****

**《操作系统实验》**

**实验报告**

**（实验三）**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学院名称** | **：** | 数据科学与计算机学院 | | | | | |
| **专业（班级）** | **：** | 16计科2班 | | | | | |
| **学生姓名** | **：** | 杨志成 | | | | | |
| **学号** | **：** | 16337281 | | | | | |
| **时间** | **：** | 2018 | 年 | 3 | 月 | 17 | 日 |

**目录**

**一，实验目的-----------------------------------3**

**二，实验要求-----------------------------------3**

**三，实验方案-----------------------------------4**

**（1）基础原理----------------------------4**

**（2）实验工具与环境----------------------7**

**（3）程序流程----------------------------8**

**（4）程序模块功能------------------------9**

**（5）代码文档组成------------------------9**

**四，实验过程及实验结果-------------------------12**

**五，实验总结-----------------------------------15**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | **成绩** | **:** |  |
| **实验三** | **：** | 开发独立内核的操作系统 | | | |

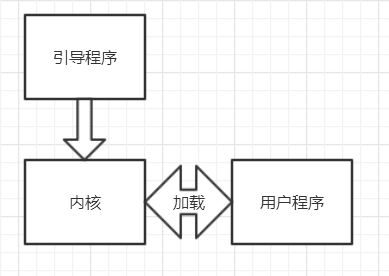
* + - 1. **实验目的**  
          1，把原来在引导扇区中实现的监控程序(内核)分离成一个独立的执行体，存放在其它扇区中，为“后来“扩展内核提供发展空间。  
          2，学习汇编与c混合编程技术，改写实验二的监控程序，扩展其命令处理能力，增加实现实验要求2中的部分或全部功能。
      2. **实验要求**  
          1，规定时间内单独完成实验。  
          2，实验三必须在实验二基础上进行，保留或扩展原有功能，实现部分新增功能。  
          3，监控程序以独立的可执行程序实现，并由引导程序加载进内存适当位星，内核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。  
          4，在指定时间内，提交所有相关源程序文件和软盘映像文件，操作使用说明和实验报告。  
          5，实验报告格式不变，实验方案丶实验过程或心得体会中主要描述个人工作，必须有展示技术性的过程细节截图和说明。

三、实验方案

本次实验我在引导程序中加载内核，然后再去通过内核加载用户程序，加载用户程序是一件不容易的事情，因为用户程序容易发生段超越等情况，导致程序无法正确返回等等。

通过让内核去执行加载用户程序这一操作甚至为以后的进程挂起等操作提供了基础，故这一点十分重要。

程序执行示意图：



（1）实验原理：

1.C语言程序和汇编语言程序间的相互引用：

* 变量互相引用

例如，showstr.asm中12行

mov bp, offset \_Message ; BP=当前串的偏移地址

引用C模块中的字符串变量Message

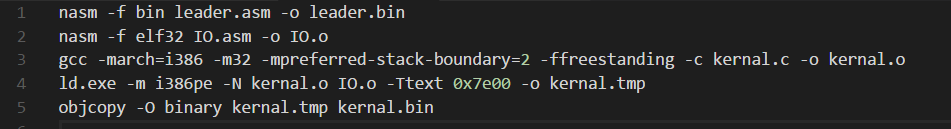
* 过程互相调用

例如， showstr.asm中11行

call near ptr \_upper

引用C模块中的函数upper()

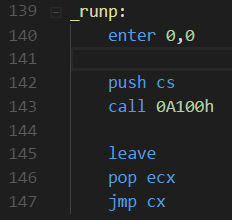
2.使用gcc+nasm编译16位代码的方式：



注意到第三行的一长串指令使用来把c语言编译成.o文件，这样为接下来的链接打基础

此外第四行的链接步骤有一个 0x7e00的数字，这串数字指示把代码以偏移量0x7e00来进行链接。

3.有关程序加载问题，如何将程序加载到指定的位置这的确是一个难题，因为这涉及到处理机控制权从内核到用户程序的问题，而且我们还需要从用户程序将控制权返回给内核，这将是一个比较棘手的问题。实际实验操作中，这一点也卡住了我很久，因为gcc+nasm编译出的16位代码是伪16位，程序在进行call操作时是运用的16位代码，但是c语言函数返回时使用的时32位代码，这就比较容易导致我们用户程序无法返回内核。解决方案如下：

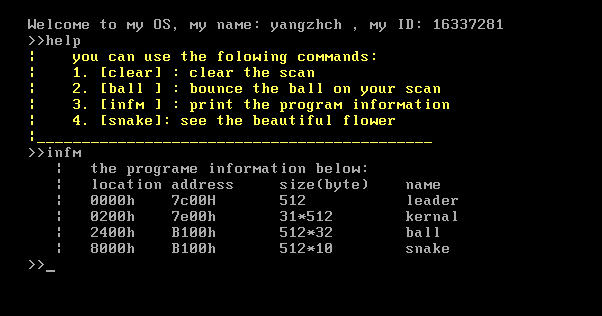


我们在写tunp函数时，只需多push一个cs这样就能把多出的16位push出来，为接下来的

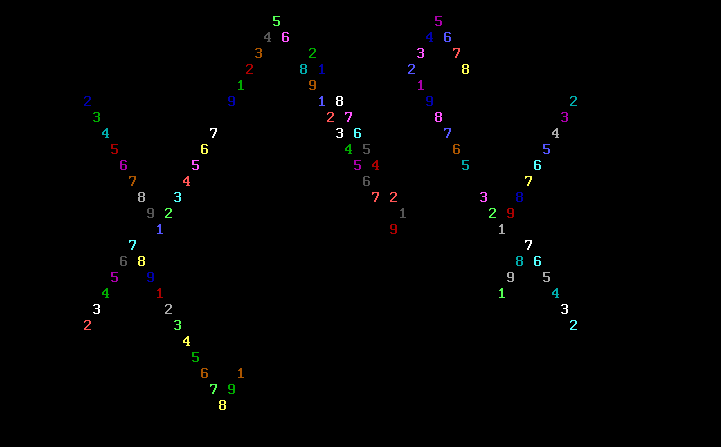
jmp cx打下了正确的基础，使得程序可以跳回内核中继续运行。

现在给出程序运行效果图：

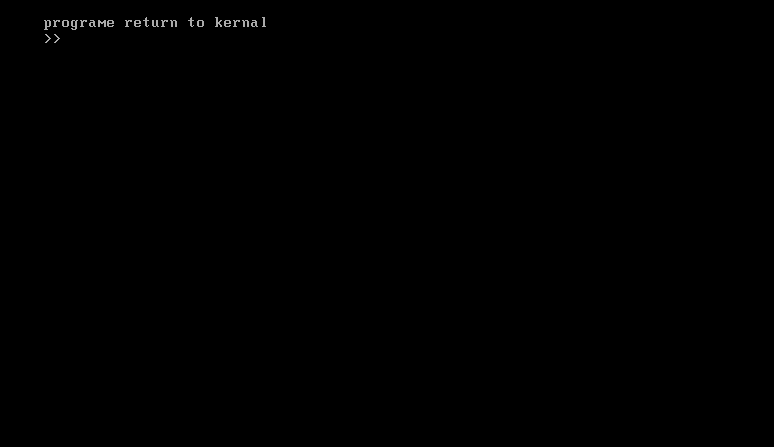
**控制台输入命令**：



其中 location是指在软盘中的位置，adress是指程序加载到内存中的位置



返回内核：



**（2）实验工具和环境**

实验支撑环境

硬件：个人计算机

主机操作系统：Windows/Linux/Mac OS/其它

虚拟机软件：VMware/VirtualPC/Bochs/其它

PC虚拟机裸机/DOS虚拟机/其它

实验开发工具

汇编语言工具：x86汇编语言

高级语言工具：标准c语言

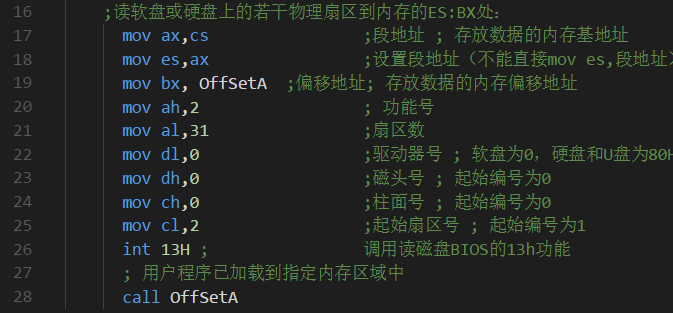
磁盘映像文件浏览编辑工具

调试工具：Bochs

**(3) 程序模块功能与代码文档组成**

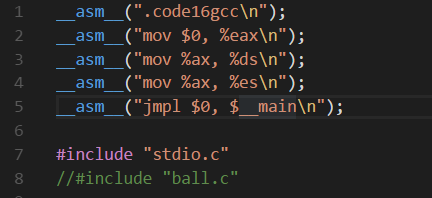
**该程序分为几大模块：**

1.引导程序



引导程序的主要代码是为了将代码引导入内存当中

2.内核：



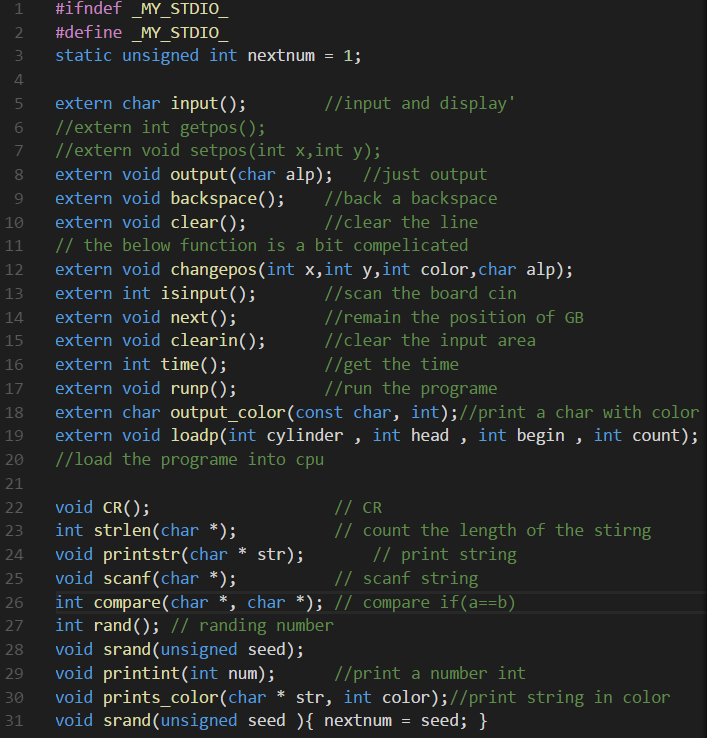
此段代码是为了将程序跳到main函数执行并保护主要的寄存器



此段代码是组成命令行的主要程序，该段代码显示了控制台的主要功能

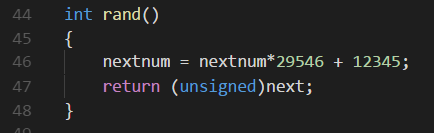
因此是必不可少的

3.输入输出控制模块：



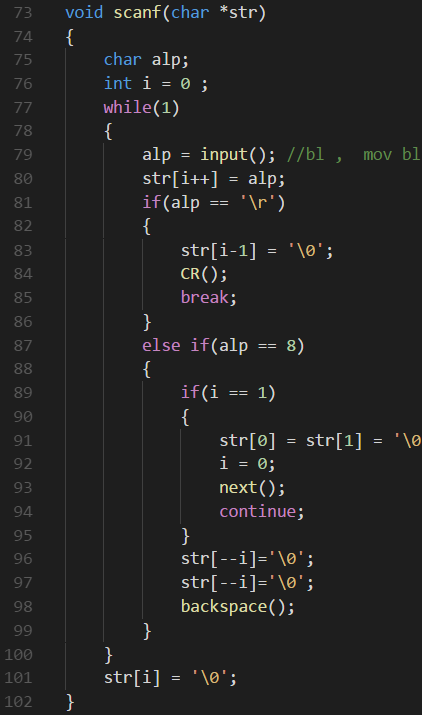
此部分为代码的声明部分，组要声明和引用了一些必要的输入输出函数。

值得介绍的是我在其中自己实现了rand函数：

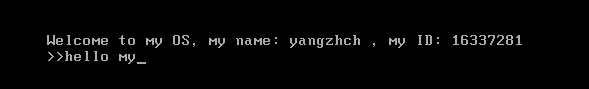
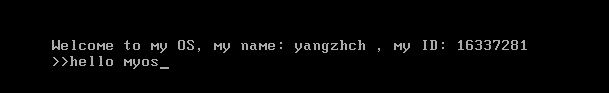


改代码中next值由另一个函数srand决定，srand可以引用时间模块

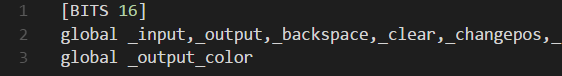
另一个值得一提的是，我的scanfs函数实现了边输入边输出的功能，且可以及时删除：



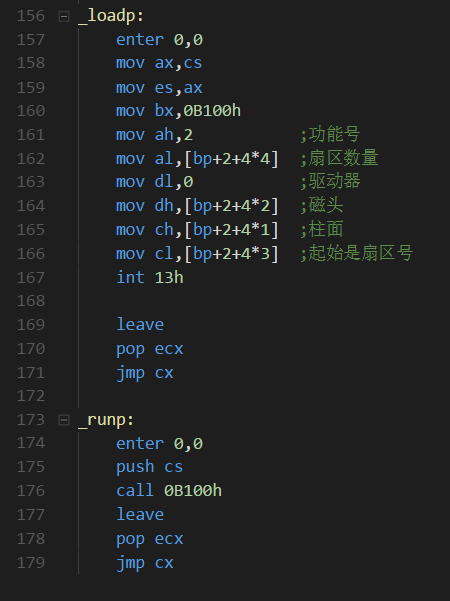
效果如下：先输入一串字符，然后删除一部分



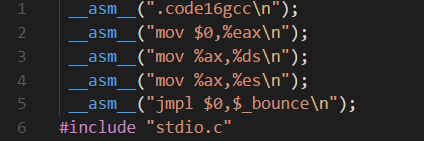
4.汇编语言实现的输入输出接口

声明部分：  
 

Load与run模块：（通过此函数时限内和加载用户程序）

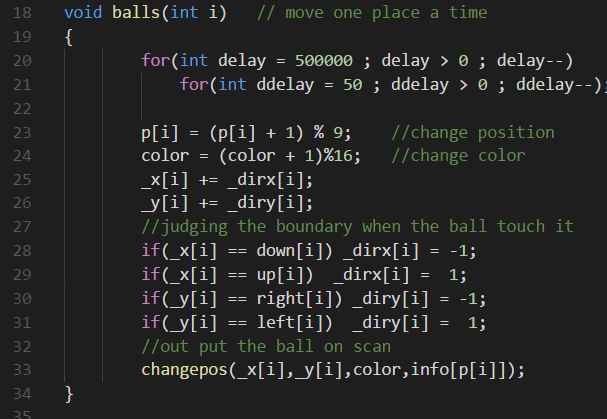


5.用户程序模块：

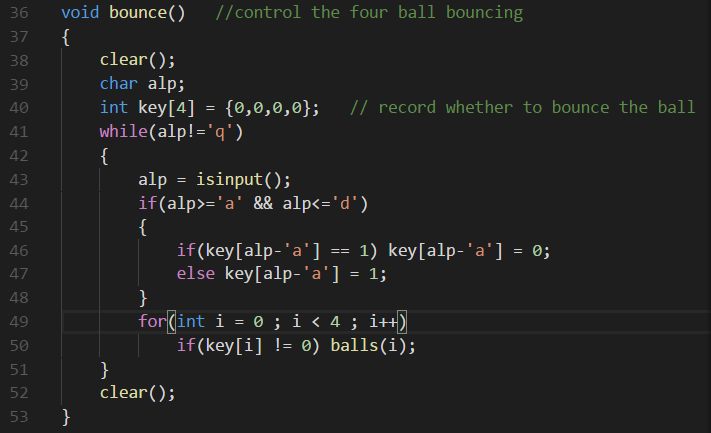


与内核程序相同，开头部分需嵌入一部分汇编语言，这样才能使程序成功运行到我们想要执行的地方。

单个小球的跳动函数：



控制多个小球的函数：



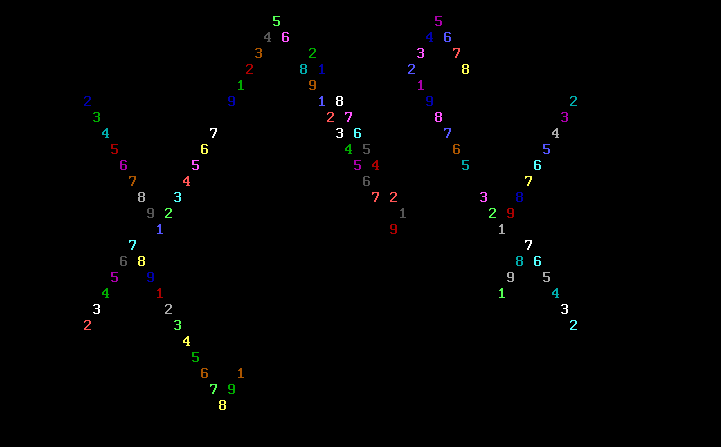
方法：一共有四个程序A,B,C,D，分别控制屏幕四个部分的显示内容与字符弹跳

输入小写a就可以使A程序运行，再输入一次就可以使A程序停止

总之想要哪个程序运动起来就按一下对应的字母键，想要哪个停下就再按下即可

四个程序是可以任意同时运行的。按q返回内核（控制台）

效果如下：



**四、实验创新点**

本次实验有以下几点创新点：  
 1.使用gcc+nasm完成16位代码的编译运行，走通这条路是不容易的，但是这个方法仍有一点点缺陷，那就是它产生的代码不是完全的16位代码，因此在函数返回以及寄存器对齐等等方面会产生很多的麻烦。

2.在用户程序ball中，我实现了四个小球同时跳动，其基本思路是简单分时（与实验一思路基本相同）

程序使用方式：

一共有四个程序A,B,C,D，分别控制屏幕四个部分的显示内容与字符弹跳

输入小写a就可以使A程序运行，再输入一次就可以使A程序停止

总之想要哪个程序运动起来就按一下对应的字母键，想要哪个停下就再按下即可

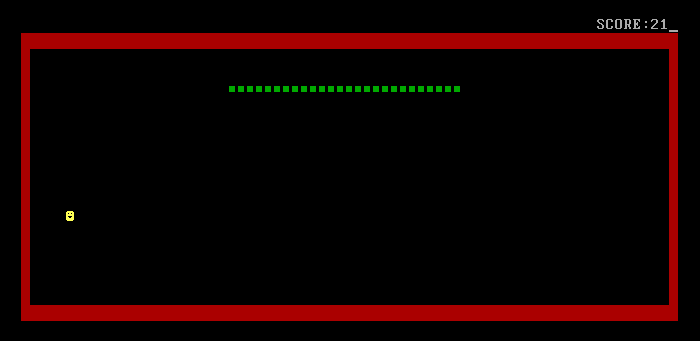
四个程序是可以任意同时运行的。按q返回内核（控制台）

3.实现了加载用户程序这个操作，这次实验有很多人都是直接把用户程序include进内核，这样加载内核时直接把用户程序加载进内存去了，这显然不是一种好的方法，因内核占用了过多内存，出于更加科学的考虑，我们需要使用这种从磁盘加载用户程序到内存指定区域的操作。

4.实现了用户程序snake：贪吃蛇游戏

操作方式：wasd分别控制上下左右即可

效果如图：



5.是用数学方法实现了srand函数，详情见我的代码

**五、实验总结：**

1.本次实验是第三次操作系统试验，相比于前两次的顺利，这次实验显得不那么容易，我花了很多时间在如何找到gcc+nasm编译16位代码的方式上，此外，在内核的架构方面，我也花了很多时间来思考与处理。在用户程序方面，我也花了很大的精力去创新。

接下来我放上我的实验思路图

用户程序创新

内核架构

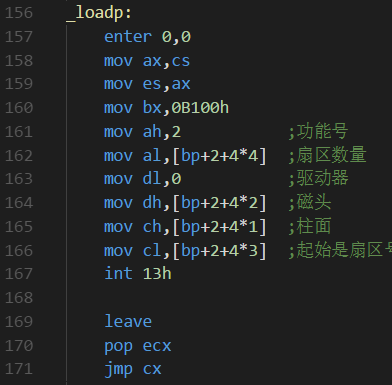
解决gcc+nasm编译16位问题

其实，根据循序渐进这个思路来讲，我觉得很符合操作系统试验课的大体思路：**从简单的程序逐渐进化为一个操作系统**。因此我以后也会将此思想贯通到我的每次实验当中。

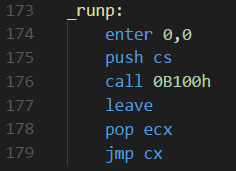
接下来说说**本次试验遇到的一些问题**：

**（1）内核如何加载用户程序并能返回内核？**

我的方法是使用BIOS终端指令int 13h加载软盘中的指定程序到内存中的指定地址即可，这样可以节省一定的内存资源，加载程序的代码如下：



另一个延伸的问题是，我们应该如何运行用户程序并返回内核？我在io.asm文件中开发了一个runp函数，它是专门用来控制进入用户程序与返回内核的。 代码如下:



这里我遇到了一个问题，那就是用户程序无法返回内核。经过bochs调试我才知道，原来这里是因为我的gcc+nasm生成的代码是伪16位代码，当call程序时使用的是16位， 但是当我返回时，它确使用的是32位，为了抵消多出来的这16位代码，我决定在ret前先多push一个cs进入堆栈，这样就完美抵消了伪16位代码的缺陷。

**（2）如何将小球同时弹跳**

我们都知道，在单核CPU中，要执行两个程序是比较困难的，需要保护环境，以及分时处理等等操作，但是很巧的事情是，在本次实验当中，我们子程序的运行并不需要保护环境这个操作，只需要将四个子程序进行简单分时处理即可。即为先处理一个程序接下来处理另外一个程序，周期性进行。即将处理机控制权在四个程序之间不停切换就能实现。

**（3）有哪些汇编语言的io接口需要我实现？**

通过观察同学的代码，我发现大部分同学都是在io接口中实现了汇编语言输出字符串的操作，但是我走了一个捷径，那就是我用c语言写了一个函数，专门用来输出字符串，其实现原理是通过不断调用output汇编函数一个一个输出字符，这样结合节省了我的代码量。

此外，为了贪吃蛇游戏和小球弹跳等用户程序，我还实现了changepos函数，它是使用了int 10h的各种中断来控制在屏幕的某个地方实现有颜色的字符输出。

**（4）操作系统的可扩展性**

这是一个让我十分头疼的问题，因为这次实验给了两个星期的时间。所以在第二个星期我花了一些时间重新组织我的代码，我决定了c语言编写内核，include一个io文件进行操作，就这样完成了可扩展性的内核。若以后的实验还需要改进，我一定会再次组织我的代码。

**2.实验感想**

本周的实验是编写一个内核，虽然是一个十分基础的内核，主要是实现了一些控制台命令并能引导用户程序，但是因为相对于前两次实验跨度较大，我还是费了不少功夫。尤其是在gcc+nasm编译16位代码的方面，我在网上搜索了很多资料，花了大量的时间阅读，才终于成功实现了gcc+nasm这套工具的使用。

另外的，此套工具并不是完美的，他有时候会引发很多问题，比如我在上面所说的用户程序调用与返回的问题，这引发了许多麻烦。不过还是被我解决了。

关于实验的创新点，我也花了一定的时间，比如实现一个四小球同时弹跳的函数，以及srand和rand函数，还有我写的贪吃蛇程序，这都是不很容易实现的，花了一定的功夫！

不知不觉已经写了三次操作系统试验了，我也感受到了操作系统实验的难度，理论与实践的结合。虽然理论课感觉操作系统知识并不很难，但是在真正实现操作系统时遇见的问题是有很多的。纸上得来终觉浅，以后的实验我应该更要认真对待，争取这学期实现一个让老师满意的操作系统