《操作系统实验》

实验报告

（实验四）

学 院 名 称 ：	数据科学与计算机学院
专业（班级）：	16 计算机类 4 班
学 生 姓 名 ：	陈泓仰
学　　号	15303009
时　　间 ：	2018　年　4　月　10　日

成 绩 ：

实验四、五 ： 中断机制编程技术

一. 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

二. 实验要求

实验的具体内容与要求。

1. 操作系统工作期间，利用时钟中断，在屏幕中显示动态旋转的摩天轮图标。还可加上动态变色的效果，调节适当的速度，方便观察。
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。

三. 实验方案

1. 实验环境

实验运行环境:MacOS

调试用虚拟机:bochs

虚拟机软件:VMware Fusion

2. 实验工具

汇编语言:NASM汇编语言 C语言

编译器:gcc 4.9.2

链接工具:GNU ld

构建工具:GNU make语言

调试工具:Bochs

合并bin文件到统一磁盘映像工具：dd

3. 相关实验原理:

1) 中断相关介绍:

中断分为软中断和硬中断，软中断是指有由指令的运行而触发的中断。但硬中断，是指由外部请求引起的中断，包括时钟中断，I/O中断，硬件故障中断。

2) 中断的相关实现:

a) 中断向量表:

CPU是根据中断号获取中断向量值，即对应中断服务程序的入口地址值，因此为了让CPU通过中断号找到对应的中断向量，操作系统就会在内存中建立一个中断向量表。在X86系统中，每个中断向量由4个字节组成，通过这四

个字节我们可以指定中断服务程序地址的段值与段内偏移量。

b) 调用中断，返回用户程序：

当用户程序需要调用中断时，只需要使用指令INT即可，此时操作系统会将当前程序的标志寄存器，以及下一个指令的段值CS，地址IP依次压栈，然后跳转到中断服务程序执行。

而当中断程序运行结束时，只需要使用指令iret，即可将IP，CS，EFLAGS依次出栈，并返回到用户程序中。

c) 中断的开启与屏蔽：

当我们进入中断程序时，为了防止其他中断的再入，我们需要将中断屏蔽，使用指令CLI，而当我们需要使用其他中断或者在中断程序结束时，则开启中断STI。对于硬件中断，我们需要在中断结束时向CPU的中断控制器8259A芯片送信号，开启中断。

d) 保护现场，恢复现场：

我们每次进入中断的时候，为了防止中断破坏用户程序的运行状态，我们需要将寄存器压栈。其他的一些细节在实现中说明。

3) 键盘中断说明：

INT 09:是由于keyboard上面的普通键被按而产生的。普通键被按后，kbc会侦测到，发出int 09中断。CPU会去执行09号中断对应的程序。这个程序包括：（通过60port）读取操作键的scancode（扫描码）->转换成ASCII码->将扫描码、

字符码存放在一个buffer（缓冲区，在内存）中。

INT 16:是在应用程序去调用它的时候产生的。它有个重要的功能是，当AH = 0时，会去buffer里面读取一个键盘的输入。这个中断的程序包括：循环检测buffer中是否有数据->若有，读取数据->将scancode送入ah,将ASCII码送入al->删除已读取的输入数据。

所以9号中断和16号中断有着不可分开的关系。如果我们改写了9号中断，那么我们会有大概率在应用程序中用不了16号中断。

4. 模块结构：

由于在上一次实验中已经将我的操作系统初步分为：

1) syscall：用汇编语言实现最底层的与I/O端口交互的功能，包括显示字符，输入字符，从软盘中读写程序等。同时用C语言将这些功能进行封装，供其他模块调用。
2) shell：主要是用C语言实现命令行运行。
3) user：存放用户程序的模块。
4) loader：引导程序，将内核程序引导到内存，并开始执行内核。
5) kernel：用C语言来调用其他模块实现的函数，起着逻辑层上的调节作用。这是一个操作系统中最为核心的地方。

因此在这次实验中，我将中断初始化，编写中断函数这些工作放在syscall这个底层模块中去实现，同时还开了一个新的C文件去存一些实现函数，简化汇编代码量。

5. 程序实现的功能描述：

1) 利用时钟中断在屏幕左上角画变色滚轮。

2) 通过键盘中断在屏幕左上角响应用户的键盘操作，显示变色的”OUCH!OUCH!”

3) 利用33，34，35，36这四个中断分别在四个用户程序中显示一些特别的字符串或图案。

4) 利用第21号中断这个系统调用号，实现读取用户输入并存入缓冲区，显示字符，回显，清屏四个功能。

四. 实验过程与结果

说明：根据需要书写相关内容，如：

1. 关键模块及程序说明：

这次实验最重要的是掌握中断程序的编写，以及理解中断的运行过程，在了解了中断的相关实现原理之后，我们来看一些比较细节的代码及说明。

1) 首先对代码结构进行优化，建立了C库stdio.h

加入了相关的输入输出函数包括：

输入字符；显示字符；输入字符串；显示字符串；显示数字；按不同的颜色显示字符串；获取字符串长度；比对字符串；

```
#ifndef STDIO_H
#define STDIO_H
extern void _printchar(char pchar, int pos,int color);
extern char _readinput();
extern void _showchar(char phar);
//extern void _RunProgress(void* addr);
void print(char const* Messeage,int row,int colume);
void prints(char const *Messeage);
void print_next_line(char const* Messeage);
void printc(char alpha);
int strlen(char const *Messeage);
void read_and_print_input();
char getch();
void  getline(char str[],int length);
void print_different_color(char const* Messeage,int row,int colume,int color);
int strcmp(char *m1,char const *m2);
#endif
```

正在加载...

2) 装载中断宏：

```
%macro SetInt 2;载入中断向量表的宏
    push es;++++++必须要加
    push ds;++++++++++++必须要加
    pusha
    mov ax, cs;++++++++++++必须要加
    mov ds, ax;++++++++++++必须要加
    mov ax,0
    mov es,ax
    mov ax,%1
    mov bx,4
    mul bx
    mov di,ax
    mov ax,%2
    mov [es:di],ax
    mov ax,cs
    mov [es:di+2],ax
    popa
    pop ds;++++++++++++必须要加
    pop es;++++++++++++必须要加
%endmacro
```

由于每次写中断函数之后，我们最重要的一步就是初始化中断函数，将中断装入中断向量表中，因此写了一个宏来简化代码量。要注意到这里，需要对所有的寄存器包括段寄存器进行保护。否则寄存器的值会被改变。

3) 保护现场与恢复现场：

保护现场:主要是对各个寄存器的保护，使用指令pusha,将ax,bx,cx,dx,bp,di,si,sp这

8个寄存器全部压入栈中，同时还需要注意对段寄存器的保护。

恢复现场:在中断程序运行结束要返回用户程序时需要对一开始为了保护现场而压入栈中的各个寄存器按逆序弹出。

4) 中断汇编函数中调用C函数:

由于我是在GCC和NASM交叉汇编的环境下编写操作系统,所以程序调用过程中总会出现各种各样的问题，其中就包括汇编函数调用了C函数之后返回不了汇编函数。在使用了bochs调试之后，我发现在C函数返回汇编函数的时候会同时弹出CS和IP（俩个字），为了平衡堆栈，所以就需要我们在调用C函数之前先在栈中压入一个字。这样才能保证最后正确返回汇编函数。

```
_SetINT35h:
    pusha
    push ds
    push gs
    push word 0;In order to get back to interupt
    call printpoem
    pop gs
    pop ds
    popa
    STI
    iret
```

5) 时钟中断

由于时钟中断是硬中断，CPU会不断的自动调用这个中断，所以我们不需要显式的调用这个中断。如果我们想利用时钟中断实现些什么的话，我们可以改写8号中断，这为我们实现分时系统打下了基础。

同样的在编写时钟中断程序的时候，我们需要保护现场，除此之外在每次结束程序的时候我们需要向中断控制器8529A发送中断使能信号。

```
_SetINT08h:
    pusha
    push gs
    dec byte [count]
    jnz end
notc:...

show:...

end:
    mov al,20h
    out 20h,al
    out 0A0H,al
    pop gs
    popa
    STI
    iret
```

6) 键盘中断：

当键盘有按键或释放按键事件时，调用09h中断。但是在这里我们必须注意到，bios的第16号中断恰恰就有用到09号中断来作为自己的一部分实现。所以如果我们破坏了9号中断而置之不理的话，我们之前使用16号中断实现的函数也会用不了。所以在使用9号中断之前我们需要先将它存起来，然后在使用完9号中断、执行完改写的9号中断任务程序之后将执行原来的9号中断任务程序。

于是我们把9号中断的装载程序独立出来：

```
_initialInt_09h:
    enter 0,0
    mov ax, [09h * 4]
    mov word [int_09_saved], ax
    mov ax, [09h * 4 + 2]
    mov word [int_09_saved + 2], ax
    SetInt 09h,_SetINT09h
    leave
    newret
```

把原有的9号中断程序放在es端中的0000中。然后加载改写的9号键盘中断程序

```
_SetINT09h:
    pusha
    mov cx, 0xB800
    mov gs, cx
    mov ah, byte [color]
handle:  ;handle the print "OUCH!OUCH!"
cont:
    inc byte [color]
    push es
    push ax
    mov ax, cs
    mov es, ax
    sti;Enable interupt
```

执行完中断程序之后，将继续执行原来的9号中断程序。

```
    pushf;push psw
    call far [es:int_09_saved]

    pop ax
    pop es
    iret
```

7) 通过系统功能号划分21号中断

调用之前，需要先将功能号压栈。

```
push    ax          ;系统调用功能号0压栈，传递给系统服务处理程序
int 21h             ;产生中断，用户程序ouch.asm中，调用0号功能。
```

而在汇编函数中，则是读取压入的参数，来决定该进行哪一个分支，执行哪一个函数。

区分过程如下：

```
_SetINT21h:
    enter 0,0
    pusha
    mov ax,[bp+8];调用int 21h,压入了ebp,和一个ip（？？？），
    cmp ah,0
    jz fn0
    cmp ah,1
    jz fn1
    cmp ah,2
    jz fn2
    cmp ah,3
    jz fn3
    popa
    leave
    iret
```

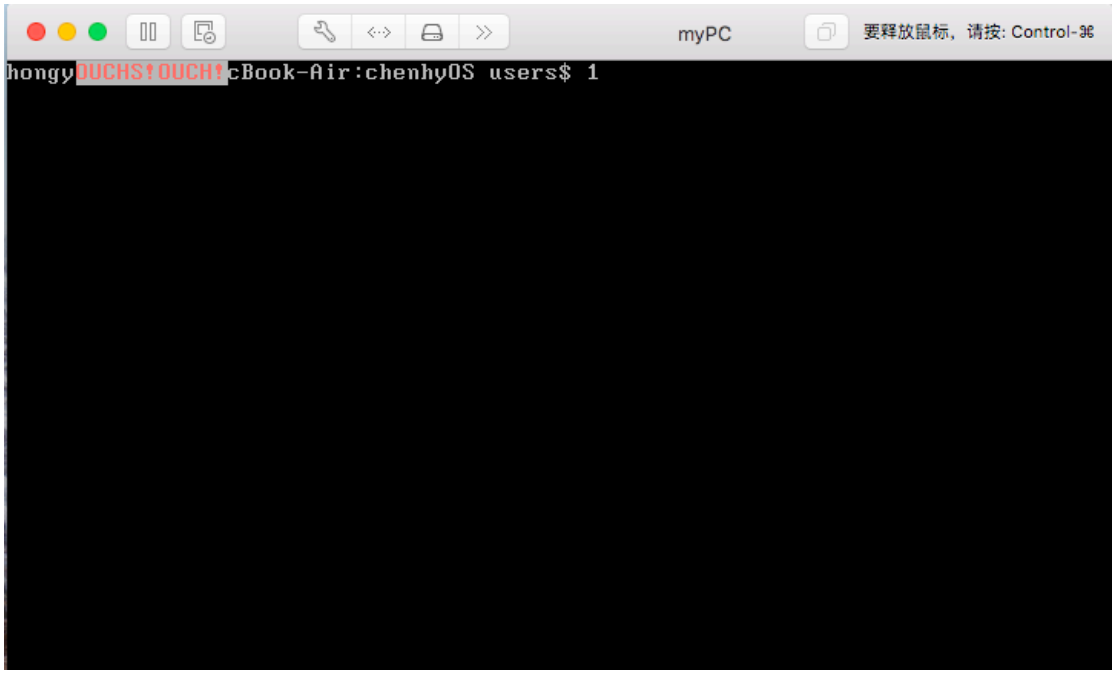
这里需要指明的是，我们在汇编函数中要取得压入的功能号，应该是从[bp+8]处取得，这一点和在C函数调用汇编函数，汇编取得C函数的参数是从[bp + 6]处开始的不同。

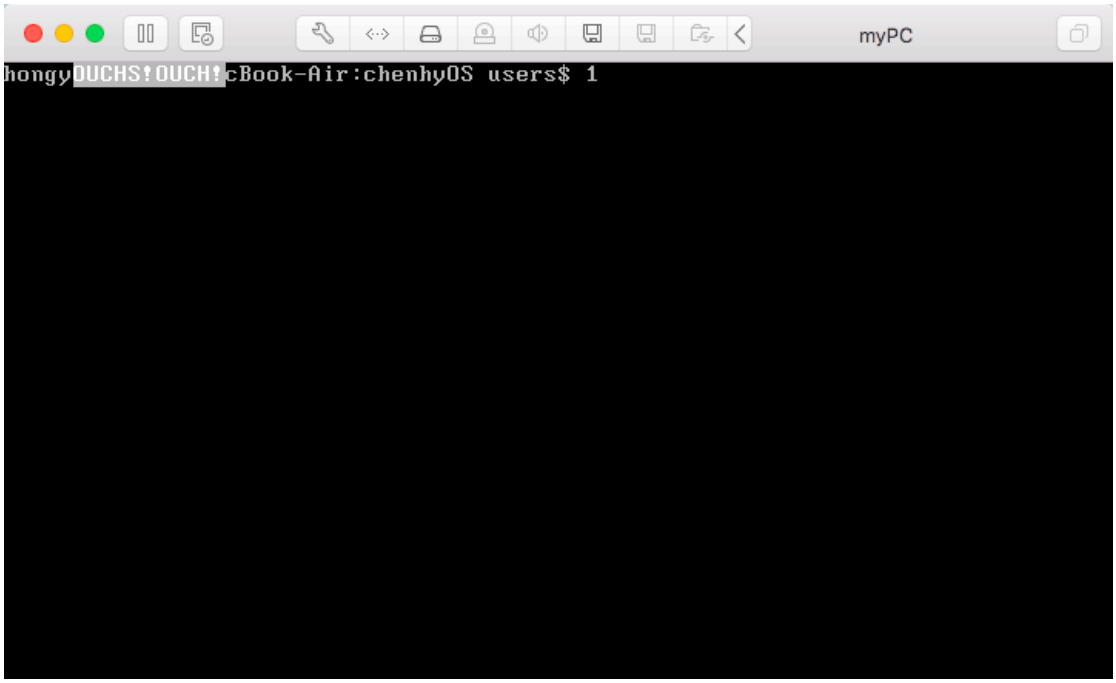
8) 在加载不同扇区的程序时候，发现了如果要读到第18个扇区以外的程序，需要经过换算才能正确的找到扇区，于是写了一个loadP和RunP的函数，传入三个变量（内存，起始扇区号，读取扇区个数），方便以后的读扇区工作。

2. 实验运行效果：

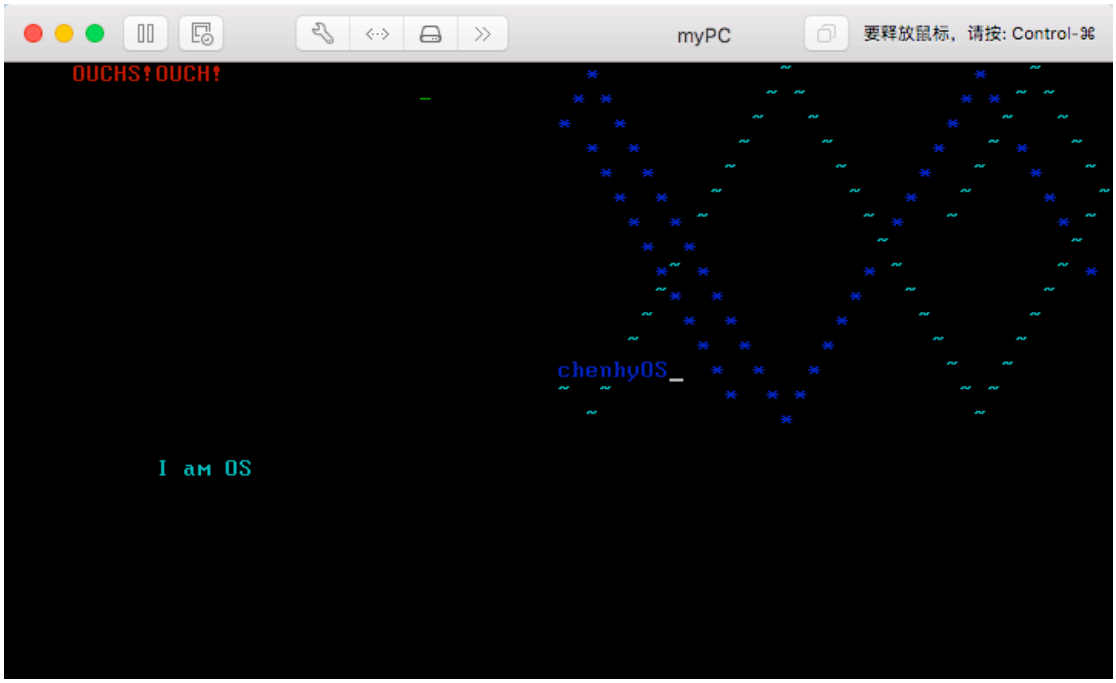
键盘中断：

输入字符的时候，字符串“OUCH！OUCH！”会变色：


myPC
要释放鼠标，请按: Control-⌘
hongyOUCHS!OUCH!cBook-Air:chenhyOS users$ 1


myPC
hongyOUCHS!OUCH!cBook-Air:chenhyOS users$ 1

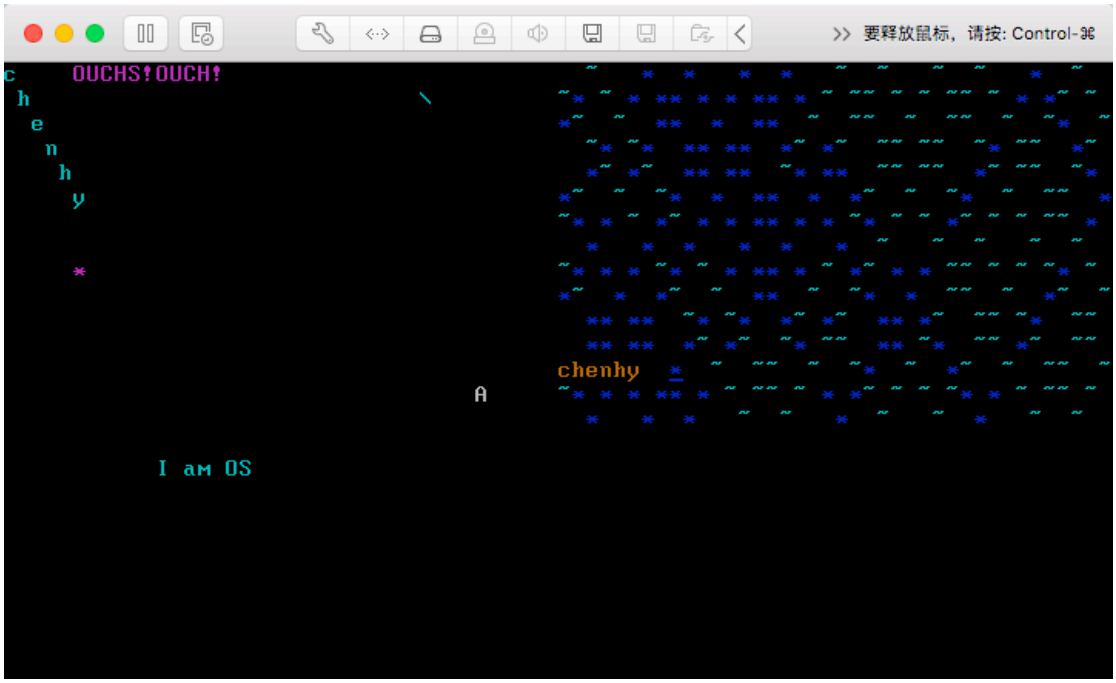
时钟中断：



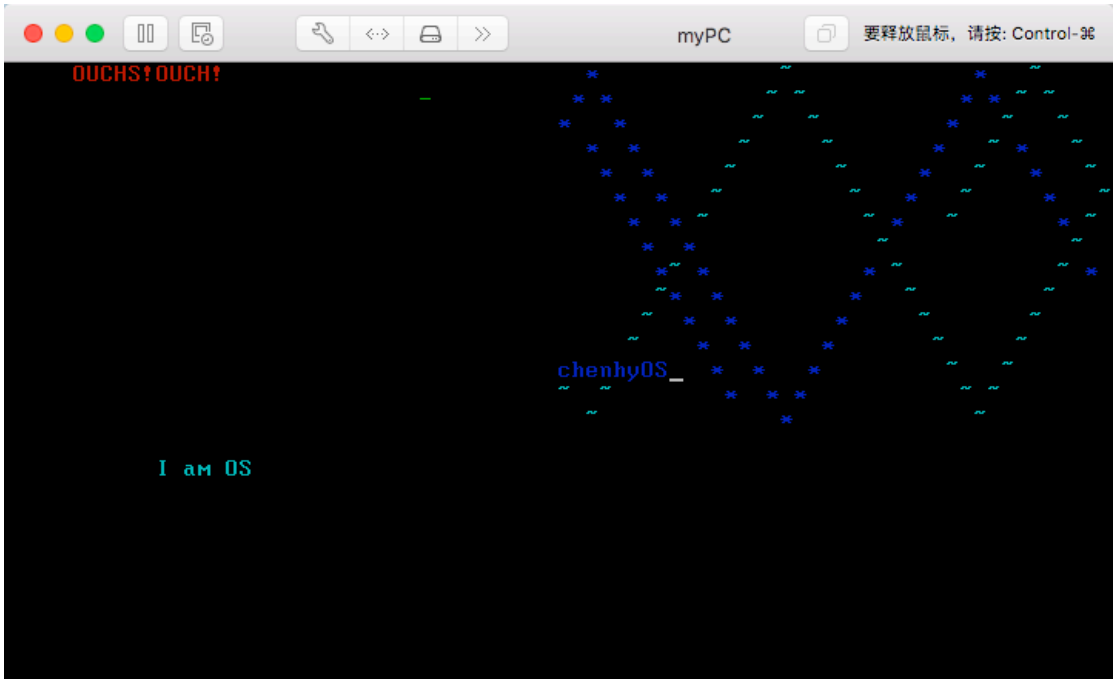
旋转的指针（但是，不知道为什么，只有在用户程序里会弹出来，而在内核里面不显示）

四个中断展示（33，34，35，36）

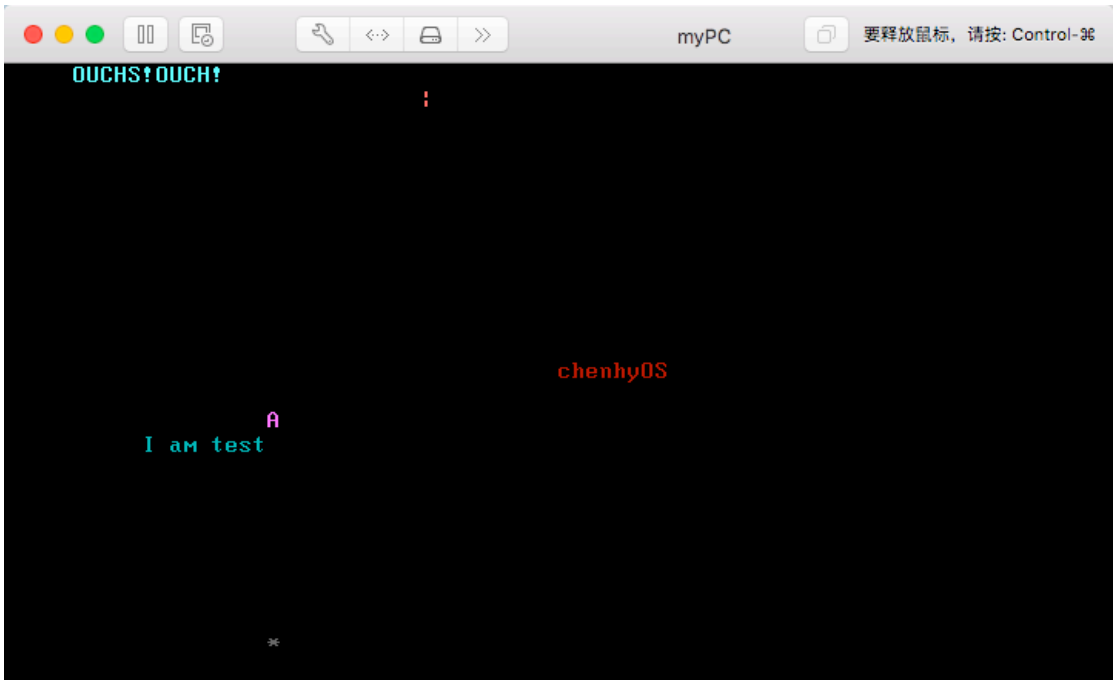
33（左上角斜着显示“chenhy”）：



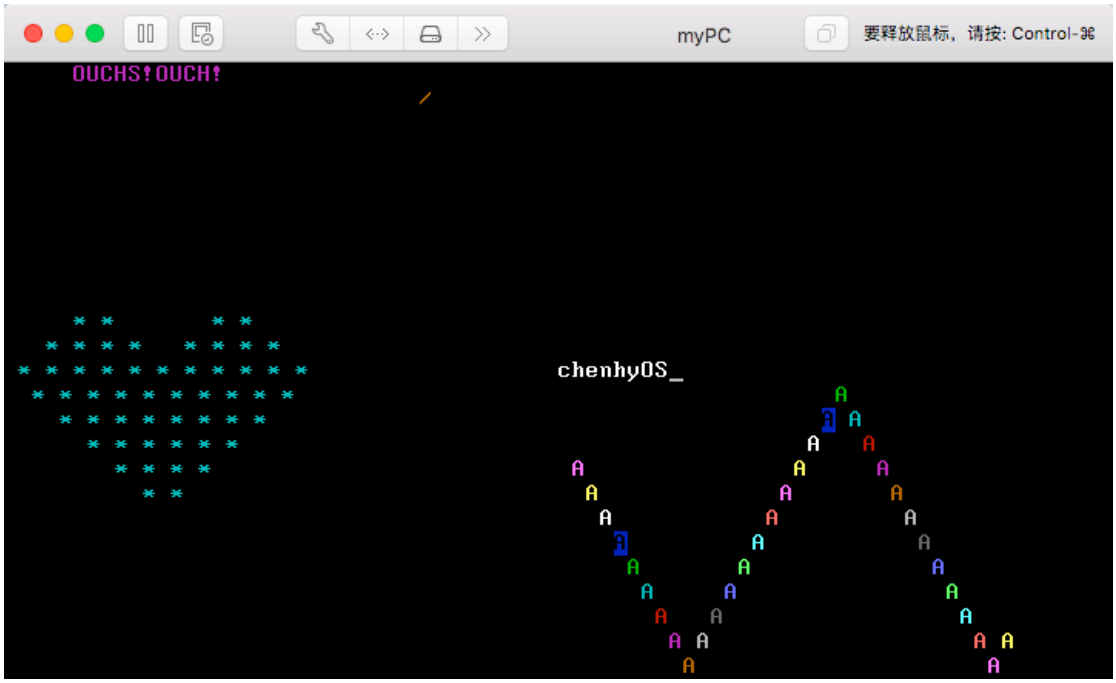
34（显示“I am OS）；



35（显示“I am test”）：



36(显示一个爱心):



21号中断共有四个功能，读取字符显示，输入字符到缓冲区，回显字符，清屏，这四个功能就不一一展示。

五. 实验心得

体会和建议。

由于之前在第二次实验，加载用户程序中，我就用过软中断来返回内核程序，所以这次实验一开始，我并没有觉得实现起来特别困难，只要克服俩个硬中断就可以。但是恰恰是时钟中断这个硬中断让我卡在这里三四天。由于它是一个硬中断，我们很难使用bochs调试工具去调试它，所以只能从代码本身入手，一步一步的去锁定出错的位置在哪里。在写时钟中断的时候，我是分成俩步去实现的。

首先，我先将老师的样例代码放在自己的程序中，但是由于并没有做任何保护现场的工作，所以这个字符一直都没有显示出来，后来在舍友的提醒之下，我终于意识到自己的错误，

将寄存器保护起来，终于能让时钟中断运行起来。但每次进去内核总得在内核里等个十几秒，才能看到字符不断的变化。在这之后，我又仔细的思考我的各种函数会造成的各种影响，适时的进行压栈。最后终于得到一个一进内核无需等待就能直接运行的时钟中断。但是在第二步，当我开始想搞点新花样，让时钟中断实现点新的东西的时候，我发现我内核里的命令行解析函数，都出现了bug。要不就是输入进去的字符串匹配的结果不正确，要不就是输入的时候操作系统就死机。这一次，我直接开始找我C语言函数的错误，比如有一个错误就是，我的显示字符串的C函数是直接对字符串的指针进行操作，执行程序之后，我的指针指向了字符串的串位，这就导致我在下一次显示相同的字符串的时候会显示不出来。我的最后一个bug就是，我在terminal函数中定义了一个用来存用户输入的字符串，这个字符串可以用来解析命令，可是在最后解析命令之后，我对这个字符串的清空操作并不正确，最后，我把清空字符串的函数重新改写之后，时钟中断终于不会对我的内核命令行里的字符串解析函数有副作用了。

同时说到键盘中断，由于9号中断和16号中断两者有着不可分割的关系，所以改写了9号中断，破坏了9号中断原有的功能的话，16号中断就无法使用。这个问题困扰了我好一段时间，但最后在舍友的建议下，我把原来9号中断的程序放在内存的一块地方，然后先执行我的程序，然后在跳到原来9号中断的那块位置执行。这样既能完成自己的键盘中断，又能不阻碍到原有的键盘中断，两全其美。

最后在将C库分离时，方法什么的都懂。就是新建一个h文件和C文件，然后把声明放在h文件中，把定义放在C文件中，然后其他文件将h文件包含进去即可。但是在链接过程中一直无法完整的读到数据。最后发现是引导程序读少了内核程序的扇区。改完这个内核就完整啦！

虽说这次实验，我顺利的实现了键盘中断和时钟中断，写了系统调用，但是还是有一些

不足的地方。首先，我的时钟中断无法在内核程序里面体现出来，必须跳到用户程序才能看到斜杠转动的效果。其次，为了减少代码量，我并没有实现老师要求的画框程序。虽然还是留着一些小小的不足之处，但是这次实验还是让我付出了一个星期的心血，特别是在实现时钟中断的时候，无法靠任何调试工具，只能靠不断的调参和人脑debug来解决程序出现的问题，所以最后能做出这个实验，我还是比较激动的。

参考资料

1.linux内核代码：https://github.com/torvalds/linux

2. Override default INT 9h:

https://stackoverflow.com/questions/12882342/override-default-int-9h