实验五：实现系统调用

学院名称：	数据科学与计算机学院
专业（班级）：	18计科教学3班
学生姓名:	夏一溥
学号:	18340178
时间:	2020 年 6 月 19 日
实验四：	实现系统调用

实验内容

- [x] 编写 `SAVE()` 和 `RESTART()` 汇编过程用于中断处理的现场保护和现场恢复，处理程序的开头都调用save()保存中断现场，处理完后都用restart()恢复中断现场。
- [x] 内核增加 `int 20h`、`int 21h` 和 `int 22h` 软中断的处理程序,分别实现 `返回` `自定调用功能` `显示INT22H` 三个处理程序。
- [x] 进行C语言的库设计，完成 `_getchar()` 、`_printf()`、 `puts()` 等输入输出库过程。

实验过程

实验环境:

- 操作系统： Ubuntu 18.04.4 LTS
- 汇编编译器： NASM
- 调试工具： Bochs
- c语言编译器： GCC

设计思路:

Part 1： 实现 SAVE() 和 RESTART() 汇编过程

参考 Minix 的实现过程， SAVE() 和 RESTART() 其本质就是，在将进入中断调用时将各寄存器的值预先存放在内存的某一块数据结构中，并当中断完成时复原这些寄存器的值。

实现方法是通过栈来实现：

```
%macro SAVE 1
    cli
    ;Stack: */fg/cs/ip
    push ds
    ;Stack: */fg/cs/ip/ds
    push cs
    ;Stack: */fg/cs/ip/ds/cs
    ; to get cs
    pop ds
    ;Stack: */fg/cs/ip/ds
    pop word[ds:_DS]
```

```
    ;Stack: */fg/cs/ip/
    mov [ds:_AX],ax
    mov [ds:_BX],bx
    mov [ds:_CX],cx
    mov [ds:_DX],dx

    mov ax,es
    mov [ds:_ES],ax

    mov ax,di
    mov [ds:_DI],ax
    mov ax,si
    mov [ds:_SI],ax
    mov ax,bp
    mov [ds:_BP],ax
    mov ax,sp
    mov [ds:_SP],ax
    mov ax,ss
    mov [ds:_SS],ax
    ;Stack: */fg/cs/ip
    pop word[ds:_IP]
    pop word[ds:_CS]
    pop word[ds:_FG]
    ;Stack: */
    ;go call int

    mov ax,[ds:_AX]
    call %1

    RESTART%1:

    mov sp,[ds:_SP]
    pop ax
    pop ax
    pop ax
    ;stand pos /*/*/*/

    mov ss,[ds:_SS]
    mov ax,[ds:_FG]
    push ax
    ;Stack: */fg/
    mov ax,[ds:_CS]
    push ax
    ;Stack: */fg/cs/
    mov ax,[ds:_IP]
    push ax
    ;Stack: */fg/cs/ip/
    mov es,[ds:_ES]
    mov di,[ds:_DI]
    mov si,[ds:_SI]
    mov bp,[ds:_BP]
    mov dx,[ds:_DX]
    mov cx,[ds:_CX]
    mov bx,[ds:_BX]
    mov ax,[ds:_DS]
    push ax
    ;Stack: */fg/cs/ip/ds/
    mov ax,[ds:_AX]
```

```
    pop ds
    ;Stack: */fg/cs/ip/
    push ax
    mov al,20h                  ; AL = EOI
    out 20h,al                  ; 发送EOI到主8529A
    out 0A0h,al                 ; 发送EOI到从8529A
    pop ax

    sti
    iret
%endmacro
```

同时因为本次实验需要保护实现的中断过程相对很多，所以我使用了宏来实现这一相同的保护-恢复过程：

```
SAVE()
call prog
RESTART()
```

同时，值得注意的是，在 c语言程序中调用汇编程序 与 汇编程序中调用c语言程序 其入栈情况有所不同，这是我在 bochs 上捕获的内容：

```
| STACK 0xff7a [0xff9a]
| STACK 0xff7c [0x0000]
| STACK 0xff7e [0x881b] <-- c调用汇编
| STACK 0xff80 [0x8878]
| STACK 0xff82 [0x80fe] <-- 汇编call
| STACK 0xff84 [0x0000]

```

所以如果要通过汇编返回c程序代码，`ret`需要变更为`o32 ret`，相应调用需要使用`call dword [addr]`但栈维护过程其实并未有改变，但需要在汇编中增加相应的pop语句，这一点体现在`lib_asm`中的函数中。

Part 2：内核增加`int 20h`、`int 21h`和`int 22h`软中断的处理程序

中断处理程序的写入与实验四没有区别，通过`writeIVT`并在监控程序中写入中断向量表：

```
WriteIVT 08h, INT08H_START
WriteIVT 20h, INT20H_START
WriteIVT 21h, INT21H_START
WriteIVT 22h, INT22H_START
```

其中，`int 08h`处理增加 save() 与 restart() 过程并未做其他处理；

`int 20h`为返回：

```
INT20H_START:
    pop word[_tmp]
    pop word[_tmp]
    pop word[_tmp]
    retf
```

因为原调用前入栈内容为

```
*/flags/cs/ip
```

故在返回前保证栈内容相同，当然其实也可以通过 `add sp,6` 实现。

重点内容是 `int 21H` 的实现：

21号中断需要根据 `ah` 的值来选择进行其中断处理程序，所以要仿照监控程序进行一个跳转的程序段，我使用以 `si` 作为offset来选择将要跳转的地址：

```
INT21H: ;8838
    mov si, cs
    mov ds, si                  ; ds = cs
    mov si, ax
    shr si, 8                   ; si = 功能号
    add si, si                  ; si = 2 * 功能号
    call [SYSINT+si]        ; 系统调用函数
    ret
    SYSINT:
        dw SYS_ShowTime
        dw SYS_reBoot
        dw SYS_PowerOff
INT21H_START:
    SAVE INT21H
```

而我选编写的中断处理程序一共有3个，分别为 `显示系统时间`、 `重启`、 `关机` 。

显示系统时间的实现值得一提，其实当前时间被储存在 CMOS RAM 中， 可以通过接口 `70H` 来访问，以获取 `年` 为例：

```
_getYear:
    mov al, 9
    out 70h, al
    in al, 71h
    mov ah, 0
    retf
```

Part 3： C语言的库设计

这一部分最难解决的问题是变参函数设计的问题，通过查阅资料，得知变参函数的实现依靠这一头文件：

```
//#include <stdarg.h>
#ifndef _STDARG_H
#define _STDARG_H

typedef char *va_list;

/* Amount of space required in an argument list for an arg of type TYPE.
   TYPE may alternatively be an expression whose type is used.  */

#define __va_rounded_size(TYPE)  \
  (((sizeof (TYPE) + sizeof (int) - 1) / sizeof (int)) * sizeof (int))

#ifndef __sparc__
#define va_start(AP, LASTARG)                       \
```

```
 (AP = ((char *) &(LASTARG) + __va_rounded_size (LASTARG)))
#else
#define va_start(AP, LASTARG)                       \
 (__builtin_saveregs (),                        \
  AP = ((char *) &(LASTARG) + __va_rounded_size (LASTARG)))
#endif

void va_end (va_list);      /* Defined in gnulib */
#define va_end(AP)

#define va_arg(AP, TYPE)                        \
 (AP += __va_rounded_size (TYPE),                   \
  *((TYPE *) (AP - __va_rounded_size (TYPE))))

#endif /* _STDARG_H */

```

其中 `va_list` 是一个指向字符串的指针 ,我们可以通过这一功能来访问变参函数中的各参数,

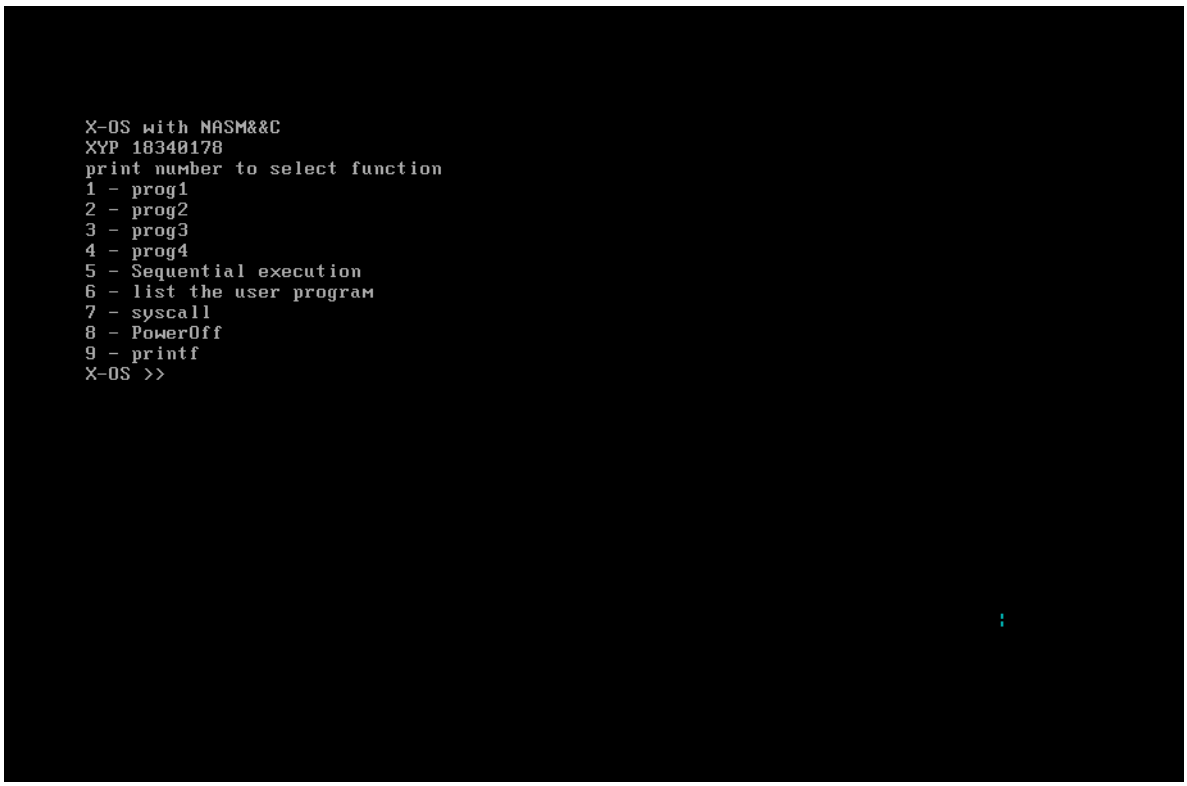
以`printf`为例:

```c
void Printf(char *s, ...)
{
    int i = 0;
    /* 可变参第一步 */
    va_list va_ptr;

    /* 可变参第二部 */
    va_start(va_ptr, s);

    /* 循环打印所有格式字符串 */
    while (s[i] != '\0')
    {
        /* 普通字符正常打印 */
        if (s[i] != '%')
        {
            _putchar(s[i++]);
            continue;
        }

        /* 格式字符特殊处理 */
        switch (s[++i])   // i先++是为了取'%'后面的格式字符
        {
            case 'd': printDeci(va_arg(va_ptr,int));
                      break;
            case 'c': _putchar(va_arg(va_ptr,int));
                      break;
            case 's': print(va_arg(va_ptr,char *));
                      break;
            default : break;
        }

        i++; // 下一个字符
    }

    /* 可变参最后一步 */
    va_end(va_ptr);
```

```
37 }
```

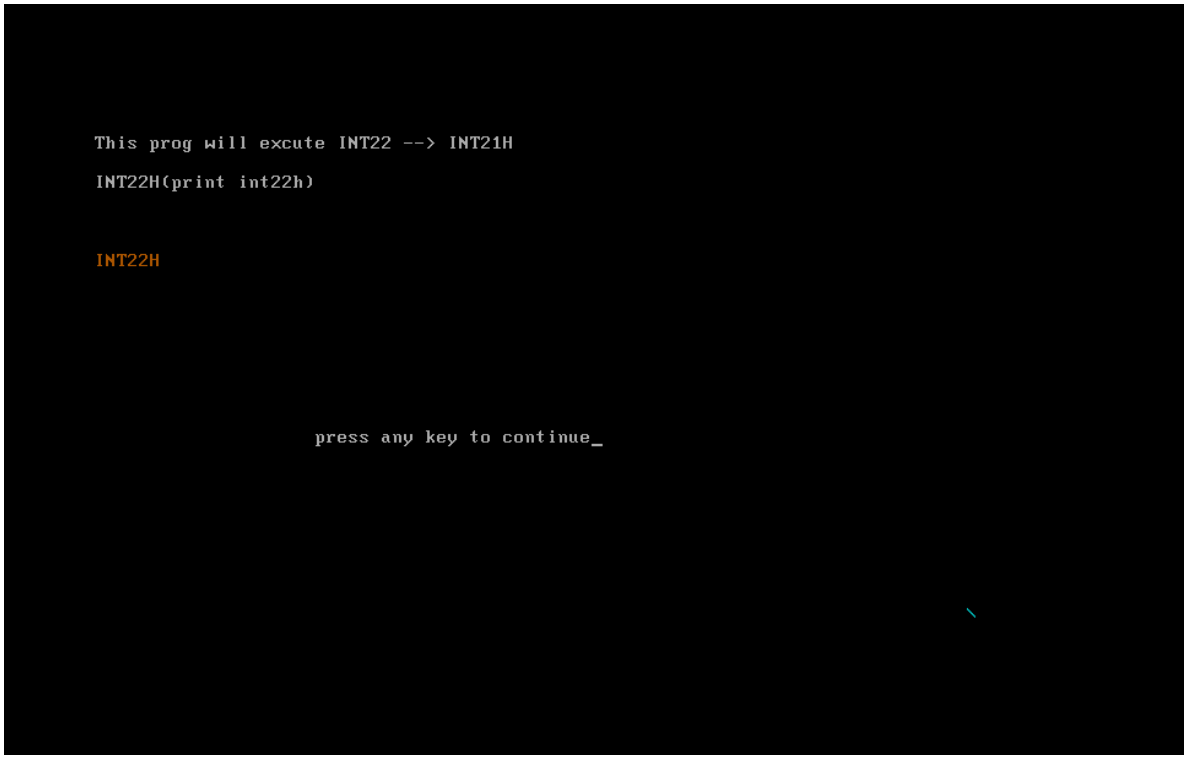
实验结果：

进入监控程序：

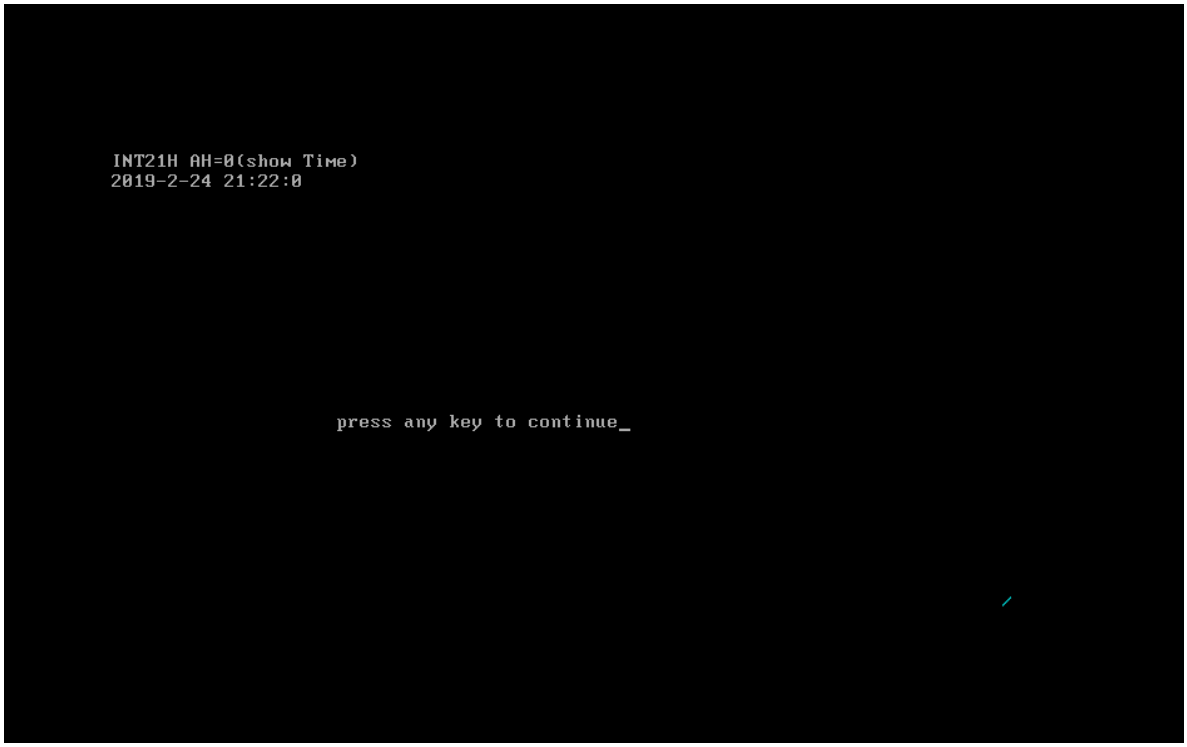


进入系统调用的展示函数：

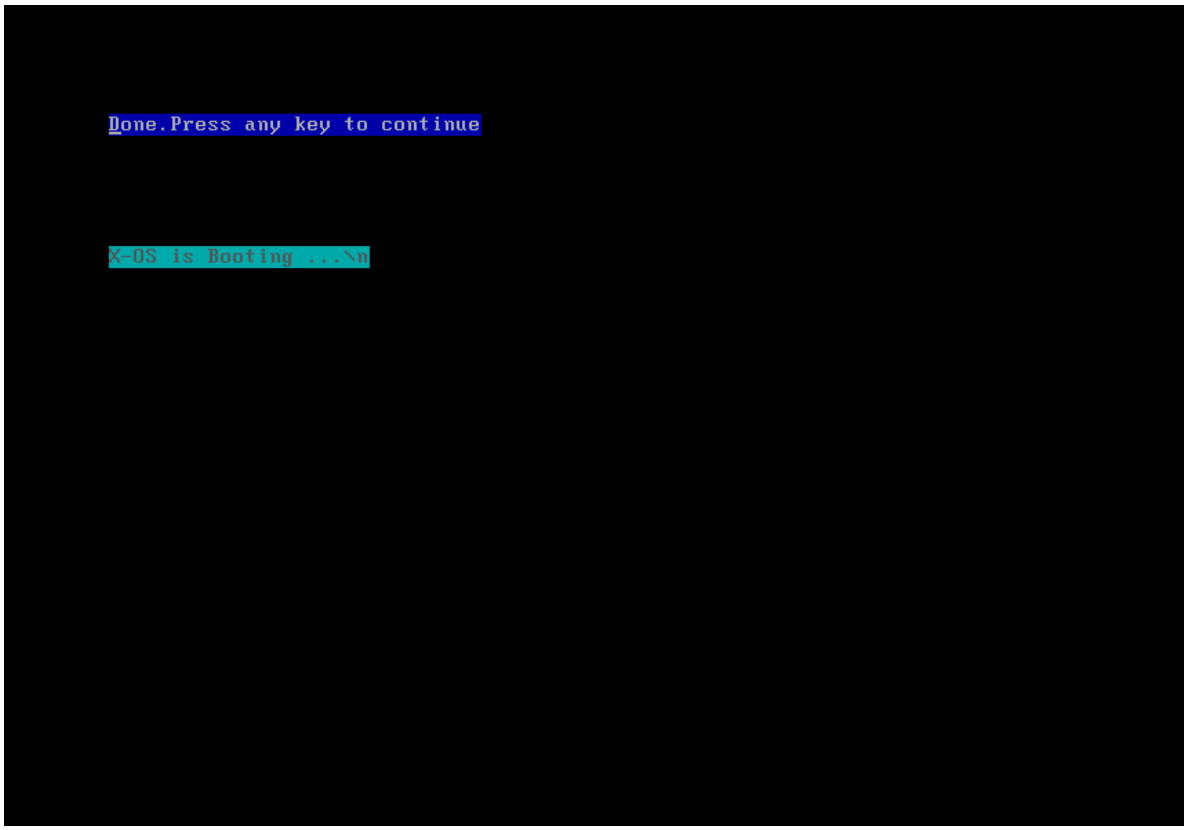
int 22H (打印 INT22H)：



int 21H AH=0 （显示时间）：

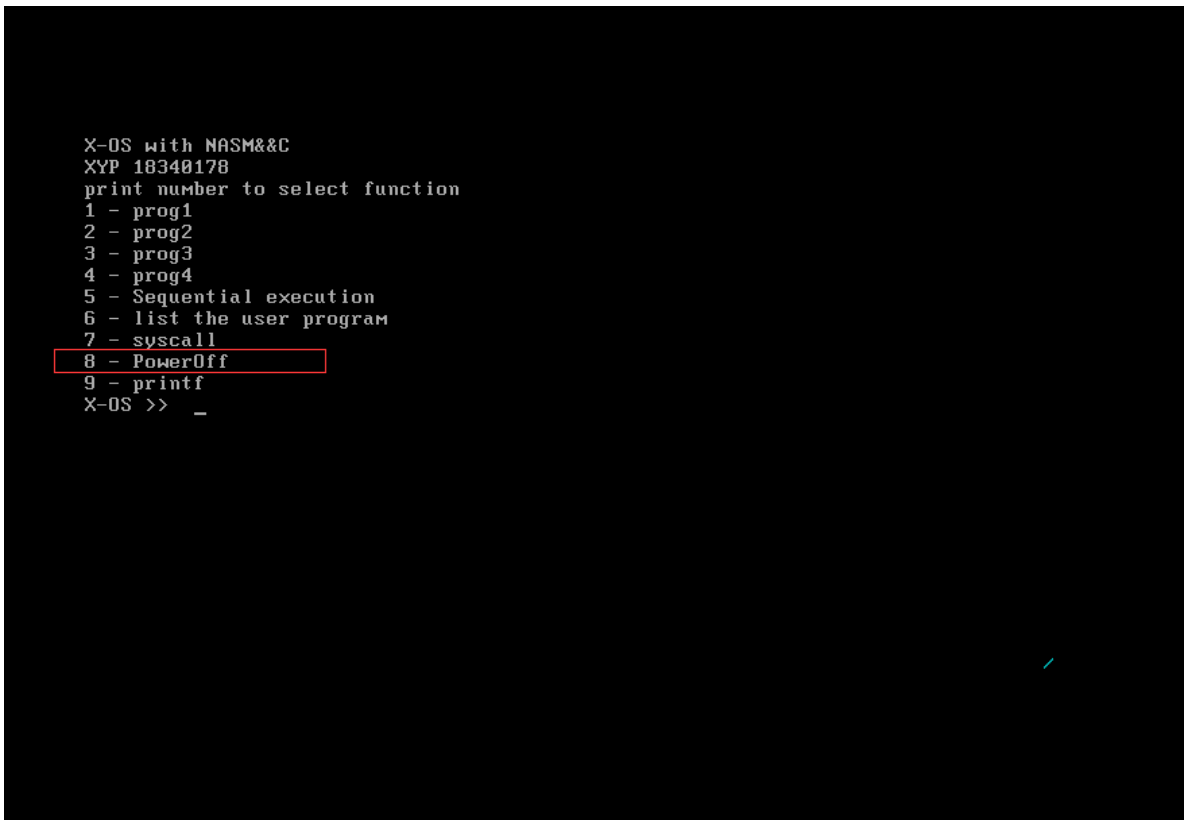


int 21H AH=1 （reBOOT）：



int 21H AH=2 （关机）：

在内核选择中实现：



纠错过程：

1. 在刚完成 `SAVE()` `RESTART()` 时，会出现进入时钟调用不能出来的情况

其实就是栈没维护好，我最开始在 `RESTART()` 过程中犯了一个致命的错误：

修改前代码：

```
    RESTART%1:

    mov sp,[ds:_SP]

    ;stand pos /*/*/*/
```

正确代码：

```
    RESTART%1:

    mov sp,[ds:_SP]
    pop ax
    pop ax
    pop ax
    ;stand pos /*/*/*/
```

其实就是在后面进行的出栈入栈操作中让现在过早恢复sp导致栈的指针错，所以需要预先弹出多出来的 `*/fg/cs/ip/` 共6字节。

2. 汇编调用C函数出现栈错误：

在 `Part1` 中已经做了说明，其实就是call C `ip` 会入栈2字节而不是1字节。

实验总结

这次实验解决了我上个实验中遗留下来的问题，上个实验中我认为中断位过分紧张不能实现多个程序共用一个中断，在和同学交流后学会了原来可以通过 参数不同，比如 `ah` 的值来控制某中断跳入不同的位置。同时我对系统调用的理解也加深了很多，特别是通过 `INT 20H` ,软中断的实现可以让程序员在用户程序执行中调用某一特定功能的中断处理函数实现其想要达成的目的。

还有就是变参函数的编写，这种区别与普通函数的函数给我编程带来了更大的便利性。

Reference

1. 自己动手写printf https://blog.csdn.net/cinmyheart/article/details/24582895
2. CMOS RAM中存储的时间信息 https://www.cnblogs.com/qintangtao/archive/2013/01/19/2867846.html
3. 《从实模式到保护模式》