**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 | **2017.04.21** | 完成日期 | **2017.04.27** |

# 【实验题目】

设置中断与编写中断服务程序

# 【实验目的】

熟悉时钟中断、键盘中断，学习如何编写中断

# 【实验要求】

1、操作系统工作期间，利用时钟中断，在屏幕24行79列位置轮流显示’|’、’/’和’\’，适当控制显示速度，以方便观察效果。

2、编写键盘中断响应程序，原有的你设计的用户程序运行时，键盘事件会做出有事反应：当键盘有按键时，屏幕适当位置显示”OUCH! OUCH!”。

3、在内核中，对33号、34号、35号和36号中断编写中断服务程序，分别在屏幕1/4区域内显示一些个性化信息。再编写一个汇编语言的程序，作为用户程序，利用int 33、int 34、int 35和int 36产生中断调用你这4个服务程序。

# 【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

# 【程序功能】

## 开机进入内核，内核提供 5种指令输入，如下图：QQ截图20170427220455

1. 时钟中断：在内核运行期间，会在屏幕边框逆时针显示字符，每次经过左上角变更字符，每次显示变更颜色。

在进入用户程序之前，恢复默认中断向量表以停止边框显示；在从用户程序返回内核后，重新设置中断向量表继续显示边框。

2. 键盘中断：在内核进入用户程序前，修改当前键盘中断向量表，使用户程序中每次按键盘都会显示“Ouch!Ouch!”，且显示位置不断变化。从用户程序返回内核后，恢复默认中断向量表，以保证内核正常的输入功能。

3. 批处理功能：指令 batch 之后，首先输入要批处理程序的序号，范围为 0~4。之后依次运行对应序号的程序。

4.分时功能：指令 batch 之后，首先输入要分时运行的程序的序号，范围为 0~4。之后同时运行对应序号的程序。

5. 显示用户程序信息：指令 show,可以显示用户程序的信息。

7. 命令行输入：左下的 Orz>处输入命令，此处模仿 cmd，在输入过程中，若输入错误，可以进行删除一定字符以修正输入。同时，命令对空格不要求，可没有空格或有多个空格。命令以回车结束。

# 【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的代码，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

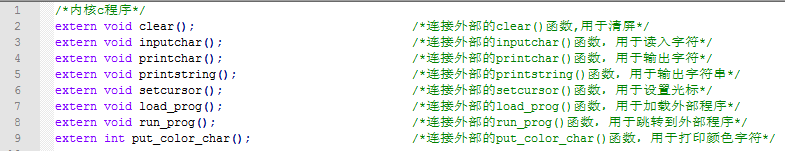
出，保持一个可观察的适当速度直线运动，碰到屏幕相应 1/4 区域的边后产生反射，改变方向运动，如

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。

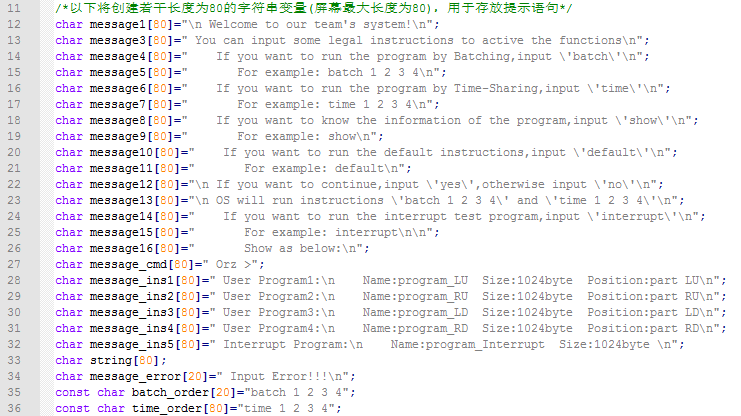
这四个程序其实和实验3的是一样的，没有进行大的改动，直接使用就可以了。

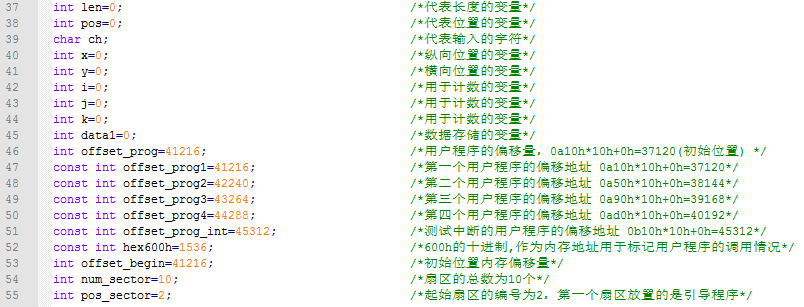
因为是以实验3的内容为基础，在里面加上几个中断，因此我们把实验3的内核拿出来使用，保留实验3的所有功能并进行改动。

首先，我们可以来看一下c语言文件：

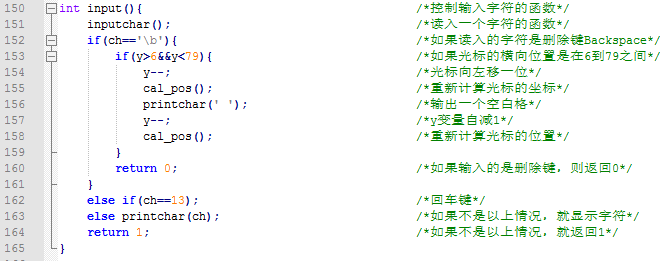
以下是外部函数的声明：

把以上函数声明好后就要声明出要显示的字符串：



还有部分需要用到的变量：

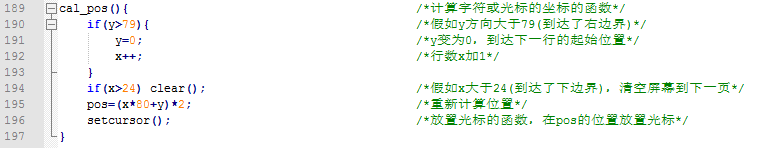
接下来就要定义一些在c语言里面比较容易实现的函数了：

输入数据的函数：

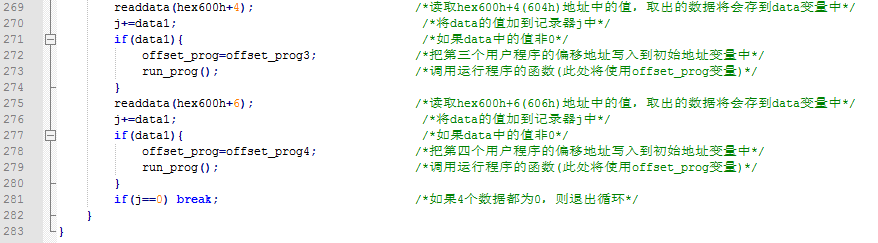
调用输入命令的函数：

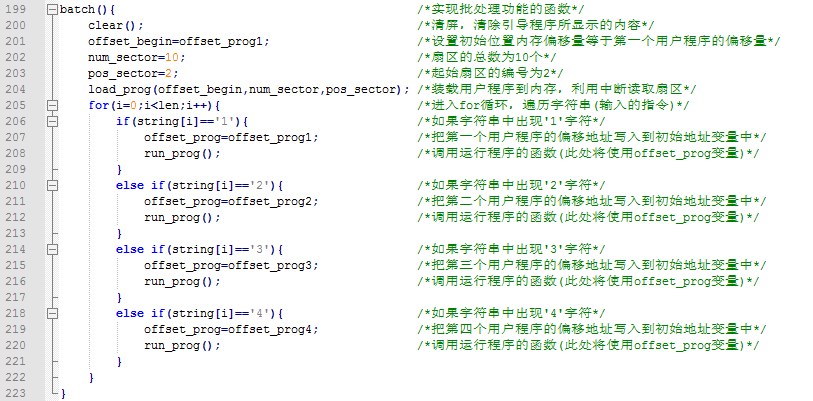
## QQ截图20170427230951

计算光标的坐标的函数：

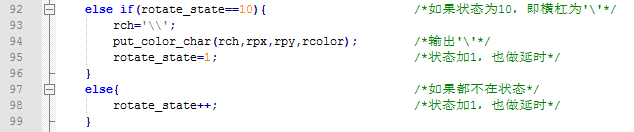


## QQ截图20170427231109实现分时功能命令的函数：

****

实现批处理功能命令的函数：

接下来就是与中断有关的函数了：

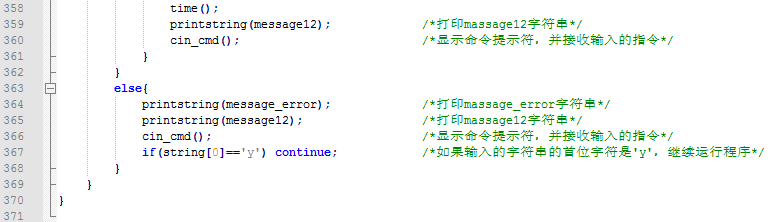


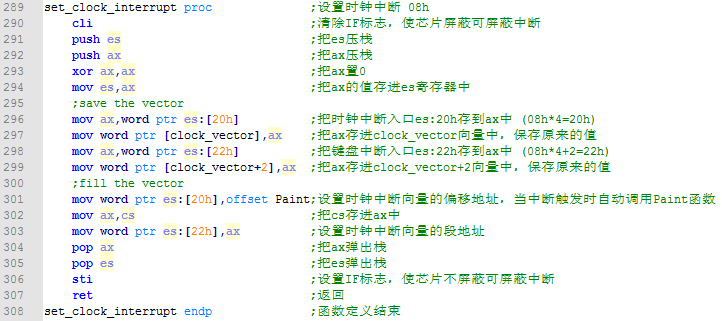




 主函数如下：







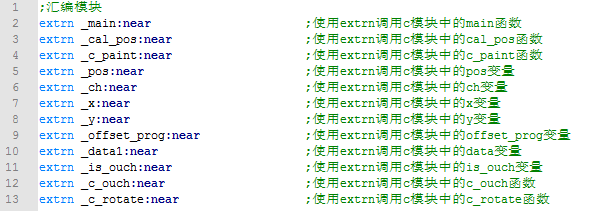
## 以上为止，c函数的编写就结束了，每一条语句的具体功能都在代码上有注释（如上）。

## 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。



汇编代码跟实验3的内容差不多，只不过加上了以下的中断代码：

## QQ截图20170428000530

## QQ截图20170428000538

## QQ截图20170428000602

## QQ截图20170428000609

## QQ截图20170428000624

## QQ截图20170428000849

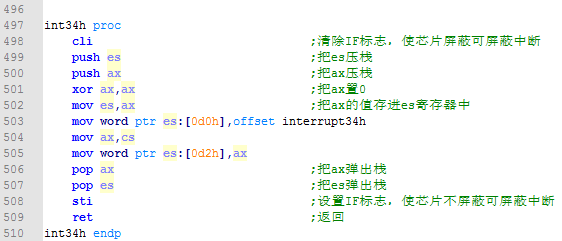
## QQ截图20170428000841

## QQ截图20170428000859

## 33QQ截图20170428000910号中断：



## QQ截图20170428001420

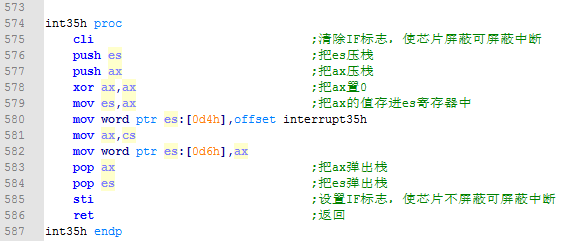
34号中断****

## QQ截图20170428001508



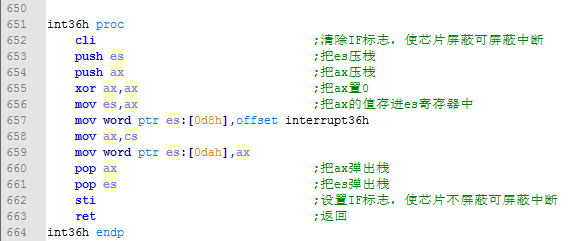
## QQ截图20170428001610

35号中断：

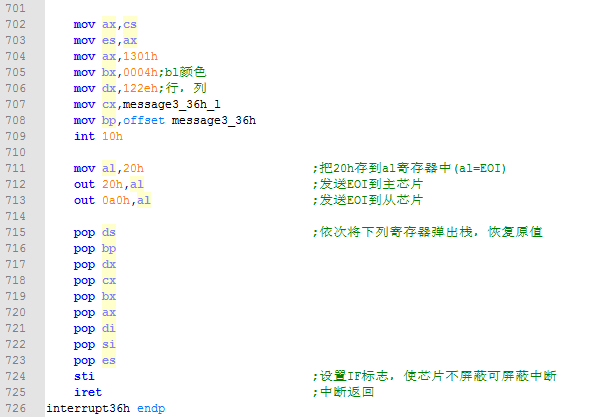




## QQ截图20170428001737

36号中断：

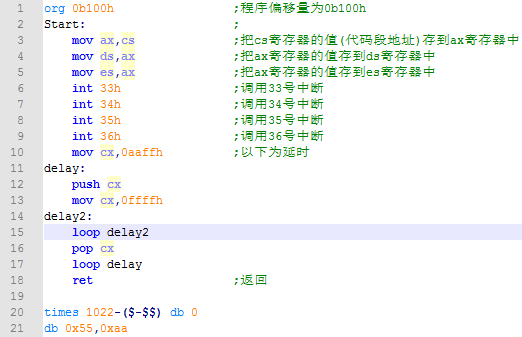
## QQ截图20170428001815



## 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。

## 接下来，就要编写调用33h，34h，35h，36h四个中断程序的用户程序了。在上面的代码可以看出，这四个代码的功能是在屏幕的左上、左下、右上、右下的位置输出四个信息，用户程序在内核中调用。

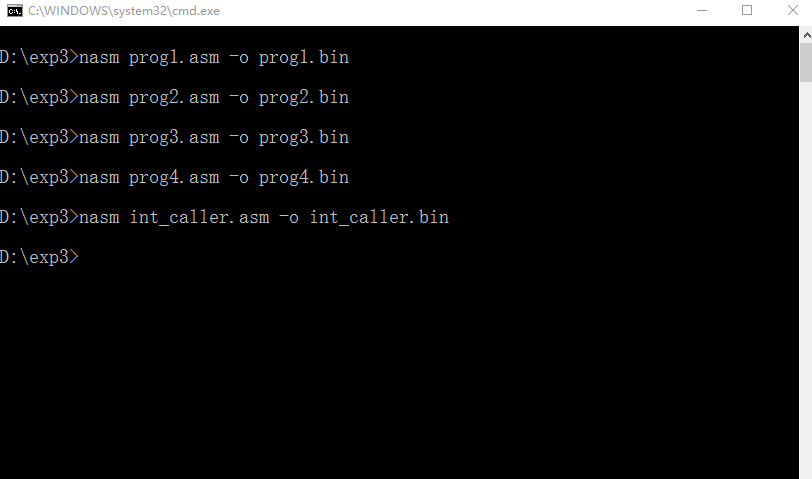
## 代码如下：



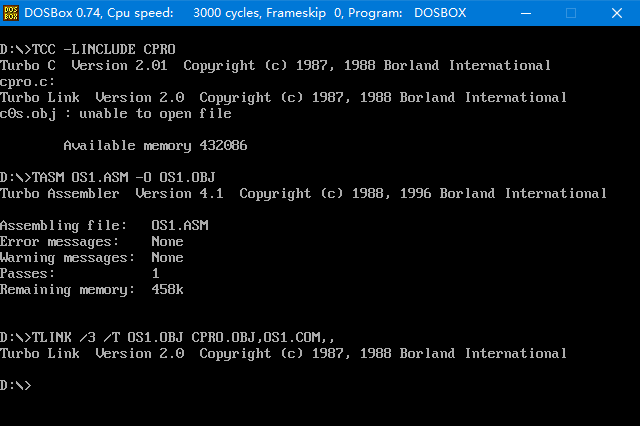
把这个用户程序编译成bin文件后和四个用户程序一起合并到loader1.bin中就完成了。

至此，所有的程序就已经完成了，真是一个很痛苦的过程！

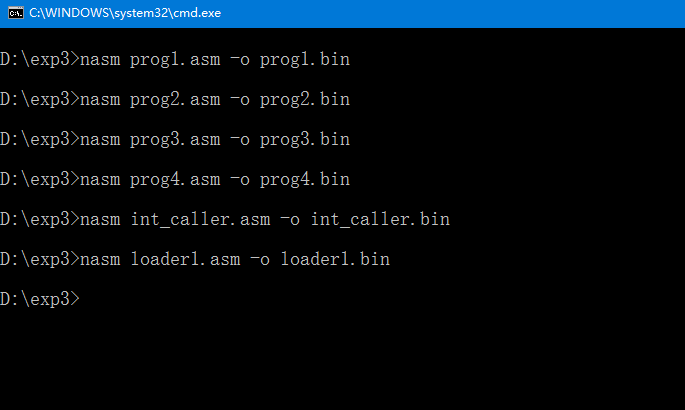
接下来就是编译的过程了，跟实验3的时候一样的编译方法：

首先使用nasm编译出5个用户程序：

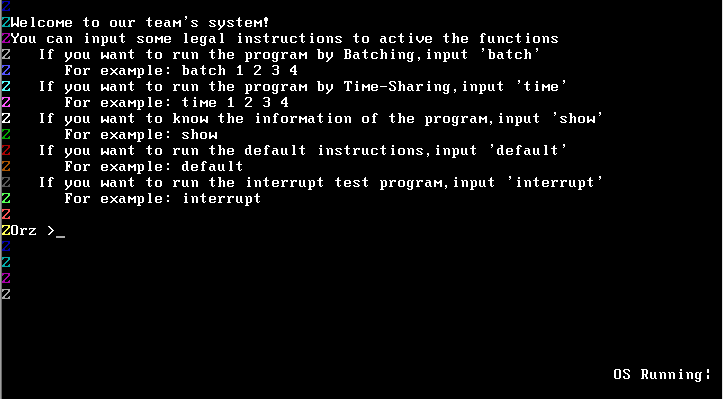
然后使用DOSBOX编译内核的汇编文件和c文件，再连接起来：



最后再编译引导程序：

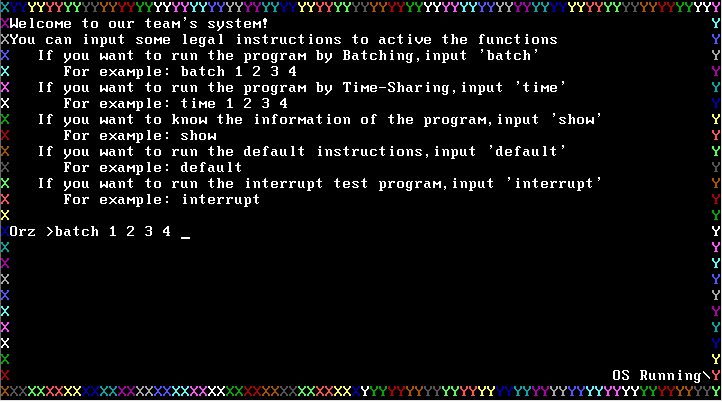


得到loader1.bin文件，把内容复制到144mb.img，导入到虚拟机里面，开始运行：

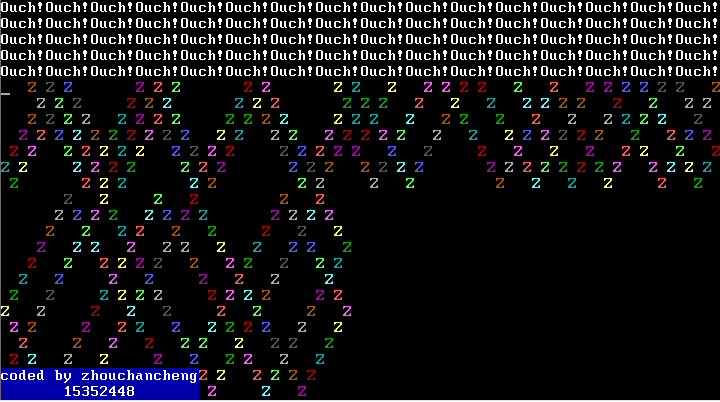
****

以上就是初始的界面，可以看到四周的边框有一个字符围绕转动的效，右下角可以看到表示操作系统正在运行的标识，还有一个不停变换的‘-’‘\’‘|’‘/’。

然后输入batch 1 2 3 4指令，批处理实现用户程序的调用，在进入后按下键盘，会发现屏幕有“Ouch!”字样出现：

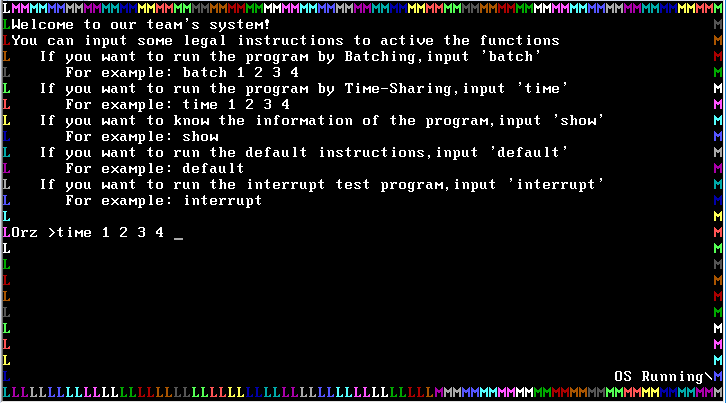


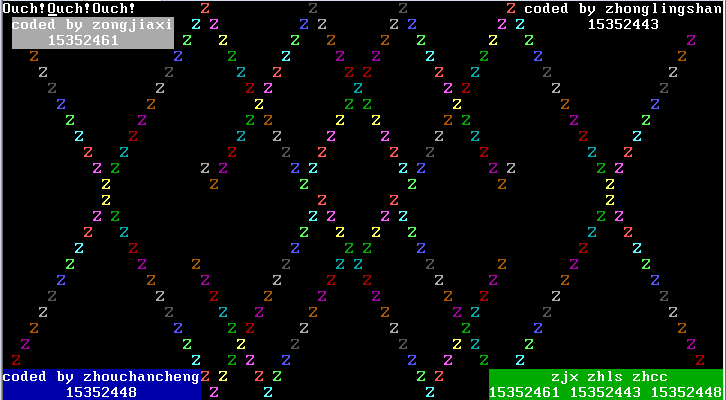




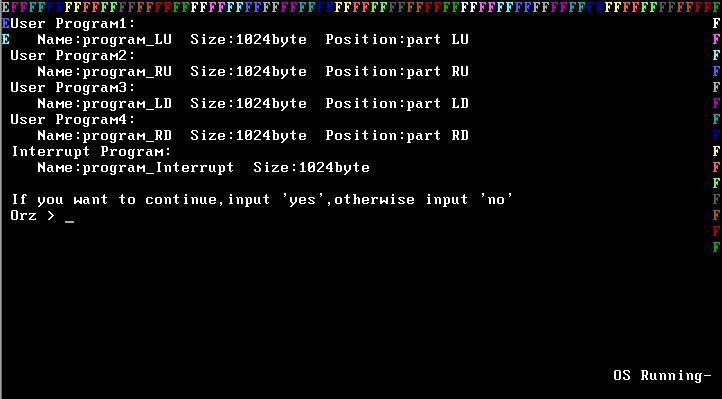
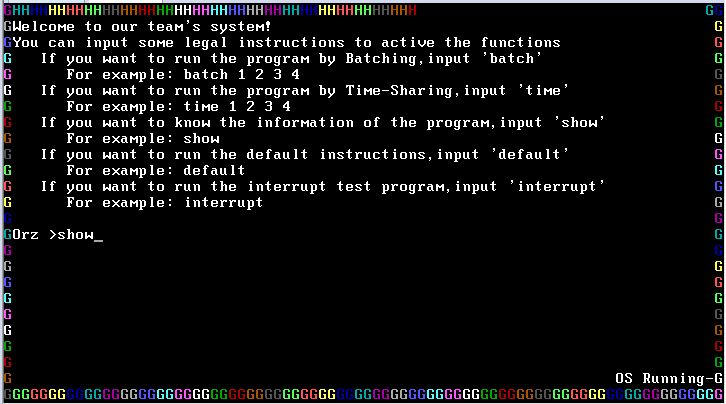
可见，键盘触发成功，每当键盘有输入时就会有Ouch!弹出，但是由于批处理是一个一个程序依次执行，因此每一个新的用户程序开始运行时，光标都会回到屏幕的左上角，重新打印Ouch!，因此后面的程序运行的时候Ouch!就会覆盖掉前面的内容。

重新回到引导程序，输入time 1 2 3 4指令，批处理实现用户程序的调用，在进入后按下键盘，会发现屏幕有“Ouch!”字样出现，但是这一个分时处理由于是不停地在调用用户程序，所以Ouch!会不停的清空并重新打印，因此当键盘输入的时候，最多会在左上角产生两到三个Ouch!：

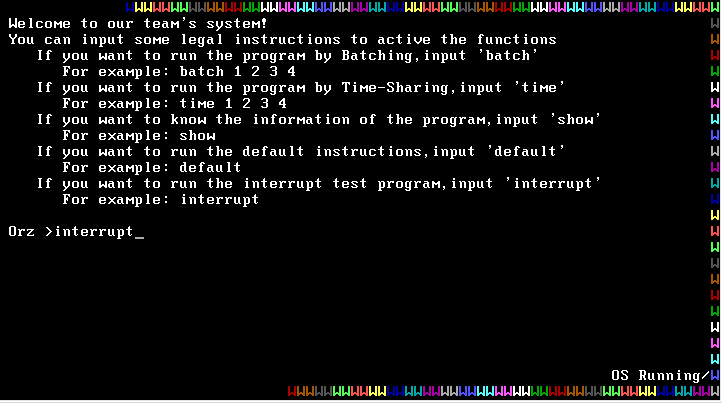


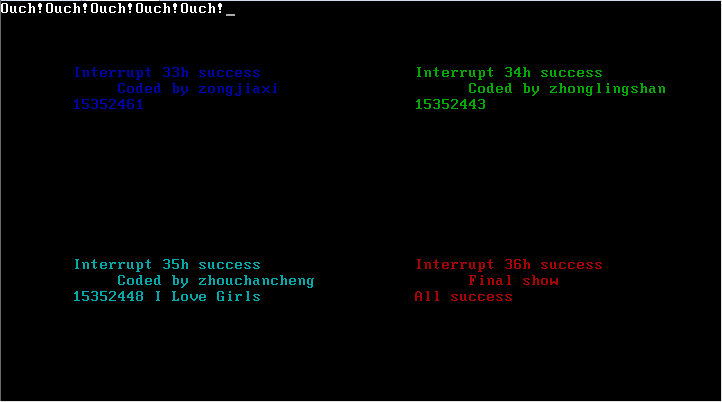


输入show：



输入interrupt，运行调用中断的用户程序，可以看到在屏幕的四个位置都显示了一些我们小组组员的信息，而且按下键盘按键还能打印Ouch!，这个中断程序的延时大概为8秒左右：





至此，本次实验的程序功能已经全部展示完毕。

# 【实验总结】

**宗嘉希 15352461：**首先吧，这一次的实验花费了我挺多的时间，比前三次的实验所花费的时间都要多，因为我没有去系统地学习x86汇编，因此对于内核、引导程序、用户程序等等在内存中是如何分配空间的，因此在本次的实验中遇到了很多困难并且花费了很多的时间去解决。现在就在此说一下遇到的困难吧：首先，老师给了我们一个能在屏幕中间利用中断去打印ascii码字符的程序，但是我不明白为什么在我的虚拟机上无法运行这一个程序，运行的结果就是只在屏幕中央显示初始化时的数字，中断过程完全没有，这个问题我烦恼了很久，大概搞了一周才发现原来是我的内存位置没有设置好，老师给的代码里面写的是org 100h，就是把代码定位到偏移量为100h的内存位置，然而我通过查阅资料发现，应该是7c00h，才可以让中断正常运行，因为7c00h的内存位置是引导程序的起始位置。所以，我就解决了这个一开始就遇到的大问题。后来，我把中断加入到了内核中，也就是c语言和汇编语言组成的内核中，但是又有一个新的问题出现了，就是这个中断它不会在内核中正常运行，这个也是花费了我很多的时间去找解决的方法（没办法，汇编语言没学好），后来才发现原来引导程序必须要放在org 8100h的位置。后来，在实验快要完成的时候又有一个新问题了，就是无法正常运行用户程序，好像卡死了一样，这个只花了我一天就找出问题来了，因为一开始的时候我是直接使用我自己编写的实验三的源代码去添加终端的内容来做实验四的，但是实验三的代码中，复杂度比不上实验四的，因此，内核占据的空间变大了，变成了14个扇区了，但是原来读取的扇区数只有8个，因此会出现读取错误，无法正常运行用户程序。还有一些小的问题，比如说现实用户程序信息的时候，发现屏幕什么字符串都没有了，原来是因为刚好字符串太多，到了下一页的开头，什么都看不到，因此在现实之前先输出几行字符串，把页面压到第二页再显示信息就可以了（挺愚蠢的方法）。当所有问题都解决了之后实验也就完成了，真的感到非常的高兴，虽然很累，但是心中有了很舒缓的感觉。这一次的实验让我发现了在汇编中对内存的掌握是多末的重要。希望能在以后的操作系统课上学习到更多的东西！

**钟凌山 15352443：**这次的操作系统实验的主要内容是进一步了解系统中断的运行机制、分类和调用方法，并用系统中断功能实现操作系统中的一些复杂功能。本次实验主要用到了时钟中断和键盘输入中断，前者实现操作系统默认状态中和用户程序中字母在屏幕上的滚动显示，以及表示操作系统正在正常运行的“-”、“|”等字符的循环显示；后者则用户程序运行时，在键盘有输入的情况下在屏幕上方依次打出“Ouch!”字样。

这些功能虽然只是上次实验的改版，但是却依然耗费了我们大量的时间去编写和调试，主要的原因还是因为对系统中断不是非常熟悉。

**周禅城 15352448：**这次实验真的很难很难啊！！！！！实验4之难，难于上青天！这次实验主要是设置中断与编写中断服务程序，主要指时钟中断与键盘中断。这次实验的过程可谓是一路坎坷。首先，在认真听了老师的理论和实验课之后，我们认为我们对中断的理解应该是比较透彻了，实验应该不会太难完成。可是，残酷的现实啊，我们还是太天真。首先，我们想跑一跑老师ppt上的代码，了解一下基本框架，但是一开始就遇到了玄学问题。我们将老师的代码编译后在DOSBOX上能跑，到了VMware上中断就不发挥作用，整个操作系统卡在中断那里不动。对于这个问题，我们小组讨论了两三天也不知道怎么解决。最后不得已求助老师。老师说那是因为VMware没有初始化中断芯片，然后给了我们一段初始化中断芯片的代码。我们把这段代码加进去之后，发现竟然还是不能跑！！！！！不能跑！！！！！真的是不努力一下，你都不知道什么叫做绝望。最后经过我们的通力合作，花了大概一周时间，终于发现是内存存放位置的问题，剩下的就比较好解决了。真的不得不说。关键时候还是要靠自己学习，希望我们以后能越做越好吧！

## 附录：

**源代码：**

用户程序 ：

prog1.asm

prog2.asm

prog3.asm

prog4.asm

int\_caller.asm

内核代码 ：

os1.asm

cpro.c

引导扇区代码：

loader1.asm

**编译文件：**

prog1.bin

prog2.bin

prog3.bin

prog4.bin

int\_caller.bin

os1.obj

cpro.obj

os1.com

loader1.bin

**镜像文件：**

144mb.img