**实验3：加载用户程序的监控程序**

姓名：姜洋帆 学号：17341068

院系：数据科学与计算机学院 专业：17级计算机科学（大数据）

【**实验题目**】用C和汇编实现操作系统内核、增加批处理能力

**【实验目的】**

1. 掌握TASM汇编语言与TURBO C语言汇合编程的方法。

2. 实现内核与引导程序分离，掌握软盘上引导操作系统方法。

3. 设计并实现一种简单的作业控制语言，建立具有较友好的控制命令的批处理原型操作系统，掌握操作系统提供用户界面和内部功能的实现方法。

【**实验要求】**

用C和汇编实现操作系统内核，增加批处理能力。

提供返回内核的一种方案。

在磁盘建表，记录用户程序的存储安排。

可以在控制台查看到用户程序的信息，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，命令中可以加载多个用户程序，依次执行，并能在控制台发出命令。

在引导系统前，将一组命令存放在磁盘中，系统可以解释执行。

【**实验方案】**

1. 硬件及虚拟机配置：Lenovo PC ；Oracle Virtual Box
2. 软件工具及作用：

Notepad++ ：写代码

Nasm：编译引导程序的asm文件到bin文件

Tasm：编译16位的内核代码

Tcc：编译16位C语言，实现C语言与x86联合编译

Sublime ：查看、编辑二进制文件，将编译完成的程序机器码写入软盘

C程序：

1. 自己编写的一个C语言程序，用来将编译生成的二进制文件填充为1.44M大小，并将后缀改为.img，生成1.44M的软盘
2. 内核程序，实现一些逻辑性较强的功能，通过调用汇编实现的一些函数来实现各种基本操作，组合出复杂的功能

86汇编：

1. 实现引导程序，偏移量位7c00h，将内核加载至内存并跳转进入内核程序
2. 实现内核程序以及一些常用的函数，如打印字符，输入字符等，可以被C语言调用
3. 程序功能

Myos1作为引导程序，设置好内核的偏移和段地址，加载内核程序，然后跳入内核程序，进行下一步操作。

内核程序实现的功能有：

1. 以表的形式显示、记录用户程序的相关信息
2. 可以在控制台输入命令，输入’table’时清空提示信息，打印上述表格，显示一些信息
3. 输入另一套命令，格式为run # #为数字，可以执行不同的程序，比如run 1执行程序1，然后跳回内核菜单，重新显示提示信息、输入 run 132，则一次执行1、3、2号程序，执行完毕后提示finish，输入任意键返回菜单，等待输入下一套命令（命令字符数量最大为20）
4. 输入run all则直接按顺序执行123，然后返回主菜单
5. 命令行实现退格操作，可以在未输入回车时，通过退格键回退，任意修改命令行字符，并且正确执行相应的指令
6. 命令行实现补全功能，比如想在内核控制台输入’table’命令，只要输入任意前几个字符，按tab键，就可以在屏幕上显示补全的命令，并且可以正确执行相应指令
7. 程序设计

引导程序采用实验二的形式，修改内核的数据段地址和内存地址，跳转进内核，即可实现引导出内核程序，内核程序大概占用5个扇区，在引导程序里直接读了10个扇区。

内核部分采用C和x86交叉调用，这里遇到了较大的问题，花了大量的时间才解决了各种奇怪的问题，主要的思路是，基本功能由汇编实现，如打印字符、从键盘读入命令（C与汇编共同实现）、清屏、回退光标、调出用户程序等；C语言部分实现逻辑性较强的功能，如判断shell命令选择执行不同的操作、打印字符串（循环调用打印字符）、控制台命令补全、打印信息、主程序循环的方式等。

用户程序有3个，分别为实验二的1，2，4号程序（功能完全一致，只修改了内存偏移和返回方式），在被内核调用时，内核用先将ds，es段寄存器入栈，再call相应的子程序，子程序执行结束后使用ret指令跳回内核程序，并将es，ds出栈即可。

五、主要代码

**Myos1引导程序：**

设置好各种偏移量，加载、跳转进内核程序即可。（在这里遇到了一个大坑，一开始忘记设置数据段偏移了）

;程序源代码（myos1.asm）

org 7c00h ; BIOS将把引导扇区加载到0:7C00h处，并开始执行

OffSetOfKernalData equ 100h

KernelSeg equ 64\*1024/16

Start:

mov ax, cs ; 置其他段寄存器值与CS相同

mov ds, ax ; 数据段

mov es, ax ; 置ES=DS

LoadnEx:

;读软盘或硬盘上的若干物理扇区到内存的ES:BX处：

mov ax,KernelSeg ;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, OffSetOfKernalData ;偏移地址; 存放数据的内存偏移地址

mov ah,2 ; 功能号

mov al,10 ;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

jmp KernelSeg:OffSetOfKernalData

AfterRun:

jmp $ ;无限循环

**内核程序（汇编部分）**

这一部分个人认为是本次实验的核心，一开始使用了老师的kliba.asm那个库，但是发现其实问题挺多的，比如打印字符串的printf函数，调用次数过多后就会出现乱码，不能正确显示后面的数据，据一些大佬说是缓存的问题，遂在C里面实现了打印字符串，就是使用while循环，调用printChar函数（这个汇编代码没有问题，直接使用老师的了），一次读入输出每个字符。另外在这里实现了清屏、读键盘输入并显示字符、实现退格键等功能，主要代码如下：

**退格功能：**

实现退格功能，大致的想法是，获取光标位置，输出空格并将重置光标位置（dl寄存器减一），实际实现的时候，需要手动在C语言中传入一个空格字符（在汇编调用int 10中断输出32时总是会遇到一些问题，所以就采用这种不太优雅的方式实现了）

public \_backspace

\_backspace proc

push bp

mov ah,3 ;获取光标位置

mov bh,0

int 10h

mov ah,2 ;置光标

mov bh,0

dec dl

int 10h

mov bp,sp

mov al,[bp+4] ;指向栈顶元素，即字符

mov bl,2

mov ah,0eh ;显示字符光标前移

int 10h

mov sp,bp

mov ah,3 ;获取光标位置

mov bh,0

int 10h

mov ah,2 ;置光标

mov bh,0

dec dl

int 10h

pop bp

ret

\_backspace endp

**打印字符：**

将指向字符串地址的bp寄存器压栈，调用中断即可，其中ah=0eh可以实现动态移动光标，使光标总是出现在当前显示字符的后一位

public \_printChar

\_printChar proc

push bp

mov bp,sp

mov al,[bp+4] ;指向栈顶元素，即字符

mov bl,2

mov ah,0eh ;显示字符光标前移

int 10h

mov sp,bp

pop bp

ret

\_printChar endp

**读取字符：**

实现比较简单，调用16号中断即可，其中chBuf是定义在C代码中的一个字符缓存，每次输入都先把字符存到chBuf中，在进行其他操作。

public \_getChar

\_getChar proc

mov ah,0

int 16h ;0号功能调用从键盘读入一个字符放入al中

mov byte ptr [\_chBuf],al

tag:

ret

\_getChar endp

还有一些其他功能，如执行某个程序，将相关的寄存器（ds、es）压栈，设偏移量，然后call即可。清屏的功能直接使用老师的代码来实现。

**内核程序（C语言部分）**

之前汇编部分都在造底层的轮子，在C语言实现一些逻辑功能，由于对C语言使用比较熟悉，而且可以不用考虑底层，尤其是那一堆寄存器和各种中断，这里写起来比较容易，不过需要主义函数定义的相对位置的问题，在C内部相互调用函数，似乎需要被调用的函数定义在调用的函数之前。之前以为编译成obj文件后，用符号表示入口，链接的时候，可以忽略这些问题。在这里实现的功能主要有输入命令行、打印字符串、打印信息、实现tab补全以及一个主函数，通过while（1）死循环，使每次程序执行完毕后，打印提示并跳回菜单。主要代码如下：

**Tab补全**

通过暴力匹配的方式来确定当前可能在输入的命令，并返回不同的值来区分不同的命令（实现写在定义好的字符串中）。若没有匹配项则返回0

int tab() {

j=0;

while(CMDline[j]!='a'){

if(CMDline[j]==shell1[j]){

j++;

}

else break;

}

if(CMDline[j]=='a'&&j>0){

for(j=0;j<5;j++)

CMDline[j]=shell1[j];

return 1;

}

j=0;

while(CMDline[j]!='a'){

if(CMDline[j]==shell2[j]){

j++;

}

else break;

}

if(CMDline[j]=='a'&&j>0){

for(j=0;j<4;j++)

CMDline[j]=shell2[j];

return 2;

}

return 0;

}

**读命令行**

采用简单的循环，调用getChar，实现输入功能，每次输入后判断，若不是退格或者回车或tab，则存入CMDline字符串，遇到回车退出，遇到退格则调用汇编写的退格功能，并将CMDline下标前移。遇到tab键则调用tab函数实现相关操作。

void ReadCommand() {

i=0;

getChar();

printChar(chBuf);

CMDline[i++]=chBuf;

while(chBuf!=13) {

getChar();

if(chBuf!=8 && chBuf!=9){

printChar(chBuf);

CMDline[i++]=chBuf;

}

else if(chBuf==8){

backspace(' ');

CMDline[i--]='a';

chBuf='a';

}

else {

printChar(' ');

backspace(' ');

flag = tab();

if(flag==1) {

for(i;i<5;i++)

printChar(shell1[i]);

}

else if(flag==2) {

for(i;i<4;i++)

printChar(shell2[i]);

}

}

}

printf("\n\r");

i=0;

}

**命令行控制模块：**

根据读入CMDline的数据，选择调用不同的功能或者报错。

void shu() /\*批处理 根据指令，执行一连串程序\*/

{

printf(">>");

ReadCommand();

num=4;

if(CMDline[0]=='r'&&CMDline[1]=='u'&&

CMDline[2]=='n'&&CMDline[3]==' '){

if(CMDline[4]=='a'&&CMDline[5]=='l'&&CMDline[6]=='l') {

mypro1();

mypro2();

mypro3();

}

else if(CMDline[num]>='1'&&CMDline[num]<='3') {

while(CMDline[num]>='1'&&CMDline[num]<='3') {

if(CMDline[num]=='1')

mypro1();

else if(CMDline[num]=='2')

mypro2();

else if(CMDline[num]=='3')

mypro3();

num++;

}

num=3;

}

else printf("Invalid Command!\n\r");

}

else if(CMDline[0]=='t'&&CMDline[1]=='a'&&CMDline[2]=='b'

&&CMDline[3]=='l'&&CMDline[4]=='e'){

printProInfo();

}

else {

printf("Invalid Command!\n\r");

}

getChar();

for(jj=0;jj<28;jj++)

CMDline[jj]='a';

printf("Press any ket to continue...\n\r");

myrestart();

}

**主程序：**

void cmain() {

while(1) {

printmsg();

shu();

}

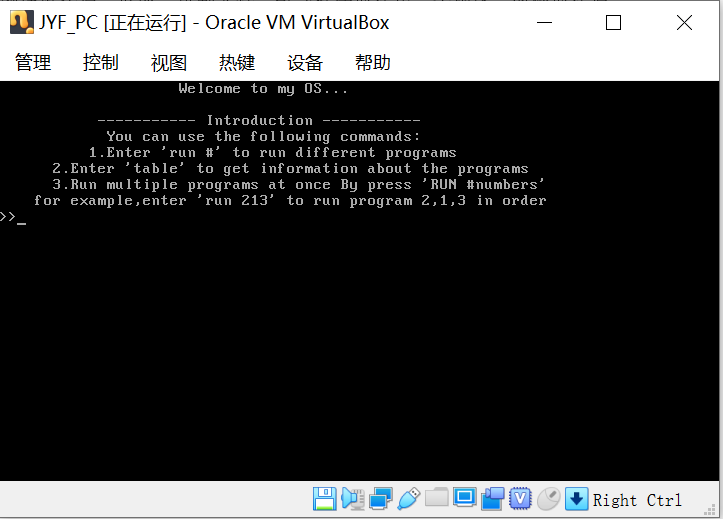
}

**【实验过程】**

**功能展示**

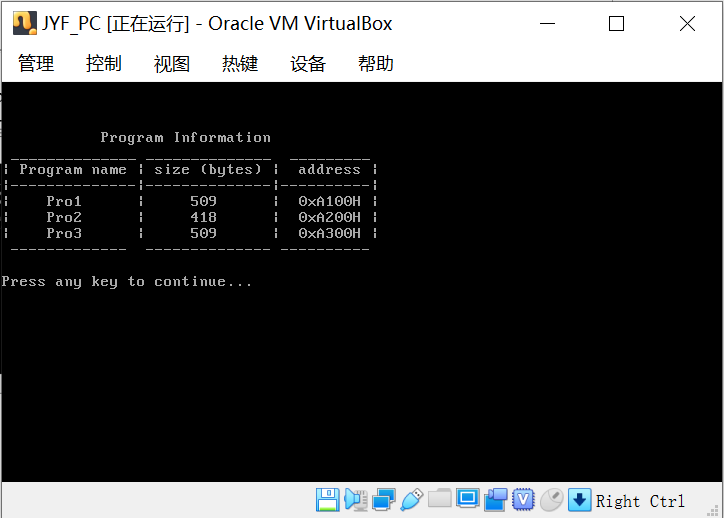
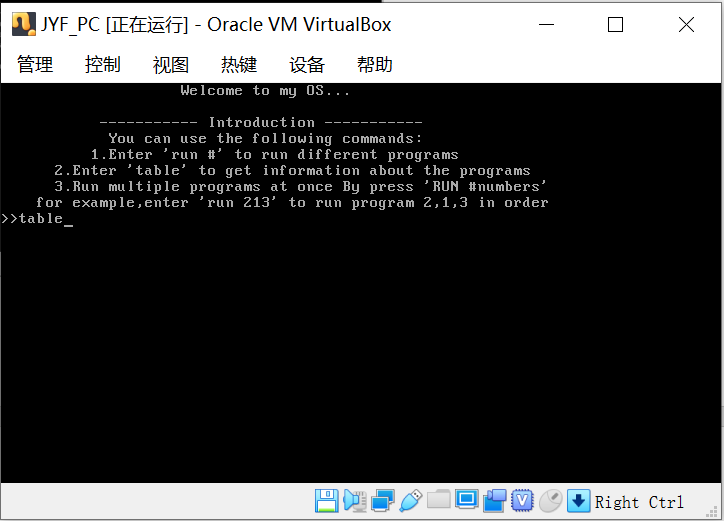
将所有代码编译链接后，形成二进制文件，引导程序放在第一个扇区，内核放在后五个扇区，然后从第十个扇区开始一次放置用户程序1，2，4

运行虚拟机，自动完成引导，加载出内核程序，显示相关信息：



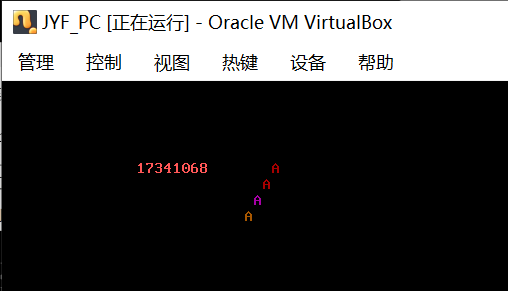
提示可以输入不同的命令，达到不同的功能

1. 输入table查看程序的信息：



输入任意键后回到主菜单

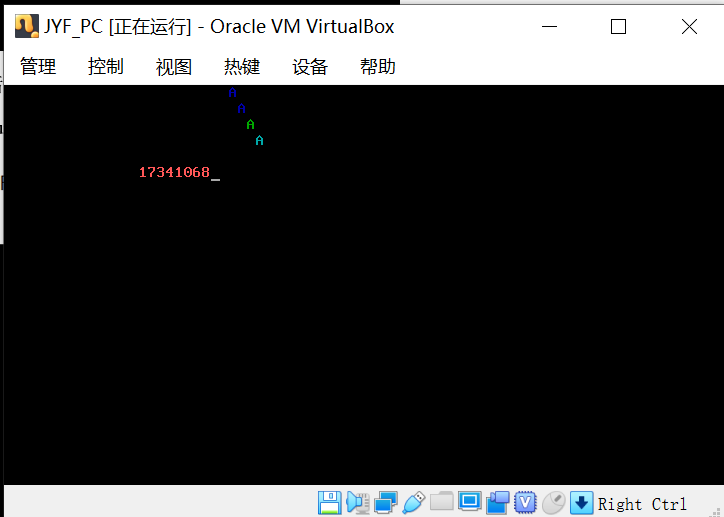
1. 输入run 1，执行一次1号程序：



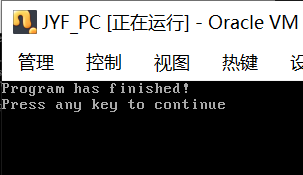
程序结束后画面停止，按任意键弹出提示信息，再输入任意键返回到主菜单

1. 输入 run 312 按顺序，执行3、1、2号程序：





1. 输入run all 命令脚本，一次执行1，2，3号程序，执行完成后输入任意键提示程序结束，再输入任意键返回菜单。



1. 命令行补全功能

比如只输入ta 然后按下tab键，会补全table



1. 退格键

再输入错误时，可以通过backspac键，取消之前的输入，并且同时改变屏幕上显示的信息



1. 输入错误的命令，输出提示Invalid Command!，回车后重置菜单，重新输入



**【实验总结】**

本次实验做得非常坎坷，花了大量的时间去试错调错。这次实验的重点应该是了解C和汇编的交叉调用，熟悉内核的工作机制。然而我自己为了实现在虚拟机上显示出字符串这一个简单的功能，花了好几天时间，中途更换过各种方法，最后才找到问题。在com文件测试时可以显示，但是放进虚拟机却不行，一开始以为是汇编代码的问题，后面初步认为是数据段的问题，可是发现汇编代码中数据段都设置正确了，到最后才找到问题，出现在引导程序，忘记去设置内核地址段的偏移了。

在开始实验之前是打算做32位保护模式的，在linux环境下，使用gcc+nasm，使用助教配备的虚拟机，做了一天的尝试，最后以失败告终，因为之前没有学过x86汇编，要从零开始造轮子，而且有一些特权指令、中断和实模式不太一样，用不了，最后认为难度太大，而且时间也不够。

然而在使用了老师的实例代码时，也遇到种种问题，比如printf函数有bug，Readcommand函数不好用等等，最后在老师提供的汇编库的代码，做了一些补充和重构，才实现各种需要的功能。一开始做思路不清晰，C和汇编交叉调用的非常多，代码结构也比较混乱，在遇到一大堆问题之后，重构了代码，把所有基础的功能放在汇编里实现，而所有高层的逻辑结构的实现，由C语言 来实现，这样debug更加方便，而且思路也会比较清晰。

有些内核汇编代码的实现比较粗糙，应该有更好办法实现，内核程序也有不少实用的功能没有实现，另外自己实现的退格功能有个小bug，会把命令提示符>>也给退掉，打算在之后的实验里再去修复这些问题，并且慢慢补全各种功能。