《操作系统原理实验》

实验报告

（实验三）

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：	朱志儒
学 号：	16337341
时 间：	2018 年 4 月 12 日

实 验 三 ： 开发独立内核的操作系统

一.实验目的

1、把原来在引导扇区中实现的监控程序(内核)分离成一个独立的执行体，存放在其它扇区中，为“后来”扩展内核提供发展空间。

2、学习汇编与c混合编程技术，改写实验二的监控程序，扩展其命令处理能力，增加实现实验要求中的部分或全部功能。

二.实验要求

1、 将实验二的原型操作系统分离为引导程序和MYOS内核，由引导程序加载内核，用C和汇编实现操作系统内核。

2、 扩展内核汇编代码，增加一些有用的输入输出函数，供C模块中调用。

3、 提供用户程序返回内核的一种解决方案。

4、 在内核的C模块中实现增加批处理能力:

（1）在磁盘上建立一个表，记录用户程序的存储安排。

（2）可以在控制台命令查到用户程序的信息，如程序名、字节数、在磁盘映像文件中的位置等。

（3）设计一种命令，命令中可加载多个用户程序，依次执行，并能在控制台发出命令。

(4)在引导系统前,将一组命令存放在磁盘映像中,系统可以解释执行。

5、监控程序以独立的可执行程序实现,并由引导程序加载进内存适当位置,内核获得控制权后开始显示必要的操作提示信息。

三. 实验方案

1、虚拟机配置

使用Vmware Workstation配置虚拟机，虚拟机的配置：核心数为1的处理器、4MB的内存、10MB的磁盘、1.44MB的软盘。

2、软件工具与作用

Notepad++：编写程序时使用的编辑器；

16位编辑器WinHex：可以以16进制的方式打开并编辑任意文件；

TAMS汇编工具：可以将汇编代码编译成对应的二进制代码；

NAMS汇编工具：可以将汇编代码编译成对应的二进制代码；

TCC编译器：可以将c代码编译成对应的二进制代码；

TLINK链接器：将多个.obj文件链接成.com文件

WinImage：可以创建虚拟软盘。

3、 基础原理

（1）引导程序引导操作系统内核的原因是实际上的操作系统功能多，程序规模大，执行代码不能直接放在一个引导扇区内容。所以我们可以设计一个引导操作系统，通过计算机硬件加载操作系统并执行，让操作系统接管硬件系统，这样就摆脱了一个扇区的限制。

（2）C与汇编的交叉调用的主要原因是C与汇编的交叉调用是现有操作系统的开发方法。而操作系统要用到汇编语言的原因是可以设置自身运行模式和环境，通过设置硬件寄存器，设置I/O端口实现I/O操作。除此之外，还可以通过汇编语言初始化中断向量表和实现中断处理。而操作系统要用C语言的原因是便于构造复杂的数据结构和相关数据结构的管理，实现复杂的功能或

算法。

（3）TCC与TASM环境使用

a. 环境：TCC编译器、TASM汇编器、TLINK链接器

b. 使用TCC编译命令： tcc -mt -c －o cfile.obj cfile.c >ccmsg.txt

c. TASM汇编命令： tasm afile.asm afile.obj > amsg.txt

d. 链接命令： tlink /3 /t cfile.obj afile.obj , showstr.com,,

4、方案思想

（1）编写一个名为lording.asm的引导程序，将这个程序放在引导扇区，用于加载操作系统并将控制权移交给操作系统。

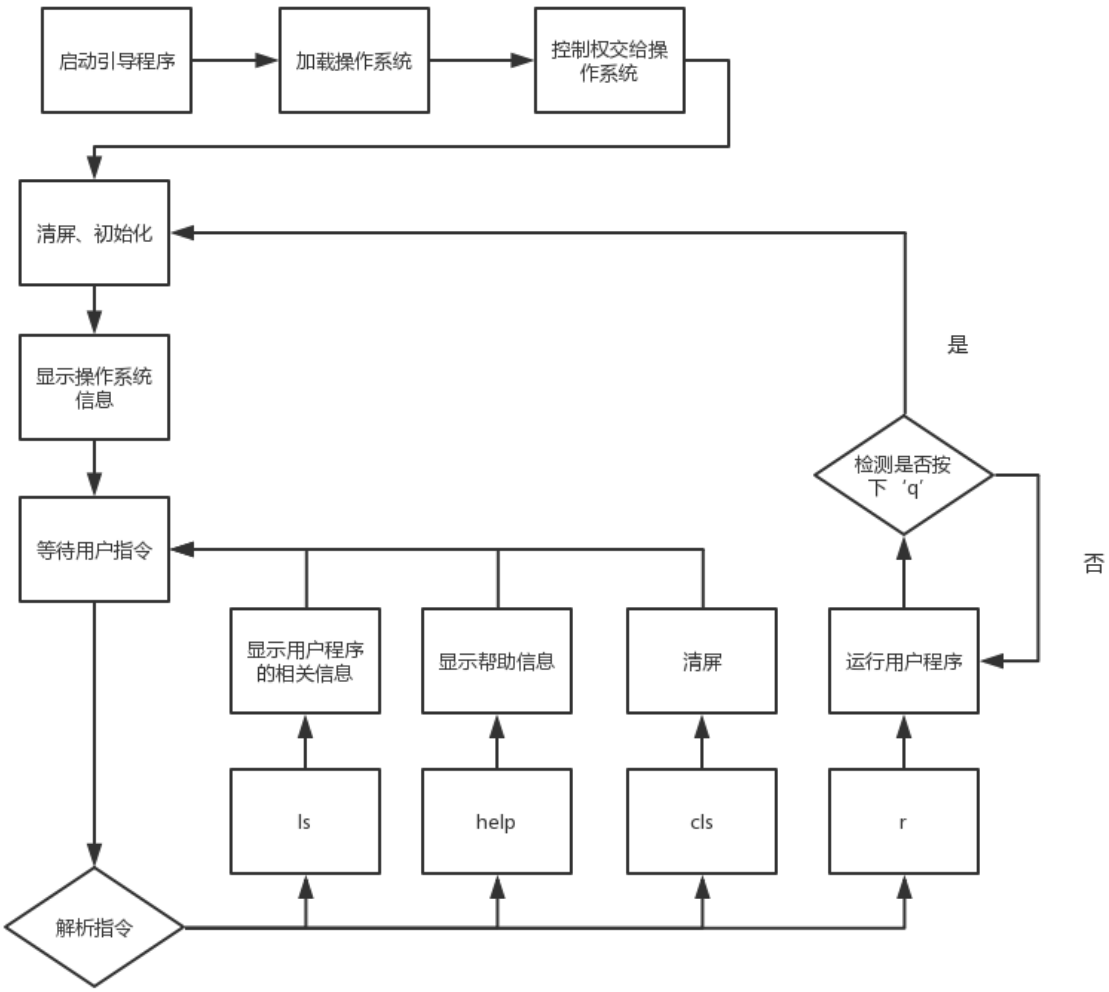
（2）编写一个名为kliba.asm的汇编代码，在这段代码中实现了清屏、加载并运行用户程序、显示一个字符、读取一个字符输入这四个基本的底层功能。

（3）编写一个名为kernal.c的c代码，导入了kliba.asm中的四个基本功能函数。在这些基本功能的基础上，拓展了一些新功能，如显示一段字符串、读取一行输入、比较两个字符串、计算一段字符串长度、获取一段字符串的子字符串等。并且还实现了初始化shell界面、列出用户程序清单、显示帮助文档、加载并运行用户程序这四个重要功能。当然，在kernal.c中还包含操作系统内核的主程序cmain，这个程序将识别用户的shell指令，然后执行相应的操作。

（4）编写一个名为MyOS.asm的汇编代码，在这段代码中，导入kernal.c中的全局变量和主函数，导入kliba.asm中的汇编代码，设置相关段寄存器后，跳转至kernal.c中的cmain程序。

（5）修改实验二的用户程序，使其能够在操作系统下运行。在用户程序中调用20h中断，响应用户的按键输入，按‘q’返回操作系统。

5、程序流程



6、算法和数据结构

算法：

（1）在kernal.c文件中，print(char *str)函数调用kliba.asm中的printChar(chars)显示字符串。str指向字符串的首地址，*str将首个字符传入printChar显示，str++将str指向下个字符，以此类推，当str指向字符串的末尾时程序停止，这样逐个显示字符以达到显示整个字符串的目地。

（2）在kernal.c文件中，getline(char *ptr, int length)函数调用kliba.asm中的getChar()读取输入字符串。其中ptr指向输入存入的字符串首地址，length指读取输入的最大长度。在getline函数中，初始化一个名为count的变量用于计数，当length为0时，不读取输入直接返回；当length不为0时，读取一个输入字符，判断该字符是否为回车键，若是回车键，则换行和回车并返回。反之，则显示该字符并将该字符存入ptr中，count加1，若count等于length，则在字符串末尾加上休止符‘\0’，再换行和回车并返回。若不相等，则读取一个输入字符。以此类推，将输入字符串存入ptr字符串中。

（3）在kernal.c文件中，strcmp(char *str1, char *str2)函数比较两个字符串是否相等。首先，比较str1和str2所指的字符是否相等，若不相等，则比较两个字符的字典序，若*str1<*str2，则返回-1，反之，返回1；若相等，则str1和str2分别指向下个字符。依此比较，直指出现休止符，然后返回*str1和*str2的相减值。

（4）在kernal.c文件中，strlen(char *str)函数计算字符串的长度。首先，初始化一个名为i的变量用于计数，然后判断str所指的字符是否为休止符，若不为休止符，则str指向下个字符，i加1。反之，返回i。

（5）在kernal.c文件中，substr(char *src, char *sstr, int pos, int len)函数返回src的一个子字符串。首先，初始化i为pos，即子字符串在src中的起始地址，然后将src中的字符复制到sstr中，当sstr的长度等于len时，程序结束并返回。

数据结构：

字符串，即由零个或多个字符组成的有限序列。

7、程序关键模块

lording.asm文件中的引导程序：

用于加载操作系统并将控制权移交给操作系统。当然，这个程序还执行一个非常重要的指令——载入中断向量 20h，在用户程序中调用 20h 中断即可返回操作系统内核。

代码如下：

```
;写入中断向量表
%macro write_inerrupt_vector 2
    pusha
    mov ax, 0h
    mov es, ax
    mov ax, %1
    mov bx, 4
    mul bx
    mov bp, ax
    mov ax, %2
    mov word[es:bp], ax
    add bp, 2
    mov ax, cs
    mov word[es:bp], ax
    popa
%endmacro

;定义 20h 中断向量
write_inerrupt_vector 20h, myinterrupt20h

myinterrupt20h:
    pusha
    print_message message1, 24, 24, 3
    mov ah, 01h
    int 16h
    jz no_input          ;没有按键，则跳转至 no_input
    mov ah, 00h
    int 16h
    cmp al, 'q'
    jne no_input         ;若没按 q，跳转至 no_input
    jmp 800h:100h
no_input:
```

```
        popa
        iret
    ;加载操作系统
    Lording_OS:
        mov ax,cs                   ;段地址 ； 存放数据的内存基地址
        mov es,ax                   ;设置段地址（不能直接 mov es,段地址）
        mov bx, offsetofos          ;偏移地址; 存放数据的内存偏移地址
        mov ah,2                    ； 功能号
        mov al,3                    ;扇区数
        mov dl,0                    ;驱动器号 ； 软盘为 0，硬盘和 U 盘为 80H
        mov dh,0                    ;磁头号 ； 起始编号为 0
        mov ch,0                    ;柱面号 ； 起始编号为 0
        mov cl,2                    ;起始扇区号 ； 起始编号为 1
        int 13H                     ;调用读磁盘 BIOS 的 13h 功能
        ;内核程序已加载到指定内存区域中

    ;将控制权转交给操作系统
    jump_to_kernel:
        jmp 800h:100h
```

kliba.asm文件中的run()程序，加载并运行用户程序：

```
public _run
_run proc
    mov ax,cs
    mov es,ax                   ;设置段地址，存放数据的内存基地址
    mov bx,0B100h               ; ES:BX=读入数据到内存中的存储地址
    mov ah,2                    ； 功能号
    mov al,1                    ； 要读入的扇区数 1
    mov dl,0                    ； 软盘驱动器号
    mov dh,0                    ； 磁头号
    mov ch,0                    ； 柱面号
    mov cl,byte ptr[_pro]       ； 起始扇区号（编号从1开始）
    int 13H                     ； 调用13H号中断

    ； 跳转到该内存地址
    mov bx, 0B100h
    jmp bx
_run endp
```

Kliba.asm文件中的cls()程序，清屏操作：

```
    public _cls
```

```
_cls proc       ; 清屏
        push ax          ;寄存器压栈
        push bx
        push cx
        push dx
          mov ax, 600h       ; AH = 6,   AL = 0
          mov bx, 700h       ; 黑底白字(BL = 7)
          mov cx, 0          ; 左上角: (0, 0)
          mov dx, 184fh      ; 右下角: (24, 79)
          int   10h          ; 显示中断

          mov ah, 02h        ;设置光标位置
          mov bh, 0
          mov dx, 0000h      ;第0行，第0列
          int 10h
      pop dx                 ;恢复寄存器信息
      pop cx
      pop bx
      pop ax
      ret
_cls endp
```

Kliba.asm文件中的printChar(char ch)程序，显示一个字符：

```
public _printChar
_printChar proc
    push bp              ;寄存器压栈
    mov bp,sp
    mov al,[bp+4]        ;读取参数ch
    mov bl,0
    mov ah,0eh
    int 10h              ;调用显示中断
    mov sp,bp
    pop bp               ; 恢复寄存器信息
    ret
_printChar endp
```

Kliba.asm文件中的getChar()程序，读取一个字符：

```
public _getChar
_getChar proc
    mov ah,0
```

```
    int 16h                     ;调用16h中断读取一个字符
    mov byte ptr [_input], al   ;将读取的字符传给input
    ret
_getChar endp
```

Kernel.c文件中print(char *shr)程序，显示一个字符串：

print(char *str)函数调用kliba.asm中的printChar(char s)显示字符串。str指向字符串的首地址，*str将首个字符传入printChar显示，str++将str指向下个字符，以此类推，当str指向字符串的末尾时程序停止，这样逐个显示字符以达到显示整个字符串的目地。

代码如下：

```
void print(char *str) {
    while(*str != '\0') {
        printChar(*str);
        str++;}}
```

kernel.c文件中getline(char *ptr, int len)程序，读取输入字符串：

getline(char *ptr, int length)函数调用kliba.asm中的getChar()读取输入字符串。其中ptr指向输入存入的字符串首地址，length指读取输入的最大长度。在getline函数中，初始化一个名为count的变量用于计数，当length为0时，不读取输入直接返回；当length不为0时，读取一个输入字符，判断该字符是否为回车键，若是回车键，则换行和回车并返回。反之，则显示该字符并将该字符存入ptr中，count加1，若count等于length，则在字符串末尾加上休止符‘\0’，再换行和回车并返回。若不相等，则读取一个输入字符。以此类推，将输入字符串存入ptr字符串中。

代码如下：

```
void getline(char *ptr, int length) {
    int count = 0;
```

```
if (length == 0) return;
else {
    getChar();
    while (input != 13) {
        printChar(input);
        ptr[count++] = input;
        if (count == length) {
            ptr[count] = '\0';
            print("\n\r");
            return;
        }
        getChar();
    }
    ptr[count] = '\0';
    print("\n\r");
    return;}}
```

kernel.c文件中的strcmp(char *str1, char *str2)程序，比较两个字符串：

strcmp(char *str1, char *str2)函数比较两个字符串是否相等。首先，比较str1和str2所指的字符是否相等，若不相等，则比较两个字符的字典序，若*str1<*str2，则返回-1，反之，返回1；若相等，则str1和str2分别指向下个字符。依此比较，直指出现休止符，然后返回*str1和*str2的相减值。

代码如下：

```
int strcmp(char *str1, char *str2) {
    while ((*str1) && (*str2)) {
        if (*str1 != *str2) {
            if (*str1 < *str2) return -1;
            return 1;
        }
        ++str1;
        ++str2;
    }
    return (*str1) - (*str2);}
```

kernel.c文件中的strlen(char *str)程序，计算字符串长度：

strlen(char *str)函数计算字符串的长度。首先，初始化一个名为i的变量用于计数，然后判断str所指的字符是否为休止符，若不为休止符，则str指向下个字符，i加1。反之，返回i。

代码如下：

```
int strlen(char *str) {
    int i = 0;
    while(*(str++)) i++;
    return i;
}
```

Kernel.c文件中的substr(char *src, char *sstr, int pos, int len)程序，得到字符串的子字符串：

substr(char *src, char *sstr, int pos, int len)函数返回src的一个子字符串。首先，初始化i为pos，即子字符串在src中的起始地址，然后将src中的字符复制到sstr中，当sstr的长度等于len时，程序结束并返回。

代码如下：

```
int substr(char *src, char *sstr, int pos, int len) {
    int i = pos;
    for (; i < pos + len; ++i)
        sstr[i - pos] = src[i];
    sstr[pos + len] = '\0';
    return 1;
}
```

kernal.c文件中的cmain()主程序：

```
cmain() {
    initial();                          //初始化界面，显示提示信息
    while(1) {
        char commands[100];
        char tmp_char[10];
        print("root@MyOS:~#");
```

```
        getline(commands, 100);      //读取用户输入，输入上限为100
        //识别用户输入，根据用户输入执行不同操作
        if (strcmp(commands, "help") == 0) help();          //显示帮助文档
        else if (strcmp(commands, "cls") == 0) cls();       //清屏操作
        else if (strcmp(commands, "ls") == 0) ls();         //显示用户程序信息
        else {
            substr(commands, tmp_char, 0, 1);
            if (strcmp(tmp_char, "r") == 0) {               //执行用户程序
                runprogram(commands);}
            else if (commands[0] == '\0') continue;
            else {
                print("Illegal command: ");          //识别用户的非法指令
                print(commands);
                print("\n\n\r");}}}}
```

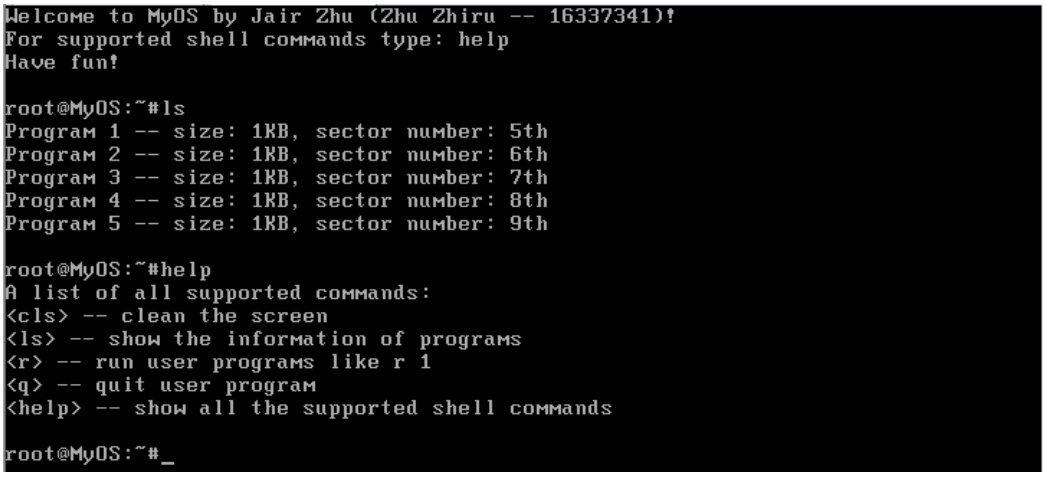
kernal.c文件中的runprogram(char *comm)函数：

```
void runprogram(char *comm) {
    int i;
    for (i = 1; i < strlen(comm); ++i) {
        if (comm[i] == ' ') continue;          //忽略用户指令中的空格
        else if (comm[i] >= '1' && comm[i] <= '5') {
            pro = comm[i] - '0' + 4;          //根据指令加载并运行相应的用户程序
            run();
            return;}
        else {
            print("invalid program number: ");//识别指令中的无效程序号
            printChar(comm[i]);
            print("\n\n\r");
            return;}}}
```

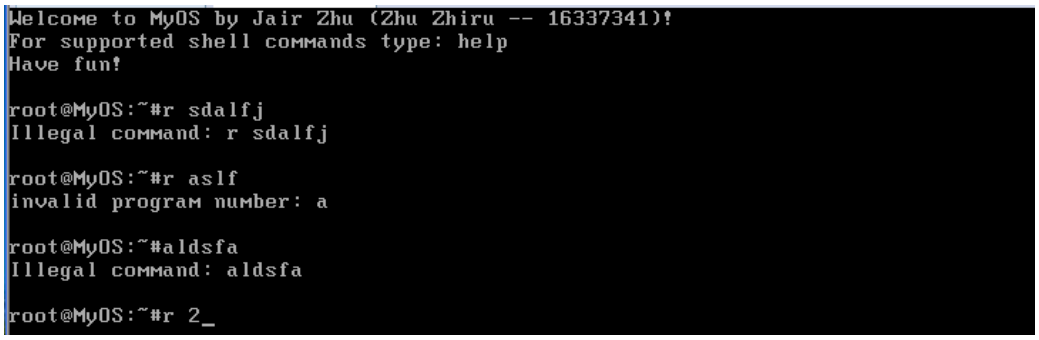
四.实验过程和结果

结果：

进入操作系统后输入ls指令显示用户程序的信息，输入help指令显示帮助信息：



操作系统也可以识别无效指令：



输入r 5指令进入用户程序5，运行效果如图所示：

五. 实验总结

总结:

我觉得这次实验的难点在于完善底层基本功能。因为我用c编写程序时调用了汇编代码中的一些功能程序，如清屏、加载并运行用户程序、显示一个字符、读取一个字符输入，在编译执行测试时出现了各种奇怪的bug，然后我就在c中调用stdis.h以测试c代码是否存在错误，发现编写的c程序几乎没有什么问题，那么这就表明之前各种奇怪的bug是底层汇编代码出现错误导致的。这次实验我花了一大半的时间用于调试汇编底层程序，深深的感觉到汇编语言没有c语言那么平易近人。

我觉得这次实验还有一个纠结点就是选择走TCC+TASM，还是走GCC+NASM，之前我本来想在WIN10 64位操作系统下直接编写、编译代码，再加上前两次实验均是以NASM语法编写汇编代码，而选择GCC+NASM这条路，但当我编写好汇编代码和c代码后，分别编译它们生成.obj文件，在链接这两个.obj文件时出现了错误:

i686-elf-ld: warning: cannot find entry symbol _start; defaulting to 00007e00

在网上查找各种资料均没有找到这个问题的解决方案，于是我放弃走GCC+NASM这条路，而转向TCC+TASM这条路。然而这条道路也充满艰险，因为TASM和NASM的语法有很大的不同，并且老师给的TCC、TASM和TLINK在WIN10 64位操作系统下不能运行，于是我在Vmware Workstation上又装了一个win7 32位操作系统的虚拟机，以便使用TCC、TASM和TLINK这些工具。

c代码和汇编代码的参数传递是通过压栈的方式传递，在编写汇编代码时需要考虑参数在栈中的位置，考虑的情况比较复杂。

为了避免考虑复杂的情况，我在kernal.c代码中声明全局变量input、pro，然后在kliba.asm导入这俩个c代码的全局变量。在getChar()底层功能代码中，将读入字符的ACSII码赋给input。这样kernal.c代码就可以调用getChar()读取用户输入，input存的就是输入字符的ACSII码，需要使用用户输入的字符时，直接使用input即可。同理，在run()底层功能代码中，将pro赋给cl寄存器指定起始扇区号。这样在kernal.c代码中先将起始扇区号赋给pro，再调用run(),即可载入并运行用户程序。

问题和解决方法：

（1）由于走TCC+TASM这条路，我之前在lording.asm引导程序使用TASM语法写的，但是，当我想使用org 7C00h这条指令来使程序访问正确的数据的时候，发现TASM编译这个文件生成.com文件时出现了错误：

Cannot generate COM file : invalid initial entry point address

查资料后才知道使用TASM生成.com文件需要将入口设为100h，即使用org 100h指令，这与我想使用org 7C00h指令相矛盾，最后我只好使用NASM语法编写lording.asm引导程序，然后使用na.bat批处理单独将这个文件编译成.com文件。

（2）由于MyOS.asm必须使用TASM编译，那么生成.com文件必须将入口设为100h，但在lording.asm这个引导程序中，我将操作系统内核加载到内存偏移量为8100h的地方，那么在MyOS.asm中想要访问正确的数据那就需要加上org 8100h指令，此时问题又再次出现。

这个问题的解决方案是，在MyOS.asm中使用org 100h指令以便正确生成.com文件，那么需要修改的就是引导程序，在lording.asm中将操作系统内核载入内存后不能直接使用jmp 8100h跳转至操作系统，而是改为jmp 800h:100h，这样也能跳转至操作系统，因为8100h和800h:100h所指的物理地址是相同的。并且使用jmp

800h:100h指令使得跳转后将段值设为800h，偏移量设为100h，这样在MyOS.asm中使用org 100h指令也能访问正确的的数据，也能在TASM下编译成.com文件。

并且在20h中断中的返回操作系统的jmp 8100h指令也应该改为jmp 800h:100h。

（3）在lording.asm中载入中断向量20h，在用户程序中使用int 20h以便响应用户的按键并返回至操作系统。当我在中断20h的代码中使用int 16h中断1号功能调用查询键盘缓冲区，响应、判断用户的输入并做出相应的操作，但在实际测试中发现，用户按下错误按键的时候，没有返回操作系统，但当用户再次按下正确的按键时，也没有返回操作系统。

对于这个问题，我先使用int 16h中断1号功能调用查询键盘缓冲区，响应用户的按键输入，再使用其0号功能从键盘读入字符送AL寄存器，然后判断是否为‘q’来决定是否返回操作系统。

（4）之前我使用原本的kliba.asm汇编代码中的printf(char *str)显示字符串，在c代码中调用printf显示字符串，当显示两三行字符串时没有出现问题，只是光标的位置不正确，但调用printf显示四行或是更多行字符串时，整个界面就铺满乱码。刚开始我以为是光标的缘故，我就在MyOS.asm中在call near ptr _cmain指令前将光标设置到第0行第0列，测试时正常显示了，但用户无法输入，因为留给用户输入的部分被乱码取代了。这就说明printf(char *str)函数是存在bug的，于是我调用kliba.asm中的printChar(char s)，在kernal.c中的print(char *str)将printChar函数包装一下使其可以显示一段字符串。测试时发现，字符串正常显示了，但无法正常显示‘\n’，即没有换行而是显示一个白底黑字的句号。

这些问题的出现说明kliba.asm中的printf(char *str)，printChar(char s)这两个底层功能函数存在bug，我只能重构显示字符这个底层功能函数。我选择调用10h中断

0Eh号功能显示单个字符，再在kernal.asm中的print(char *str)包装该函数使其可以显示字符串。测试时发现，字符串正常显示了，要想换行则需加上‘\n’和‘\r’，即换行和回车。

（5）同样，在原本的kliba.asm中的cls()清屏功能在测试时出现了问题，虽然将屏幕上的字符全部清除了，但再次显示字符时，显示的位置出现了问题，它没有在第0行第0列开始显示，而是接在清除的字符之后。

对于这个问题，我在cls()中清屏操作后加上了重置光标操作，将光标置于第0行第0列。这样，屏幕上的字符全部清除后，再次显示字符时，显示的位置是第0行第0列。

六. 参考文献

1、 《x86 PC汇编语言，设计与接口》

2、 汇编中的10H中断int 10h详细说明

http://www.itzhai.com/assembly-int-10h-description.html#read-more

3、 键盘I/O中断调用（INT 16H）

https://blog.csdn.net/qingkongyeyue/article/details/68490194