

# 操作系统实验报告

---

计科（大数据、人工智能方向）17341155 王永康

## （实验三：C与汇编开发独立批处理的内核）

### 一、实验目的

初步熟悉C与汇编联合开发操作系统的步骤，同时设计出操作系统的内核，使之具备一定的批处理功能。

### 二、实验内容

- 将实验二的原型操作系统分离为引导程序和kernel内核，由引导程序加载内核，用C和汇编实现操作系统内核
- 扩展内核汇编代码，增加了一些有用的、较为完善的输入输出函数，供C模块中调用
- 提供用户程序返回内核的一种解决方案
- 在内核的C模块中实现增加批处理能力：
  - 实现了从磁盘第二扇区读一个信息表，并显示在终端内。
  - 实现了一种命令，加载多个用户程序。
  - 实现了原语的操作。

### 三、实验方案

#### 操作系统实验工具与环境

- 实验支撑环境
  - 硬件：个人计算机
  - 主机操作系统：Windows10
  - 虚拟机软件：VMware,Dosbox
- 实验开发工具：
  - 汇编语言工具：x86汇编语言
  - 汇编编译工具：TASM+TCC
  - 磁盘写入工具：Winhex

### 四、实验过程和结果

## 分离引导程序boot与kernel内核程序

把kernel程序放置在第3-5扇区，并把加载进内存的地址偏移设为5000h,然后在引导程序里面先把kernel从第3、4、5扇区加载进内存偏移为5000h的位置，利用中断int 13H，之后再利用 `jmp 5000h` 指令，把控制权移交给内核，实现跳转进入内核。

## 内核的设计与开发

这一部分，使用C与X86汇编联合开发，主要思想是：建立两个文件：kernel.asm与cmain.c,二者协同完成内核功能。

### 主要功能

- 用户输入，系统输出响应，完成简单的交互过程
- 提供一组内置操作系统语言，输入可以做出相应功能；

```
1  <ls          >Directory view
2  <clear       >Clear screen
3  <run         >The next line to input program to
    run
4  <help        >Show the commands
5  <init.cmd    >Execute the file
```

### 代码部分

其中，kernel.asm提供输入、输出函数给cmain.c调用，然后kernel.asm调用cmain,由cmain完成内核的相应功能。

- **kernel.asm 部分**

kernel.asm提供的函数有：

```
1  public _printChar
2  extern _Message:near
3  extern _row:near
4  extern _col:near
5  ;在C里面封装，对应
6  void putChar(char message,int Row,int Col)
7  ;把一个字符显示在第row行，第col列
8
9  public _getChar
10 extern _GetM:near
11 extern _Flag:near
12 ;在C里面封装，对应
13 void getC(char* dt)
14 ;把读取的字符的值通过GetM传递给*dt;
15 ;其中Flag参数是用来判断是否读取Backspace
```

```

16
17     public _Load
18     extern _sector:near
19     ;在C里面封装, 对应
20     void load(int sector)
21     ;表示加载sector扇区进内存, 并执行程序
22
23     public _clear
24     ;void clear(), 清空屏幕
25
26     public _read
27     extern _des:near
28     ;相当于
29     void Read(char* s)
30     ;读取磁盘第二扇区的表, 返回字符串首地址des
31
32     public _readInit
33     extern _Initdes:near
34     ;相当于
35     void Readinit(char * s)
36     ;读取第11扇区的原语, 返回字符串首地址Initdes

```

其中 `_printChar` 函数还需要考虑**屏幕滚动**的功能：当用户输入达到**屏幕底端**，需要屏幕**向上滚动**；

`getChar` 函数考虑读入 `enter` 和 `backspace` 等键，为屏幕用户输入输出做处理；

- **cmain.c 部分**

```

1 void Clears(); //清空屏幕
2 void putChar(char a, int h, int l); //打印字符
3 void getc(char* dt); //读取一个字符
4 void getline(char* st); //读取一行字符
5 void printf(const char*s); //打印字符串
6 int len(const char*s); //获取字符串的长度
7 int cmp_equ(char*s, char*t); //判断字符串是否相等
8 void show_mem(); //读取磁盘, 并且展示磁盘信息表
9 void cmain(); //实现kernel功能

```

其中 `void printf(const char*s)` 代码如下：

```

1 void printf(const char*s)
2 {
3     int i = 0;
4
5     for(i=0;i<len(s);++i){
6         putchar(s[i],disp_pos,i);//通过调用打印一
           个字符的函数void putchar();
7     }
8     disp_pos++; //全局变量，控制显示的行。
9 }

```

`void getline(char* st)`代码如下：因为`getline`需要做到读一个字符，就要在屏幕上**同时显示**一个字符的功能，并且还需要实现功能：用户输错，可以通过backspace回退。

```

1 void getline(char* st)
2 {
3
4     int i = 0;
5     int cntt = 0;
6     int init_col = col;//记录初始行
7
8     while(1){
9         Flag = 0;
10        getC(ttt+i);
11
12        if(Flag == 1){
13            if(col>init_col){/*解决backspace回
           退，和光标问题*/
14                putchar(' ',row,col);
15                putchar(' ',row,col+1);
16                putchar('\b',row,col);
17                col--;
18            }
19
20            if(col>init_col){
21                col--;
22                i--;
23            }
24            continue;
25        }
26        i++;
27        col++;
28        if(ttt[i-1] == ' '|| ttt[i-1] ==
           '\n'){//判断一行读入结束
29            break;
30        }
31        putchar(ttt[i-1],row,col);

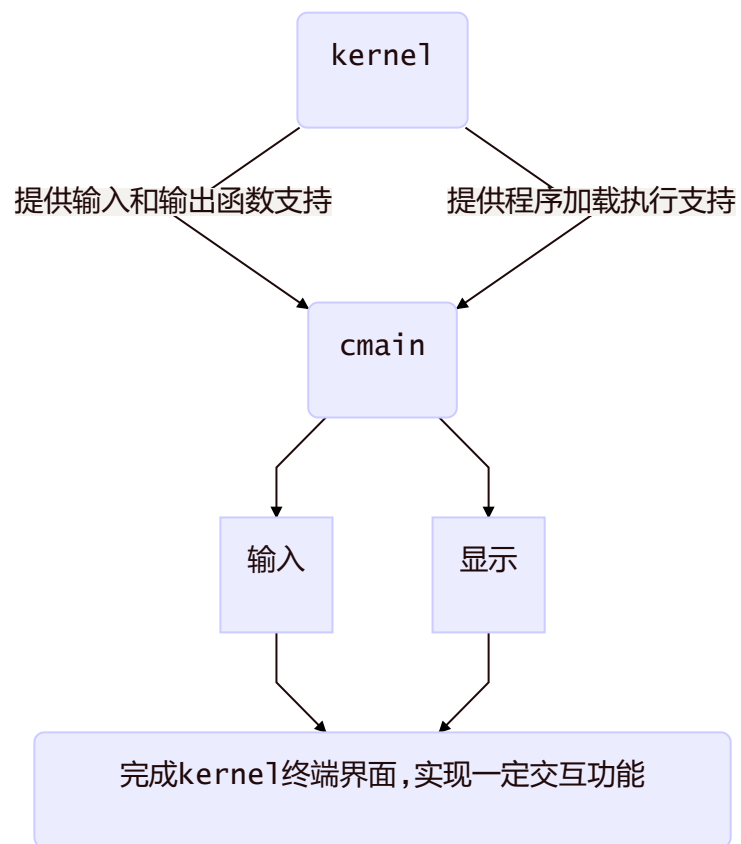
```

```

32     }
33
34     putchar('$', ++disp_pos, 0); //在屏幕上显示一个
    $
35
36     //把读入在缓存的char*ttt读入目标char*st;
37     for(cntt = 0; cntt < i; ++cntt){
38         st[cntt] = ttt[cntt];
39     }
40     st[i-1] = '\0';
41 }

```

## 程序流程



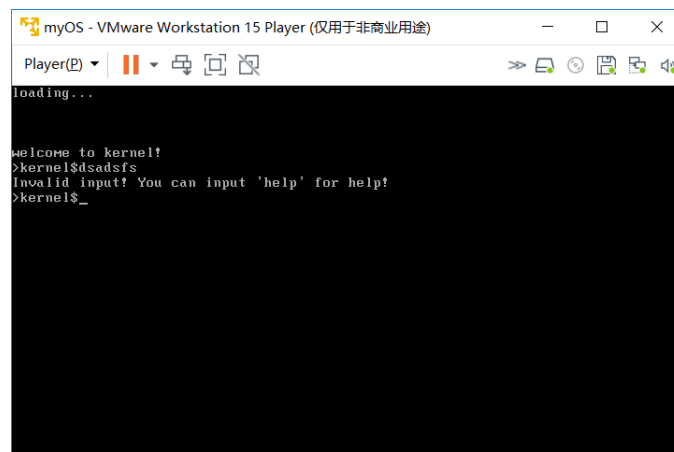
## 实验结果

- 编写源文件kernel.asm和.cmain.c

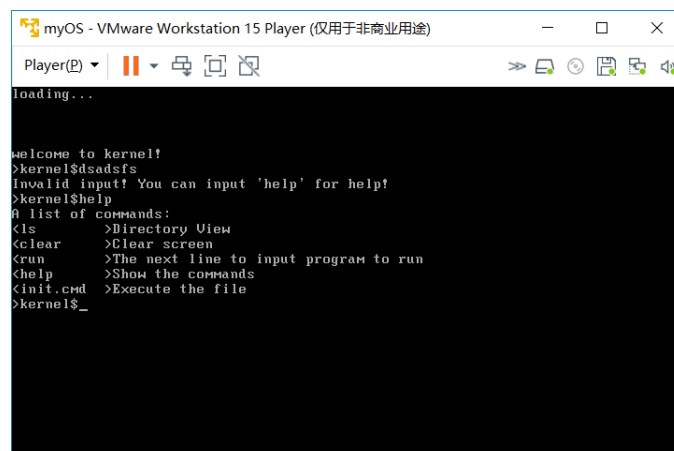
- 链接生成ker.com
- 利用Winhex分别把  
myOS.bin放入第一扇区  
a.txt软盘信息表放入第二扇区  
ker.com放入软盘第3、4、5扇区  
用户程序pro1,pro2,pro3放入6, 7, 8扇区  
init.cmd放入第11扇区
- 运行虚拟机结果如下：
  - 初始界面：



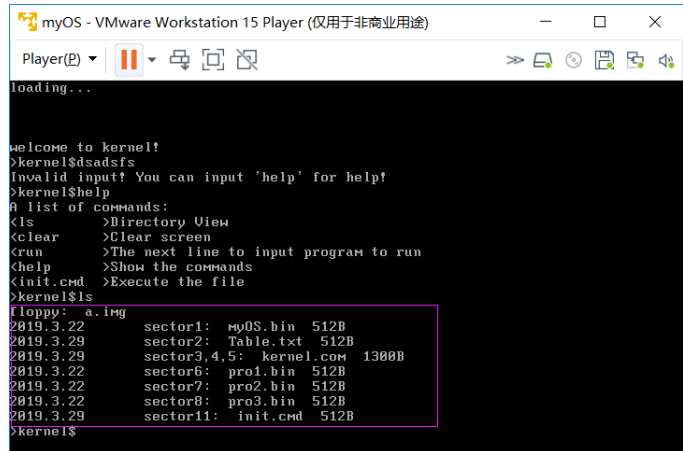
- 随机输入：



- help



- 展示软盘信息：ls



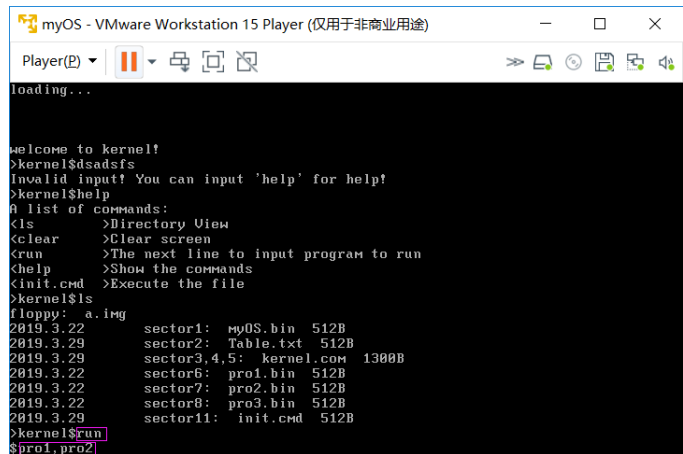
```
loading...

welcome to kernel!
>kernel$dsadsfs
Invalid input! You can input 'help' for help!
>kernel$help
A list of commands:
<ls      >Directory View
<clear   >Clear screen
<run     >The next line to input program to run
<help    >Show the commands
<init.cmd>Execute the file
>kernel$ls
floppy: a.img
2019.3.22      sector1: myOS.bin  512B
2019.3.29      sector2: Table.txt 512B
2019.3.29      sector3,4,5: kernel.com 1300B
2019.3.22      sector6: pro1.bin  512B
2019.3.22      sector7: pro2.bin  512B
2019.3.22      sector8: pro3.bin  512B
2019.3.29      sector11: init.cmd 512B
>kernel$
```

- run批处理程序：

run

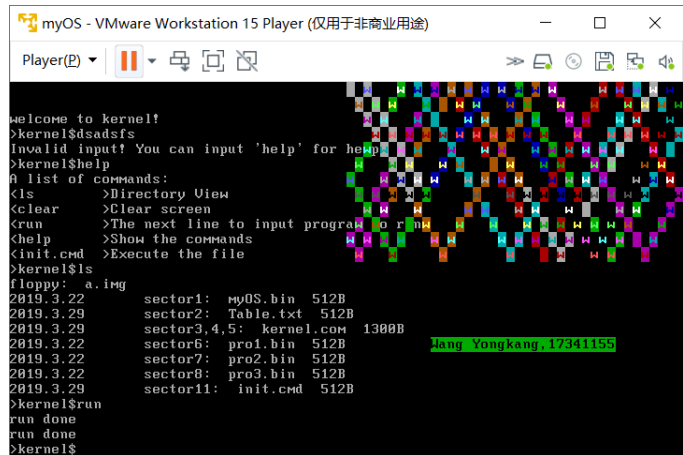
pro1,pro2



```
loading...

welcome to kernel!
>kernel$dsadsfs
Invalid input! You can input 'help' for help!
>kernel$help
A list of commands:
<ls      >Directory View
<clear   >Clear screen
<run     >The next line to input program to run
<help    >Show the commands
<init.cmd>Execute the file
>kernel$ls
floppy: a.img
2019.3.22      sector1: myOS.bin  512B
2019.3.29      sector2: Table.txt 512B
2019.3.29      sector3,4,5: kernel.com 1300B
2019.3.22      sector6: pro1.bin  512B
2019.3.22      sector7: pro2.bin  512B
2019.3.22      sector8: pro3.bin  512B
2019.3.29      sector11: init.cmd 512B
>kernel$run
$pro1,pro2
```

Enter之后：

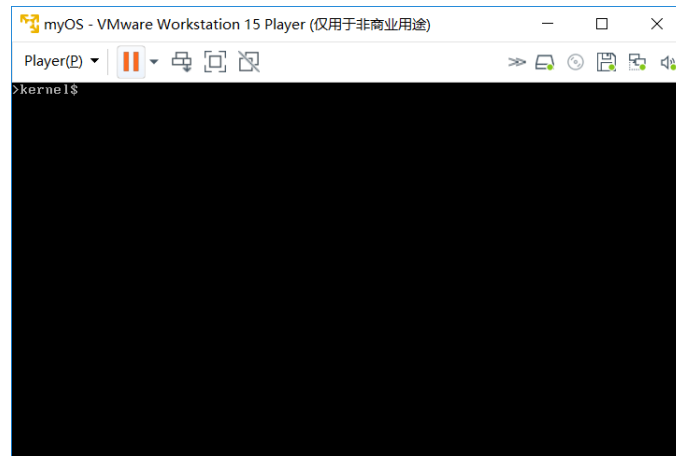


```
loading...

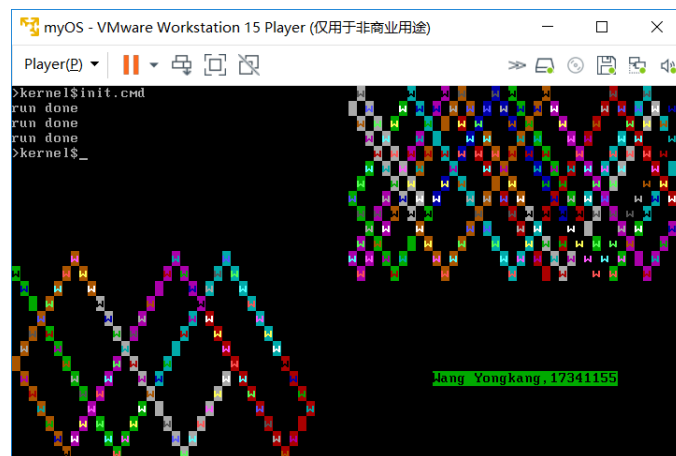
welcome to kernel!
>kernel$dsadsfs
Invalid input! You can input 'help' for help!
>kernel$help
A list of commands:
<ls      >Directory View
<clear   >Clear screen
<run     >The next line to input program to run
<help    >Show the commands
<init.cmd>Execute the file
>kernel$ls
floppy: a.img
2019.3.22      sector1: myOS.bin  512B
2019.3.29      sector2: Table.txt 512B
2019.3.29      sector3,4,5: kernel.com 1300B
2019.3.22      sector6: pro1.bin  512B
2019.3.22      sector7: pro2.bin  512B
2019.3.22      sector8: pro3.bin  512B
2019.3.29      sector11: init.cmd 512B
>kernel$run
run done
run done
>kernel$
```

完成了两个程序先后完成执行，同时注意到屏幕向上发生了滚动。

- clear清屏



- init.cmd原语执行



说明一下：左下角是pro3,右上角是pro2,右下角是pro1;

## 五、实验总结

这一次实验，我想谈谈一些真实感想。

### 关于历史与发展的问题

从实验三作业安排下来，一直在走保护模式的路，一直到这星期二（3.26）才渐渐意识到周六要交作业，保护模式应该来不及了，所以才放弃转为实模式。

不得不说，实验三的要求对于保护模式要求实在是太高了，如果要完成实验三，以于渊的《一个操作系统的实现》来看，至少要看前七章（296页），中间的各种文件系统，进保护模式，设置gtd等等，实在是内容巨多，当时周二看完第5章，进了一下保护模式的内核，就觉得时间绝对来不及，所以转向实模式去开发了。从linux到windows,从32位到16位，从gcc到tcc,从nasm到tasm,从21世纪到上世纪80年代，哦，用一套30年前年龄比我还大的工具（TCC、TASM）去开发操作系统了，有那么一瞬间，觉得时代在发展，有些东西却永远不会过时的，有些旧事物势力还是过于强大，对于新生的、现代的力量摧残是立竿见影的，于是一个充满着对现代与时俱进技术探索的年轻人终于在它面前低下了头，去经历上个世纪的历史沧桑。不过，也好这样也就能在20岁年纪拥有30年的工作经验了。

### 其次学生主动与老师交流的问题



对于操作系统实验课程安排的不合理问题，其实大家私下都吐槽很多的，但是却没有人直接向老师反映，这是一个问题，我记得以前也有类似课程设置不合理，不过有人写email给老师，老师立马就采取措施更正了过来。这一次，却没有人这么做了，直到星期五上实验课，老师才知道有些同学在走32位保护模式的道路（坚持下来的已经不多了），于是他断然鼓励这些同学，并且直接延期4周，宽限时间给他们。而我实模式下已经做完了，顿时心里五味杂陈。也由此可见，大学老师都是比较通情达理的，学生与老师的交流还是非常必要且有效的，这样才能更好促进教学过程。

好了，谈完这些，就是关于实验过程的一些感悟：

### TCC下C语言和TASM下汇编语言语法不熟悉对实验干扰极大

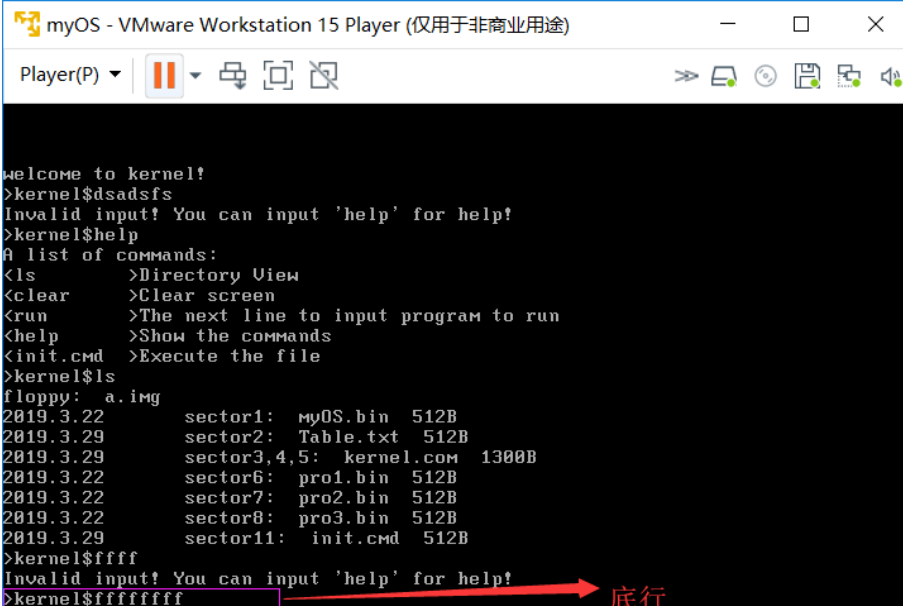
TCC下C语言注释不能用`//`，TCC下C语言的局部变量需要在最前面声明，不允许之间声明；TASM变量要用ptr等等。因为这部分内容比较古老，网上也几乎搜不到相应语法资料，只能自己研究老师的代码来获取相应使用规则，这是干扰非常大的。

### 输入过程删除后光标问题

还有一点遇到的问题就是对于读字符的问题，因为要模拟这个过程，就必须要把用户输入的字符也同时显现出来，这个过程还是有些复杂，还有是backspace退格问题，我的思路是判断是退格后，就马上在光标前一个位置输出2个空格，然后再\b退格，这样可以模拟一个删除之前字符的操作，同时把光标也移动到了前一个位置。

### 底行屏幕上滚问题

个人也考虑了屏幕到底行之后，整个屏幕向上滚动的操作，只是有一些小问题：最后一行（底行）无法向上翻滚，但是其他的行却可以，包括下一次本应该在底行输出的也可以向上滚动，这个过程我是一直没有解决。（示意图如下：）



```
welcome to kernel!
>kernel$dsadsfs
Invalid input! You can input 'help' for help!
>kernel$help
A list of commands:
<ls      >Directory Uiew
<clear   >Clear screen
<run     >The next line to input program to run
<help    >Show the commands
<init.cmd>Execute the file
>kernel$ls
floppy: a.img
2019.3.22      sector1:  myOS.bin  512B
2019.3.29      sector2:  Table.txt  512B
2019.3.29      sector3,4,5: kernel.com 1300B
2019.3.22      sector6:   pro1.bin  512B
2019.3.22      sector7:   pro2.bin  512B
2019.3.22      sector8:   pro3.bin  512B
2019.3.29      sector11:  init.cmd  512B
>kernel$fffff
Invalid input! You can input 'help' for help!
>kernel$ffffff
```

Enter之后，本来会在下一行输出Invalid.....;

可以看到最后一行没有实现上滚。这个问题一直尝试无法解决。

### 最后再谈一谈实模式和保护模式

现在，我实模式做了，保护模式也可以说做了一点点，我有以下一些体会：

保护模式的原理可以说是非常复杂，涉及的知识是非常多的，也是比较现代，与时俱进的，可以说，走这一条路，学到知识是非常多的。但是，实际上，这一部分从打码实现角度来谈，其实大部分代码都是直接复制过来用就好了，因为自己实现我觉得是不太可能的，太多，汇编寄存器之间技巧太强，所以一直复制过来，是可以直接进入保护模式的内核的，甚至说，几乎可以不写汇编，就可以完成内核的构建，只需要自己写一些C语言代码就可以完成，从这个角度来看，其实打码难度其实是比实模式低的。相比而言，实模式下，完成靠个人写码实现，对于汇编的操作还是更多的，难度还可能稍微大一些。

### 参考文献

[1] 凌应标. “03实验课.ppt”，中山大学计算机科学系，2015-3

[2] 于渊. 《一个操作系统的实现》[M]. 电子工业出版社，2009-6-1