

**实验报告**



**题目： 键盘驱动程序的分析与修改**

**班 级： 2023211307**

**学 号： 2023211075**

**姓 名： 魏生辉**

**学 院： 计算机学院**

**2024年 12月 8日**

1. 实验目的
2. 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；

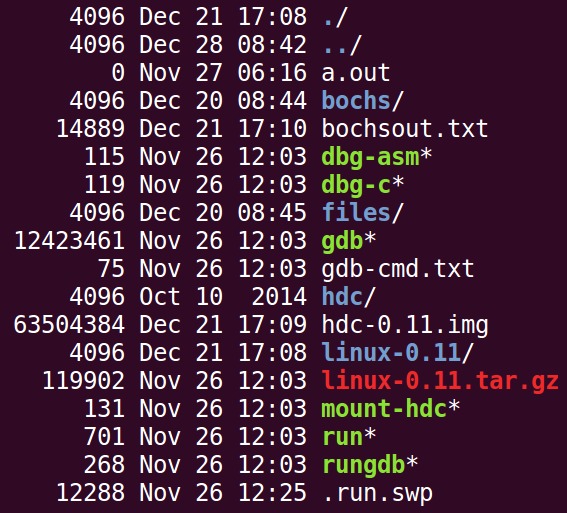
4.通过阅读源代码，进一步提高C语言和汇编程序的编程技巧以及源代码分析能力；

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

#### ①必要知识了解

在Linux 0.11的体系中，终端的管理可谓是复杂且极具层次感的，系统通过两类终端设备来与用户进行交互：一类是控制台终端，另一类则是通过串行端口连接的硬件终端设备。控制台终端不仅承担着与用户的交互任务，还被赋予了一个至关重要的角色：它由内核中的关键程序——keyboard.S和console.c进行管理。这两个程序看似平凡，却在幕后执行着繁琐且高度协调的工作，确保用户能够通过键盘输入指令、在屏幕上看到响应的输出。

在这个庞大的系统中，每一个控制台终端都对应着一个tty\_struct数据结构。这个结构体仿佛是终端设备的“身份证”，它记录了终端当前的状态和各类参数，包含着对字符输入输出的严格管理。其内部设有三个主要的缓冲队列——read\_q、write\_q和secondary，它们各司其职，充当着输入输出的枢纽。

* **read\_q**：保存着键盘输入的原始字符序列，几乎是输入的原始“原料”。
* **write\_q**：这里存储着准备在控制台屏幕上显示的数据，意味着这一队列承载着视觉的呈现。
* **secondary**：这个队列则存放了经过“行规则”程序处理后的数据，经过copy\_to\_cooked等函数的处理，字符已经“被煮熟”，准备好呈现给用户。

这些缓冲区的存在，既保障了输入输出的顺畅流动，也为系统提供了灵活的处理能力。再看tty\_io.c，它是上层接口与硬件之间的桥梁，负责提供一系列函数来操控终端设备，如tty\_read和tty\_write。每当用户进行文件操作时，内核便会通过这些接口函数与终端进行互动，处理字符的读写。

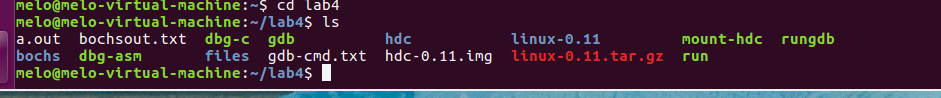
然而，键盘输入的整个流程远不止于此。真正的幕后功臣是keyboard.S，这个汇编程序充当着中断处理的核心角色。当用户按下键盘时，keyboard.S会先检测到按键的状态，并根据特定的扫描码，通过一张精心编排的跳转表来调用相应的子程序，将字符放入read\_q。之后，这些字符会通过中断机制，传递给tty\_io.c中的do\_tty\_interrupt函数，这个函数就像是调度员，它会将字符通过copy\_to\_cooked函数进行必要的加工处理，最后把处理好的字符移入secondary队列。

如果用户启用了回显功能，这些字符还会被再次传送到write\_q，并在屏幕上进行显示。此时，console.c中的con\_write函数便开始发挥作用。它从write\_q队列中提取字符序列，进行字符的显示、光标移动、字符擦除等操作。con\_write不仅负责将字符按顺序呈现给用户，它还需要灵活应对各种控制字符的需求。

#### ②环境配置与lab4解压

环境根据云平台的指导， 同时我还参考了csdn上的这篇文章，快速地完成了配置<https://www.cnblogs.com/bestsheng/p/6350250.html>

下图是解压工作：



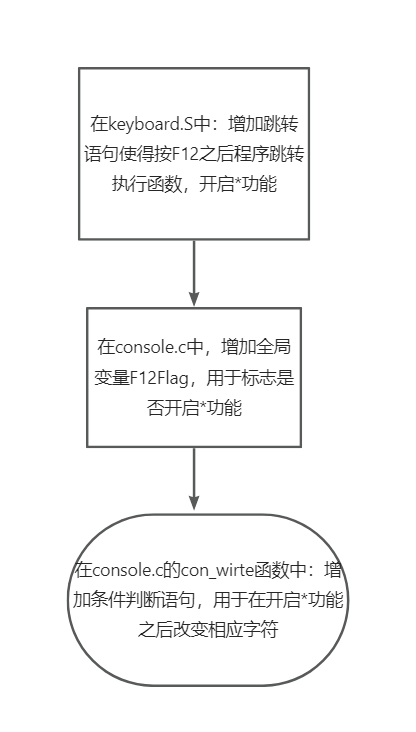
### Phase 1

#### ①解题思路与流程图

从keyboard.S源代码中，我们可以清晰地看到，key\_table是一个关键的跳转表，用来根据键盘扫描码将按键与相应的处理函数进行绑定。对于功能键如F1到F12，它们的扫描码通过该跳转表指向对应的处理函数，这些处理函数在按下特定功能键时会被触发，执行特定的操作。

我们的任务是扩展这一机制，新增对F12按键的支持，使得当用户按下F12时，程序能够跳转到一个新的处理函数。

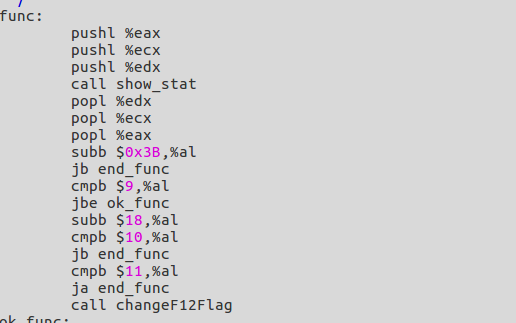
**于是得出解题流程图：**



下面进行具体操作：

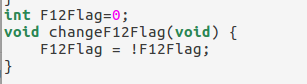
#### ②修改keyboard.S （注意大小写）里面的func

在后面增加一行：call changeF12Flag

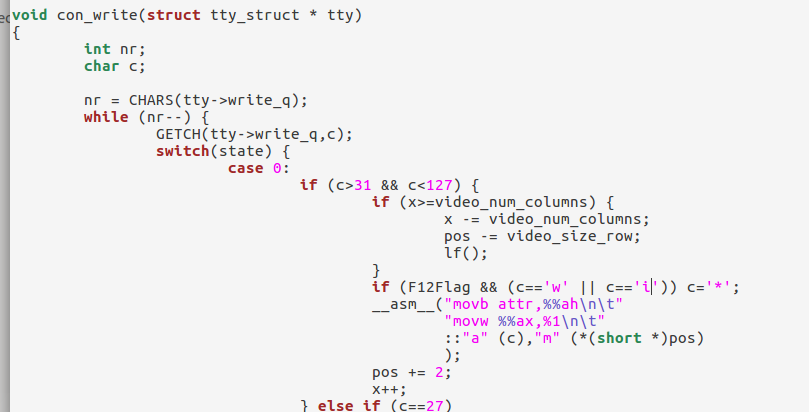


也就是：如果按下了F12键，就跳转至changeF12Flag函数

#### ③修改console.c

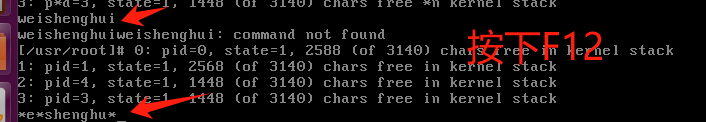


这个函数表示每按一次F12，F12Flag就翻转一次



**这里说明** ：我的名字是weishenghui 所以输入w 和i时就输出\*

#### ④测试



顺利完成第一关！

### Phase 2

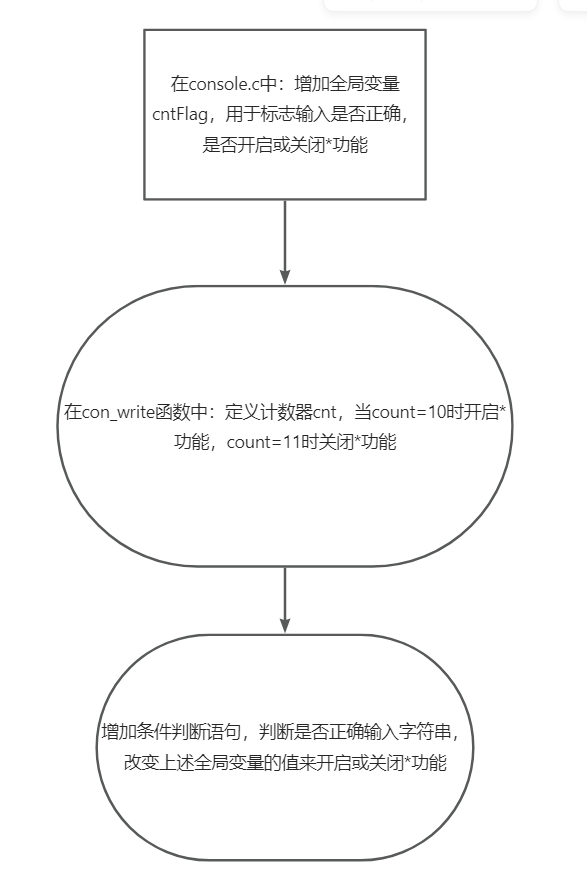
#### ①解题思路与流程图

对输入的每个字符进行逐步检查，并用标识位来判断每个字符是否正确，从而实现逐步开启和关闭功能。具体的实现思路是：

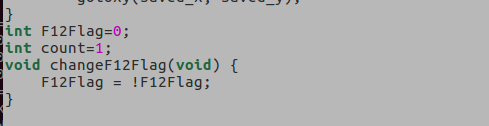
1. **逐个字符检查**：首先记录输入的每个字符的正确性，并逐步检查。
2. **开启功能的条件**：当前10个字符（即学号2023211075）全部输入正确时，激活\*功能。
3. **关闭功能的条件**：若输入的第11个字符是-，则关闭该功能。

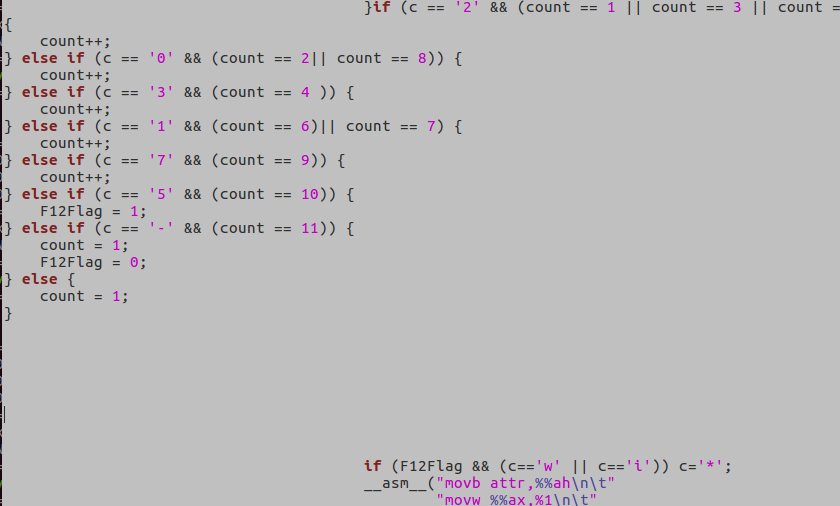
**具体步骤：**

1. 用一个标识位（如count）记录已经输入正确的字符数。
2. 当用户输入字符时，从第1个字符开始逐一验证每个字符是否符合要求。
3. 如果前n个字符（n从1到10）都输入正确，则更新标识，表示前n个字符正确。
4. 当所有前10个字符都正确时，检查是否启用\*功能。如果未开启，启用该功能；如果已经开启，检查第11个字符是否为-，若是，则关闭\*功能。

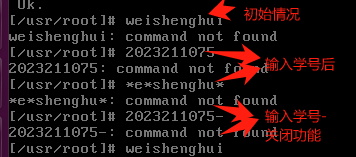


#### ②修改console.c:





#### ③运行测试



一开始正常输入“weishenghui”，键入 学号 后，输入“weishenghui”,显 示出“\*e\*shenghu\*”，再键入 学号- 后，又正常显示出。

五、总结体会

这次实验无疑是一场充满波折的历程，充斥着无数的挑战与反思，仿佛在不断与虚拟机和代码中的细节进行博弈。回顾整个过程，我不禁感叹自己经历了许多“跌宕起伏”的时刻，从虚拟机配置到源代码修改，每一环节都不曾轻松。最终，当所有的困惑逐渐被解决，成就感油然而生，回望这一路走来的心路历程，更是充满了对计算机系统深刻的理解与认识。

**配置虚拟机的漫长与迷茫**

最初的配置阶段，困扰我最大的不是代码本身，而是虚拟机的调试与配置。作为大一时已摸索过虚拟机的配置，但当时对细节并没有足够的重视。此次重装虚拟机，竟然因“不能同时安装两个workstation”的问题，差点让人心生绝望。卸载本就不易，我决定在大一配置的虚拟机上继续作业。尽管我几乎忘记了之前的配置过程，但幸运的是，终于在时间与心力的挤压下，我成功地调整好设置，最终避免了重新安装的麻烦。

**环境调试的阻滞与突破**

然而，虚拟机的配置远未结束。最初，明明曾信心满满地认为自己领先一步，但由于虚拟机版本不对，连不上网络的问题让我陷入了“死循环”。一个接一个的小问题不断涌现——配置的内容总觉得缺了些东西，怎么也找不到关键的设置。在那时，我的耐心几近崩溃，甚至怀疑自己早期的努力是否都白费。于是，焦虑中度过了一个凌晨，直到第二天清晨，我才恍若醒悟，原来“重启”有时比任何设置都有效。终于，网络问题迎刃而解，接下来的步骤也渐渐顺利了许多，下载了所需工具，虚拟机的配置才算告一段落。

**第一阶段的反复与调整**

第一阶段的挑战，主要是对Linux内核源代码的修改。最开始，我并没有充分借鉴《Linux内核完全注释》，于是花费了不少时间在F12功能的处理函数上。不断的查阅资料与调试代码，在我逐步深入理解的同时，也让我意识到，自己在设置全局变量位置时的一个小错误，竟然导致无法生成image文件。这个问题让我陷入了长时间的困惑，直到通过报错信息找到并修正错误，问题才终于解决。

在修改的过程中，我深刻体会到调试的艰辛：通过对源代码的修改，尤其是在处理F12按键时，反复测试每个细节，直到成功。这不仅考验了我对代码的理解深度，还锤炼了我的问题排查能力。尽管一开始的错误不断，甚至在做出修改后忘记保存，但每次发现问题并解决，都会让我感到一种跃跃欲试的兴奋。

**第二阶段的思考与突破**

当进入第二阶段时，我已积累了一些经验，并逐渐理解了之前的失误。经过深思熟虑后，我设计出了更合理的算法，但问题依然没有消失——尽管我知道修改在哪里，代码依旧没有按预期输出\*符号。深夜反复琢磨后，我终于意识到，部分变量需要设置为全局变量，而这一细节恰恰是解决问题的关键。修改完成后，代码终于如愿以偿地输出了预期的功能，所有的错误和困难在成功的那一刻都得到了偿还。

经历了长时间的调试和修改，整个实验终于迎来了尾声。尽管实验过程中的每一个问题都曾让我感到疲惫不堪，但最终能够亲手修改并实现Linux内核的功能，不仅让我对Linux系统有了更深入的理解，也让我更加熟悉了虚拟机的使用与调试。更重要的是，这次实验加深了我对计算机底层架构和I/O系统的理解，让我明白了如何通过与硬件交互来控制程序的执行。

最终，我不仅成功完成了实验，甚至还进一步优化了原有虚拟机的配置，安装了更多有用的工具，甚至为虚拟机加上了中文输入法，增强了使用体验。