实验四 ：中断机制编程技术

目录

一/二.实验目的和要求

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，学习握掌 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.实验工具与环境（gcc+nasm+ld）

nasm 编译汇编代码生成.o 文件，gcc 编译 C 模块生成.o 文件；ld 链接.o 文件生成.bin 文件；dd 将.bin 写入映像文件.img；vmware 上添加.img 文件到软盘后运行。

2.方案的思想

1.相关原理：

①中断处理的过程：

中断处理有三个步骤。（其中一三步需要自己在中断响应程序中实现。第二步由系统自身实现，但需自己修改中断向量表。）

第一步是保护断点的现场。在这一步有两件事情要做，首先是将**标志寄存器FLAGS压栈**，再**清除它的IF位和TF位**。清除IF位和TF位是为了避免在中断中响应其他中断，所以如果你在自己的中断中想要响应其他中断的话，则必须先把if和tf位置一，可以使用sti命令。然后则是要将当前的**代码段寄存器CS和指令指针寄存器IP压栈**，以保证中断执行完后可以返回当前代码段。

第二步则是执行中断处理程序。处理器拿到中断号后，它将该**号码乘以4**（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，然后，处理器就开始执行中断处理程序了。

第三步是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。

②关于系统调用

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

③ret, retf, iret指令的差异：

ret指令是段内返回指令，只从堆栈返回ip

retf指令是段间返回指令，从堆栈返回ip和cs

iret指令是中断返回指令，从堆栈返回ip，cs和FLAGS

④获取当前时间：

当前的时间存放在CMOS RAM中。每个时间占一个字节，存储格式是BCD码，高四位放十位数，第四位放个位数。年只存放最后两位，如现在的年份只存储18.存放单元如下：

秒：0　　分：2　　时：4　　日：7　　月：8　　年：9

要读取时间时，先向地址端口70h写入要访问的单元的地址，然后就可以从数据端口71h中取得指定单元中的数据，如取当前小时：

```
mov al,4
out 70h,al
in al,71h
```

⑤关于关机和重启：

写0x2001进端口0x1004可以使电脑关机；写0xfe进端口0x64可以使电脑重启。

2.实验内容

1.设计自己的时钟中断响应程序：

在屏幕的右下角沿矩形动态画框，并循环变色。

2.改写键盘响应中断程序：

每当有按键输入的时候，正常获取按键后，在屏幕又下方显示“OUCH”，在时钟中断中将该“OUCH”清空，使得每次按键输入都闪现“OUCH”

3.设计自己的中断响应程序int 34，int 35，int 36，int 37：

分别在左上角，右上角，左下角，右下角播放逐字符飞入动画显示个人信息。

4.设计自己的系统调用程序，通过ah的值调用int 21h中断的不同功能：

①获取当前时间并显示于右下角 ah = 0

②切换背景颜色　ah = 1

③切换前景颜色　ah = 2

④关机 ah = 3

⑤重启 ah = 4

5.设计一个简单的用户子程序来测试所有的中断与系统调用

3.实现方法：

①中断处理实现方法：

首先要设计中断响应程序，中断响应程序开头保存断点环境，结尾恢复环境并返回断点。而中断响应程序的功能则根据具体需要来设计。

设计好中断响应程序之后，要把中断响应程序的首地址(即中断向量)放到中断向量表中。所有中断响应程序都需要把程序首地址放到对应的中断向量表。在发生中断时，处理器会通过中断号自动执行中断响应程序。

②时钟中断响应程序实现方法：

为实现字符的显示与移动，我们需要记录当前字符所在的位置和它的移动方向。因此我们用变量x，y代表字符在屏幕上的坐标，用rdul表示当前字符的方向。每次进入时钟中断的时候，就根据rdul的值改变x，y的值，而后通过判断x，y的值是否超出界限来改变方向rdul的值，并重新对x，y赋值。得到更新后的x，y，后即可在屏幕上对它进行显示。要实现字符变换以及颜色变化的话，只需要用变量记录字符和颜色的状态，并在方向改变时改变字符的值，在显示的时候改变颜色的值。

③重写键盘中断响应程序的实现方法：

在这个程序前，要**事先将旧的9号中断的地址存储起来，进入这个程序后先响应旧的9号中断**，然后再在VGA上显示出“OUCH”，即可。

④34,35,36,37中断响应程序：

由于这四个中断实现的功能相似，我只解释34号中断响应程序，其他类似。动画的播放其实就是控制时延使得文字一个一个的显示。这需要保存字符的位置，然后每隔一段时延将字符向下移动一格。时延控制好就达到动画的效果了。

⑤21h系统调用响应程序：

首先是关于获取系统时间，在基本原理处我已经将如何获取时间说的比较清楚，得到从汇编获取时间的6个接口后，我使用自己写的类型转换函数将int转换为字符串类型，然后用自己写的连接字符串函数将几个字符串串联起来，做简单输出即可。

然后是背景颜色和前景颜色的切换。这个需要两个底层汇编端口读取颜色和设置颜色。通过遍历25*80个位置，获取该位置背景色和前景色，再重新设置背景色或前景色即可，要注意避免背景色与前景色颜色相同。

至于关机和重启，只要写特定数据进对应端口即可。

3.程序流程：

1.引导程序加载内核。
2.(内核首先修改中断向量表),内核作为控制台，通过命令加载用户子程序并跳转到子程序运行。
3.用户子程序执行期间响应硬件中断，软中断以及系统调用。
4.从用户子程序返回内核响应中断响应程序，执行中断响应程序。
5.从中断响应程序返回用户子程序并继续执行。

4.程序关键模块：

1. 时钟中断响应程序模块：

```
int_8:
    pushf
    pushad
    cli
    ;计时
    dec byte[cs:count]
    jnz end_
    mov byte[cs:count], delay
    ;移动字符,输出
dic:
    cmp byte[cs:rdul],1
    jz DN
    cmp byte[cs:rdul],2
    jz RT
    cmp byte[cs:rdul],3
    jz UP
    cmp byte[cs:rdul],4
    jz LF
DN:
    inc byte[cs:x]
    cmp byte[cs:x],25
    jnz show
    mov byte[cs:rdul],2
    dec byte[cs:x]
    inc byte[cs:y]
    jmp show
```

首先进中断要先保存各个寄存器的值，避免改变了寄存器的值导致外部程序崩溃。x，y 代表当前字符坐标，rdul 代表方向。首先判断当前字符方向，根据方向跳转到不同的地方做不同的移动，移动后判断边界条件，如果未出界则显示，出界则重新设置坐标方向。

2. 键盘中断响应程序模块：
初始化 9 号中断向量地址：

```
Setaddr:
;判断是否初始化了,已设置就跳转
    cmp byte[cs:is_init],1
    jz start
    inc byte[cs:is_init]
;时钟中断响应地址
    xor ax,ax
    mov es,ax
    mov word[es:20h],int_8
    mov word[es:22h],cs
;键盘中断响应地址
    mov ax,word[es:24h]
    mov word[cs:pre_int9_addr],ax
    mov ax,word[es:26h]
    mov word[cs:pre_int9_addr+2],ax
    mov word[es:24h],int_9
    mov word[es:26h],cs
```

之所以要判断是否初始化，是考虑从子程序返回内核，如果再次初始化，则会把

自己安装的 9 号中断地址传送到用于存放旧的 9 号中断地址的变量 pre_int9_addr。所以只能赋值一次。
9 号中断内容：

```
int_9:
;调用旧的9号中断
    pushf
    cli
    call far[cs:pre_int9_addr]
    pushf
    pushad
    push gs
    mov ax,0b800h
    mov gs,ax
    mov byte[gs:(23*80+72)*2],'O'
    mov byte[gs:(23*80+72)*2+1],0fh
    mov byte[gs:(23*80+73)*2],'U'
    mov byte[gs:(23*80+73)*2+1],0fh
    mov byte[gs:(23*80+74)*2],'C'
    mov byte[gs:(23*80+74)*2+1],0fh
    mov byte[gs:(23*80+75)*2],'H'
    mov byte[gs:(23*80+75)*2+1],0fh
    pop gs
    popad
    popf
    iret
```

进入自己新设置的九号中断后，先调用旧的九号中断，以确保键盘可以正常使用，由于 16 号中断调用 9 号中断，而且不知道旧的九号中断实现了什么功能，所以如果不调用他，会导致 16 号中断失效。

3. 34 号中断响应程序模块：

```
int_34:
    pushf
    pushad
    cli
    xor eax,eax
    mov ax,_drawLfUp
    call eax
    popad
    popf
    iret
```

在左上角区域显示个人信息：

```
void drawLfUp()
{
    // 输出信息
    int i, j;
    for(i = 0; i < 9; i++)
    {
        char cr = my_inf[0][i];
        if(cr == ' ') continue;
        j = 0;
        Setchar(0,14+i,cr);
        while(j++ < 6)
        {
            timedelay(5);
            Setchar(j-1,14+i,' ');
            Setchar(j, 14+i, cr);
        }
    }
}
```

Setchar 是一个指定位置输出的函数，封装了汇编的接口，timedelay 则是一个时延函数，用于控制时间。

4. 系统调用响应程序模块：

分支控制

```
int_21h:
    pushf
    pushad
    xor ecx,ecx
    cmp ah,0
    jz int_21h_0
    cmp ah,1
    jz int_21h_1
    cmp ah,2
    jz int_21h_2
int_21h_end:
    popad
    popf
    iret
int_21h_0:
    mov ecx,_Disptime
    call ecx
    jmp int_21h_end
```

显示当前时间：

```
_Getminute:
    ;发送需要的时间参数到指定端口70h，从71h获得其BCD码
    push bx
    push cx
    mov al,2
    out 70h,al
    in al,71h
    mov ah,al
    mov cl,4
    shr al,cl
    and ah,00001111b
    ;ah 代表 个位数，al 代表 十位数
```

分钟的单元为 2，发送 2 到 70h 端口，即可从 71h 端口获得当前分钟。

```
void Disptime()
{
    times[0] = '2';
    times[1] = '0';
    times[2] = 0;
    //分秒要补零，
    int year, month, day, hour, minute, second, i;
    year = Getyear();
    Str(year, temp);
    strcat(times, temp);
    temp[0] = '/';
    temp[1] = 0;
    strcat(times, temp);
    month = Getmonth();
```

Str 是类型转换函数，用于将 int 变量 year 转换为字符串存于 temp 变量。Strcat 是自己写的字符串拼接函数，用于将后面的字符串拼接到前面的字符串。

切换背景色或前景色：

```
void changebcolor()
{
    int i, j, color;
    for(i = 0; i < 25; i++)
        for(j = 0; j < 80; j++)
        {
            color = Readcolor(i, j);
            color += 0x10;
            if(color > 0xff) color -= 0x100;
            int fcolor = color / 16;
            int bcolor = color % 16;
            if(bcolor == fcolor || fcolor == (bcolor+0x08) || bcolor == (fcolor+0x08))
            {
                color += 0x10;
                if(color > 0xff) color -= 0x100;
            }

            Setcolor(i, j, color);
        }
}
```

bcolor 是背景颜色，fcolor 是前景色，Readcolor 和 Setcolor 函数是汇编实现的接口，用于读取和设置固定位置的颜色

关机和重启：

这两个中断的实现相对简单，只要写特定数据进对应端口即可。

写端口的汇编接口：

```
;说明:      void Port_out_16(int port, int value)
;注意:      只由C调用，返回32位地址
_Port_out_16:
    push bp
    mov bp,sp
    push dx
    push ax
        mov dx, word[bp + 6]        ; port
        mov ax, word[bp + 10]
        out dx, ax
        nop                         ; 一点延迟
        nop
    pop ax
    pop dx
    pop bp
    retf
```

关机与重启函数

```
void Shutdown()
{
    Port_out_16(0x1004, 0x2001);
}
void Restart()
{
    Port_out_16(0x64,0xfe);
}
```

将 0x2001 写入端口号 0x1004 即是关机；将 0xfe 写入端口号 0x64 即是重启。这个的难点来自于资料比较少，在谷歌上才搜到了。

四.实验过程与结果

1.实验过程：

【编译链接的命令】

使用的命令基本和实验三一样，唯一不同的是我使用了 dd 命令来将.bin 文件写入映像文件中，指定大小的块拷贝一个文件，并在拷贝的同时进行指定的转换。命令如下：

dd if=kernel.bin of=pro4.img bs=512 seek=1 count=16

if 表示输入文件，of 表示输出文件，bs 表示复制的块大小，以字节为单位，seek 表示从开头跳过的块数，count 表示复制的块数。

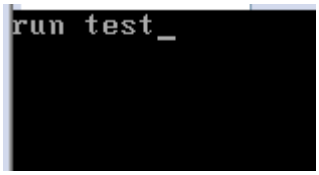
【磁盘和内存安排方案】

与上次实验一致。

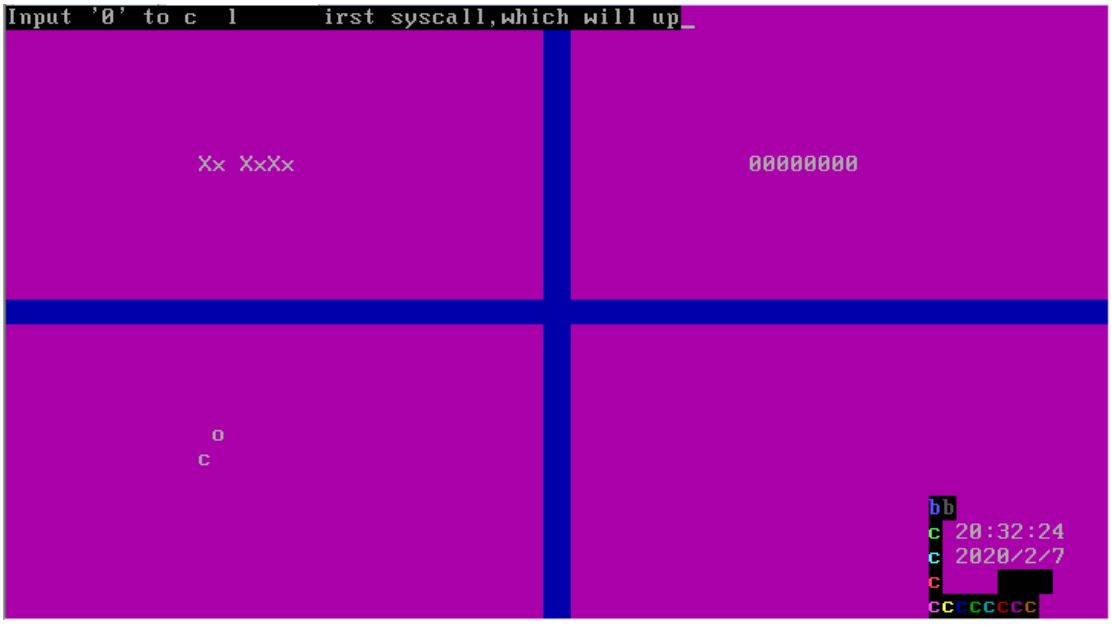
2.运行结果：

时钟中断与键盘中断响应结果：

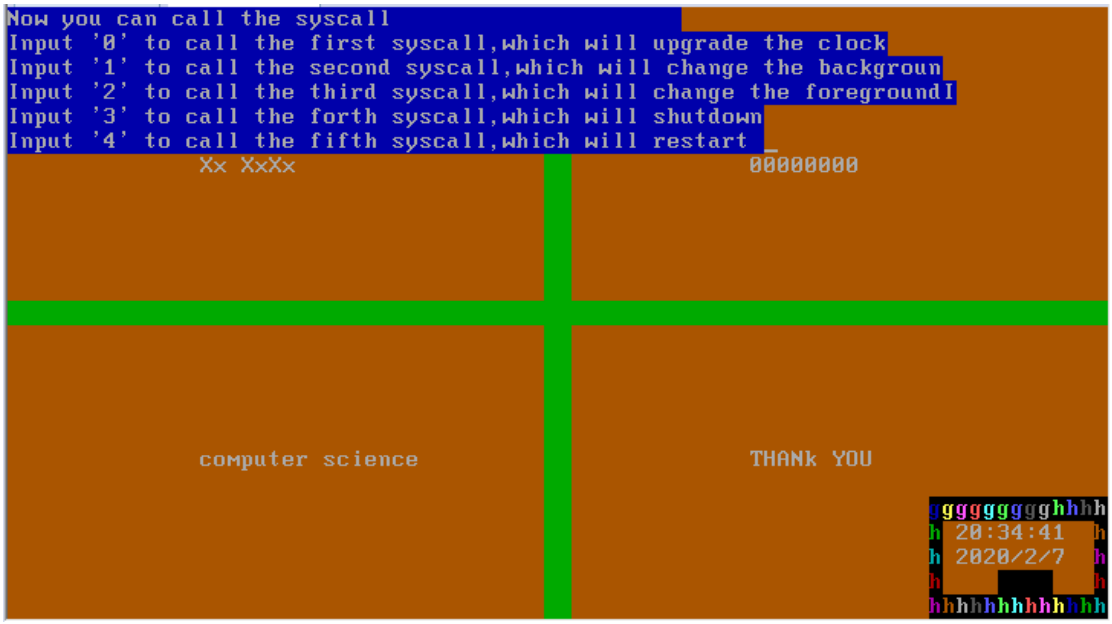
测试程序命令：

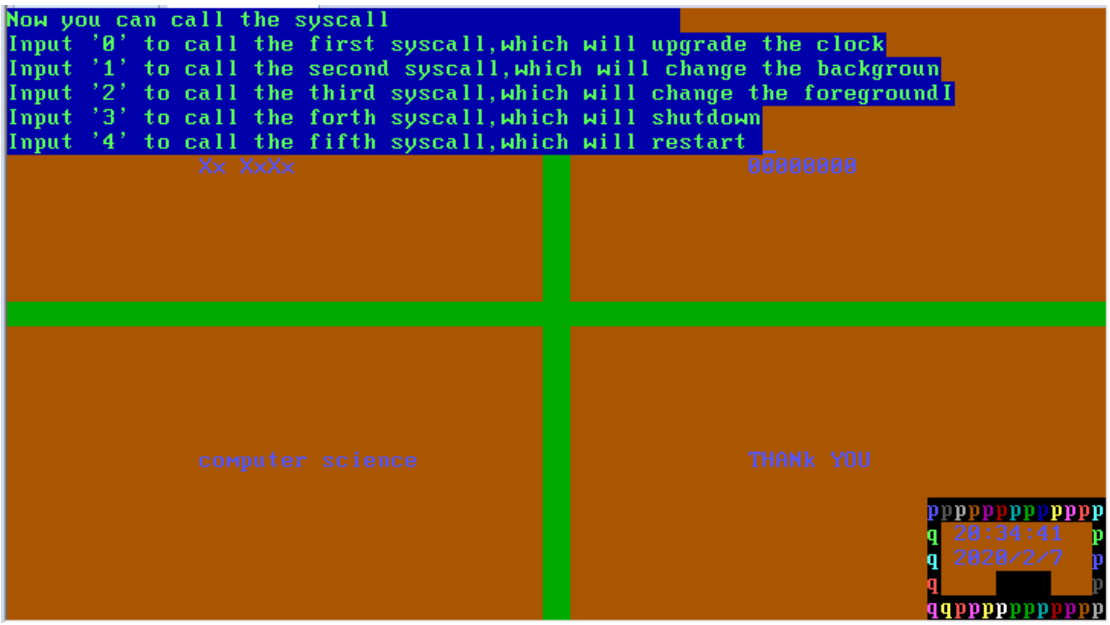


中断动画播放(34~37号中断):



切换背景颜色或前景颜色(系统调用):





更新时间：

3.遇到的问题及解决情况：

① 之前我的汇编函数的返回使用了ret。而其实这是有隐患的。如果是只由C调用的汇编函数应当用retf返回。

ret指令返回ip，而retf返回ip，cs。而C调用函数时，会把cs和ip都压栈，因此如果一个汇编函数时专门写来给C调用的，应该用retf返回。如果C调用该函数的次数少问题不大，但是如果调用的次数多了，就会爆栈，导致程序崩溃。如果有某些调用次数不多又需要同时给C和汇编使用的函数，则使用ret问题也不大。

建议把由C调用的函数与汇编调用的函数区分开。

②时钟中断中没有使用pushad,导致光标位置输出了一些在其他位置的字符串。

在时钟中断中，我只把自己使用到的寄存器保存到栈里面去。而在子程序调用中我调用了16号中断输出字符串。结果光标位置出现了该字符串，而这个字符串在光标处是没有输出的，（虽然他在自己应该输出的地方有输出），运行结果如图：

右下角处是正确的输出，左上角是出错的地方。

而当我改成pushad将所有的寄存器都保存起来后，这个问题解决了，我猜测，某些寄存器的值被改变了（虽然我在时钟中断程序中没有使用到）。

因此，不管在中断响应程序中有没有使用到某个寄存器，你都应该将其保存下来。这是比较稳妥的

③在新的 9 号中断程序中调用旧的 9 号中断程序出错

首先是没有加 pushf 就 call 了旧的 9 号中断，这个会出错，因为中断的返回 iret 会 pop ip，cs，flag 寄存器。

在看过王爽的《汇编语言（第三版）》以后，我使用了这个指令：　call dword [pre_int9_addr]　;pre_int9_addr 存放旧 9 号中断的地址。这个指令是有问题的。如图：

```
int_9:
    pushf
    call dword[cs:pre_int9_addr]
    iret
```


这个指令会使得程序跳到一个不知名的地方。解决方法是把 dword 改为 far。这是 nasm 的 call 指令语法的问题，段间转移使用的指令有两种形式。下图是在 nasm 中文手册中找

到的。这样便可以在新的 9 号中断中调用旧的 9 号中断，实现扩展 9 号中断的功能的目的。

'segment'和'offset'都以立即数的形式出现。所以要调用一个远过程，可以如下编写代码：

```
call    (seg procedure):procedure
call    weird_seg:(procedure wrt weird_seg)
```

（上面的圆括号只是为了说明方便，实际使用中并不需要）

NASM 支持形如'call far procedure'的语法，跟上面第一句是等价的。

五.实验创新点

1.对不同类型的 C 函数按类型分文件存放，并将汇编函数做成接口。

这次实验中我将函数分成了四个函数库，io.h(定义输入输出函数),time.h(定义时间函数),stdlib.h(定义杂项函数)和 string.h(字符串处理函数)，这样可以减少数据冗余，并且方便管理。将必须汇编实现的功能都做成接口，如获取系统时间，输入，输出等，再用 C 函数来包装这些底层接口。将不同类型功能的 C 函数放不同的文件中，模拟 C 标准库的形式建立一个可以由 C 实现所有功能的函数库，让日后的编写程序可以和汇编分开。即使需要使用汇编也是调用 C 的函数完成主体部分。

2.实现了个人信息的动画播放

3.实现了关机与重启的中断功能

通过写端口实现关机和重启。

4.实现了系统时间的获取，直接与硬件接触。

访问 CMOS RAM。使用 in，out 指令从特定数据端口获取当前的时间。

5.切换背景前景颜色

六.实验总结：

本次实验主要内容在于中断响应程序，具体就是保存恢复断点环境以及修改中断向量表，这在原理和实现部分都讲得差不多了。本次实验有时钟中断，键盘中断，34~37 号中断，还有系统调用；其框架基本是一样的。另外就是自己实现了不少 C 函数，建立了四个 C 函数库，主要的是输入输出方面的，也算是加深了对 C 的认识。因此这次的实验相比上次没那么难，环境搭好了，打代码就容易多了。虽然遇到的难题不大，但是小坑还是很多的。

在堆栈的保护方面有一个比较深的体会。这次实验在堆栈方面遭遇了不少的困难。首先没理解好 ret，iret，retf 的意义就造成了一些错误，导致我的程序跳转不到准确的位置并且篡改了标志寄存器的值。另外就是进入中断后没有使用 pushad 将所有寄存器保存下来，特别是时钟中断，或者说硬件中断，硬件中断的响应是未知的，所以我们不知道他会在哪个地方响应，所以它随时有可能篡改寄存器的值。另外，如果中断要从 eax 返回值，则要小心 ax 有没有放进栈中。堆栈实在是个很值得小心的东西。其实问题的关键是数据的维护，现在单进程数据的维护已经这么麻烦，我实在担心多进程时候的数据维护的难度。

这次实验的另外的小问题就是内存的安排不够合理，以及函数库的建立不够合理。这属于个人的编程习惯不够良好，导致了不必要的 bug，浪费了时间。我上个实验是把内核加载到 08100h，子程序加载到 0A100h，而这次实验内核大了，超过了 2000h，就使得加载子程序时数据覆盖了内核程序，应该是需要给内核预留足够多的空间的。

另外，之前我把所有的函数都扔到了一个文件中，每次链接的时候都要链接上文件中所有的函数，就加大了子程序的大小，有数据的冗余问题。管理起来也极为不方便。这次我就仿照 C 的函数库的形式，将时间类型函数放 time 文件，输入输出函数放 io 文件，字符串处理函数放 string 文件，杂项函数放 stdlib 文件。并且我将汇编完成的任务都做成接口，这样就可以减少编程时汇编部分出问题的可能，找起 bug 来就方便多了。这样使用起来就舒服的多。

参考文献：

1．《80x86 汇编语言程序设计教程》 （杨季文）
2．《nasm 中文手册》 （csdn）
3．《一个操作系统的实现》 于渊
4．《汇编语言（第三版）》 王爽