

# 中山大学数据科学与计算机学院本科生实验报告

## (2016 学年秋季学期)

课程名称：操作系统实验

任课教师：凌应标

教学助理 (TA):

年级	15 级	专业 (方向)	软件工程 (移动信息工程)
学号	15352461	姓名	宗嘉希 (组长)
学号	15352443	姓名	钟凌山
学号	15352448	姓名	周禅城
电话	18022724490	Email	zongjx@mail2.sysu.edu.cn
开始日期	2017.04.07	完成日期	2017.04.17

### 【实验题目】

用 C 和汇编设计内核

### 【实验目的】

1. 将实验二的原型操作系统分离为引导程序和 OS 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

### 【实验要求】

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

## 【实验方案】

### 一、 硬件及虚拟机配置

硬件：操作系统为 win10 的笔记本电脑

虚拟机配置：无操作系统，10MB 硬盘，4MB 内存，启动时连接软盘

### 二、 软件工具及作用

Nasm:用于编译汇编程序，生成.bin 文件

WinHex:用于向软盘写入程序

VMware Workstation 12 Player：用于创建虚拟机，模拟裸机环境

Notepad++: 用于编辑汇编语言文件

TCC：用于编译 C 文件，生成 OBJ 文件

TASM：用于编译 ASM 文件，生成 OBJ 文件

TLINK：用于把两个 OBJ 文件连接，生成 COM 文件

Dosbox：用于提供 16 位运行环境，为 TCC+TASM+TLINK 编译链接提供一个环境

## 【实验过程】

首先，配置好虚拟机，创建好一个 1.44MB 的虚拟软盘并连接到虚拟机。

然后我们首先设计了四个不一样的有输出的用户可执行程序代码，分别编译出四份可执行程序。四个程序的内容是：

1、程序显示在屏幕的左上角的四分之一的位置，用字符 'Z' 从屏幕左上角位置 45 度角下斜射出，保持一个可观察的适当速度直线运动，碰到屏幕相应 1/4 区域的边后产生反射，改变方向运动，如此类推，不断运动。在最左上角打印出组员 1 的姓名和学号。

2、程序显示在屏幕的右上角的四分之一的位置，用字符 'Z' 从屏幕右上角位置 45 度角下斜射出，保持一个可观察的适当速度直线运动，碰到屏幕相应 1/4 区域的边后产生反射，改变方向运动，如此类推，不断运动。在最右上角打印出组员 2 的姓名和学号。

3、程序显示在屏幕的左下角的四分之一的位置，用字符 'Z' 从屏幕左上角位置 45 度角上斜射出，保持一个可观察的适当速度直线运动，碰到屏幕相应 1/4 区域的边后产生反射，改变方向运动，如此类推，不断运动。在最左下角打印出组员 3 的姓名和学号。

4、程序显示在屏幕的右下角的四分之一的位置，用字符 'Z' 从屏幕左上角位置 45 度角上斜射出，保持一个可观察的适当速度直线运动，碰到屏幕相应 1/4 区域的边后产生反射，改变方向运动，如此类推，不断运动。在最右下角打印出所有组员的姓名和学号。

这四个程序跟实验二的程序内容上是差不多的，但是由于我们将要用到分时运行，所以会在程序里做出一些改动，这将会在实验报告后面讲到。

下一步我们就要开始设计 c 程序和 os 内核了，首先要确定要做什么功能，由于汇编学的不是很好，所以我们就决定模仿师兄的程序里的功能，实现批处理功能和分时功能，虽然不是很懂这些名词是什么意思，但是我经过自己的理解后，认为批处理是按顺序运行几个程序，当一个程序运行完了，再运行下一个程序；分时是同时运行几个程序。

首先决定实现程序要使用哪些功能，一共有以下几个函数，先把声明写在 c 语言文件的开头，使用

extern 标志来定义，因为这些函数的实现是写在此 c 文件关联的汇编文件中的。

```
2 extern void clear(); /*连接外部的clear()函数,用于清屏*/
3 extern void inputchar(); /*连接外部的inputchar()函数,用于读入字符*/
4 extern void printchar(); /*连接外部的printchar()函数,用于输出字符*/
5 extern void printstring(); /*连接外部的printstring()函数,用于输出字符串*/
6 extern void setcursor(); /*连接外部的setcursor()函数,用于设置光标*/
7 extern void load_prog(); /*连接外部的load_prog()函数,用于加载外部程序*/
8 extern void run_prog(); /*连接外部的run_prog()函数,用于跳转到外部程序*/
9 extern void filldata(); /*连接外部的filldata()函数,用于填入数据*/
10 extern void readdata(); /*连接外部的readdata()函数,用于读入数据*/
```

把以上函数声明好后就要声明出要显示的字符串：

```
12 /*以下将创建若干长度为80的字符串变量(屏幕最大长度为80),用于存放提示语句*/
13 char message1[80]="Welcome to our team's system!\n";
14 /*char message2[80]="Coded by Zongjiaxi Zhonglingshan Zhouchancheng\n";*/
15 char message3[80]="You can input some legal instructions to active the functions\n";
16 char message4[80]=" If you want to run the program by Batching,input \'batch\'\n";
17 char message5[80]=" For example: batch 1 2 3 4\n";
18 char message6[80]=" If you want to run the program by Time-Sharing,input \'time\'\n";
19 char message7[80]=" For example: time 1 2 3 4\n";
20 char message8[80]=" If you want to know the information of the program,input \'show\'\n";
21 char message9[80]=" For example: show\n";
22 char message10[80]=" If you want to run the default instructions,input \'default\'\n";
23 char message11[80]=" For example: default\n\n";
24 char message12[80]="If you want to continue,input \'yes\',otherwise input \'no\'\n";
25 char message13[80]="OS will run instructions \'batch 1 2 3 4\' and \'time 1 2 3 4\'\n";
26 char message_cmd[80]=" Orz >";
27 char message_ins1[80]="User Program1:\n Name:program_LU Size:1024byte Position:part LU\n";
28 char message_ins2[80]="User Program2:\n Name:program_RU Size:1024byte Position:part RU\n";
29 char message_ins3[80]="User Program3:\n Name:program_LD Size:1024byte Position:part LD\n";
30 char message_ins4[80]="User Program4:\n Name:program_RD Size:1024byte Position:part RD\n";
31 char string[80];
32 char message_error[80]="Input Error!!!\n";
33 const char batch_order[80]="batch 1 2 3 4";
34 const char time_order[80]="time 1 2 3 4";
```

还有需要用到的变量：

```
35 int len=0; /*代表长度的变量*/
36 int pos=0; /*代表位置的变量*/
37 char ch; /*代表输入的字符*/
38 int x=0; /*纵向位置的变量*/
39 int y=0; /*横向位置的变量*/
40 int i=0; /*用于计数的变量*/
41 int j=0; /*用于计数的变量*/
42 int k=0; /*用于计数的变量*/
43 int data1=0; /*数据存储的变量*/
44 int offset_prog=37120; /*用户程序的偏移量,0910h*10h+0h=37120(初始位置)*/
45 const int offset_prog1=37120; /*第一个用户程序的偏移地址 0910h*10h+0h=37120*/
46 const int offset_prog2=38144; /*第二个用户程序的偏移地址 0950h*10h+0h=38144*/
47 const int offset_prog3=39168; /*第三个用户程序的偏移地址 0990h*10h+0h=39168*/
48 const int offset_prog4=40192; /*第四个用户程序的偏移地址 09d0h*10h+0h=40192*/
49 const int hex600h=1536; /*600h的十进制,作为内存地址用于标记用户程序的调用情况*/
50 int offset_begin=37120; /*初始位置内存偏移量*/
51 int num_sector=8; /*扇区的总数为8个*/
52 int pos_sector=2; /*起始扇区的编号为2,第一个扇区放置的是引导程序*/
```

接下来就要定义一些在 c 语言里面比较容易实现的函数了：

输入数据的函数：

```
54 int input() { /*控制输入字符的函数*/
55     inputchar(); /*读入一个字符的函数*/
56     if(ch=='\b'){ /*如果读入的字符是删除键Backspace*/
57         if(y>8&&y<79){ /*如果光标的横向位置是在8到79之间*/
58             y--; /*光标向左移一位*/
59             cal_pos(); /*重新计算光标的坐标*/
60             putchar(' '); /*输出一个空白格*/
61             y--; /*y变量自减1*/
62             cal_pos(); /*重新计算光标的位置*/
63         }
64         return 0; /*如果输入的是删除键，则返回0*/
65     }
66     else if(ch==13); /*回车键*/
67     else putchar(ch); /*如果不是以上情况，就显示字符*/
68     return 1; /*如果不是以上情况，就返回1*/
69 }
```

调用输入命令的函数：

```
71 cin_cmd() { /*输出命令提示符的函数*/
72     printstring(message_cmd); /*打印字符串的函数*/
73     i=0; /*初始字符串下标*/
74     while(1){ /*while循环*/
75         if(input()){ /*如果输入的不是删除键*/
76             if(ch==13) break; /*如果输入的字符是回车键，则跳出while循环*/
77             string[i++]=ch; /*否则把该字符添加到string的末尾*/
78         }
79         else if(i>0) i--; /*如果输入的是删除键，就删除一个字符*/
80     }
81     for(j=0;j<i;j++) /*进入for循环，遍历string字符串*/
82         if(string[j]==' '){ /*如果字符串首位是空格*/
83             for(k=1;k<i;k++) string[k-1]=string[k]; /*把首位空格去掉*/
84             i--; /*字符串长度减1*/
85         }
86     else break; /*如果字符串首位不是空格，就跳出循环*/
87     string[i]='\0'; /*字符串末尾补上一个空格，也可以把i-1*/
88     len=i; /*记录下字符串的长度i*/
89     printstring("\n"); /*换行*/
90 }
```

计算光标的坐标的函数：

```
92 cal_pos() { /*计算字符或光标的坐标的函数*/
93     if(y>79){ /*假如y方向大于79(到达了右边界)*/
94         y=0; /*y变为0，到达下一行的起始位置*/
95         x++; /*行数x加1*/
96     }
97     if(x>24) clear(); /*假如x大于24(到达了下边界)，清空屏幕到下一页*/
98     pos=(x*80+y)*2; /*重新计算位置*/
99     setcursor(); /*放置光标的函数，在pos的位置放置光标*/
100 }
```



实现批处理功能命令的函数：

```
102 batch() {
103     clear();
104     offset_begin=offset_prog1;
105     num_sector=8;
106     pos_sector=2;
107     load_prog(offset_begin,num_sector,pos_sector);
108     filldata(hex600h,0);
109     filldata(hex600h+2,0);
110     filldata(hex600h+4,0);
111     filldata(hex600h+6,0);
112     for(i=0;i<len;i++){
113         if(string[i]=='1'){
114             offset_prog=offset_prog1;
115             run_prog();
116         }
117         else if(string[i]=='2'){
118             offset_prog=offset_prog2;
119             run_prog();
120         }
121         else if(string[i]=='3'){
122             offset_prog=offset_prog3;
123             run_prog();
124         }
125         else if(string[i]=='4'){
126             offset_prog=offset_prog4;
127             run_prog();
128         }
129     }
130 }
```

/\*实现批处理功能的函数\*/  
/\*清屏，清除引导程序所显示的内容\*/  
/\*设置初始位置内存偏移量等于第一个用户程序的偏移量\*/  
/\*扇区的总数为8个\*/  
/\*起始扇区的编号为2\*/  
/\*装载用户程序到内存，利用中断读取扇区\*/  
/\*向hex600h(600h)这个内存位置填进0，表示第一个用户程序暂不运行\*/  
/\*向hex600h+2(602h)这个内存位置填进0，表示第二个用户程序暂不运行\*/  
/\*向hex600h+4(604h)这个内存位置填进0，表示第三个用户程序暂不运行\*/  
/\*向hex600h+6(606h)这个内存位置填进0，表示第四个用户程序暂不运行\*/  
/\*进入for循环，遍历字符串(输入的指令)\*/  
/\*如果字符串中出现'1'字符\*/  
/\*把第一个用户程序的偏移地址写入到初始地址变量中\*/  
/\*调用运行程序的函数(此处将使用offset\_prog变量)\*/  
/\*如果字符串中出现'2'字符\*/  
/\*把第二个用户程序的偏移地址写入到初始地址变量中\*/  
/\*调用运行程序的函数(此处将使用offset\_prog变量)\*/  
/\*如果字符串中出现'3'字符\*/  
/\*把第三个用户程序的偏移地址写入到初始地址变量中\*/  
/\*调用运行程序的函数(此处将使用offset\_prog变量)\*/  
/\*如果字符串中出现'4'字符\*/  
/\*把第四个用户程序的偏移地址写入到初始地址变量中\*/  
/\*调用运行程序的函数(此处将使用offset\_prog变量)\*/

实现分时功能命令的函数：

```
132 time() {
133     clear();
134     offset_begin=offset_prog1;
135     num_sector=8;
136     pos_sector=2;
137     load_prog(offset_begin,num_sector,pos_sector);
138     filldata(hex600h,0);
139     filldata(hex600h+2,0);
140     filldata(hex600h+4,0);
141     filldata(hex600h+6,0);
142     for(i=0;i<len;++i){
143         if(string[i]=='1')
144             filldata(hex600h,1);
145         else if(string[i]=='2')
146             filldata(hex600h+2,1);
147         else if(string[i]=='3')
148             filldata(hex600h+4,1);
149         else if(string[i]=='4')
150             filldata(hex600h+6,1);
151     }
152     while(1){
153         j=0;
154         readdata(hex600h);
155         j+=data1;
156         if(data1){
157             offset_prog=offset_prog1;
158             run_prog();
159         }
160         readdata(hex600h+2);
161         j+=data1;
162         if(data1){
163             offset_prog=offset_prog2;
164             run_prog();
165         }
166         readdata(hex600h+4);
167         j+=data1;
168         if(data1){
169             offset_prog=offset_prog3;
170             run_prog();
171         }
172         readdata(hex600h+6);
173         j+=data1;
174         if(data1){
175             offset_prog=offset_prog4;
176             run_prog();
177         }
178         if(j==0) break;
179     }
180 }
```

/\*实现伪分时处理的函数\*/  
/\*清屏，清除引导程序所显示的内容\*/  
/\*设置初始位置内存偏移量等于第一个用户程序的偏移量\*/  
/\*扇区的总数为8个\*/  
/\*起始扇区的编号为2\*/  
/\*装载用户程序到内存，利用中断读取扇区\*/  
/\*向hex600h(600h)这个内存位置填进0，表示第一个用户程序暂不运行\*/  
/\*向hex600h+2(602h)这个内存位置填进0，表示第二个用户程序暂不运行\*/  
/\*向hex600h+4(604h)这个内存位置填进0，表示第三个用户程序暂不运行\*/  
/\*向hex600h+6(606h)这个内存位置填进0，表示第四个用户程序暂不运行\*/  
/\*进入for循环，遍历字符串(输入的指令)\*/  
/\*如果字符串中出现'1'字符\*/  
/\*向hex600h(600h)这个内存位置填进1，表示第一个用户程序将会运行\*/  
/\*如果字符串中出现'2'字符\*/  
/\*向hex600h(602h)这个内存位置填进1，表示第二个用户程序将会运行\*/  
/\*如果字符串中出现'3'字符\*/  
/\*向hex600h(604h)这个内存位置填进1，表示第三个用户程序将会运行\*/  
/\*如果字符串中出现'4'字符\*/  
/\*向hex600h(606h)这个内存位置填进1，表示第四个用户程序将会运行\*/  
/\*进入while循环，程序将会分时运行\*/  
/\*记录器j初始为0，此处用于记录是否有程序运行\*/  
/\*读取hex600h(600h)地址中的值，取出的数据将会存到data变量中\*/  
/\*将data的值加到记录器j中\*/  
/\*如果data中的值非0\*/  
/\*把第一个用户程序的偏移地址写入到初始地址变量中\*/  
/\*调用运行程序的函数(此处将使用offset\_prog变量)\*/  
/\*读取hex600h+2(602h)地址中的值，取出的数据将会存到data变量中\*/  
/\*将data的值加到记录器j中\*/  
/\*如果data中的值非0\*/  
/\*把第二个用户程序的偏移地址写入到初始地址变量中\*/  
/\*读取hex600h+4(604h)地址中的值，取出的数据将会存到data变量中\*/  
/\*将data的值加到记录器j中\*/  
/\*如果data中的值非0\*/  
/\*把第三个用户程序的偏移地址写入到初始地址变量中\*/  
/\*调用运行程序的函数(此处将使用offset\_prog变量)\*/  
/\*读取hex600h+6(606h)地址中的值，取出的数据将会存到data变量中\*/  
/\*将data的值加到记录器j中\*/  
/\*如果data中的值非0\*/  
/\*把第四个用户程序的偏移地址写入到初始地址变量中\*/  
/\*调用运行程序的函数(此处将使用offset\_prog变量)\*/  
/\*如果4个数据都为0，则退出循环\*/



## 主函数：

```

182 main(){
183     while(1){
184         clear();
185         printstring(message1);
186         /*printstring(message2);
187         printstring(message3);
188         printstring(message4);
189         printstring(message5);
190         printstring(message6);
191         printstring(message7);
192         printstring(message8);
193         printstring(message9);
194         printstring(message10);
195         printstring(message11);
196         cin_cmd();
197         if(string[0]=='b'){
198             batch();
199             printstring(message12);
200             cin_cmd();
201             if(string[0]=='y') continue;
202         }
203         else if(string[0]=='t'){
204             time();
205             printstring(message12);
206             cin_cmd();
207             if(string[0]=='y') continue;
208         }
209         else if(string[0]=='s'){
210             printstring(message_ins1);
211             printstring(message_ins2);
212             printstring(message_ins3);
213             printstring(message_ins4);
214             printstring(message12);
215             cin_cmd();
216             if(string[0]=='y') continue;
217         }
218         else if(string[0]=='d'){
219             printstring(message13);
220             printstring(message12);
221             cin_cmd();
222             if(string[0]=='y'){
223                 for(i=0;i<80;i++){
224                     string[i]=batch_order[i];
225                     if(batch_order[i]=='\0'){
226                         len=i;
227                         break;
228                     }
229                 }
230                 batch();
231                 printstring(message12);
232                 cin_cmd();
233                 if(string[0]!='y') continue;
234                 for(i=0;i<80;i++){
235                     string[i]=time_order[i];
236                     if(time_order[i]=='\0'){
237                         len=i;
238                         break;
239                     }
240                 }
241                 time();
242                 printstring(message12);
243                 cin_cmd();
244             }
245         }
246         else{
247             printstring(message_error);
248             printstring(message12);
249             cin_cmd();
250             if(string[0]=='y') continue;
251         }
252     }
253 }

```

```

/*主函数*/
/*进入while循环，开始运行程序*/
/*清屏操作*/
/*打印message1字符串*/
/*打印message2字符串*/
/*打印message3字符串*/
/*打印message4字符串*/
/*打印message5字符串*/
/*打印message6字符串*/
/*打印message7字符串*/
/*打印message8字符串*/
/*打印message9字符串*/
/*打印message10字符串*/
/*打印message11字符串*/
/*显示命令提示符，并接收输入的指令*/
/*如果接收到的指令的首位字符是'b'，则意味着调用了批处理功能的函数*/
/*调用batch函数，实现批处理功能*/
/*打印message12字符串*/
/*显示命令提示符，并接收输入的指令*/
/*如果输入的字符串的首位字符是'y'，则重新进入while循环*/

/*如果接收到的指令的首位字符是't'，则意味着调用了伪分时功能的函数*/
/*调用batch函数，实现批处理功能*/
/*打印message12字符串*/
/*显示命令提示符，并接收输入的指令*/
/*如果输入的字符串的首位字符是'y'，则重新进入while循环*/

/*如果接收到的指令的首位字符是's'，则显示每个程序的信息*/
/*打印message_ins1字符串*/
/*打印message_ins2字符串*/
/*打印message_ins3字符串*/
/*打印message_ins4字符串*/

/*显示命令提示符，并接收输入的指令*/
/*如果输入的字符串的首位字符是'y'，则重新进入while循环*/

/*如果接收到的指令的首位字符是'd'，则显示每个程序的信息*/
/*打印message13字符串*/
/*打印message12字符串*/
/*显示命令提示符，并接收输入的指令*/
/*如果输入的字符串的首位字符是'y'*/
/*进入for循环，将批处理功能的指令存到string中*/
/*字符串(批处理指令)转移*/
/*如果到达字符串尾部*/
/*记录下字符串的长度*/
/*跳出for循环*/

/*调用batch函数，实现批处理功能*/
/*打印message12字符串*/
/*显示命令提示符，并接收输入的指令*/
/*如果输入的字符串的首位字符是'y'，继续运行程序*/
/*进入for循环，将批处理功能的指令存到string中*/
/*字符串(伪分时指令)转移*/
/*如果到达字符串尾部*/
/*记录下字符串的长度*/
/*跳出for循环*/

/*调用time函数，实现批处理功能*/
/*打印message12字符串*/
/*显示命令提示符，并接收输入的指令*/

/*打印message_error字符串*/
/*打印message12字符串*/
/*显示命令提示符，并接收输入的指令*/
/*如果输入的字符串的首位字符是'y'，继续运行程序*/

```



以上为止，c 函数的编写就结束了，每一条语句的具体功能都在代码上有注释（如上）。

在这一个 c 程序中实现的其实是等于前一次实验的引导扇区的功能，输出提示字符，然后提供输入命令的功能和显示信息的功能。

完成 c 文件后，就要着手去写和它相呼应的汇编文件了。

首先还是先要用 extrn 标识符去定义一些在 c 文件里面的变量，作用是与 c 文件里面一样的，都是关联两个文件，使这些变量可以互用。要注意，在这里定义变量的时候要在变量名前加一个下划线 “\_”，因为 c 文件编译成 obj 文件的时候，变量名前都会加上 “\_”（具体原因我不清楚），因此要加上才能关联上对应的变量。

```
1 ;汇编模块
2 extrn _main:near ;使用extrn调用c模块中的main函数
3 extrn _cal_pos:near ;使用extrn调用c模块中的cal_pos函数
4 extrn _pos:near ;使用extrn调用c模块中的pos变量
5 extrn _ch:near ;使用extrn调用c模块中的ch变量
6 extrn _x:near ;使用extrn调用c模块中的x变量
7 extrn _y:near ;使用extrn调用c模块中的y变量
8 extrn _offset_prog:near ;使用extrn调用c模块中的offset_prog变量
9 extrn _data1:near ;使用extrn调用c模块中的data变量
```

设置好段定义：

```
11 _TEXT segment byte public 'CODE' ;段定义伪操作
12 DGROUP group _TEXT, _DATA, _BSS ;数据段的设置
13 assume cs:_TEXT ;明确段寄存器与段的关系(cs寄存器存放_TEXT段)
14 org 100h ;起始偏移地址为100h
```

开始程序，把程序放置到合适的地址：

```
16 start: ;
17 mov ax,cs ;把cs寄存器的值(代码段地址)存到ax寄存器中
18 add ax,800h ;此处为cs+800h，先将800h左移了四位再加上偏移地址得到物理地址
19 mov ds,ax ;把ax寄存器的值存到ds寄存器中
20 mov es,ax ;把ax寄存器的值存到es寄存器中
21 mov ss,ax ;把ax寄存器的值存到ss寄存器中(栈的段地址)
22 mov sp,100h ;设置sp寄存器的值为100h(栈的长度)
23 call near ptr _main ;调用c模块中的主函数
24 jmp $ ;无限循环
```

下面是 C 语言中关联的各个函数的定义：

```
26 public _setcursor          ;定义设置光标的函数
27 _setcursor proc           ;子程序定义伪指令
28     push ax                ;此处分别将ax, bx, dx按顺序压栈, 用于保护数据
29     push bx                ;由于函数中的中断操作将会用到这三个寄存器
30     push dx                ;因此先将这几个寄存器里的原始数据压栈保护
31     mov ah,02h             ;功能号设置为02h, 光标定位
32     mov dh,byte ptr [_x]   ;设置dh为x变量的值, 表示起始的行数
33     mov dl,byte ptr [_y]   ;设置dl为y变量的值, 表示起始的列数
34     mov bh,0               ;bh设置为0, 表示第0页
35     int 10h                ;设置中断号为10h, 进行中断
36     pop dx                 ;中断调用完成后恢复各个寄存器中的原始值
37     pop bx                 ;因此按顺序弹出dx, bx, ax的原始值
38     pop ax                 ;分别放回原处
39     ret                    ;返回
40 _setcursor endp            ;子函数的定义结束
```

```
42 public _printchar         ;定义打印一个字符的函数
43 _printchar proc           ;子程序定义伪指令
44     push ax                ;此处分别将ax, es, bp, bx按顺序压栈, 用于保护数据
45     push es                ;由于函数中的中断操作将会用到这四个寄存器
46     push bp                ;因此先将这几个寄存器里的原始数据压栈保护
47     push bx                ;
48     call _setcursor        ;调用c模块中的setcursor函数, 放置光标
49     mov bp,sp              ;把栈顶位置寄存器赋值给bp寄存器
50     mov ax,0b800h          ;把显存起始位置存到ax寄存器中
51     mov es,ax              ;把ax寄存器的值存到段地址
52     mov al,byte ptr [bp+10] ;把传入函数的数据存到al寄存器中,表示要输出的字符
53     mov ah,0fh             ;把0fh存进ah寄存器中,表示要输出的字符的样式为白字黑底
54     mov bx,word ptr [_pos]  ;把pos变量存进bx寄存器中,代表输出字符的坐标
55     mov word ptr es:[bx],ax ;在屏幕上打印该字符
56     inc word ptr [_y]       ;y变量加1(横向坐标加1)
57     call near ptr _cal_pos  ;调用cal_pos函数,用于计算新的坐标
58     pop bx                 ;中断调用完成后恢复各个寄存器中的原始值
59     pop bp                 ;因此按顺序弹出bx, bp, es, ax的原始值
60     pop es                 ;分别放回原处
61     pop ax                 ;
62     ret                    ;返回
63 _printchar endp            ;子函数的定义结束
```

```
102 public _inputchar        ;定义输入字符的函数
103 _inputchar proc           ;子程序定义伪指令
104     push ax                ;把ax压栈(保护ax里的数据)
105     call _setcursor        ;调用c模块中的setcursor函数, 放置光标
106     mov ax,0               ;把ax寄存器里的值置0(功能号, ah=0h: 入口 输入的字符返回到al寄存器)
107     int 16h                ;设置中断号为16h, 进行中断
108     mov byte ptr [_ch],al  ;把al寄存器的值(输入的字符)存到ch变量里
109     pop ax                 ;中断调用完成后恢复各个寄存器中的原始值
110     ret                    ;返回
111 _inputchar endp            ;子函数的定义结束
```



65	public _printstring	;定义打印一个字符串的函数
66	_printstring proc	;子程序定义伪指令
67	push bp	;此处分别将bp, es, ax按顺序压栈, 用于保护数据
68	push es	;由于函数中的中断操作将会用到这三个寄存器
69	push ax	;因此先将这几个寄存器里的原始数据压栈保护
70	mov bp,sp	;把栈顶位置寄存器赋值给bp寄存器
71	mov ax,0b800h	;把显存起始位置存到ax寄存器中
72	mov es,ax	;把ax寄存器的值存到段地址
73	mov si,word ptr [bp+8]	;把传入函数的数据存到si寄存器中,表示要输出的字符串
74	mov di,word ptr [_pos]	;把pos变量存进di寄存器中,代表输出字符串的起始坐标
75	.1:	;第一种情况: 该字符非空字符或换行符
76	mov al,byte ptr [si]	;将si代表的字符串上的一位字符存到al寄存器中
77	inc si	;si自加1,表示读入下一个字符
78	test al,al	;将两个操作数作逻辑与运算,检验al中存放的是否为空字符,并将结果存到zf中
79	jz .3	;如果zf寄存器的内容是1,则跳转到cond3
80	cmp al,0ah	;比较al寄存器中存的内容是否为换行符,并将结果存到zf中
81	jz .2	;如果zf寄存器的内容是1,则跳转到cond2
82	mov ah,0fh	;把0fh存进ah寄存器中,表示要输出的字符的样式为白字黑底
83	mov word ptr es:[di],ax	;在屏幕上打印该字符
84	inc byte ptr [_y]	;y变量加1(纵向坐标加1)
85	call near ptr _cal_pos	;调用cal_pos函数,用于计算新的坐标
86	mov di,word ptr [_pos]	;把新的坐标(pos)存进di寄存器中
87	jmp .1	;跳转到cond1
88	.2:	;第二种情况: 读取到的是换行符
89	inc word ptr [_x]	;x变量加1(纵向坐标加1)
90	mov word ptr [_y],0	;y变量变为0(由于换行,横向坐标为0)
91	call near ptr _cal_pos	;调用cal_pos函数,用于计算新的坐标
92	mov di,word ptr [_pos]	;把新的坐标(pos)存进di寄存器中
93	jmp .1	;跳转到cond1
94	.3:	;第三种情况: 读取到的是空字符(结束输出)
95	call _setcursor	;调用c模块中的setcursor函数,放置光标
96	pop ax	;中断调用完成后恢复各个寄存器中的原始值
97	pop es	;因此按顺序弹出ax, es, bp的原始值
98	pop bp	;分别放回原处
99	ret	;返回
100	_printstring endp	;子函数的定义结束

113	public _clear	;定义输入字符的函数
114	_clear proc	;子程序定义伪指令
115	push ax	;此处分别将ax, bx, cx, dx按顺序压栈, 用于保护数据
116	push bx	;由于函数中的中断操作将会用到这四个寄存器
117	push cx	;因此先将这几个寄存器里的原始数据压栈保护
118	push dx	;
119	mov ax,0600h	;把0600h存进ax寄存器(ah=06h: 表示窗口上卷功能, al=00h: 卷动整个窗口)
120	mov bx,0700h	;把0700h存进bx寄存器(bh=07h: 白字 bl=00h: 黑底)
121	mov cx,0	;把0存进cx寄存器(ch=0h: 起始纵坐标为0 cl=0h: 起始横坐标为0)
122	mov dx,184fh	;把184fh存进寄存器(dh=18h: 终止纵坐标为24 dl=4fh: 终止横坐标为79)
123	int 10h	;设置中断号为10h, 进行中断
124	mov word ptr [_x],0	;把x的值初始化为0
125	mov word ptr [_y],0	;把y的值初始化为0
126	mov word ptr [_pos],0	;把pos的值初始化为0
127	call _setcursor	;调用c模块中的setcursor函数,放置光标
128	pop dx	;中断调用完成后恢复各个寄存器中的原始值
129	pop cx	;因此按顺序弹出dx, cx, bx, ax的原始值
130	pop bx	;分别放回原处
131	pop ax	;
132	ret	;返回
133	_clear endp	;子函数的定义结束



135	public _load_prog	;定义加载程序的函数
136	_load_prog proc	;子程序定义伪指令
137	push ax	;此处分别将ax, bx, cx, dx, es, bp按顺序压栈, 用于保护数据
138	push bx	;由于函数中的中断操作将会用到这六个寄存器
139	push cx	;因此先将这几个寄存器里的原始数据压栈保护
140	push dx	;
141	push es	;
142	push bp	;
143	mov bp,sp	;把栈顶位置寄存器赋值给bp寄存器
144	mov ax,cs	;把cs的值(代码段的段地址)放进ax寄存器中
145	mov es,ax	;把ax寄存器的值存到段地址(设置段地址)
146	mov bx,word ptr [bp+12+2]	;把传入函数的第三个数据存到bx寄存器中, 表示偏移地址
147	mov ah,2	;功能号为2, 表示读磁盘
148	mov al,byte ptr [bp+12+4]	;把传入函数的第二个数据存到al寄存器中, 表示扇区数
149	mov dl,0	;驱动器号
150	mov dh,0	;磁头号
151	mov ch,0	;柱面号
152	mov cl,byte ptr [bp+12+6]	;把传入函数的第一个数据存到cl寄存器中, 表示起始扇区号
153	int 13h	;设置中断号为13h, 进行中断
154	pop bp	;中断调用完成后恢复各个寄存器中的原始值
155	pop es	;因此按顺序弹出bp, es, dx, cx, bx, ax的原始值
156	pop dx	;分别放回原处
157	pop cx	;
158	pop bx	;
159	pop ax	;
160	ret	;返回
161	_load_prog endp	;子函数的定义结束

163	public _run_prog	;定义运行程序的函数
164	_run_prog proc	;子程序定义伪指令
165	push ax	;此处分别将ax, bx, cx, dx, es, ds按顺序压栈, 用于保护数据
166	push bx	;由于函数中的中断操作将会用到这六个寄存器
167	push cx	;因此先将这几个寄存器里的原始数据压栈保护
168	push dx	;
169	push es	;
170	push ds	;
171	call word ptr [_offset_prog]	;跳至要运行的程序的地址并运行程序
172	pop ds	;中断调用完成后恢复各个寄存器中的原始值
173	pop es	;因此按顺序弹出ds, es, dx, cx, bx, ax的原始值
174	pop dx	;分别放回原处
175	pop cx	;
176	pop bx	;
177	pop ax	;
178	ret	;返回
179	_run_prog endp	;子函数的定义结束

181	public _filldata	;定义填充数据的函数
182	_filldata proc	;子程序定义伪指令
183	push ax	;此处分别将ax, bx, cx, dx, es, ds按顺序压栈, 用于保护数据
184	push bx	;由于函数中的中断操作将会用到这六个寄存器
185	push cx	;因此先将这几个寄存器里的原始数据压栈保护
186	push dx	;
187	push es	;
188	push ds	;
189	mov bp,sp	;把栈顶位置寄存器赋值给bp寄存器
190	mov ax,0	;把ax置为0
191	mov es,ax	;设置段地址
192	mov bx,word ptr [bp+12+2]	;把传入函数的第二个数据存到bx寄存器中, 表示偏移地址
193	mov ax,word ptr [bp+12+4]	;把传入函数的第一个数据存到ax寄存器中, 表示数据
194	mov word ptr es:[bx],ax	;把ax的值存到该地址中
195	pop ds	;中断调用完成后恢复各个寄存器中的原始值
196	pop es	;因此按顺序弹出ds, es, dx, cx, bx, ax的原始值
197	pop dx	;分别放回原处
198	pop cx	;
199	pop bx	;
200	pop ax	;
201	ret	;返回
202	_filldata endp	;子函数的定义结束

204	public _readdata	;定义填充数据的函数
205	_readdata proc	;子程序定义伪指令
206	push ax	;此处分别将ax, bx, cx, dx, es, ds按顺序压栈, 用于保护数据
207	push bx	;由于函数中的中断操作将会用到这六个寄存器
208	push cx	;因此先将这几个寄存器里的原始数据压栈保护
209	push dx	;
210	push es	;
211	push ds	;
212	mov bp,sp	;把栈顶位置寄存器赋值给bp寄存器
213	mov ax,0	;把ax置为0
214	mov es,ax	;设置段地址
215	mov bx,word ptr [bp+12+2]	;把传入函数的第一个数据存到bx寄存器中, 表示偏移地址
216	mov ax,word ptr es:[bx]	;把该地址中的数据存到ax寄存器中
217	mov word ptr [_data1],ax	;把ax寄存器的值存到data变量中
218	pop ds	;中断调用完成后恢复各个寄存器中的原始值
219	pop es	;因此按顺序弹出ds, es, dx, cx, bx, ax的原始值
220	pop dx	;分别放回原处
221	pop cx	;
222	pop bx	;
223	pop ax	;
224	ret	;返回
225	_readdata endp	;子函数的定义结束

以上就是所有函数的定义了, 定义完成后结束数据段:

227	_TEXT ends	;代码段结束
228		
229	_DATA segment word public 'DATA'	;_DATA段开始
230	_DATA ends	;_DATA段结束
231		
232	_BSS segment word public 'BSS'	;_BSS段开始
233	_BSS ends	;_BSS结束
234		
235		
236	end start	;start结束



至此，与 c 文件关联的汇编文件也完成了，这两个文件编译并关联后就是这次实验的“内核”了。

然后就要编写引导程序了，这一次的引导程序只需要把“内核”放在正确的内存地址中就可以了，还要把四个用户程序的 bin 文件和内核 com 文件包入其中：

```
1  org 7c00h                ; BIOS将把引导扇区加载到0:7C00h处，并开始执行
2  OffSetOfKernal equ 8100h  ;把内核的地址设为8100h
3  Start:                   ;读软盘或硬盘上的若干物理扇区到内存的ES:BX处:
4      mov ax,cs             ;段地址；存放数据的内存基地址
5      mov es,ax             ;设置段地址（不能直接mov es,段地址）
6      mov bx,OffsetOfKernal ;偏移地址；存放数据的内存偏移地址
7      mov ah,2              ;功能号
8      mov al,8              ;扇区数
9      mov dl,0              ;驱动器号；软盘为0，硬盘和U盘为80H
10     mov dh,0              ;磁头号；起始编号为0
11     mov ch,0              ;柱面号；起始编号为0
12     mov cl,10             ;起始扇区号；起始编号为1
13     int 13H               ;调用读磁盘BIOS的13h功能
14     jmp OffSetOfKernal    ;跳转至内核的地址，运行程序
15 AfterRun:                 ;
16     jmp $                  ;无限循环
17
18     times 510-($-$$) db 0  ;512字节的设置
19     db 0x55,0xaa           ;
20     incbin 'prog1.bin'      ;导入的几个程序
21     incbin 'prog2.bin'
22     incbin 'prog3.bin'
23     incbin 'prog4.bin'
24     incbin 'os1.com'
```

至此，所有的程序就已经完成了，真是一个很痛苦的过程！

如果要想实现批处理和分时功能，用户程序上也要动一些手脚，首先是程序存放的地址，由于要分时运行，所以要分配不同的内存地址。

<pre>1 ;prog1 2 START: 3     mov ax,0910h          ;决定程序内存地址的起始为0910h 4     mov ds,ax             ;把7c0h存进数据段寄存器ds中，指向起始位置 5     mov ax,0b800h         ;把显存的起始位置0b800h存进ax寄存器 6     mov es,ax             ;把0b800h存进附加段寄存器es中，指向显存起始位置 7     cmp byte[run],1       ;判断run变量是否是1，如果是就把zf变量变为1 8     jz BEGIN              ;如果zf变量为1，就跳转到BEGIN 9     mov word[x],-1        ;要显示的字符的起始纵坐标 10    mov word[y],-1        ;要显示的字符的起始横坐标 11    mov byte[dir],d_r      ;决定方向弹射方向的变量 12    mov byte[cnt],2        ;计数变量 13    mov byte[color],1fh    ;决定颜色的变量 14    mov byte[count],0      ; 15    mov byte[run],1        ;把run变量赋值为1</pre>	<pre>1 ;prog2 2 START: 3     mov ax,0950h          ;决定程序内存地址的起始为0950h 4     mov ds,ax             ;把7c0h存进数据段寄存器ds中，指向起始位置 5     mov ax,0b800h         ;把显存的起始位置0b800h存进ax寄存器 6     mov es,ax             ;把0b800h存进附加段寄存器es中，指向显存起始位置 7     cmp byte[run],1       ;判断run变量是否是1，如果是就把zf变量变为1 8     jz BEGIN              ;如果zf变量为1，就跳转到BEGIN 9     mov word[x],-1        ;要显示的字符的起始纵坐标 10    mov word[y],79         ;要显示的字符的起始横坐标 11    mov byte[dir],d_r      ;决定方向弹射方向的变量 12    mov byte[cnt],2        ;计数变量 13    mov byte[color],1fh    ;决定颜色的变量 14    mov byte[count],0      ; 15    mov byte[run],1        ;把run变量赋值为1</pre>
---	--



1	;prog3			
2	START:			
3	mov ax,0990h	;决定程序内存地址的起始为0990h		
4	mov ds,ax	;把7c0h存进数据段寄存器ds中,指向起始位置		
5	mov ax,0b800h	;把显存的起始位置0b800h存进ax寄存器		
6	mov es,ax	;把0b800h存进附加段寄存器es中,指向显存起始位置		
7	cmp byte[run],1	;判断run变量是否是1,如果是就把zf变量变为1		
8	jz BEGIN	;如果zf变量为1,就跳转到BEGIN		
9	mov word[x],24	;要显示的字符的起始纵坐标		
10	mov word[y],-1	;要显示的字符的起始横坐标		
11	mov byte[dir],d_r	;决定方向弹射方向的变量	1	;prog4
12	mov byte[cnt],2	;计数变量	2	START:
13	mov byte[color],1fh	;决定颜色的变量	3	mov ax,09d0h
14	mov byte[count],0		4	mov ds,ax
15	mov byte[run],1	;把run变量赋值为1	5	mov ax,0b800h
			6	mov es,ax
			7	cmp byte[run],1
			8	jz BEGIN
			9	mov word[x],24
			10	mov word[y],79
			11	mov byte[dir],d_r
			12	mov byte[cnt],2
			13	mov byte[color],1fh
			14	mov byte[count],0
			15	mov byte[run],1
				;把run变量赋值为1

还有判断是否分时运行的代码段：

```

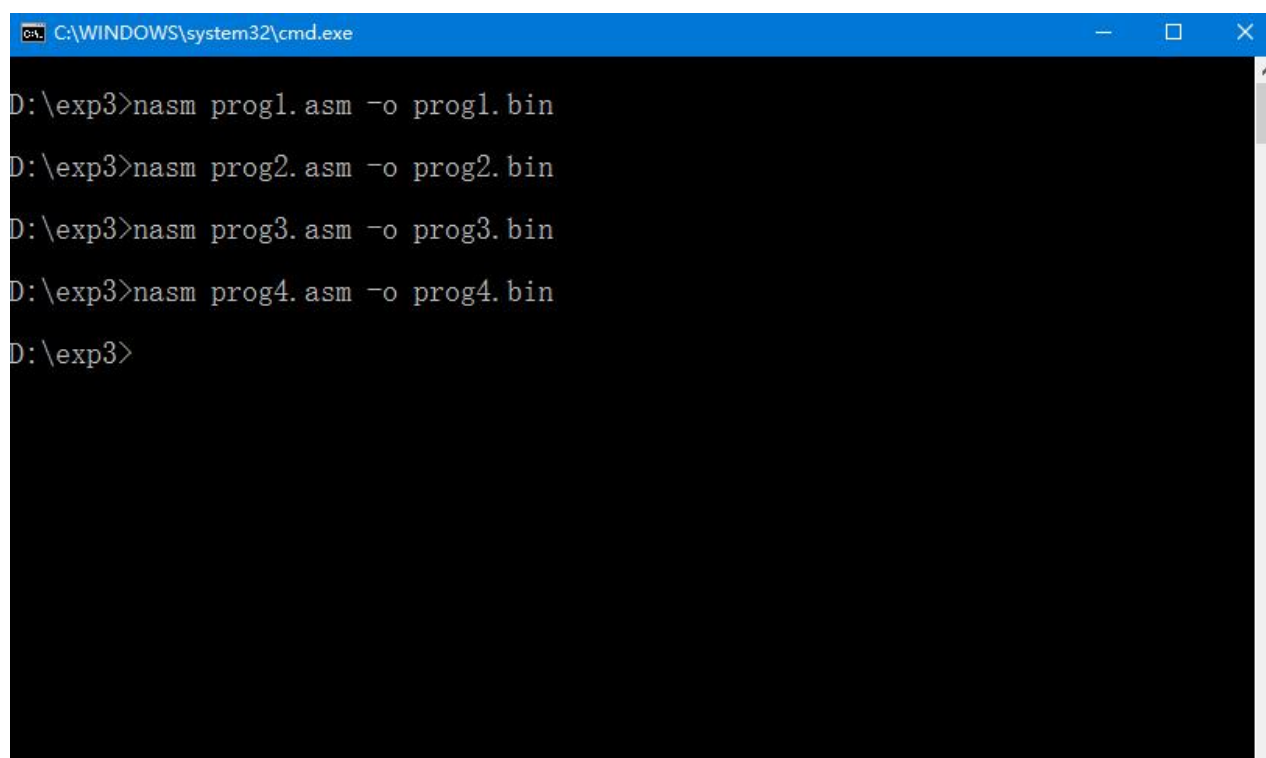
16 BEGIN:
17     mov ax,0100h      ;把0100h存进ax寄存器中(ah=01h:功能号 al=00h:ASCII码为0,键盘无输入则为0)
18     int 16h          ;中断,中断号为16h,功能号为01h,即检查键盘缓存区是否有数据
19     ;如果有数据,把数据存进al寄存器中,且zf等于1,否则为0
20     jnz READKEY      ;检查zf是否为0,如果非0,则跳转到READKEY段代码,否则不进行操作
21     jmp NOTREAD      ;跳转到BEGIN段代码
22 READKEY:
23     mov ax,0          ;把00h存进ax寄存器中(ah=00h:功能号 al=00h:ASCII码为0,键盘无输入则为0)
24     int 16h          ;中断,中断号为16h,功能号为00h,即读入键盘的输入,存进al寄存器中
25     cmp al,'1'        ;比较al寄存器中的值是否与'1'相等,如果相等就把zf置为1,否则置为0
26     jz RETURN        ;如果zf为1,则跳转至RETURN段代码
27 NOTREAD:
28     inc byte[count]   ;count变量加1
29     cmp byte[count],0afh ;把count与0afh作比较,如果相等,则把zf置为1,否则为0(程序运行时间)
30     jz RETURN        ;检查zf是否为0,如果非0,则跳转到READKEY段代码,否则不进行操作
31     jmp SEC          ;跳转到SEC段
32 RETURN:
33     mov ax,0          ;把00h存进ax寄存器中
34     mov es,ax         ;把ax存进es寄存器中(初始地址为0)
35     mov word[es:600h],0 ;把0存到600h的地址中
36     mov byte[run],0   ;把0存进run变量
37     ret              ;返回

```

其中 count 变量用作计时，程序运行一定时间就会自动停止。

至此，代码部分的内容全部结束，接下来我们就进入到编译过程的部分：

代码都放在 d 盘的 exp3 文件夹，首先把四个用户程序使用 nasm 编译成 bin 文件：

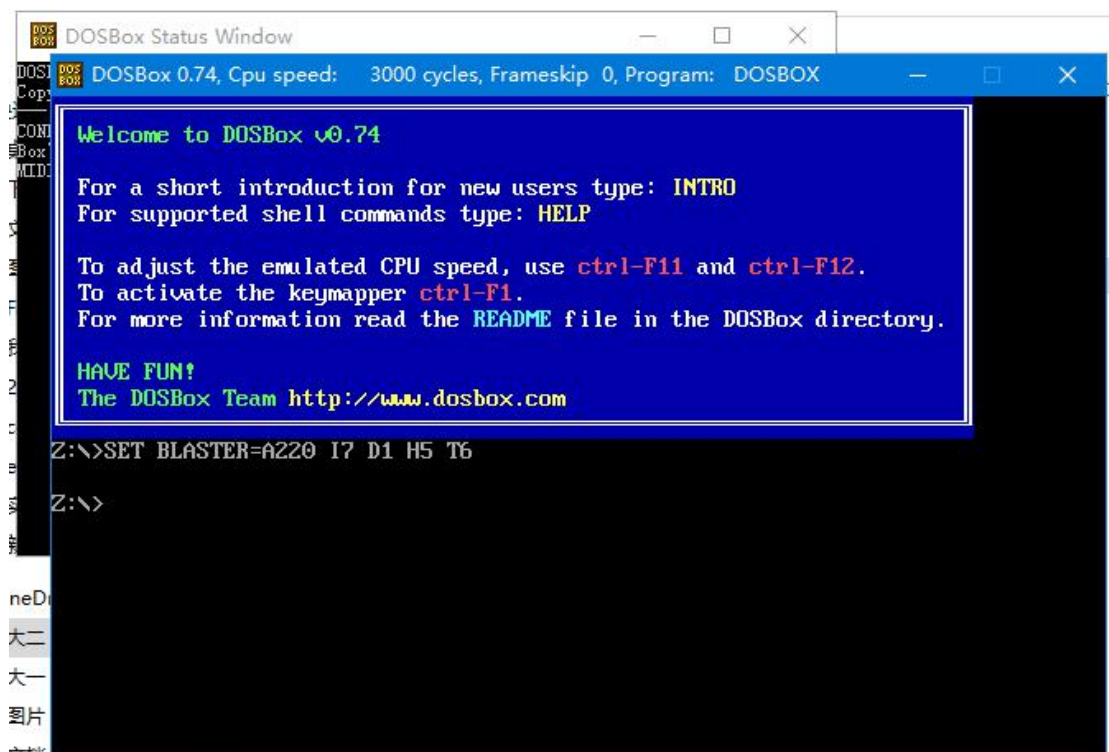


```
C:\WINDOWS\system32\cmd.exe

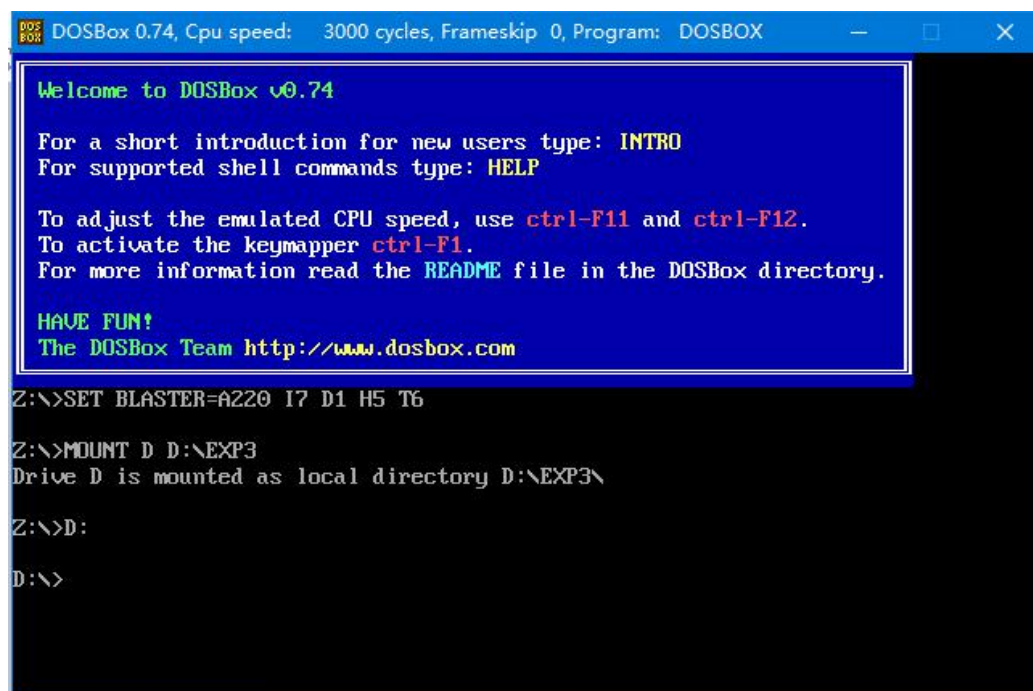
D:\exp3>nasm prog1.asm -o prog1.bin
D:\exp3>nasm prog2.asm -o prog2.bin
D:\exp3>nasm prog3.asm -o prog3.bin
D:\exp3>nasm prog4.asm -o prog4.bin
D:\exp3>
```

编译完成！

接下来就要使用 TCC 和 TASM 分别去编译 c 文件和相关联的汇编文件了，在使用 TLINK 去连接两个 OBJ 文件。但是我发现在我的 64 位系统里面不能够运行 TCC 和 TASM 和 TLINK 编译器，因此我们使用一个叫做 DOSBOX 的工具去模拟 16 位的计算机环境，从而进行编译。



首先把我的放置编译器和代码的文件地址导入到 DOSBOX 的虚拟盘中，再把 DOSBOX 的当前地址切换到该地址：



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

Welcome to DOSBox v0.74

For a short introduction for new users type: INTRO
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com

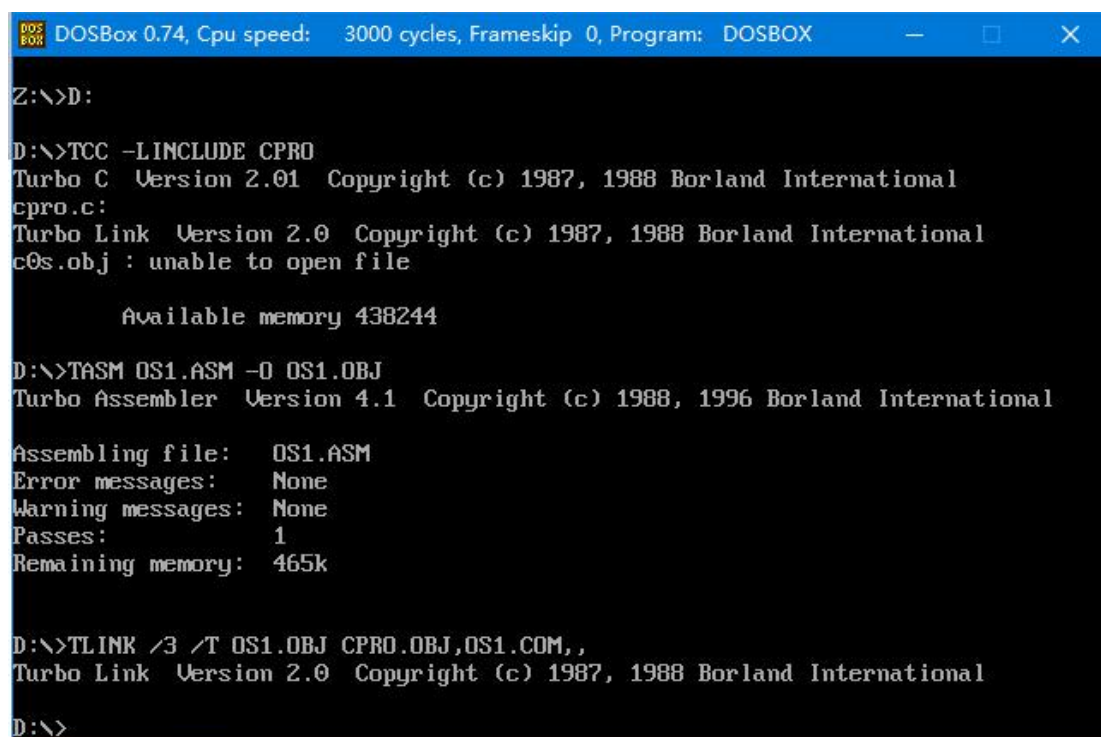
Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>MOUNT D D:\EXP3
Drive D is mounted as local directory D:\EXP3\

Z:\>D:

D:\>
```

然后就可以进行编译了：



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

Z:\>D:

D:\>TCC -LINCLUDE CPRO
Turbo C Version 2.01 Copyright (c) 1987, 1988 Borland International
cpro.c:
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International
c0s.obj : unable to open file

        Available memory 438244

D:\>TASM OS1.ASM -O OS1.OBJ
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International

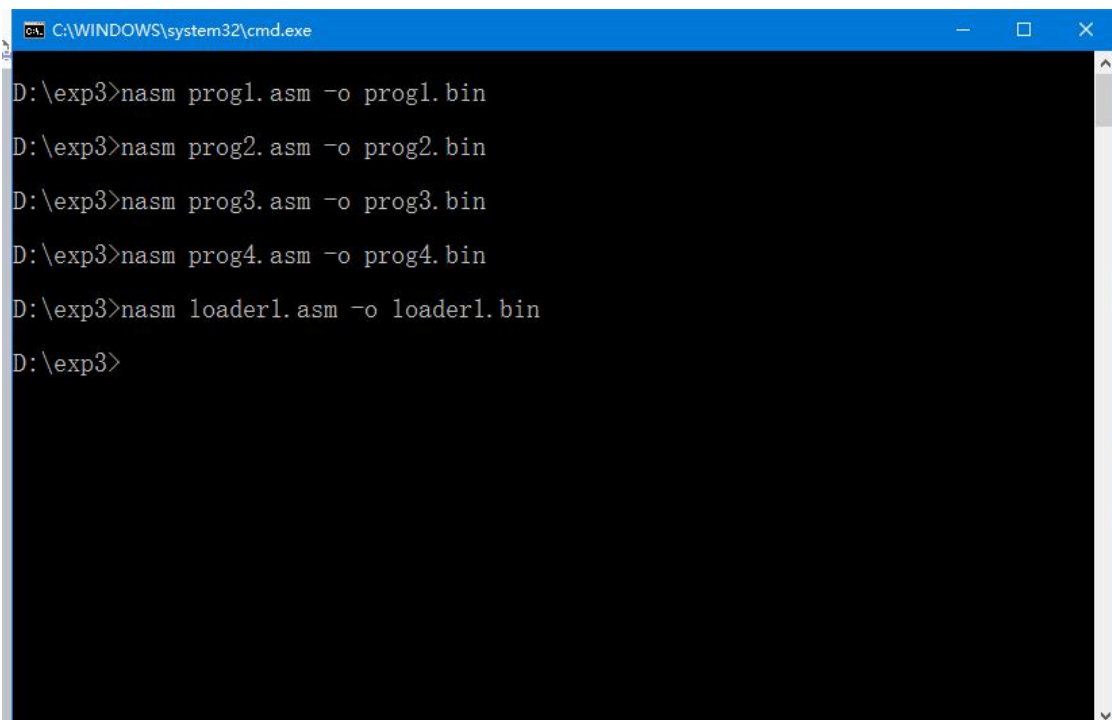
Assembling file:  OS1.ASM
Error messages:   None
Warning messages: None
Passes:          1
Remaining memory: 465k

D:\>TLINK /3 /T OS1.OBJ CPRO.OBJ,OS1.COM,,
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International

D:\>
```



从以上图片我们可以看到，c 文件和汇编文件都编译出 OBJ 文件了，并且两个 OBJ 文件都连接起来了，生成了“内核” OS1.COM 文件。然后再使用 nasm 编译 loader1.asm 文件，最后一步完成！



```
C:\WINDOWS\system32\cmd.exe
D:\exp3>nasm prog1.asm -o prog1.bin
D:\exp3>nasm prog2.asm -o prog2.bin
D:\exp3>nasm prog3.asm -o prog3.bin
D:\exp3>nasm prog4.asm -o prog4.bin
D:\exp3>nasm loader1.asm -o loader1.bin
D:\exp3>
```

得到的 loader1.bin 就是最后的文件了！把里面的内容复制到 1.44mb 的虚拟软盘中，链接到虚拟机上，就可以运行了 !!!

下面为运行结果的实现：

初始引导界面：

```
Welcome to our team's system!
You can input some legal instructions to active the functions
  If you want to run the program by Batching,input 'batch'
    For example: batch 1 2 3 4
  If you want to run the program by Time-Sharing,input 'time'
    For example: time 1 2 3 4
  If you want to know the information of the program,input 'show'
    For example: show
  If you want to run the default instructions,input 'default'
    For example: default

Orz >_
```

输入 show 命令，显示用户程序的属性内容：

```
Welcome to our team's system!
You can input some legal instructions to active the functions
  If you want to run the program by Batching,input 'batch'
    For example: batch 1 2 3 4
  If you want to run the program by Time-Sharing,input 'time'
    For example: time 1 2 3 4
  If you want to know the information of the program,input 'show'
    For example: show
  If you want to run the default instructions,input 'default'
    For example: default

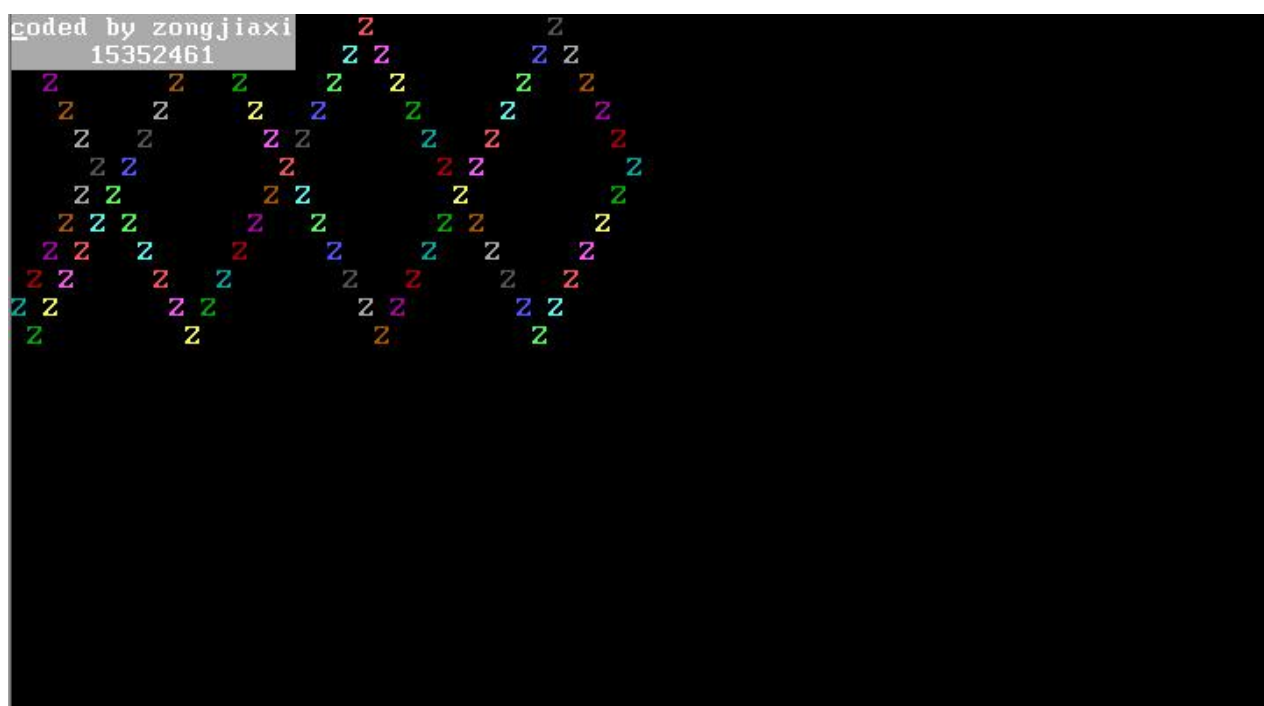
Orz >show
User Program1:
  Name:program_LU   Size:1024byte   Position:part LU
User Program2:
  Name:program_RU   Size:1024byte   Position:part RU
User Program3:
  Name:program_LD   Size:1024byte   Position:part LD
User Program4:
  Name:program_RD   Size:1024byte   Position:part RD
If you want to continue,input 'yes',otherwise input 'no'
Orz >_
```

输入批处理指令（运行 1，4 程序）：

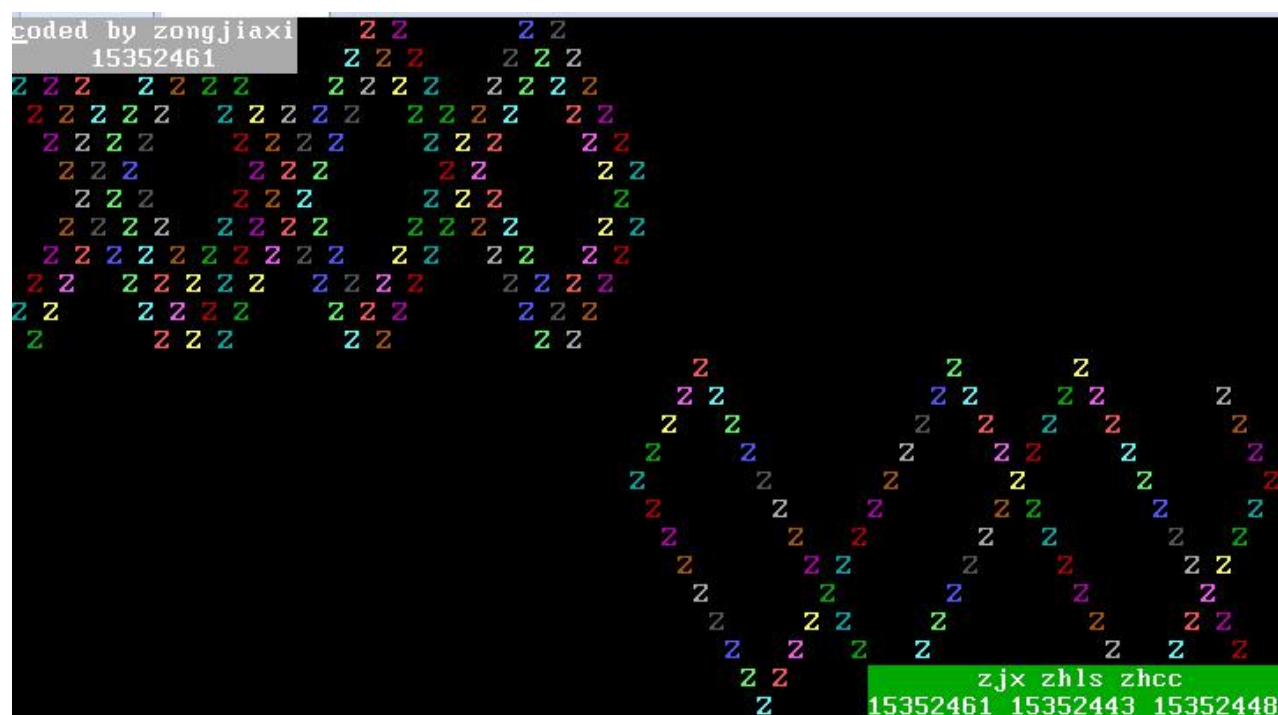
```
Welcome to our team's system!
You can input some legal instructions to active the functions
  If you want to run the program by Batching,input 'batch'
    For example: batch 1 2 3 4
  If you want to run the program by Time-Sharing,input 'time'
    For example: time 1 2 3 4
  If you want to know the information of the program,input 'show'
    For example: show
  If you want to run the default instructions,input 'default'
    For example: default

Orz >batch 1 4_
```

刚开始运行时：



运行第二个程序：





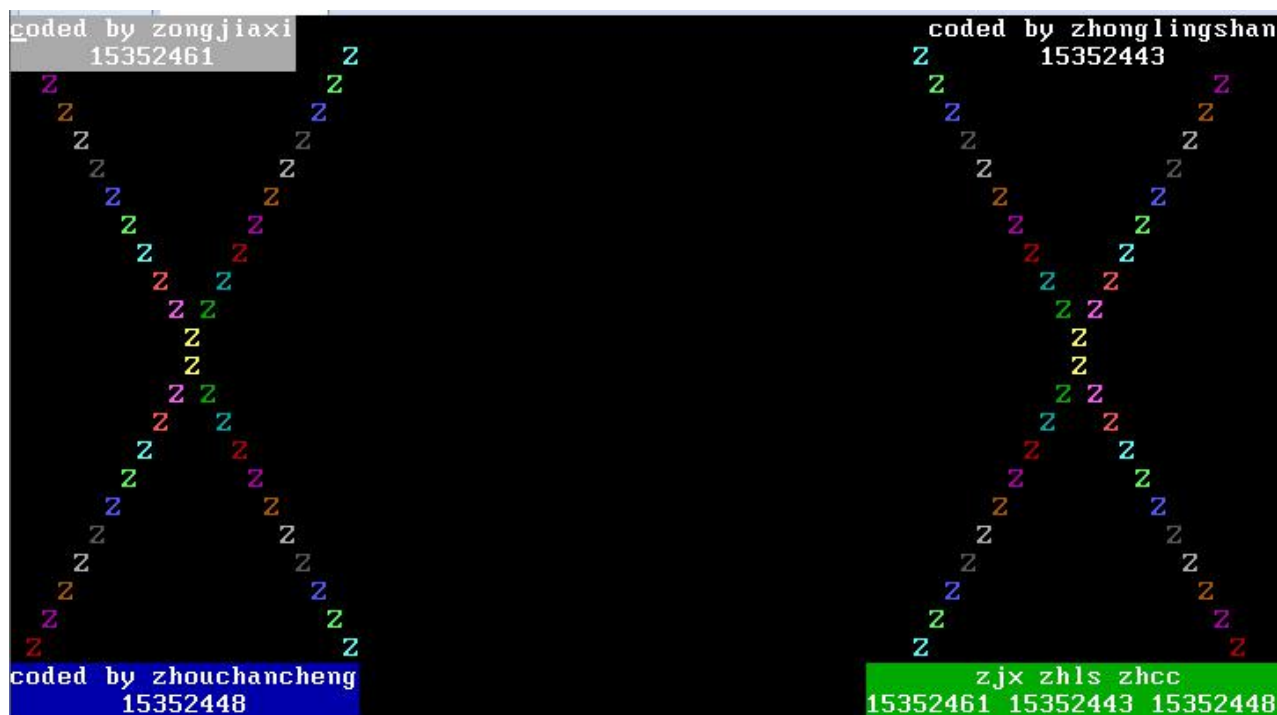
运行完毕，输出提示是否继续运行（其实输入 yes 或 no 都一样，返回主界面）：

[illegible]

输入分时指令（运行 1, 2, 3, 4 程序）:

```
Welcome to our team's system!  
You can input some legal instructions to active the functions  
  If you want to run the program by Batching,input 'batch'  
    For example: batch 1 2 3 4  
  If you want to run the program by Time-Sharing,input 'time'  
    For example: time 1 2 3 4  
  If you want to know the information of the program,input 'show'  
    For example: show  
  If you want to run the default instructions,input 'default'  
    For example: default  
  
Orz >time 1 2 3 4_
```

刚开始运行时：



运行结束：



如果输入的命令有误，输出错误提示，弹出重新输入的命令：

```
Welcome to our team's system!
You can input some legal instructions to active the functions
  If you want to run the program by Batching,input 'batch'
  For example: batch 1 2 3 4
  If you want to run the program by Time-Sharing,input 'time'
  For example: time 1 2 3 4
  If you want to know the information of the program,input 'show'
  For example: show
  If you want to run the default instructions,input 'default'
  For example: default

Orz >ns
Input Error!!!
If you want to continue,input 'yes',otherwise input 'no'
Orz >_
```

输入 default 命令，先运行批处理，后运行分时：

```
Welcome to our team's system!
You can input some legal instructions to active the functions
  If you want to run the program by Batching,input 'batch'
  For example: batch 1 2 3 4
  If you want to run the program by Time-Sharing,input 'time'
  For example: time 1 2 3 4
  If you want to know the information of the program,input 'show'
  For example: show
  If you want to run the default instructions,input 'default'
  For example: default

Orz >default
OS will run instructions 'batch 1 2 3 4' and 'time 1 2 3 4'
If you want to continue,input 'yes',otherwise input 'no'
Orz >_
```



coded by zongjiaxi 15352461 coded by zhonglingshar 15352443

coded by zhouchancheng 15352448

coded by zongjiayi  
15352461

Z coded by zhonglingshan  
15352443

coded by zhouchancheng  
15352448

zjx zhls zhcc  
15352461 15352443 15352448

- 24 -

## 【实验总结】

宗嘉希 15352461： 这一次的实验让我学习到了很多东西，也解决了我上一节课的一个疑问：既然引导扇区放置引导程序，而引导扇区的大小却只有 512 字节，那么如果我想在引导扇区里面实现更多的功能那该要怎么办？引导程序输出几个字符串就差不多写满了，那要怎么样去实现更多的功能呢？这一次实验给出的答案就是把引导程序写成一个“内核”，再写一个引导程序把这个内核导入到内存中运行就可以了。那么这个内核必然是要实现很多的功能的，但是学习了汇编之后，都知道要是用汇编去实现这么多功能是有难度的。因此我们就可以使用 c 语言和汇编交叉调用的方式去写这一个“内核”，这跟用纯汇编语言编写程序比起来要简单一点。

为什么要用两种语言呢？老师在课堂上已经解释了，首先，使用汇编语言是必须的，因为它有几个必备功能：①设置自身运行模式和环境，需要设置硬件寄存器，②设置 I/O 端口实现 I/O 操作，③初始化中断向量和实现中断处理，④实现控制原语。其次，使用 c 语言也是有它的好处的：①适合构造复杂的数据结构和相关数据结构的的管理，②实现复杂的功能或算法。

因此，函数的声明和调用就要在实现复杂功能的 c 语言模块里面，而这些函数的实现就要在汇编语言模块里面了，因为要使用中断，I/O 操作等等。

根据以上这一些原理，还有对师兄的代码的参考，我们编写出了一个像师兄一样的除了能实现实验的基本要求功能，还能实现批处理与分时功能的程序。这一个过程真的是很辛苦，因为对代码的不熟悉和粗心大意，导致要花很多时间去学习，后期调试的时候还出现了很多的错误要去排除，比如说输出了乱码，才发现自己的函数的调用出现了错误，导致输出的字符错误，还有段地址设置错误等问题导致用户程序不能够正确地运行。在输入命令的时候也出现了一些小问题，不过并不影响整体程序的正确运行。

总的来说，这一次的实验虽然挺难的，知识点也挺多的，但是学习到的东西也很多。原来 c 语言和汇编交叉编译的功能真的挺大的，可惜我的时间与能力不足，不能写出更多的功能。希望能在以后继续努力吧！

钟凌山 15352443：这次的实验要求我们用汇编和 C 编写一个引导程序，引导操作系统内核运行。操作系统要求能够根据输入的指令来调用不同的用户程序来运行。

首次运行的时候就遇到了不小的问题。运行时屏幕上出现了几行乱码，进行输入后乱码消失，但是并没有进入系统。后来仔细检查所有代码后发现是因为段定义处少了一句（DGROUP）。

本次实验需要更深入地了解磁盘扇区的概念，以及熟悉读取用户程序的方法。

周禅城 15352448：不得不说，相比上一次实验，这次实验的难度真可谓是飞跃啊！！！！这次实验主要是要设计操作系统的内核，除了要实现一些基本功能以外，还要拓展内核代码，增加一些有用的输入输出函数以及实现批处理能力和分时能力。与很多其他同学一样，我们看到实验题目的第一感觉也是：题目要我们做什么？实话实说，一开始我们真的无从下手，完全不知道要做什么。后来我们觉得非常有必要先学习一下。于是我们找到了王烁辰师兄这次实验的实验报告，对他的实验报告进行了深入的学习。通过王师兄的报告，我们知道了王师兄是如何在内核里实现批处理和分时功能的。基于对王师兄报告的学习和我们对本次实验的理解，我们便着手编写我们自己的操作系统内核。虽然是用我们比较熟悉的 C 语言编写，但是在实验过程中我们遇到了非常多的问题，我们一边继续学习王师兄的报告，一边讨论，完成代码的修改，解决一个又一个问题，其中的辛苦，只有真正去做的人才知道。这次实验虽然耗费了我们很多的时间和精力，但是对于加深对操作系统内核的理解可谓是大有裨益。



附录：

**asm 文件：**

loader1.asm

prog1.asm

prog2.asm

prog3.asm

prog4.asm

os1.asm

**c 文件：**

cpro.c

**bin 文件：**

loader1.bin

prog1.bin

prog2.bin

prog3.bin

Prog4.bin

**obj 文件：**

os1.obj

cpro.obj

**com 文件：**

cpro.com

**img 文件 :**

144mb.img