

**实验报告**



**题目： 键盘驱动程序的分析与修改**

**班 级： 2020211310**

**学 号： 2020211502**

**姓 名： 王小龙**

**学 院： 计算机学院**

**2021年 12 月 19 日**

一、实验目的  
1.理解I/O系统调用函数和C标准I/O函数的概念和区别；

2.建立内核空间I/O软件层次结构概念，即与设备无关的操作系统软件、设备驱动程序和中断服务程序；

3.了解Linux-0.11字符设备驱动程序及功能，初步理解控制台终端程序的工作原理；

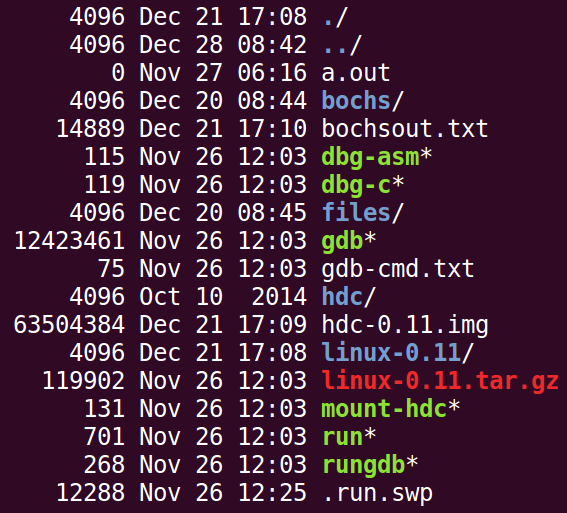
4.通过阅读源代码，进一步提高C语言和汇编程序的编程技巧以及源代码分析能力；

5.锻炼和提高对复杂工程问题进行分析的能力，并根据需求进行设计和实现的能力。

1. 实验环境
2. 硬件：学生个人电脑（x86-64）
3. 软件：Windows 10，VMware Workstation 15 Player，32位Linux-Ubuntu 16.04.1
4. gcc-3.4编译环境
5. GDB调试工具

三、实验内容

从网盘下载lab4.tar.gz文件，解压后进入lba4目录得到如下文件和目录：



实验常用执行命令如下：

* 执行./run ，可启动bochs模拟器，进而加载执行Linux-0.11目录下的Image文件启动linux-0.11操作系统
* 进入lab4/linux-0.11目录，执行make编译生成Image文件，每次重新编译（make）前需先执行make clean
* 如果对linux-0.11目录下的某些源文件进行了修改，执行./run init 可把修改文件回复初始状态

本实验包含2关，要求如下：

* Phase 1

键入F12，激活\*功能，键入学生本人姓名拼音，首尾字母等显示\*

比如：zhangsan，显示为：\*ha\*gsa\*

* Phase 2

键入“学生本人学号” ：激活\*功能，键入学生本人姓名拼音，首尾字母等显示\*

比如：zhangsan，显示为：\*ha\*gsa\*，

再次键入“学生本人学号-” ：取消显示\*功能

提示：完成本实验需要对lab4/linux-0.11/kernel/chr\_drv/目录下的keyboard.s、console.c和tty\_io.c源文件进行分析，理解按下按键到回显到显示频上程序的执行过程，然后对涉及到的数据结构进行分析，完成对前两个源程序的修改。修改方案有两种：

* 在C语言源程序层面进行修改
* 在汇编语言源程序层面进行修改

实验4的其他说明见lab4.pdf课件和爱课堂中虚拟机环境搭建相关内容。linux内核完全注释(高清版).pdf一书中对源代码有详细的说明和注释。

四、源代码的分析及修改

准备工作：

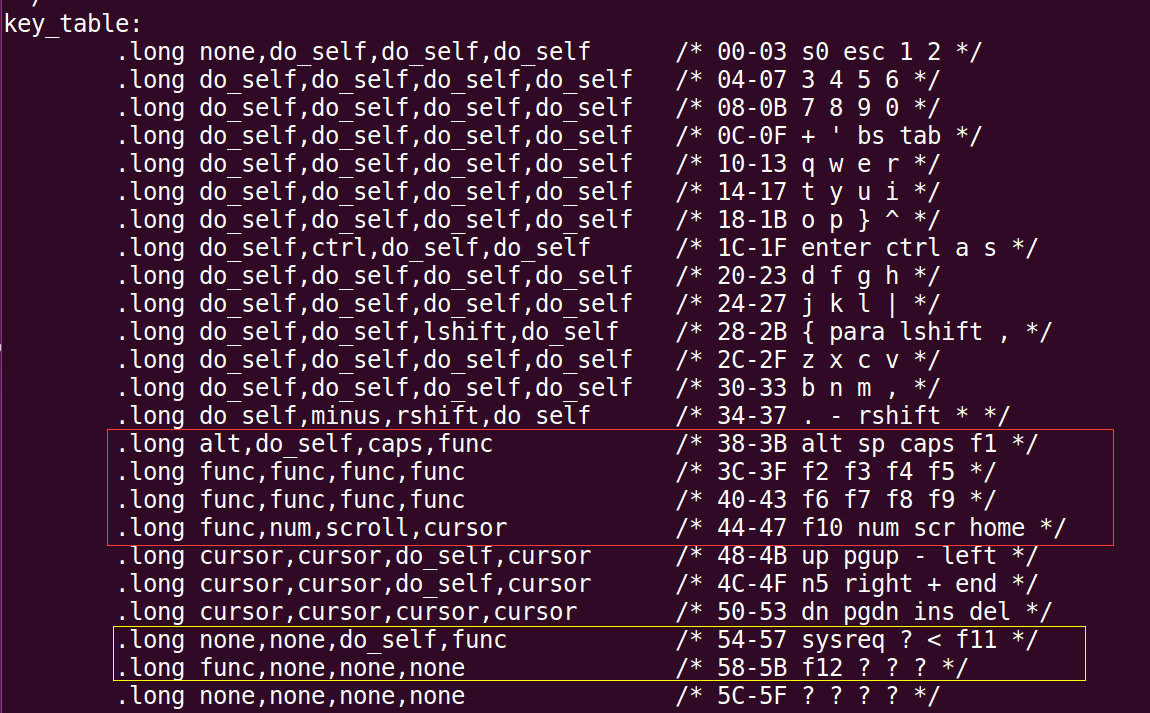
在互联网上下载并在x86-64电脑上安装虚拟机软件VMware Workstation 15 Player，然后利用虚拟机软件安装了32位的Linux-Ubuntu 16.04.1。接下来根据实验提示安装了gcc-3.4工具链和as86汇编程序。

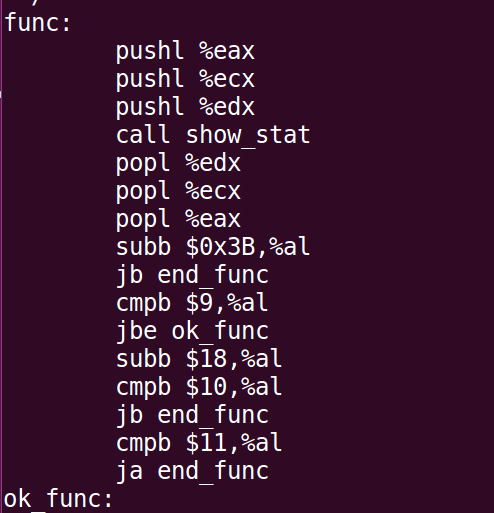
将提供的“lab4.tar.gz”文件传输至虚拟机的存储空间，然后利用“tar –xvf lab4.tar.gz”进行文件的解压。

阶段一：

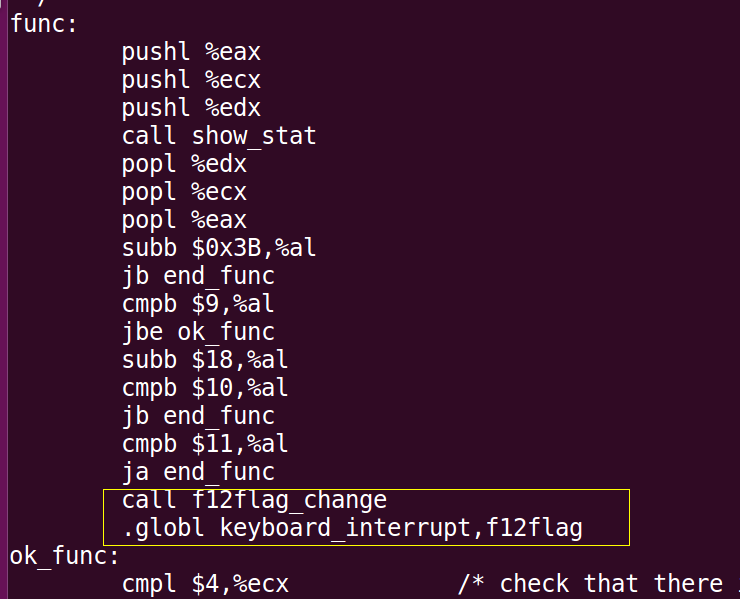
对于我而言，名字拼音为“wangxiaolong”，故需要修改的是“w”和“g”字符，使其开启功能后回显为“\*”

在Linux-0.11目录下的/kernel/chr\_drv/keyboard.S源代码文件中查阅“key\_table”，可知键盘上的F1-F12的扫描码使用函数func()进行处理，于是查看func()函数的具体汇编代码。如下所示：





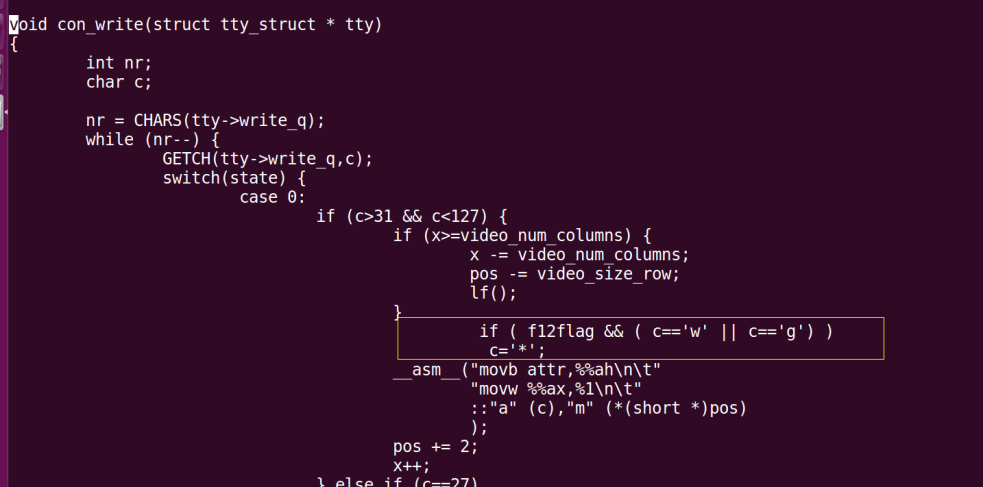
通过分析知需添加如下汇编代码，使其跳转至我们欲实现的功能代码上：



下面我们需要使用C语言代码编写以下代码并将其插入到“console.c”文件中，如下图所示：

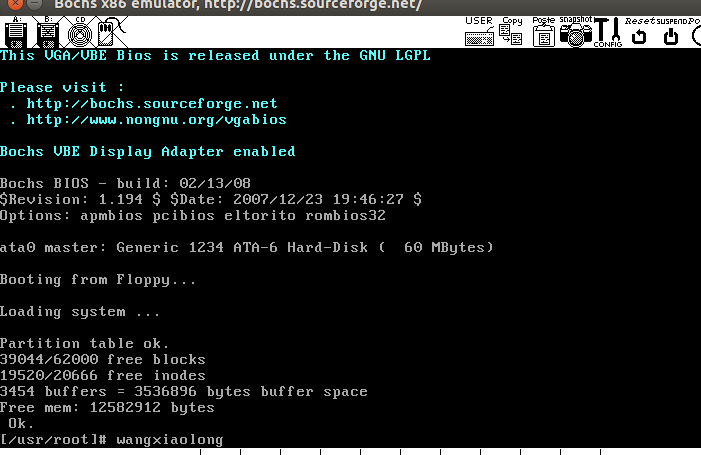
![Q)P2Z2%M$CX](9EI1)KOU9F](data:image/png;base64,)

由于欲修改回显字符为“w”与“g”，因此可编写如下C代码并插入到函数“con\_write”中，将字符c在写到内存pos处前，将其更改为\*即可：

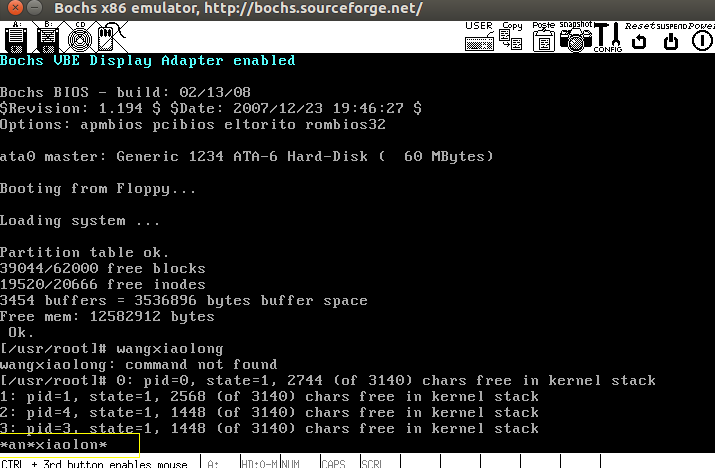


下面就可以在Linux-0.11目录下分别使用“make clean”与“make all”命令对修改过的内核进行编译。并在lab4目录下执行“./run”命令，开始对实验的理论分析进行验证。

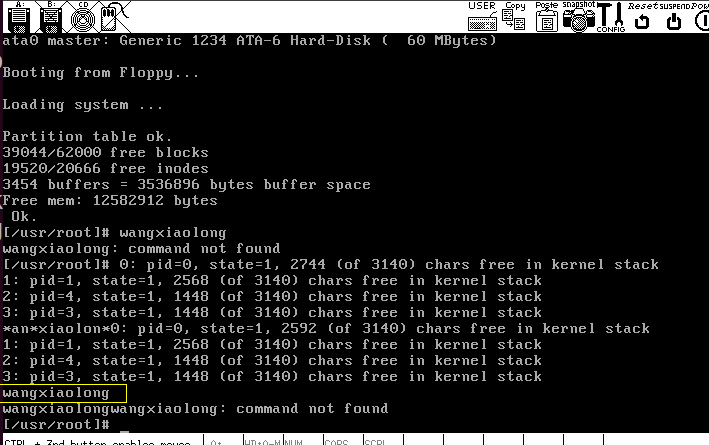
使用“Bochs x86 emulator”运行linux-0.11，先键入“wangxiaolong”，我们可以看到目前的显示是完全正常的。在初始化时理论上该功能也是处于关闭状态，这是符合之前的设计的。如下图所示：



按下F12后，再次输入“wangxiaolong”，我们可以看到所有回显字符中的“w”与“g”全部被修改为“\*”。如下图所示：



再次按下F12后，输入“wangxiaolong”，我们可以看到功能已被关闭，回显的字符完全正常。如下图所示：



到此，本次实验的第一个任务已顺利完成。接下来开始对实验中的第二个任务进行分析与操作。

阶段二：

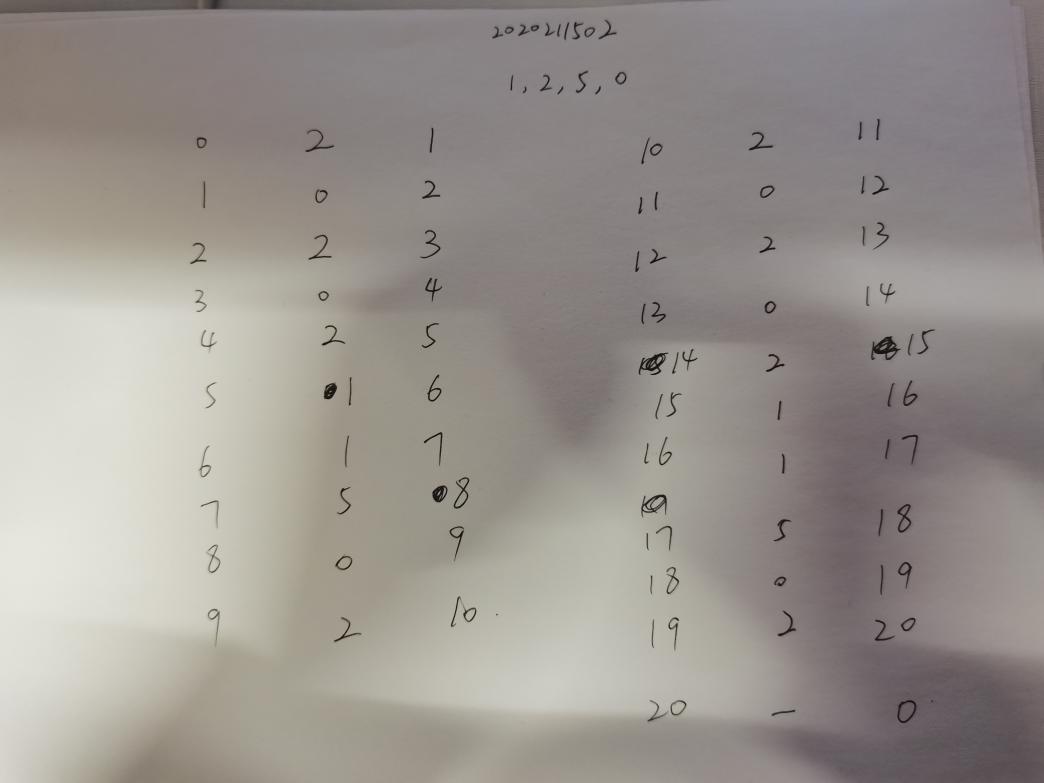
对于我而言，名字拼音为“wangxiaolong”，故需要修改的是“w”和“g”字符，使其开启功能后回显为“\*”。

同样对于我而言，输入“2020211502”即为“\*”功能开启的条件；而输入“2020211502-”即为“\*”功能关闭的条件。

对欲实现的功能制定如下算法策略：

① 设定全局变量“flag”作为标志位，初始值为0。当其值大于等于10时，“\*”功能开启，否则关闭。

② 由于实验者的学号中仅包含数字“1”、“2”、“5”、“0”，故只需对上述的4个数字键与“-”键进行热键功能的设计即可。我们可设计21个状态使其达到我们想要的效果，具体如下：



由于待检测的按键序列为“2020211502”与“2020211502-”，因此我们可对此设计21个状态，当用户按照规则键入按键时，flag的值将会在这21个值之间跳转，代表在21个状态之间跳转。

1）初始状态下，flag=0。当按下“2”时，flag将会跳转至“1”；

2）当flag=1时。按下“0”，flag将会跳转至“2”；

3）当flag=2时。按下“2”，flag将会跳转至“3”；

···

10）当flag=9时。按下“2”，flag将会跳转至“10”；

在用户键入“2020211502”后，flag的值达到了10，那么根据上述的设计思路，当flag的值大于等于10时，“\*”功能开启，用户输入的字符中，所有“w”与“g”都将被替换为“\*”；

接下来，

11）当flag=10时。按下“2”，flag将会跳转至“11”；

12）当flag=11时。按下“0”，flag将会跳转至“12”；

13）当flag=12时。按下“2”，flag将会跳转至“13”；

···

20）当flag=19时。按下“2”，flag将会跳转至“20”；

21）当flag=20时。按下“-”，flag将会跳转至“0”；

在用户键入“2020211502-”后，flag的值将会回到0，也即在此刻关闭“\*”功能

下面将其以C语言代码实现。具体设计的C代码为：

extern int flag=0;

void one\_change()

{

   if(flag==5)     flag=6;

   else if(flag==6)  flag=7;

  else if(flag==15)  flag=16;

else if(flag==16)  flag=17;

}

void two\_change()

{

   if(flag==0)     flag=1;

   else if(flag==2)  flag=3;

   else if(flag==4)  flag=5;

  else if(flag==9)  flag=10;

else if(flag==10)  flag=11;

else if(flag==12)  flag=13;

else if(flag==14)  flag=15;

   else if(flag==19)  flag=20;

}

void five\_change()

{

   if(flag==7)     flag=8;

   else if(flag==17) flag=18；

}

void zero\_change()

{

   if(flag==1)     flag=2;

   else if(flag==3)  flag=4;

else if(flag==8)  flag=9;

   else if(flag==11) flag=12;

  else if(flag==13) flag=14;

else if(flag==18) flag=19;

}

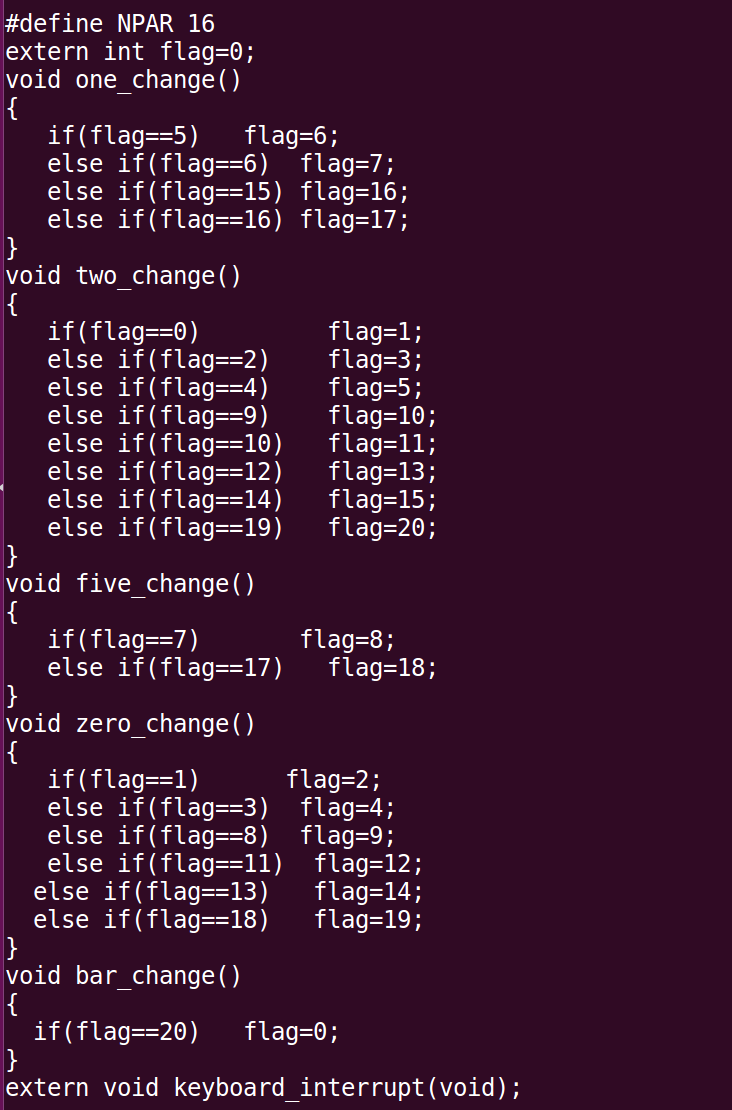
void bar\_change()

{

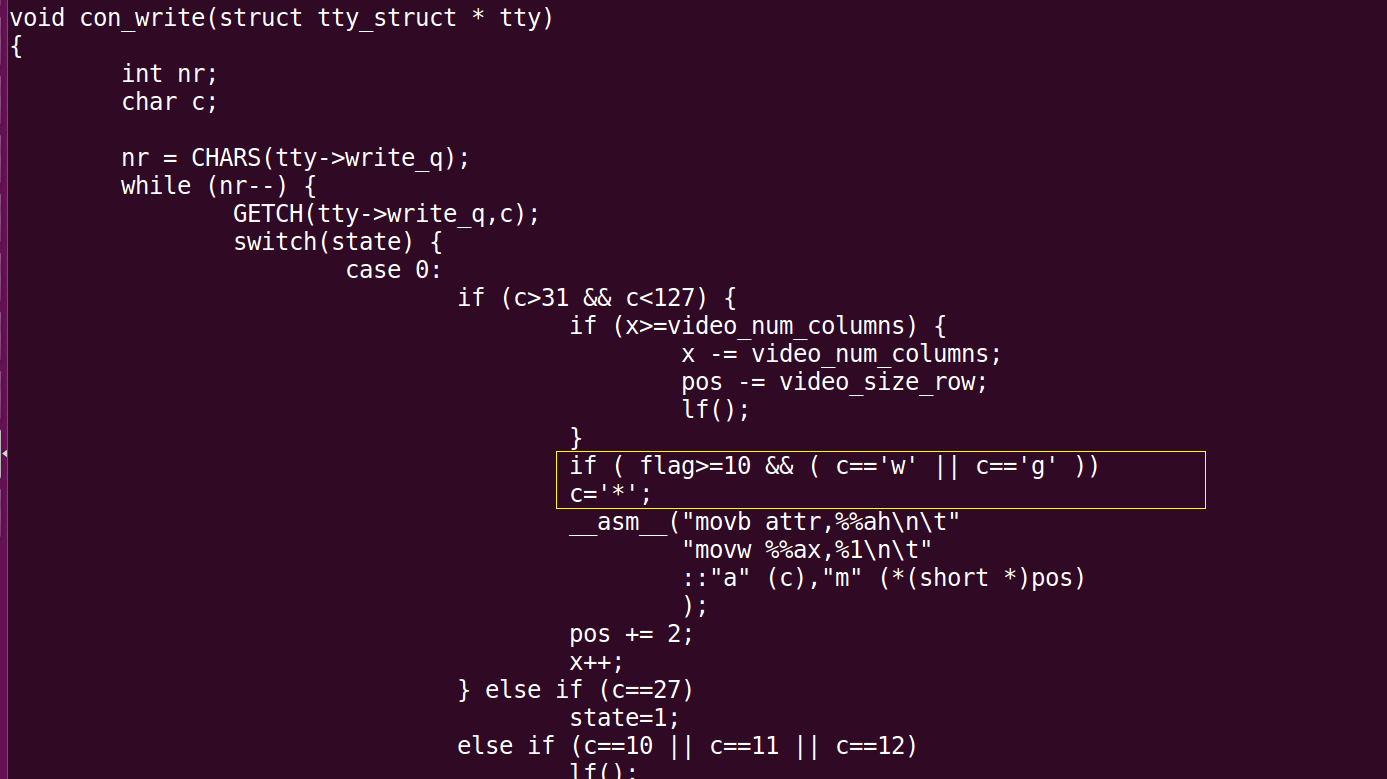
   if(flag==20)    flag=0;

}

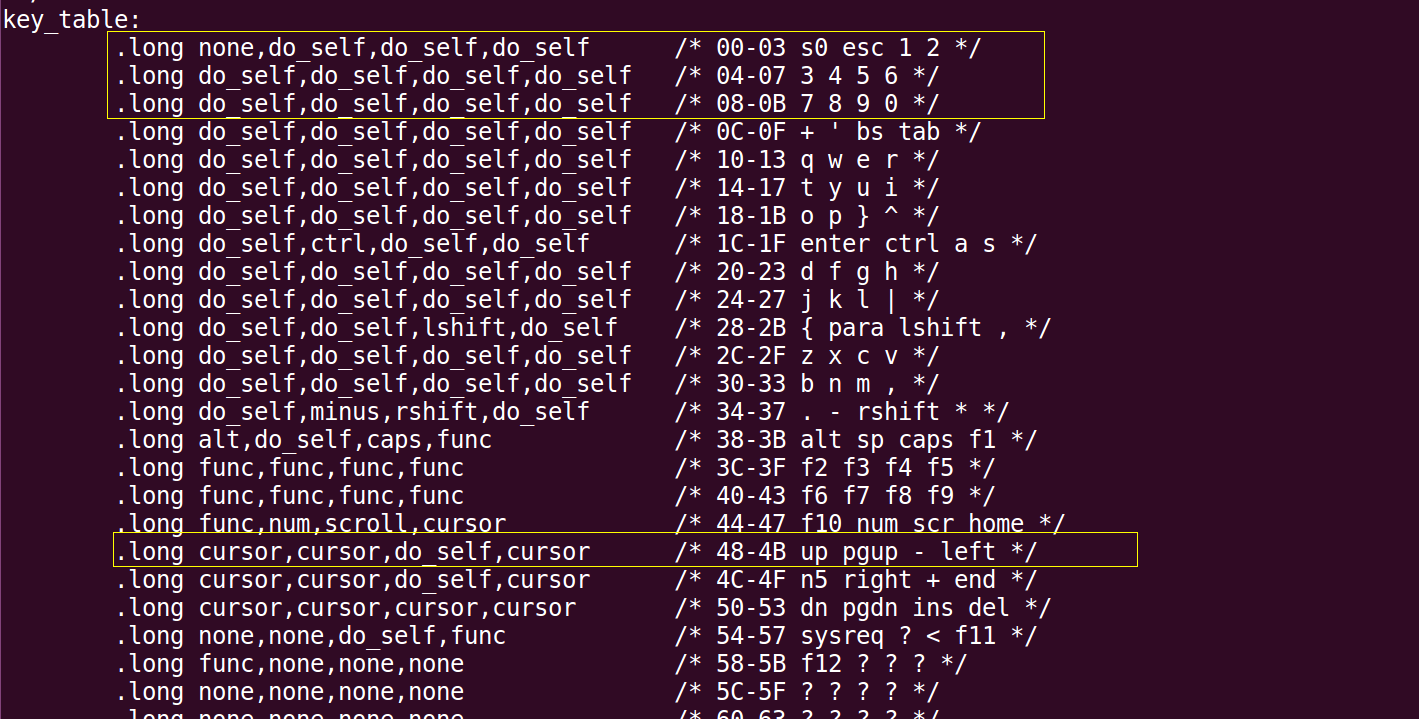
为了实现目的，设计了5个函数：“one\_change”、“two\_change”、“five\_change”、“zero\_change”与“bar\_change”。分别对应按下按键“1”、“2”、“5”、“0”与“-”时状态的改变。接下来将其插入至“console.c”文件中，如下图所示：



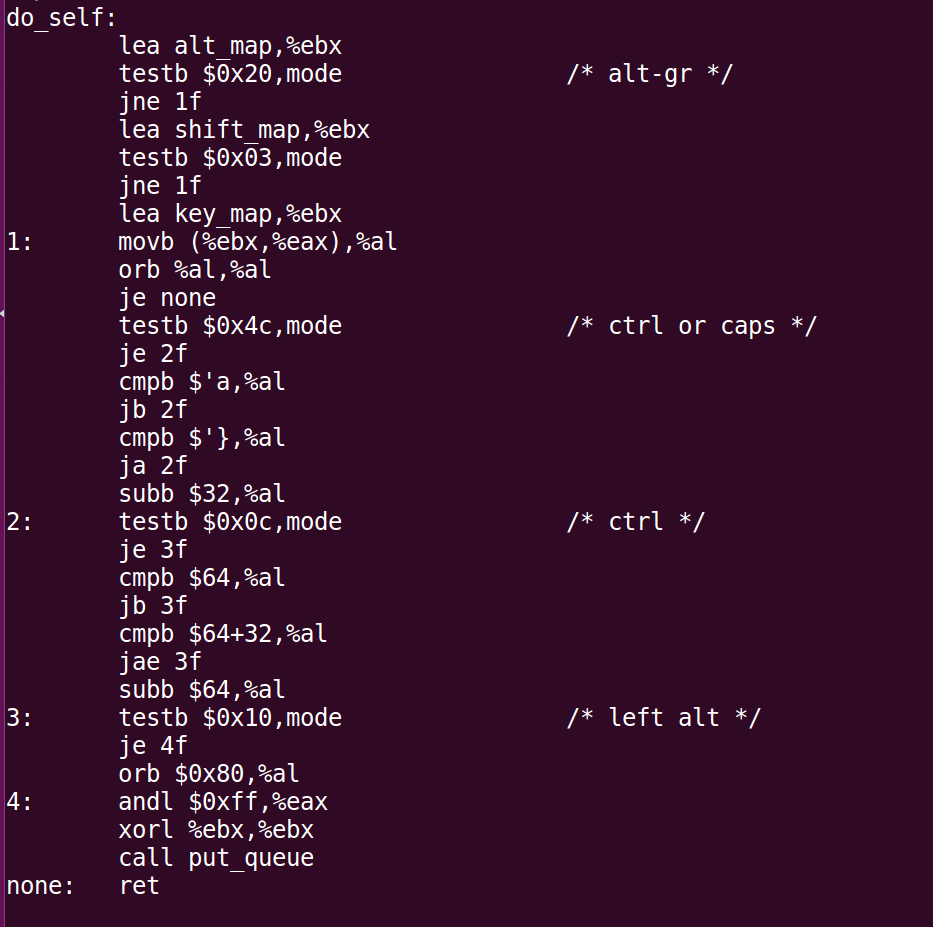
上文中提到过，我们欲使当flag大于等于10，即用户已经完成了“2020211502”的输入后，“\*”功能开启。因此类似任务一，我们可在con\_write()函数中插入如下所示的代码：



再次查找“keyboard.S”文件中的“key\_table”，我们可发现，数字键0-9以及按键“-”的扫描码均由函数“do\_self”处理。如下所示：



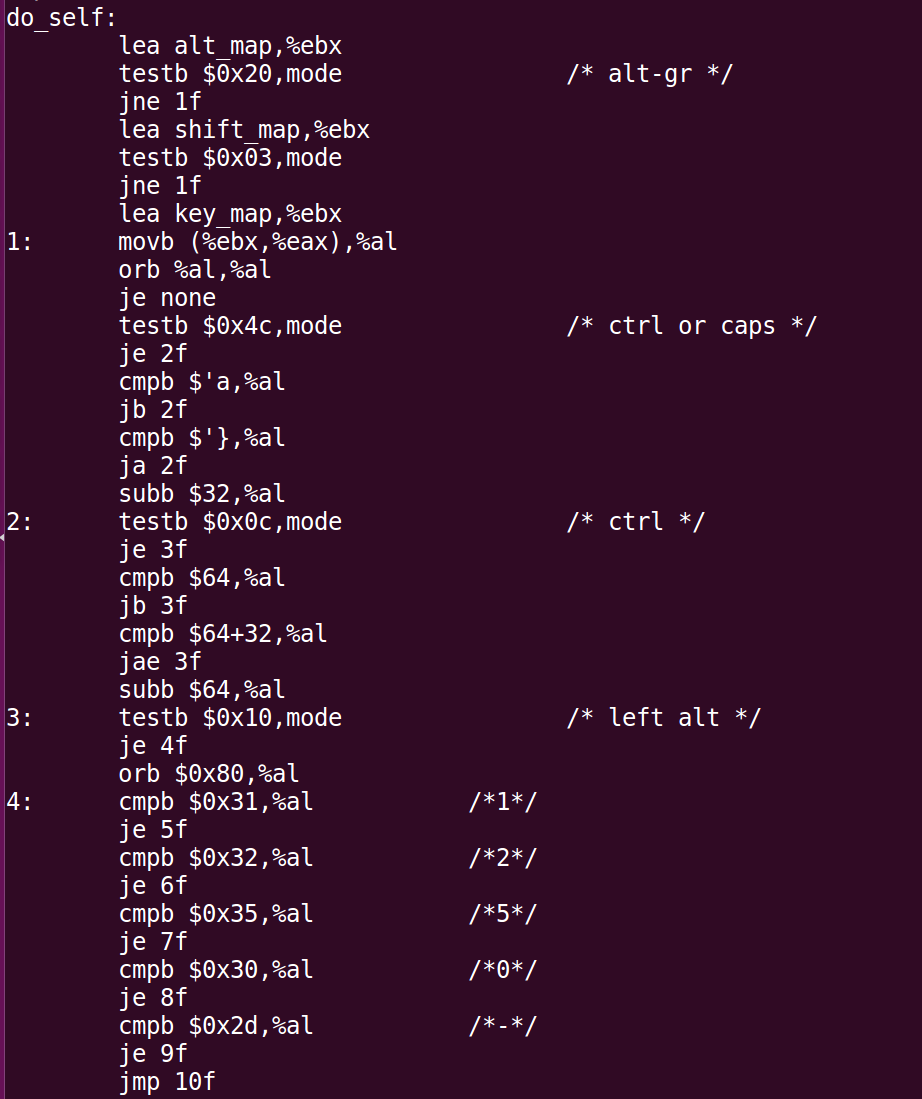
找到“keyboard.S”中的“do\_self”函数汇编代码段，如下所示：

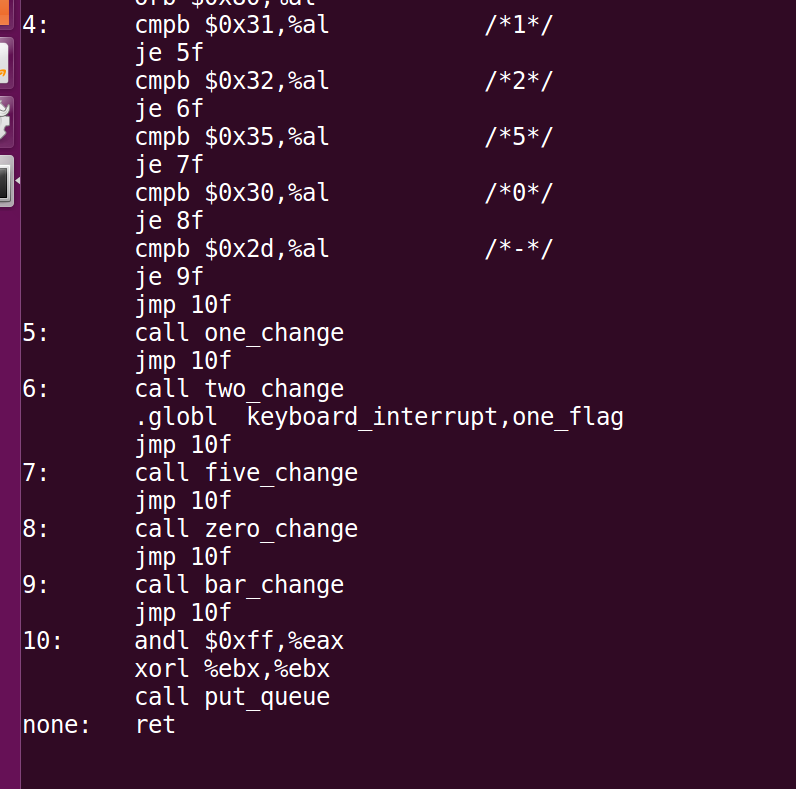


对该段汇编代码进行仔细分析不难看出，在程序开始时，首先根据mode标志选择alt\_map、shift\_map或key\_map映射表之一，然后根据扫描码取映射表中对应的ASCII字符。若没有对应字符，则返回（跳转至none）；若ctrl键已按下或caps键锁定，并且字符存在‘a’-‘}’(0x61 – 0x7D)的范围内，则将其转成大写字符；若ctrl键已经按下，并且字符在‘`’-‘\_’(0x40 – 0x5F)之间，即是大写字符，则将其转换成控制字符；若左alt键同时按下，则将字符的位7置位。即此时生成值大于0x7F的扩展字符集中的字符。在程序运行至‘4’之后，其将会把al中的字符放入读缓冲队列中。

由以上的分析可知，若我们只考虑正常情况下学号的键入，则仅需对‘4’中以及之后的代码进行修改，使得当指定按键被按下时，能够跳转至我们需要的函数并开始运行即可。因此可得出本关的解题策略为：

在‘4’中，将寄存器%al中的值分别于‘1’、‘2’、‘5’、‘0’和‘-’的ASCII码进行比较。若按下的键为其中之一，则跳转至指定的函数上。下面是根据此策略对“do\_self”函数进行修改得到的汇编源代码：





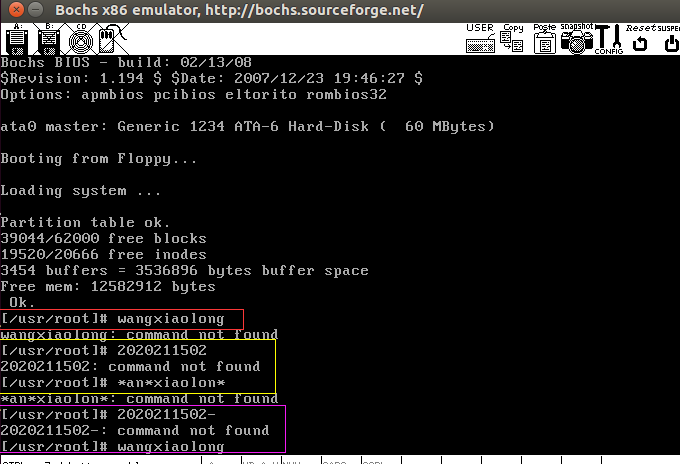
设计这段代码的目的在于，当判断按键为‘1’时，程序将会跳转至函数“one\_change”，当该函数运行结束后，程序将会跳转至‘11’处继续运行，并且‘11’处之后的代码为原汇编代码，其他部分代码的功能与此类似。

接下来在Linux-0.11目录下分别使用“make clean”与“make all”命令对修改过的内核进行编译。并在oslab目录下执行“./run”命令，我们开始对实验的理论分析进行验证。

使用“Bochs x86 emulator”运行linux-0.11，先键入“wangxiaolong”，我们可以看到目前的回显是完全正常的。

在初始化时理论上该功能也是处于关闭状态，这是符合之前的设计的。而在键入“2020211502”后，我们可以观察到，所有回显的字符中，‘c’与‘n’均被替换为‘\*’；

而在键入“2020211502-”后，该功能被关闭，回显自动恢复至正常状态,如下图：



综上，本次实验的第二个任务已完成。

五、总结体会

总共大约花费了一下午和半个晚上的时间，总体来说是很有难度的，并且后一关的难度比第一关要大很多，此次实验的过程虽然很困难，但最终使也我获益良多。