

**实验报告**



**题目： 键盘驱动程序的分析与修改**

**班 级： 2023211307**

**学 号： 2023211088**

**姓 名： 罗浩**

**学 院： 计算机学院**

**2024年 12 月 8 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；

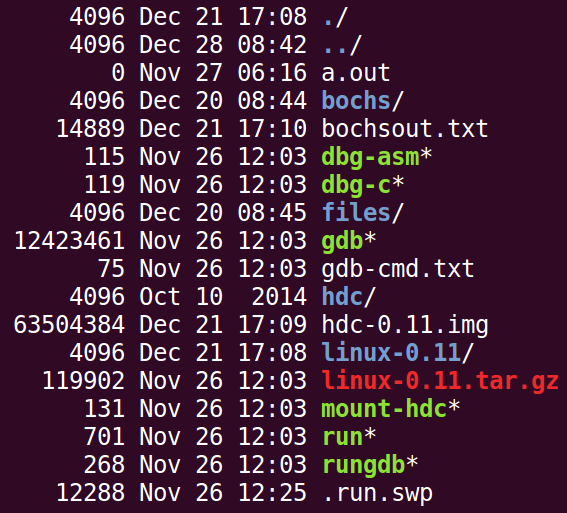
5.锻炼和提高对复杂工程问题进行分析的能力，并根据需求进行设计和实现的能力。

# 二、实验环境

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

针对一次按键操作对源代码keyboard.s、console.c和tty\_io.c的进行分析，说明分析过程，要配有流程图（不能从书中进行截图）进行说明，给出各阶段的修改思路和代码实现。各阶段需要有较详细的文字说、运行截图、分析过程的内容。

### 按键操作分析

要深入分析一次按键操作在 Linux 0.11 中是如何在源代码中处理的，我们需要追溯键盘输入的流程，分析源代码文件 keyboard.s、console.c 和 tty\_io.c 的协作方式。这些文件涉及到输入设备的管理、终端输入的处理以及字符设备的输出等内容。

**前提：**

键盘驱动的核心在于tty结构体。

根据Linux-0.11的Linux内核完全注释，我们可以知道在字符驱动程序中，每个tty\_struct结构体都对应一个终端设备，例如控制台（console），也即计算器显示器就是其中一个终端设备，对应一个tty\_struct结构体（其中，这样的结构体在linux-0.11/linux/inlcude文件夹中的tty中定义）。以下列出其定义：

**struct tty\_struct** {

struct termios termios;

int pgrp;

int stopped;

void (\*write)(struct tty\_struct \* tty);

struct tty\_queue read\_q;

struct tty\_queue write\_q;

struct tty\_queue secondary;

};

extern struct tty\_struct tty\_table[];

**struct tty\_queue** {

unsigned long data;

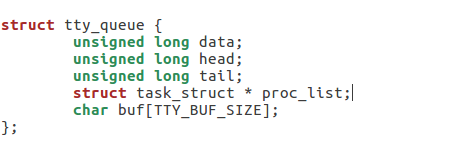
unsigned long head;

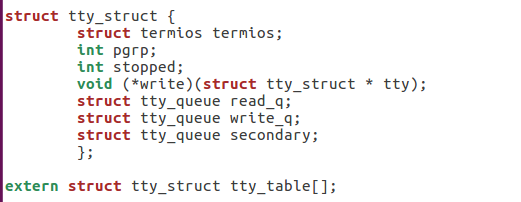
unsigned long tail;

struct task\_struct \* proc\_list;

char buf[TTY\_BUF\_SIZE];

};





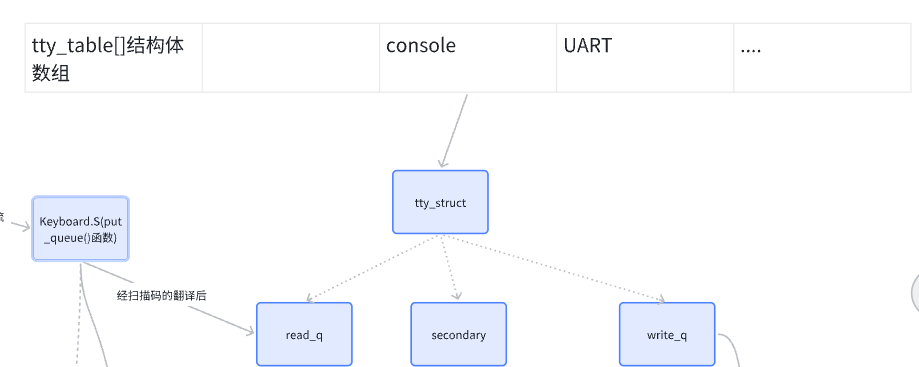
其中tty\_table用于保存系统中每个终端设备的信息。

终端所处理的数据被保存在3个tty\_queue结构的字符缓冲队列中(或称为字符表)。

每个字符缓冲队列的长度是1K字节。其中读缓冲队列read\_q用于临时存放从键盘或串行终端输入的原始(raw)字符序列;写缓冲队列write\_q用于存放写到控制台显示屏或串行终端去的数据;根据ICANON标志，辅助队列secondary用于存放从read\_q中取出的经过行规则程序处理(过滤)过的数据或称为熟(cooked)模式数据。这是在行规则程序把原始数据中的特殊字符如删除(backspace)字符变换后的规范输入数据，以字符行为单位供应用程序读取使用。上层终端读函数tty\_read()即用于读取secondary队列中的字符。

在读入用户键入的数据时，中断处理汇编程序只负责把原始字符数据放入输入缓冲队列中，而由中断处理过程中调用的C函数(copy\_to\_cooked())来处理字符的变换工作。例如当进程向一个终端写数据时，终端驱动程序就会调用行规则函数copy\_to\_cooked()，把用户缓冲区中的所有数据数据到写缓冲队列中，并将数据发送到终端上显示。在终端上按下一个键时，所引发的键盘中断处理过程会把按键扫描码对应的字符放入读队列read\_q中,并调用规范模式处理程序把read\_q中的字符经过处理再放入辅助队列 secondary 中。与此同时,如果终端设备设置了回显标志(LECHO),则也把该字符放入写队列 write\_q中，并调用终端写函数把该字符显示在屏幕上。通常除了像键入密码或其他特殊要求以外，回显标志都是置位的。我们可以通过修改终端的termios 结构中的信息来改变这些标志值。

流程图：



**1. keyboard.s - 键盘中断处理与扫描码**

键盘的输入是通过键盘控制器向 CPU 发送中断信号进行的。按下每一个键时，键盘会产生一个扫描码，这个扫描码会被中断服务程序（ISR）处理。

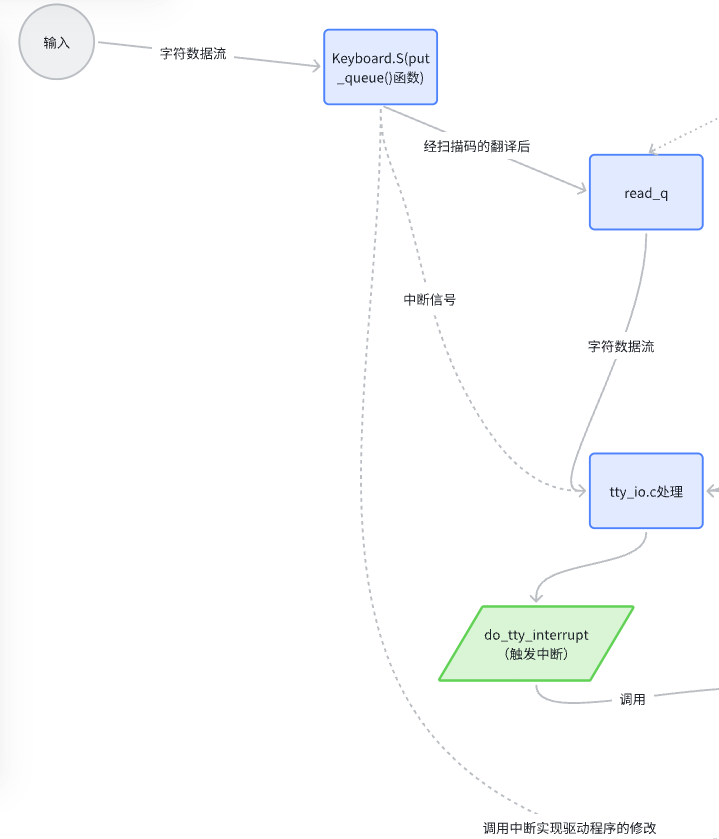
keyboard.s主要用于把用户键入的字符放入read\_q缓冲队列中

关键流程

1.键盘中断：当按下键盘上的任意按键时，键盘控制器会产生一个中断信号，CPU 会跳转到键盘中断处理程序。

2.keyboard.s 文件：这个汇编文件中包含了处理键盘中断的代码。该文件负责从键盘控制器读取扫描码并将其传递给内核处理。

流程图：



**2. console.c - 显示与输入处理**

本文件是内核中最长的程序之一，但功能比较单一。其中的所有子程序都是为了实现终端屏幕写函数con\_write()以及进行终端屏幕显示的控制操作。

当往一个控制台设备执行写操作时，就会调用con\_write()函数。这个函数管理所有控制字符和换码字符序列，这些字符给应用程序提供全部的屏幕管理操作。

函数con\_write()主要由转换语句组成，用于每次处理一个字符的有限长状态自动转义序列的解释。在正常方式下，显示字符使用当前属性直接写到显示内存中。该函数会从终端tty\_struct结构的写缓冲队列 write\_q中取出字符或字符序列，然后根据字符的性质(是普通字符、控制字符、转义序列还是控制序列)，把字符显示在终端屏幕上或进行一些光标移动、字符除等屏幕控制操作。

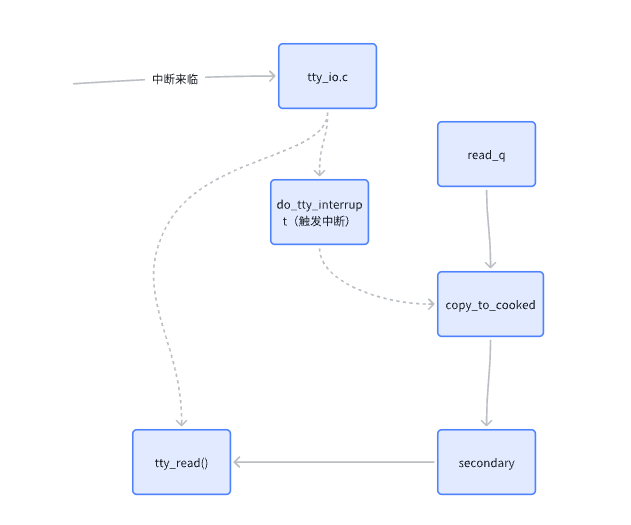
终端屏幕初始化函数 con\_init()会根据系统初始化时获得的系统信息，设置有关屏幕的一些基本参数

值，用于con\_write()函数的操作。

**3. tty\_io.c - 终端输入输出**

每个tty设备有3个缓冲队列，分别是读缓冲队列(read\_q)、写缓冲队列(write\_q)和辅助缓冲队列(secondary)，定义在tty\_struct 结构中(include/linux/tty.h)。对于每个缓冲队列，读操作是从缓冲队列的左端取字符，并且把缓冲队列尾(tail)指针向右移动。而写操作则是往缓冲队列的右端添加字符，并且也把头(head)指针向右移动。这两个指针中，任何一个若移动到超出了缓冲队列的末端，则折回到左端重新开始。

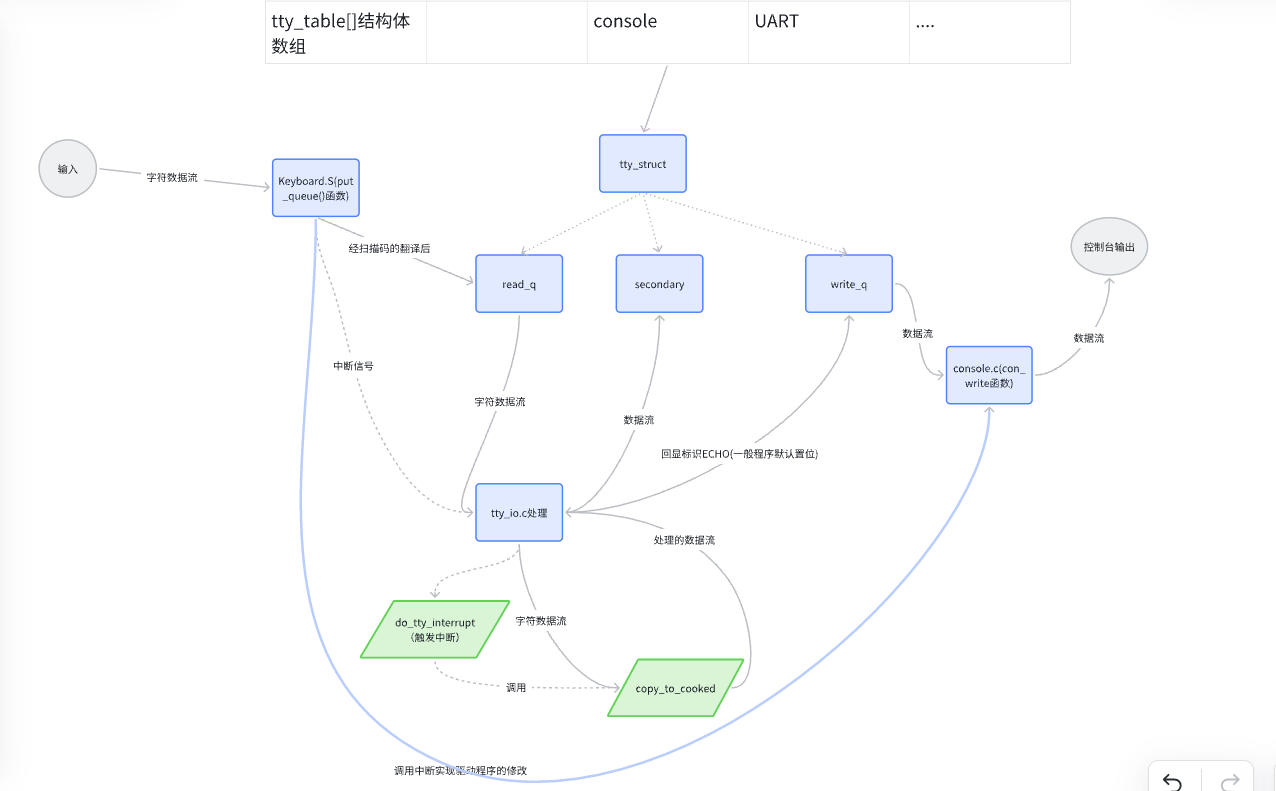
copy\_to\_cooked()函数由键盘中断过程调用(通过 do\_tty\_interrupt())，用于根据终端 termios 结构中设置的字符输入/输出标志(例如INLCR、OUCLC)对read\_q队列中的字符进行处理，把字符转换成以字符行为单位的规范模式字符序列，并保存在辅助字符缓冲队列(规范模式缓冲队列)(secondary)中，供上述 tty\_read()读取。



### 总流程

1. **按键触发**：当用户按下一个键时，硬件会通过键盘控制器发送一个中断信号，触发 keyboard.s 中的中断处理程序。
2. **扫描码处理**：在 keyboard.s 中，扫描码会被读取并缓存在缓冲区，供内核进一步处理。
3. **字符转换与回显**：console.c 会读取扫描码并将其转换为字符，同时根据配置（例如回显设置）决定是否将其显示在屏幕上。
4. **字符存储与处理**：tty\_io.c 负责将字符插入到 tty 输入缓冲区，处理控制字符，并根据需要进行屏幕显示、缓冲区管理或输出处理。

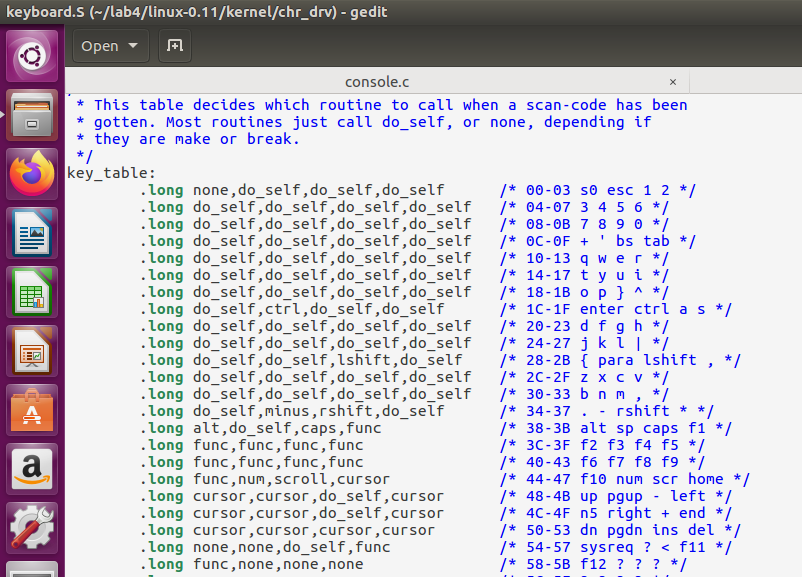
流程图：



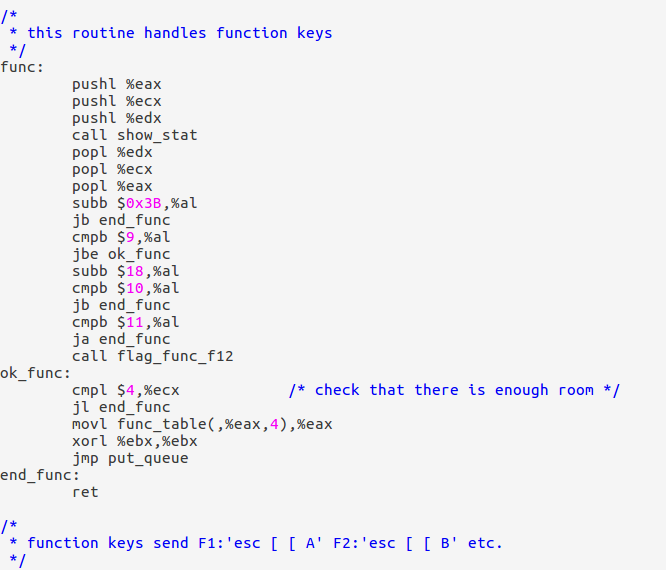
## Phase 1

要求键入 F12，激活功能，键入学生本人的姓名拼音，首尾字母等显示。

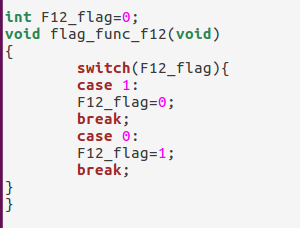
1. 因为任务只要求涉及单键，可以在汇编层面做修改，因为keyboard.s能够有效的触发中断，因此只需要为F12键设置独属于他的中断函数即可。
2. 找到Keyboard.s中对不同键盘按键设计的中断函数



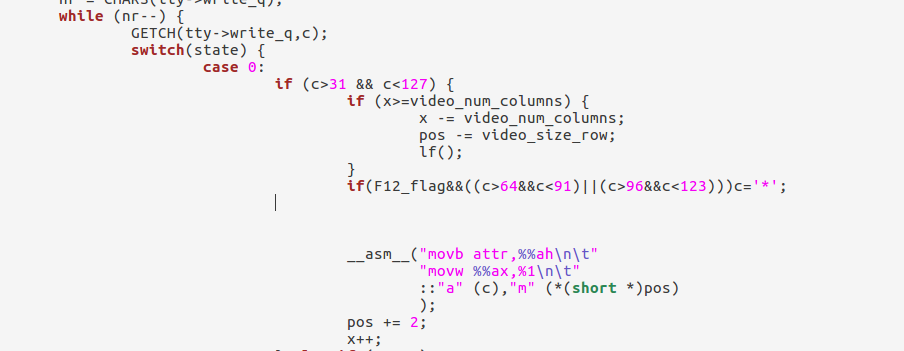
1. 发现F12对应中断func函数，也能发现f1~f10也都对应同一个函数func。定位到func函数，



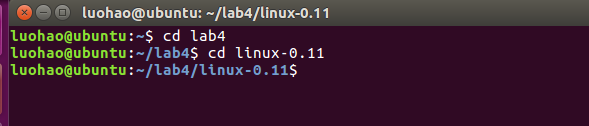
1. 根据汇编语言，我们大致知晓func在进一步区分不同的扫描码时为每个情况设置了一个跳转表，而对f12的判断恰巧就在cmpb $11,%al后面，因此为f12设置单独的处理函数flag\_func\_f12,我们将相应的函数写在console.c中，因为最后编译后函数会被链接在一起。
2. 在console.c中编写，设置一个标志位用于区分什么时候进行”\*”号显示:



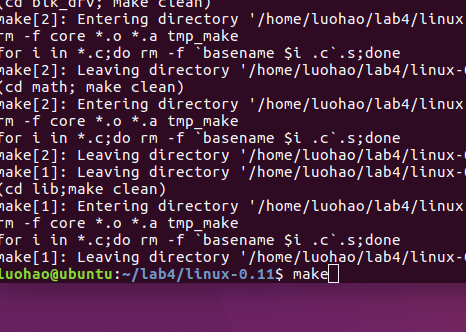
1. 在con\_write函数中编写，以将对应的结果打印在控制台上。



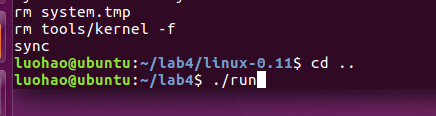
1. 进行编译，打开终端，将目录切换至linux-0.11下：



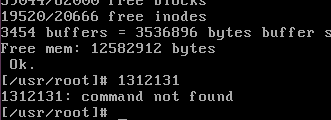
1. make clean清除所有的.o以及.exe文件，再用make指令重新进行编译



1. 切换出该目录回到lab4目录，进行运行：

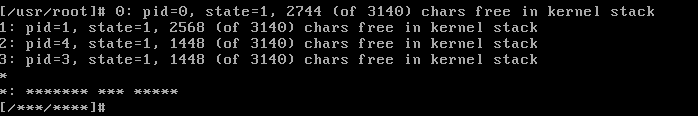


1. linux终端出现，



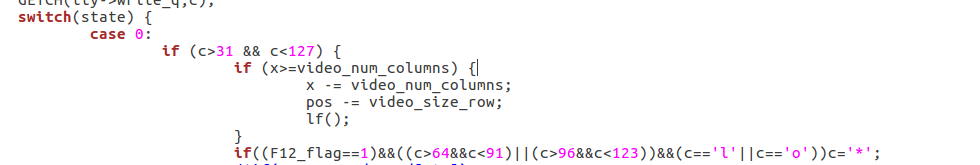
现在正常显示。

键入F12键，观察现象：

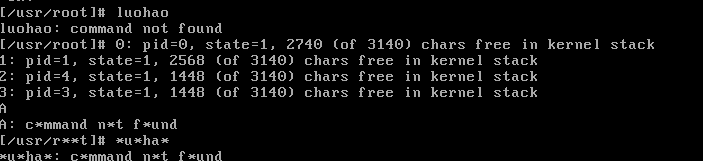


F12触发的处理函数flag\_func\_f12成功将flag置为1，因此屏蔽功能体现

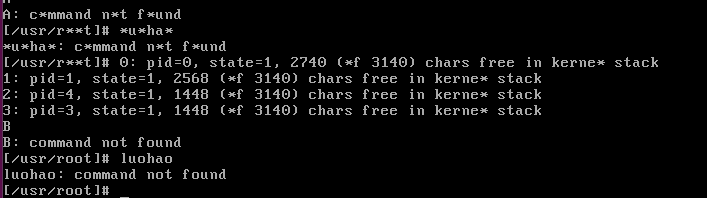
1. 返回修改con\_write,让它只屏蔽姓名拼音的首尾字母，luohao就应该屏蔽’l’’o’,修改如下：



结果如下，符合要求，按下f12后，姓名首尾字母变为’\*’



再次按下f12键即可恢复：



至此实验1完美完成。

## Phase 2

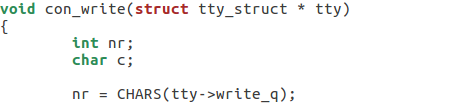
实验二目标是：键入“学生本人学号” ：激活\*功能，键入学生本人姓名拼音，首尾字母等显示\* 比如：zhangsan，显示为：\*ha\*gsa\*， 再次键入“学生本人学号-” ：取消显示\*功能。

因此不再需要F12单键触发，因为对于一串字符无法触发对应的中断，只有单字符可以。因此我选择不再在Keyboard.S汇编层面处理，因为Keyboard.S只适用于单键的中断触发，无法满足学号触发。

我们考虑在console.c文件中进行处理，根据之前的分析，con\_write函数读取wirte\_q队列中的回显字符。该队列中的字符经过read\_q缓冲区进入tty\_io.c进行cooked煮熟处理后放入了read\_q和secondary中。

那么我们可以通过直接读取read\_q队列中扫描码对应的字符串直接进行对应的模式触发。

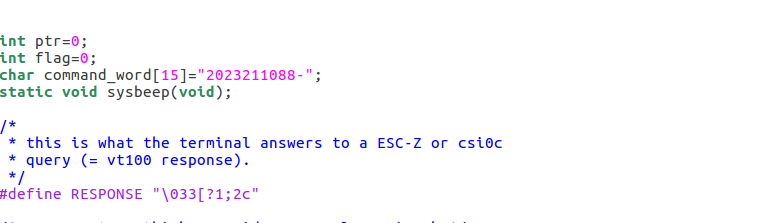
Con\_write函数接受tty结构体，因此我们可以尝试直接使用tty中的wirte\_q队列进行目标字字串的提取来改变我们对应的模式；



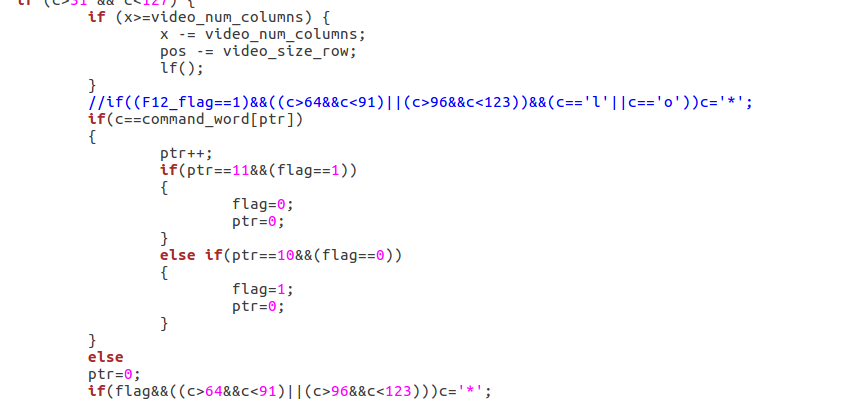
但也可选择采用状态机的方式一个字一个字单独读入，每次根据字符进行状态转移。

我选择状态转移的方法，这样的效率要高于前者。

1. 设置全局变量flag，用于开启或关闭\*功能；设置ptr表示当前已匹配的的字符个数；command\_word为触发指令，和ptr配合进行字符匹配。可知现在我的学号为2023211088



1. 在con\_write函数中编写状态转移方程：

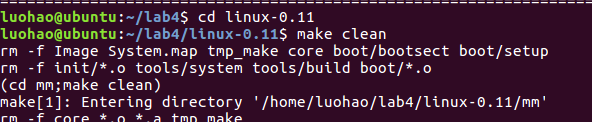


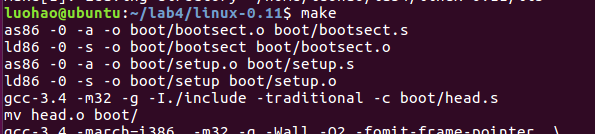
每次从第一个字符开始匹配，如果匹配成功，ptr+1，如果不成功，则重新从第一个字符开始匹配。

因为匹配成功第10个字符时，说明2023211088输入进来，此时开启flag，使得flag=1，即开启\*功能；

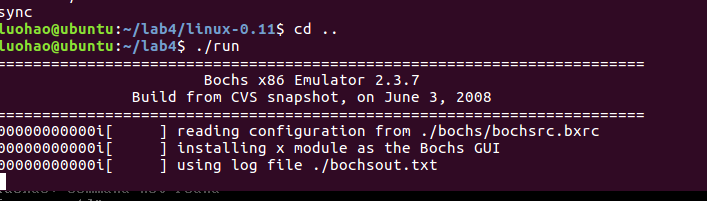
但如果我再输入2023211088-想要关掉这个功能的时候，会再次进入if语句中，无法再进行关闭的功能，因此在if判断中加入(ptr==10&&(flag==0))，这样能保证\*功能开启后，不再继续将flag置1，而是继续判断直到识别到2023211088-，因而将该功能关闭。

3.我们进入到终端中，使用make clean中清理缓存，进行make重编译。：

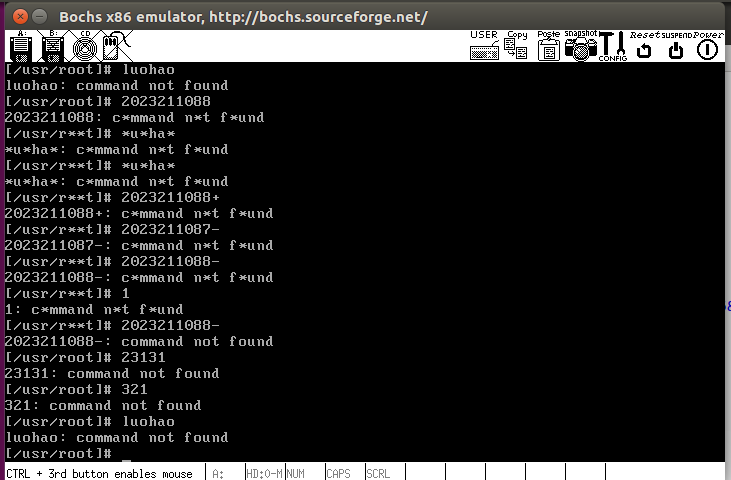




1. 回到父目录，进行运行：



1. 验证：



1. 分析

1.第一次输入时，\*功能未开启，正常运行，luohao正常打印

2.之后输入学号2023211088之后\*功能开启，输入luohao首尾字母被隐藏。

3.输入其他字符串没反应，符合预期

4.**但是**为什么我输入了2023211088-没有关闭\*功能呢，原因很简单，因为之前已经有一部分的字符被匹配了导致我们设置的全局变量ptr没有被置零，此时输入一个无关字符‘1’使得ptr置位即可。

5.再次输入2023211088-复原功能，如图。符合预期，实验二完成，完美收场。

# 五、总结体会

本次实验的主要任务是分析和修改 Linux 0.11 内核中的键盘驱动程序，通过对 keyboard.s、console.c 和 tty\_io.c 等源代码的学习，理解了按键事件的处理过程，掌握了键盘中断、字符输入输出以及终端回显等功能的实现原理。

在实验过程中，我首先对一次按键操作的整个流程进行了深入分析，理解了从键盘中断到最终字符显示的每个环节。通过阅读和调试源代码，我意识到键盘输入的处理不仅涉及硬件中断，还涉及到内核空间的多层次管理，例如缓冲队列、字符转换等。这些操作不仅是内核实现的基础，而且是开发实际操作系统中设备驱动的关键。

在第一阶段中，我通过修改汇编语言层面的 keyboard.s 文件，成功实现了按下 F12 键时，显示姓名拼音首尾字母为 \* 的功能。这一过程让我进一步理解了中断服务程序如何处理扫描码，并通过终端驱动函数进行字符输出。通过此过程，我不仅加深了对汇编语言的理解，也提高了调试技能。

在第二阶段中，我使用状态机的方式，修改了 console.c，使得输入学号后能够启用或禁用 \* 功能。这一过程中，我逐步掌握了如何在字符流中匹配特定的模式，并使用标志位控制程序行为。这个过程让我认识到，状态机是一种非常高效且灵活的编程方法，适合处理复杂的字符匹配和转换任务。

实验中遇到了一些挑战，特别是在第二阶段匹配学号并控制功能启停时，由于状态变量未及时重置，导致功能无法正常关闭。通过仔细分析源代码和逐步调试，我最终解决了问题，确保了程序按预期运行。

整体来看，这次实验让我深入理解了 Linux 操作系统中字符设备驱动的基本原理，同时锻炼了我调试和源代码分析的能力。我意识到，操作系统的内核编程不仅需要扎实的理论基础，还需要极高的细致性和耐心。通过这次实验，我也增强了自己在面对复杂工程问题时的分析与解决能力。

在未来的学习中，我希望能进一步巩固这些基础知识，并尝试实现更多复杂的设备驱动程序。同时，也建议今后的实验报告中能够加入更多的调试过程和步骤描述，以帮助同学们更好地理解每个细节。