**实验报告**

院系：数据科学与计算机学院 专业、年级：17级计算机科学与技术 学号：17341097 姓名：廖永滨 指导教师：凌应标

**【实验题目】**

开发独立内核的操作系统：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，存放在其他扇区中，为以后扩展内核提供发展空间。

操作系统内核：

* 可加载多个用户程序
* 汇编模块
* C模块
* 在磁盘上建立一个表，记录用户程序的存储安排；
* 可以在控制台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；
* 设计一种命令，并能在控制台发出命令，执行用户程序；
* 批处理

**【实验目的】**

1. 学习C与汇编混合编程，掌握GCC+NASM交叉编译技术；
2. 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2中的部分或全部功能。

**【实验要求】**

1. 实验三必须在实验二基础上进行，保留或扩展原有功能，实现部分新增功能。
2. 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星，内核获得控制权后开始显示必要的操作提示信息，实现若干命令，方便使用者(测试者)操作。
3. 制作包含引导程序，监控程序和若干可加载并执行的用户程序组成的1.44M软盘映像。

**【实验方案】**

1. 实验工具：

Notepad++：编写程序时使用的编辑器；

Subline：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

VmWare虚拟机: 创建裸机环境，生成虚拟磁盘

1. 实验基本操作方法：

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

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

c. 链接命令：tlink /3 /t cfile.obj afile.obj , final.com

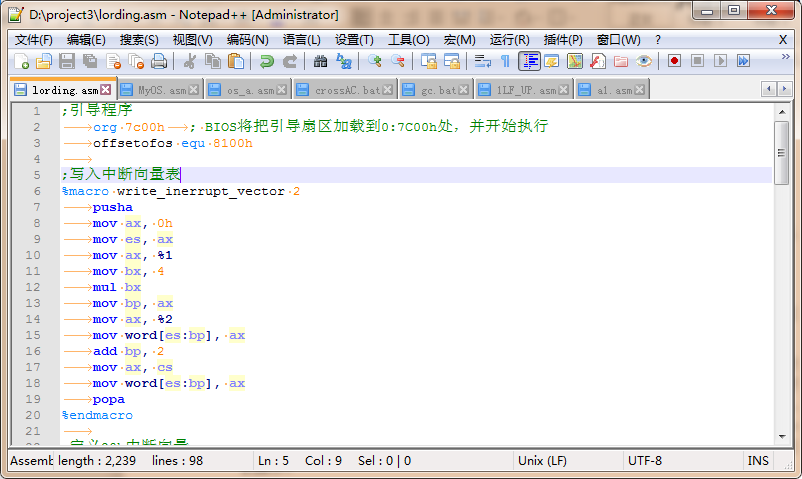
d. NASM汇编指令：NASM afile.asm

1. 实验原理
2. 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。
3. 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。

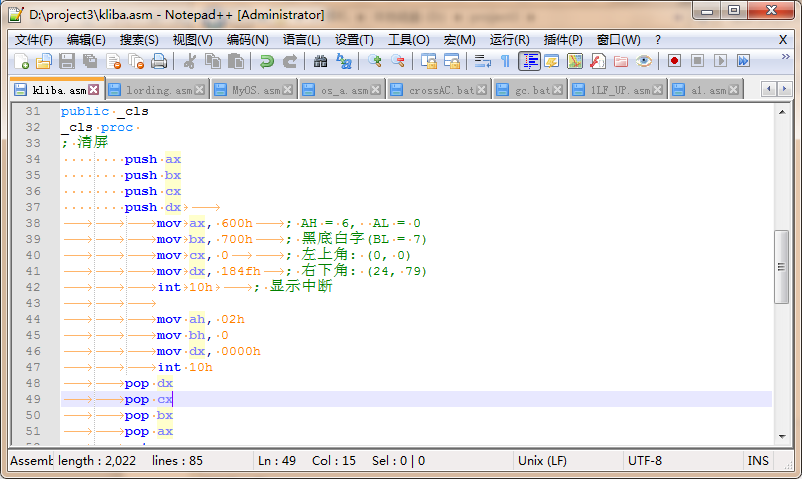
**【实验过程】**

1、实验主要编写方法：（后面有代码解析）

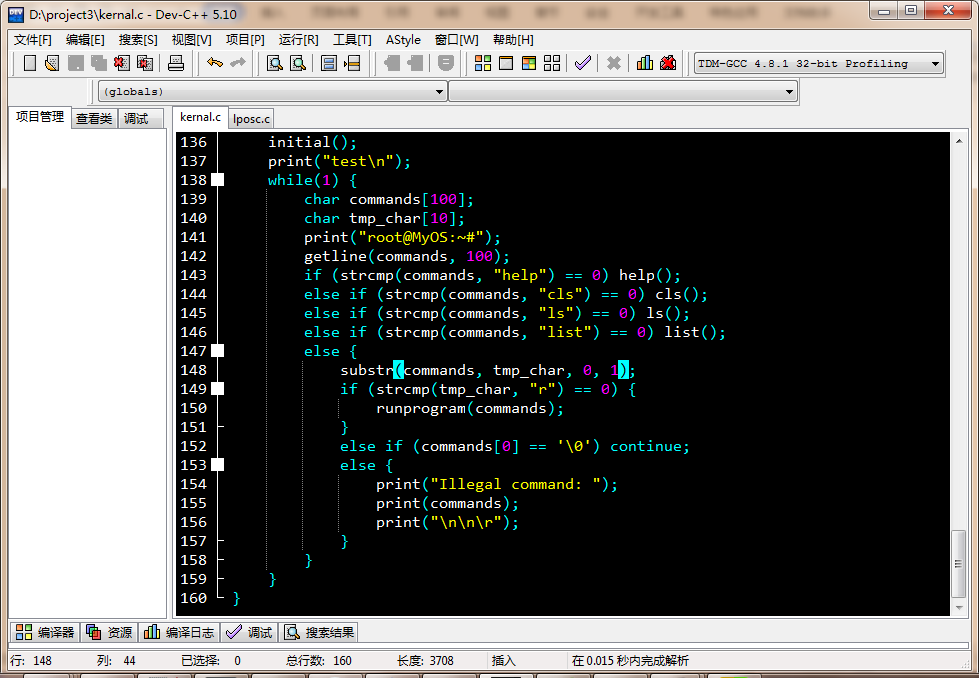
1. 编写一个名为lording.asm的引导程序，将这个程序放在引导扇区，用于加载操作系统并将控制权移交给操作系统。（此处要定义中断表以便后面子程序跳回OS）



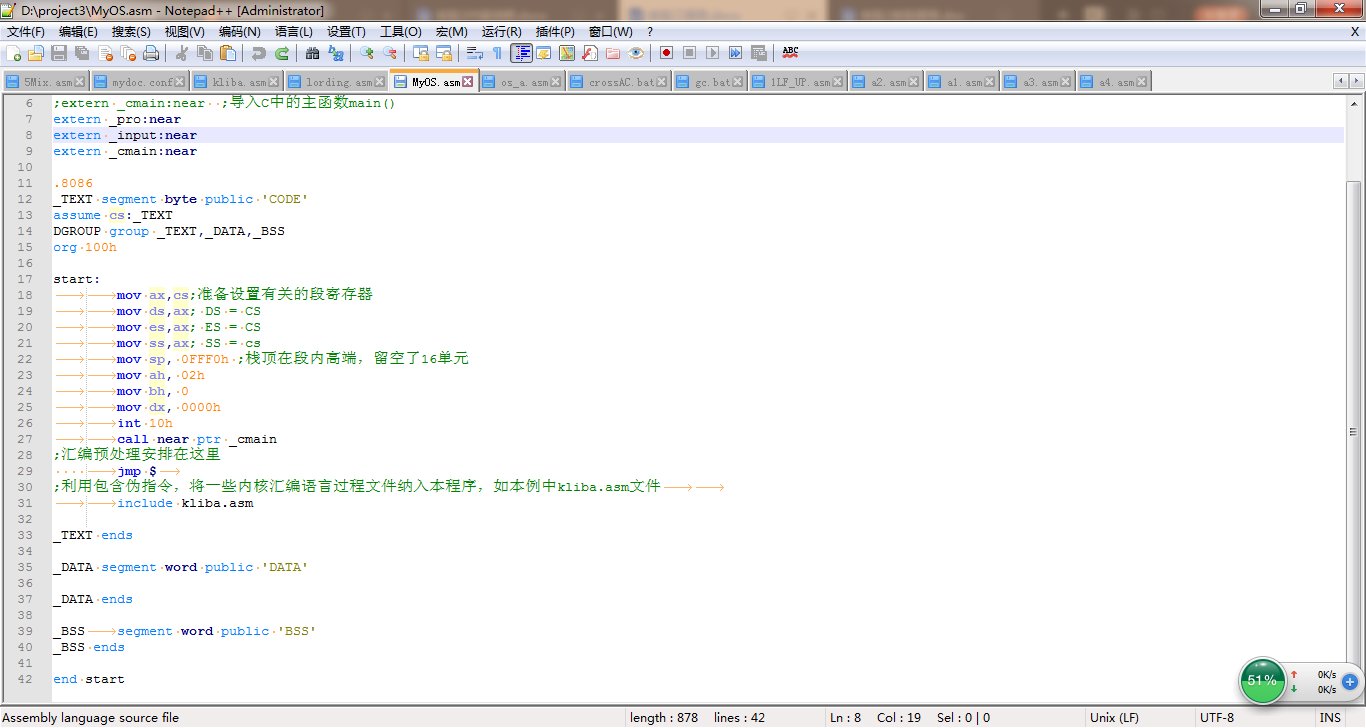
1. 编写一个名为kliba.asm的汇编代码，在这段代码中实现了清屏、加载并运行用户程序、显示一个字符、读取一个字符输入这四个基本的底层功能。



1. 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。



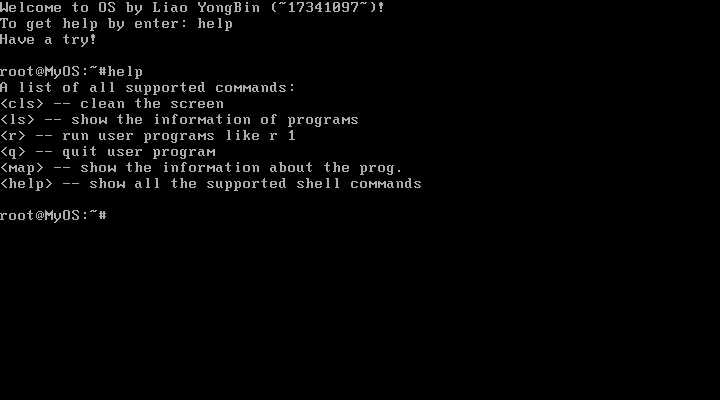
1. 编写一个名为MyOS.asm的汇编代码，在这段代码中，导入kernal.c中的全局变量和主函数，导入kliba.asm中的汇编代码，设置相关段寄存器后，跳转至kernal.c中的cmain程序。



1. 子程序从实验二中进行修改，此处除了我自己的子程序外，我还修改了别人的子程序进行操作系统泛用性的测试。发现，主要在代码段中插入自己中断20h就可以正常使用。

## 【实验结果】

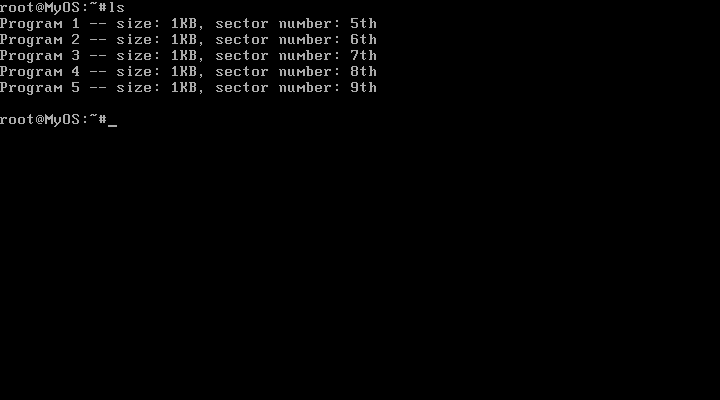
1、help指令



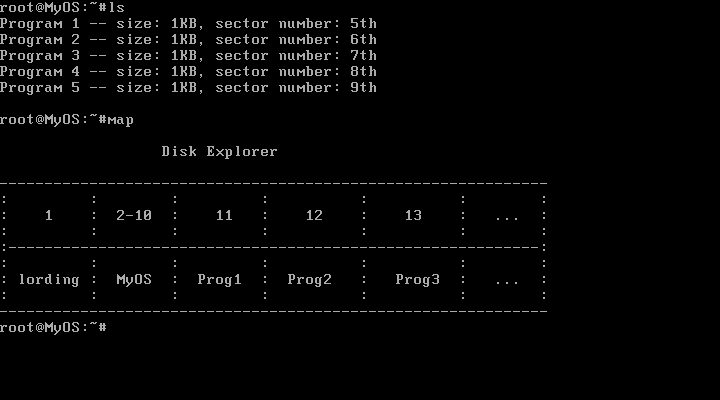
1. cls指令

（由于直接被清屏了，没有截图的意义。。。）

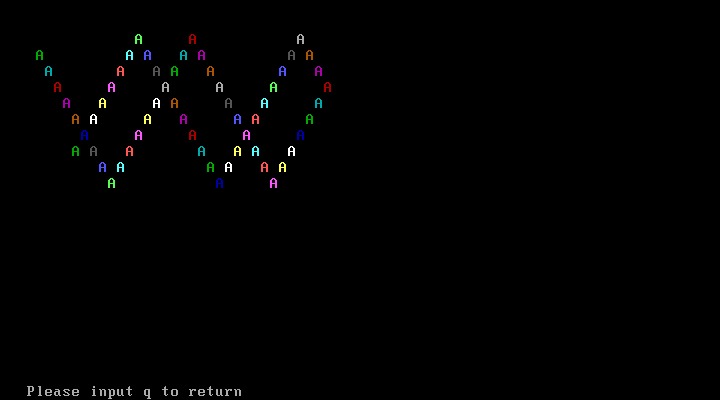
3、ls指令



4、map指令



1. r指令(可以批处理，例如输入r 1 2 3，会依次运行1,2,3)



## 【代码解析】

1. 读入键盘输入

void getline(char \*ptr, int length) {

int count = 0;

if (length == 0) {

return;

}

else {

getChar();

while (input != '\n' && input != '\r') {

if (input == '\b'){ /\*此处就是当读入del键时，先判断当前字符 串长度，如果为0则跳过，>0就退一格并输出空白后再退一格，并将计数器减1，达到 视觉上和效果上了del \*/

if (count > 0){

printChar(input);

printChar(' ');

printChar(input);

ptr[--count] = 0;

}

}else{

printChar(input);

ptr[count++] = input;

if (count == length) {

ptr[count] = '\0';

print("\n\r");

return;

}

}

getChar();

}

ptr[count] = '\0';

print("\n\r");

return;

}

}

1. 程序指令执行

cmain() {

if(now>0)

runprogram(commands);/\*这是实现批处理的一个小操作，就是先判断之前那个语句是否执行结束，如果没有就跳到那里继续跑\*/

initial();

while(1) {

char tmp\_char[10];

print("root@MyOS:~#");

getline(commands, 100);

if (strcmp(commands, "help") == 0) help();

else if (strcmp(commands, "cls") == 0) cls();

else if (strcmp(commands, "ls") == 0) ls();

else if (strcmp(commands, "map") == 0) map();

else {/\*连续的if判断就是方便指令执行相应的函数，如果指令错误就会报 错并跳过这个语句\*/

substr(commands, tmp\_char, 0, 1);

if (strcmp(tmp\_char, "r") == 0) {

runprogram(commands);

}

else if (commands[0] == '\0') continue;

else {

print("Illegal command: ");

print(commands);

print("\n\n\r");

}

}

}

3、RUN函数的实现

void runprogram(char \*comm) {

int i;

int flag = 0;

for (i = 1; i < strlen(comm); ++i) {

if (comm[i] == ' ') continue;/\*就是会自动跳过空格键且只有四种子程序\*/

else if (comm[i] >= '1' && comm[i] <= '5') {

pro = comm[i] - '0' + 10;

if(flag == now){/\*先判断全局变量now是否与该语句的执行阶段一致，是就执行并将now加1进入新阶段，否则该语句将跳过当前阶段继续判断，这里是为了实现批处理\*/

now++;

run();

}

else{

flag ++;/\*就像上面讲的一样，进入新阶段继续判断\*/

continue;

}

return;

}

else {

print("invalid program number: ");

printChar(comm[i]);

print("\n\n\r");

return;

}

}

now = 0;/\*整个语句执行完毕，初始化执行阶段\*/

}

4、汇编程序，以run为例子

public \_run ;声明函数

\_run proc ;开始

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,这里的0B100h是规定好 ;的，我所有子程序的org也都是0B100h

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）,pro是传过来的参数

int 13H ; 调用13H号中断

mov bx, 0B100h

jmp bx ; 跳转到该内存地址

\_run endp ;结束

1. loading汇编实现（局部,此处代码是网上找的有关中断的设置）

;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

注：诸如其他cls，print，getch函数的汇编实现，大家都差不多（也没法搞创新），这里不累述

## 【技术点与创新点】

1、诡异的批处理方法（只应用于运行指令r）

我在c文件代码的最上面声明了一个全局变量now，并初始化为0，当这个值为0时，表现系统并没有没执行完的语句，而但这个值不为0时，表示系统正在执行指令，且处于now阶段。所以，cmain的开始，我判断now是否大于0，是的话，就跳到run函数那里，继续执行第now阶段的指令。所以run函数那里我设置了flag标志来判断运行阶段，每拆解一次r指令，都判断一次，如果阶段符合，就跑程序，不符合就继续往后读，直到读完跳出循环。下面以r 1 2指令为例子解析流程。

（1）now = 0，读指令，进入run函数

（2）Flag = 0，发现与now相等，now++，执行第flag个程序，执行完跳回cmain

（3）cmain发现now==1，直接进入run函数，flag=0，与run不同，flag++，flag=1，此时flag与now相同，now++，执行第flag个程序，执行完跳回cmain

（4）cmain发现now==2，直接进入run函数，flag=0，与run不同，flag++，flag=1，flag++，flag=2此时flag与now相同，now++，执行第flag个程序，执行完跳回cmain

（5）cmain发现now==3，直接进入run函数，flag=0，与run不同，flag++，flag=1，flag++，发现超出字符串成都也没有找到使flag = 2的字符串结构，跳出循环，now = 0

2、dosbox的自动指令使用。就是复制并修改dosbox的conf文件，让dosbox启动时自动执行对应指令，可以省去很多时间。依照网上做法，只要把dosbox的快捷方式后面接一个-conf “路径”就行了，很好用。可以搞多个conf文件和对应的快捷方式。

## 【实验总结】

心得体会：

不得不说，这次实验是我写得最头痛的一次实验。这实验，就算是抄别人的代码估计都

得改半天，更何况是自己慢慢搞。这次实验历时两周，我一直用的是TCC+TASM。这套工具的好处是TASM编译出的代码就是16位的，能与TASM汇编代码完美结合，因此就不用考虑在底层C与汇编相互之间传递参数和返回的一系列令人头疼的问题。但是，TASM汇编的语法实在是令人抓狂。首先，TASM的入口地址只能是100h，因此为了能让正确地把控制权交给用户程序，caller就要用基地址:偏移地址的方式得到用户程序的物理地址。也就是使用 jmp 800h:100h这句代码。这和NASM比起来真的很不方便，因为NASM是任意入口的，org可以是任意合理的值，caller直接跳到用户程序org的地址就可以了。其次，TASM的语法实在是很冗余。比如，为了获取某个地址处的值，需要加PTR，为了获取某个变量，需要交OFFSET。而在NASM中，不用中括号括住的变量就是变量本身的值，用中括号括住表示变量（被当做地址）处的值，多简洁！搞得我很难受，不少代码都是看了别人的才写出来的。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代码中声明了不少全局变量来解决问题。而在这之中，我也遇到很多很多的编译或语法问题。但是搞定输入输出就熬了不少夜，bug层出不穷。

问题和解决方法：

1. 无法正常生成obj : 把-o删掉，原理不明
2. 无法正常生成com文件：调换顺序
3. 编译出现错误 : 正常的语法错误，慢慢改
4. ......

## 【代码清单】

project3

--操作系统

--FINALOS（机器码）

--FINALOS.MAP

--KERNAL.OBJ

--KERNAL.C

--KLIBA.ASM

--MYOS.ASM

--MYOS.OBJ

--工具包

--DOSBox.exe（只是快捷方式）

--mydoc.conf (DOSBOX的配置文件)

--nasm.exe

--nasmpath.bat

--TASM.EXE

--TCC.EXE

--TLINK.EXT

--软盘文件

--实验3.flp

--引导程序

--lording.asm

--lording.com

--用户程序

--a1

--a1.com

--a2

--a2.com

--a3

--a3.com

--a4

--a4.com