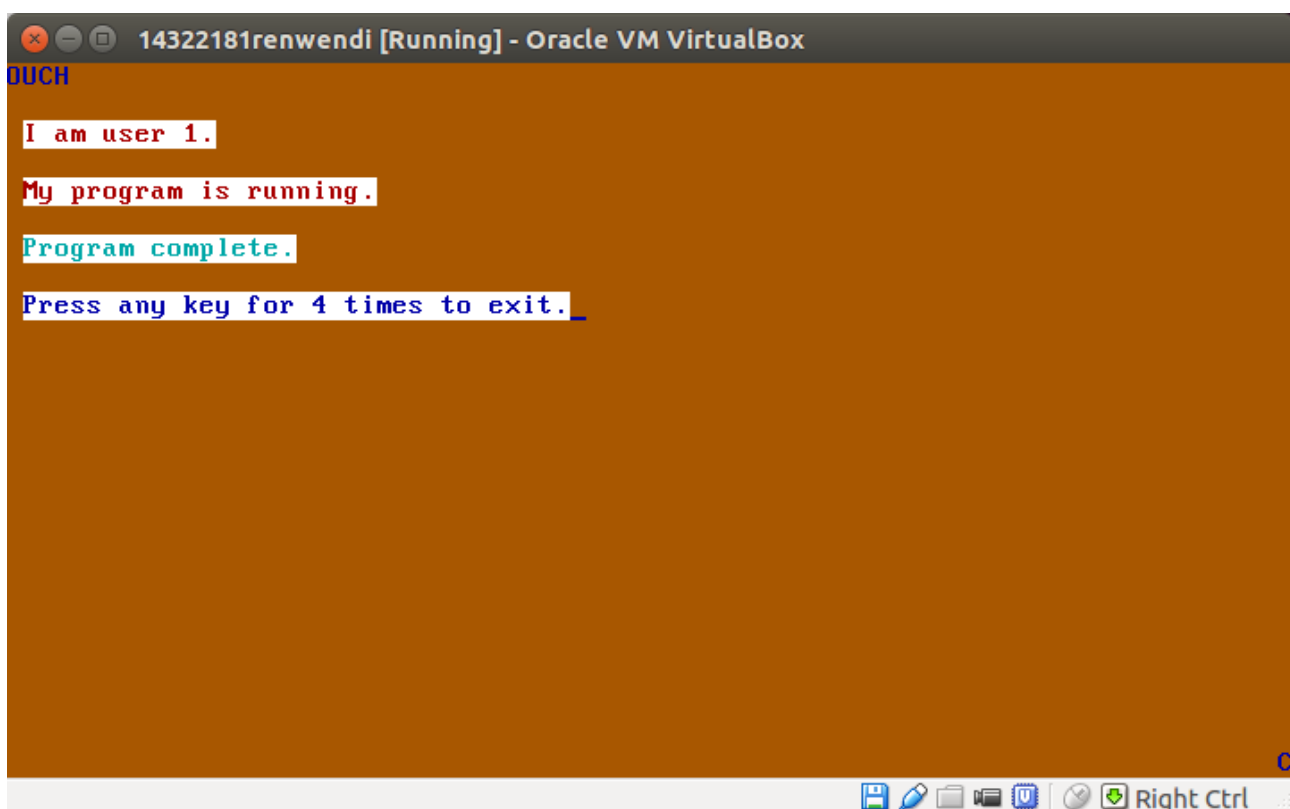
daaaaaaaaaaaaaaaaaaaab
d      b
d      b
ren@ren-Indpiron:~$ int 33h b
ren@ren-Indpiron:~$ int 34h b
ren@ren-Indpiron:~$ int 35h b
ren@ren-Inspiron:~$ _ b
d      b
d      b
d      b
dcccccccccccccccccccc
```

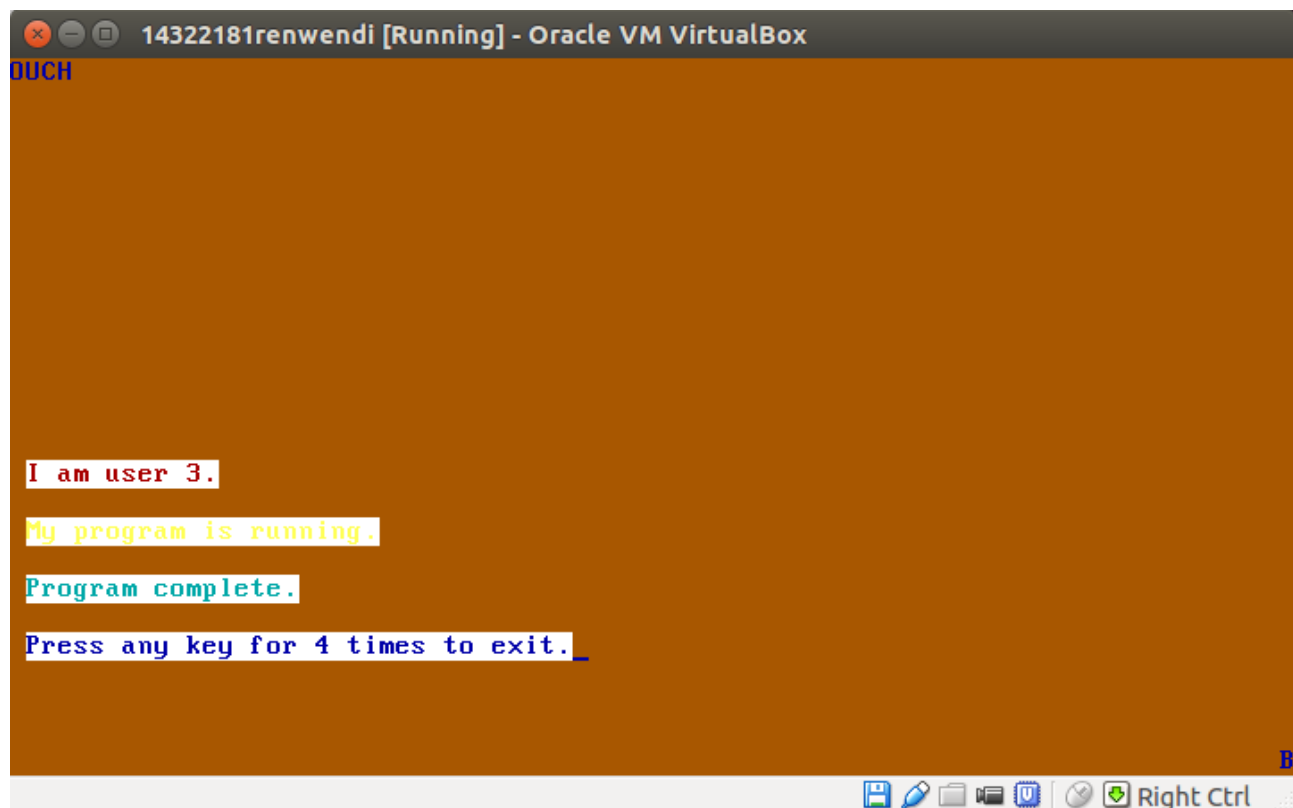
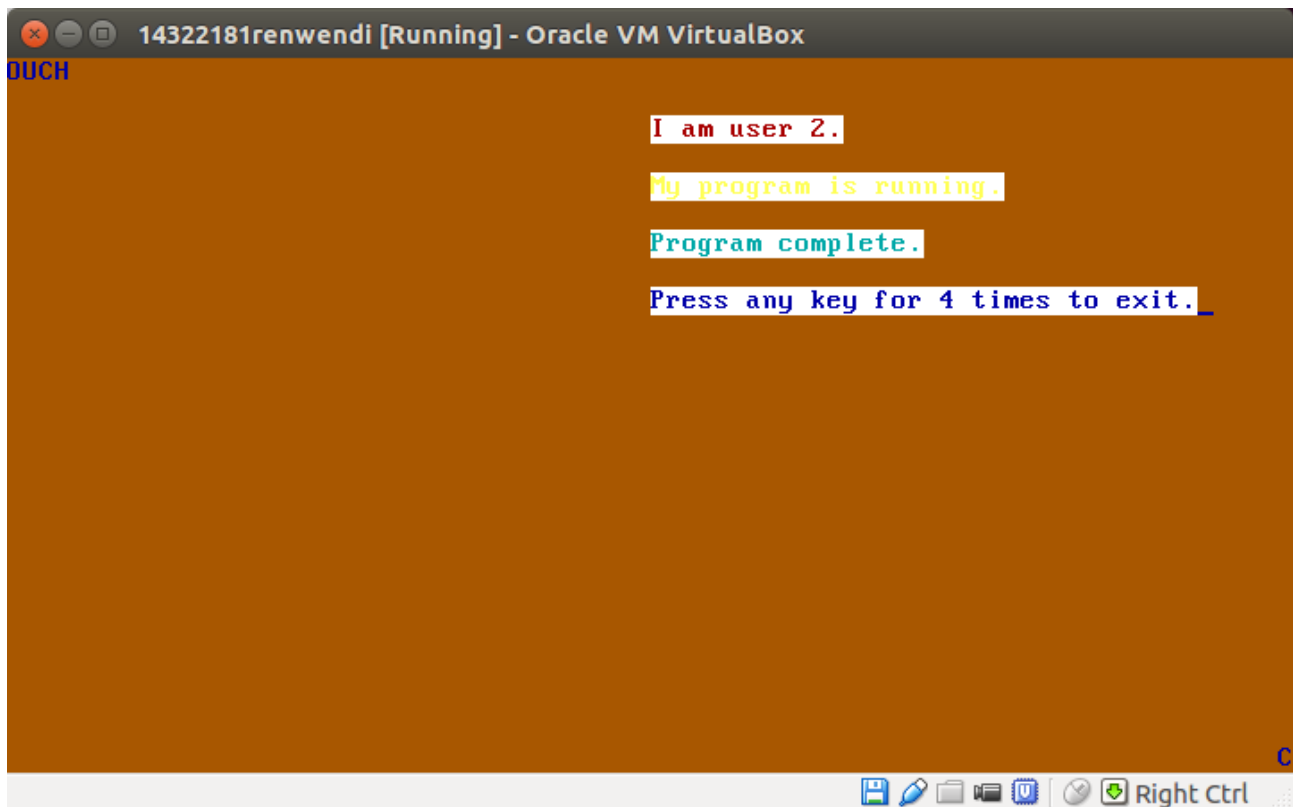


(6) 运行4个用户程序

输入run1234

注意左上角的OUCH



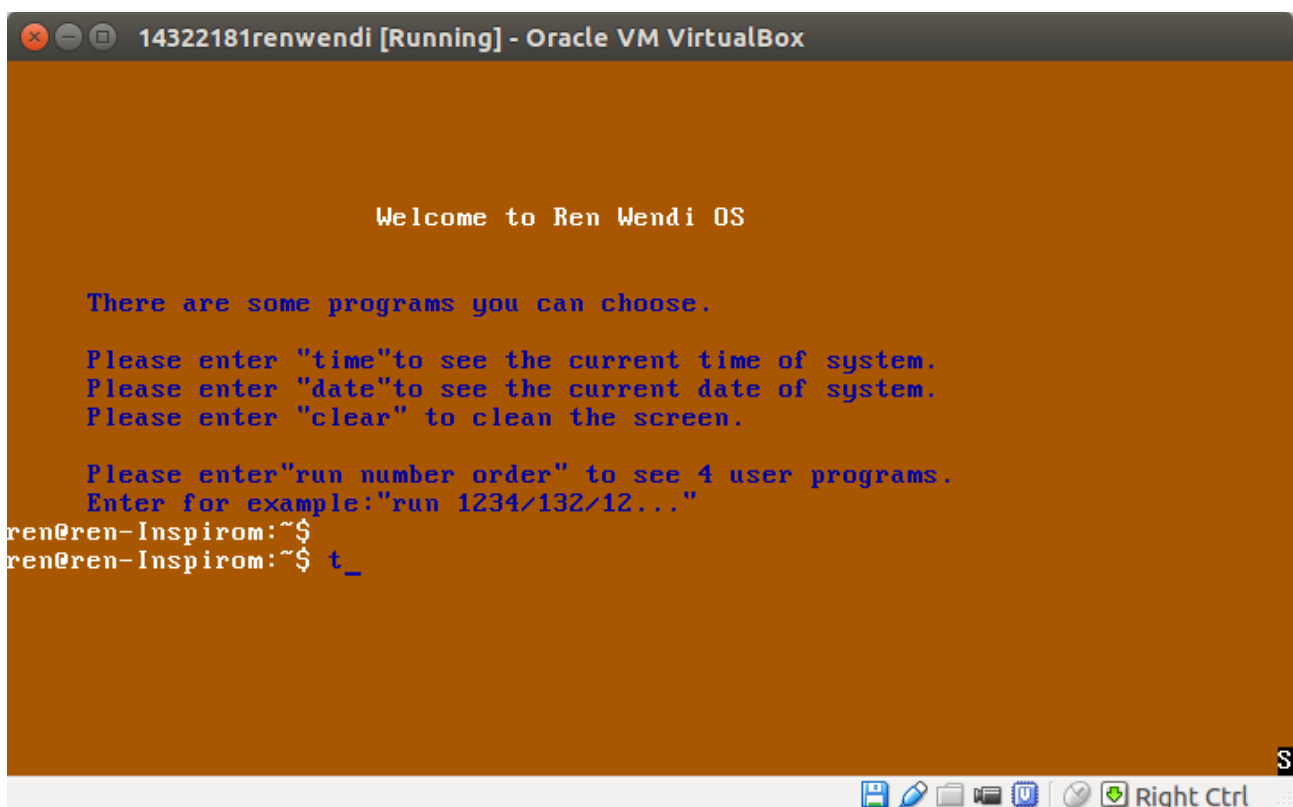




4 遇到的问题及解决情况

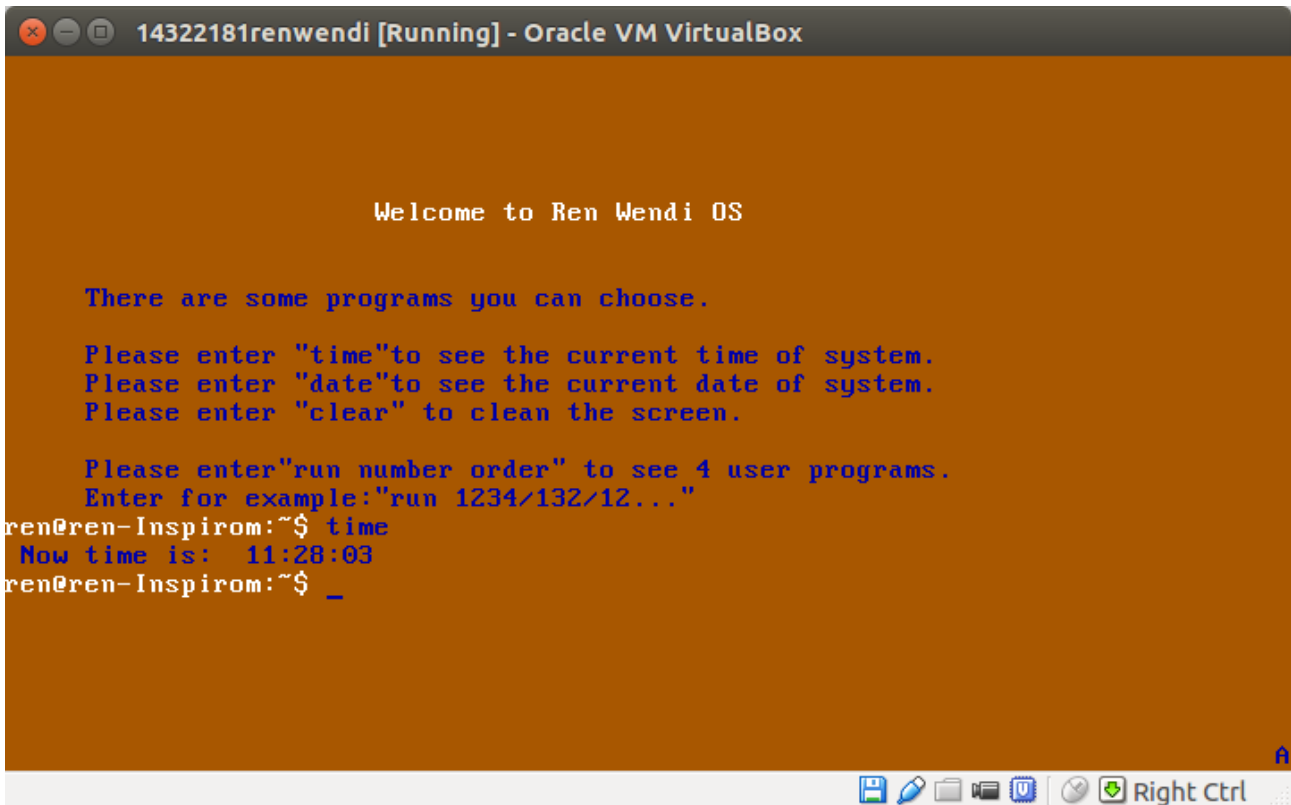
(1) 时钟中断的相关问题

在我设置好初始化时钟中断之后，使用老师所给的参考代码循环显示字符时，会出现时钟中断无法在内核启动时就产生，必须要按下一个键，同时一旦开始进入中断以后便无法再进行其它的按键操作，会一直死循环显示字符。



后来仔细看了看代码觉得直接这样使用是不太合适的，因为代码中有`jmp $`，就是会进入死循环，因此我自己重新编写了显示字符的代码。

但是又出现了新的问题，其它操作是可以正常进行了，如显示日期时间，但字符是不变的，一直显示A。



```
14322181renwendi [Running] - Oracle VM VirtualBox

Welcome to Ren Wendi OS

There are some programs you can choose.

Please enter "time" to see the current time of system.
Please enter "date" to see the current date of system.
Please enter "clear" to clean the screen.

Please enter "run number order" to see 4 user programs.
Enter for example: "run 1234/132/12..."
ren@ren-Inspirom:~$ time
Now time is: 11:28:03
ren@ren-Inspirom:~$ _
```

后来查阅了相关资料，发现硬件中断`08h`会自动触发中断号为`1ch`的时钟中断,又叫`time-tick`中断。通过改写`1ch`的中断处理程序即可实现时钟中断效果。这是我之前不知道的，抱着试一试的心态，将中断向量号从`08h`改为`1ch`之后，果然可以正常发生时钟中断，字符开始变化。

(2) 突然无法调用中断和用户程序

这个问题我觉得非常奇怪，在编写前三个中断向量的时候，字符都可以自动显示，在编写好`36h`向量准备测试显示时，所有的中断都无法成功显示，输入`int3*h`时都会报错，出现提示语`No such file or Dictionary`。

然后再用`run number`调用用户程序后，也出现同样的问题。内核报错，出现提示语`No such file or Dictionary`。

这个问题我能想到的可能原因时内核占用空间超过第14个扇区，但我打开软盘（命令`vim -v test.img / %!xxd`显示软盘二进制文件）后发现并没有超过。其它的原因就不是很清楚了，然后我重新修改了`Makefile`文件，把软盘名新改了一遍，再`make`之后形成新的软盘，在虚拟机中运行又恢复正常。

报错的截图如下：

```
14322181renwendi [Running] - Oracle VM VirtualBox

Welcome to Ren Wendi OS

There are some programs you can choose.

Please enter "time" to see the current time of system.
Please enter "date" to see the current date of system.
Please enter "clear" to clean the screen.

Please enter "run number order" to see 4 user programs.
Enter for example: "run 1234/132/12..."
ren@ren-Inspirom:~$ int 33h
No such file or Directory
ren@ren-Inspirom:~$ int 34h
No such file or Directory
ren@ren-Inspirom:~$ int 35h
No such file or Directory
ren@ren-Inspirom:~$ _
```

```
14322181renwendi [Running] - Oracle VM VirtualBox

Welcome to Ren Wendi OS

There are some programs you can choose, and you can enter h for help.

ren@ren-Inspirom:~$ h
Please enter 'time' to see the current time of system.
Please enter 'date' to see the current date of system.
Please enter 'cls' to clean the scree
Please enter 'run number order' to see 4 user programs.
Enter for example: 'run 1234/132/12...'
Please enter 'int numberh' to see 4 interrupt programs.
Enter for example: 'int 33h/34h/35h/36'
ren@ren-Inspirom:~$ time
Now time is: 20:12:35
ren@ren-Inspirom:~$ date
Now Date is: 2016-05-22
ren@ren-Inspirom:~$ run 1
No such file or Directory
ren@ren-Inspirom:~$ run1
No such file or Directory
ren@ren-Inspirom:~$ int 33h
No such file or Directory
ren@ren-Inspirom:~$ _
```

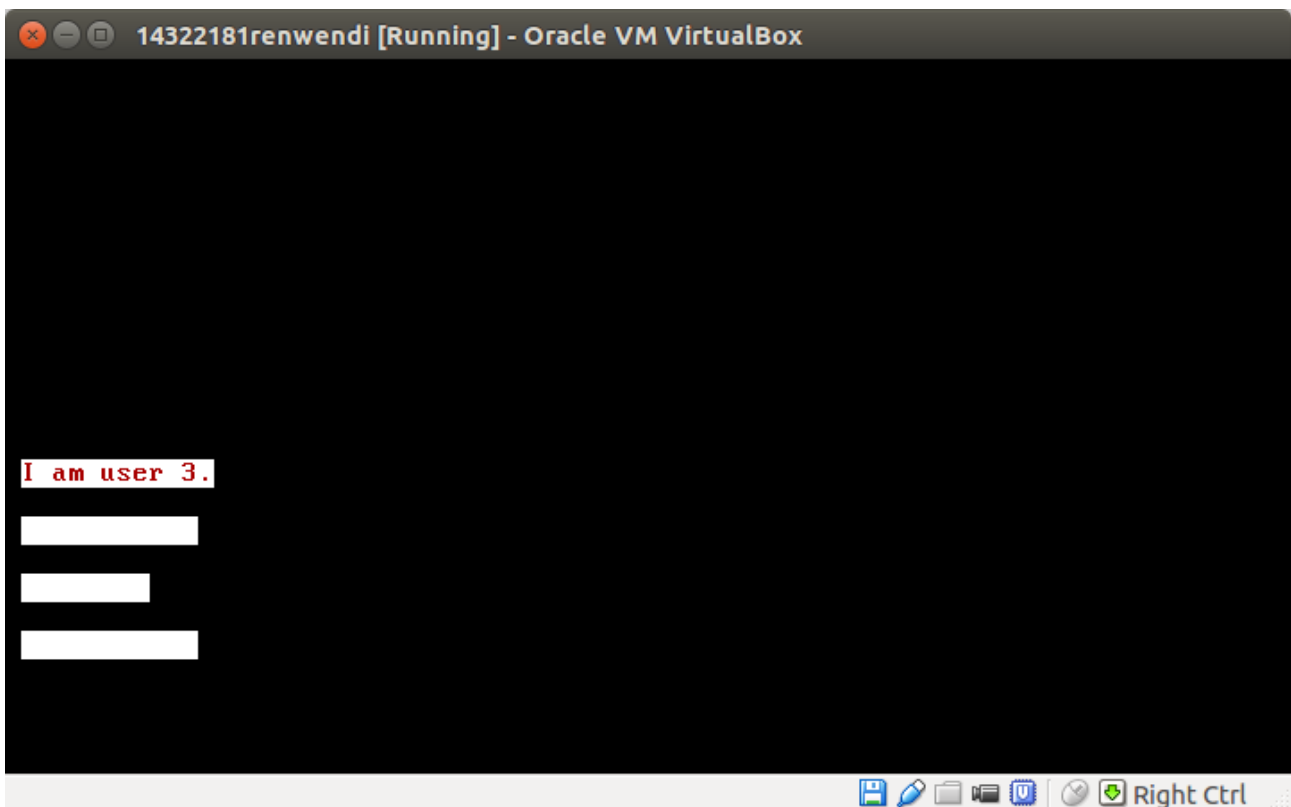
(3) 用户程序字符串不显示

刚开始没有对用户程序进行修改就直接拿之前的用了，然后肯定会出现问题，因为这里是按键4次返回而不是像之前的一样按键一次返回，之前用的是int 16h中断监听用户输入。

以下是修改之后的代码，同时删除了清屏函数来出现黑色底，直接使用内核的清屏函数：

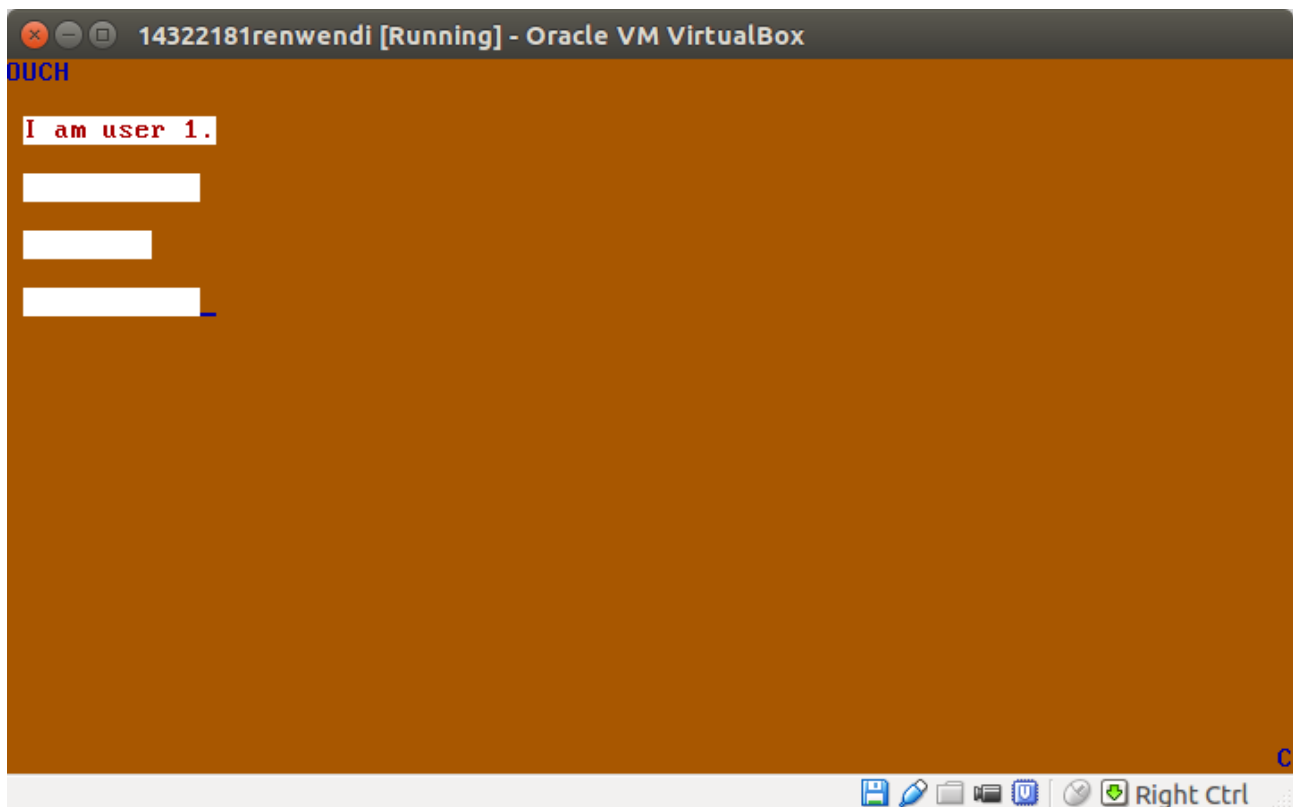
```
64 ;LISTEN_EXIT----  
65 listen:  
66   mov ch,'A'  
67   cmp ch,cl  
68 jne listen  
69  
70 ;-----
```

错误运行截图：



修改之后OUCH可以成功显示了，背景颜色也正常，但是字符串依然不完全显示，只显示第一行，因此我推测是第一行之后出现了什么问题。于是我试着删除了之前的call delay函数（功能：字符串分开一句一句显示），虽然四行字符串只能同时显示，但最终还是都正常显示出来了。

错误运行截图：



【实验总结】

实验心得

本次实验较上一次进行的顺利很多，相比而言中断更加容易理解和实现。通过本次实验，在手动编写中断处理程序和修改添加中断向量表中的中断向量后，更加清楚的了解x86下中断的机制和原理。

看到上一次实验有一些同学用到了内联汇编，因此本次实验中我也进行了尝试。在网上搜索资料学习gcc的内联汇编语法之后，在这次实验中用到了少量的内联汇编，感觉在一些地方用内联汇编会使得代码更为清晰和精练。

中断的实现需要对硬件芯片的结构原理等有所了解，还好在上一个学期的计算机组成原理课上，学习过8259A的相关知识，包括对芯片的初始化编程等，因此本次会进行的容易一些。做好操作系统不仅仅需要编程的知识，这些较为底层的实现还需要理解计算机硬件的知识，这方面也是更加抽象的，在计算机组成原理课上就对芯片这些理解的不够清楚。通过本次实验，手动修改中断向量表之后，对硬件芯片业有了更好的理解。

本次实验看到有同学实现了保护模式下的中断，我也查了一些资料，阅读了于渊的书上中断的相关内容。在《Orange's》上作者主要描述的就是如何实现保护模式下的中断，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。感觉这个难度还是比较大的，因此也就仅限看看理论知识，无法自己实现了。

【参考文献】

于渊 《Orange's——一个操作系统的实现》

GCC内联汇编基础 <http://blog.chinaunix.net/uid-20605433-id-1617453.html>