|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 课程名称 | 操作系统设计与实践 | | | 成 绩 |  | 教师签名 |  |
| 实验名称 | I/O子系统 | | | 实验序号 | 9 | 实验日期 |  |
| 姓 名 | 张子航 | 学 号 | 2021302181026 | | | | 组长 |
| 姓 名 | 辜汝曦 | 学 号 | 2021302141194 | | | | 组员 |
| 姓 名 | 杨馨悦 | 学 号 | 2021302181212 | | | | 组员 |
| 姓 名 | 赵敏 | 学 号 | 2021302181215 | | | | 组员 |

**《操作系统设计与实践》实验报告**

# 实验目的及实验内容

（本次实验所涉及并要求掌握的知识；实验内容等）

## 实验目的

* 键盘输入的基本原理
* 显示器输出的基本原理
* TTY任务子系统
* 合理配置不同程序于不同环级
* printf的机理

## 参考资料

《操作系统设计实验系列（九）：I/O子系统》PPT

## 实验要求

* 验证键盘输入原理
* 了解显示器的输出概念
* 了解TTY的基本架构以及相关扩展功能
* 了解如何设置不同程序处于不同环级
* 了解printf的实现机理

# 实验环境及实验步骤

（列出本次实验所使用的软件、工具；简要概括实验步骤）

## 实验环境

虚拟机工具：VMWare Workstation 16

虚拟机版本：Ubuntu 14.04.6(内存4GB，硬盘40GB，双核处理器)

开发与调试工具：bochs 2.6.8

## 实验步骤

阅读本章节的资料，并根据PPT内容进行理解并调试，最终实现相应任务

# 实验过程分析

（详细记录实验过程，通过截图展示得到的结果。特别是对于实验中发生的故障和问题，要进行故障分析，说明故障排除的过程及方法。）

## 验证键盘输入的处理程序，并回答如下问题：

### 解释键盘输入缓冲区的作用，以及处理过程

8042的输入缓冲区大小只有一个字节，所以当一个扫描码有不止一个字符时，实际上会产生不止一次中断。也就是说，如果我们按一下 Shift+A ，产生的 0x2A0x1E0x9E0xAA 是 4 次中断接收来的。这就给我们的程序实现带来了困难，因为第一次中断时收到的 0x2A 无法让我们知道用户最终会完成什么，说不定是按下Shift又释放，也可能是 Shift+Z 而不是 Shift+A 。

于是，当接收到类似 0x2A 这样的值的时候，需要先把它保存起来，随后慢慢解析用户到底做了什么。

保存一个字符可以用全局变量来完成。可是，由于扫描码的值和长度都不一样，这项工作做起来可能并不简单。而且可以想像，键盘操作必将是频繁而且复杂的，如果把得到扫描码之后相应的后续操作都放在键盘中断处理中，最后keyboard\_handler会变得很大，这不是一个好主意。

**所以我们可以建立一个缓冲区，让 keyboard\_handler 将每次收到的扫描码放入这个缓冲区，然后建立一个新的任务专门用来解析它们并做相应处理。**

**总结一下作用：**

(1)临时存储输入数据：键盘输入缓冲区是一个内存区域，用于临时存储来自键盘的输入数据，包括按下和释放按键时所产生的字符和控制码。

(2)解耦输入和处理：它允许输入和处理操作分离，即使处理数据的速度与输入不匹配，也能确保输入不会丢失。键盘可能以不同的速率发送输入，而系统可能不总是能立即处理所有输入。

**处理过程：**

（1）接收键盘输入：键盘输入数据首先由硬件接收，然后被发送到键盘控制器。

（2）存储到缓冲区：键盘控制器将输入数据放置到键盘输入缓冲区中。这些数据包括扫描码和控制码，表示按键按下、释放等信息。

（3） 处理器读取输入：处理器通过访问键盘输入缓冲区来读取键盘输入。通常，我们的处理键盘缓冲区的代码函数被周期性地调用，用于从缓冲区中读取输入并进行处理。

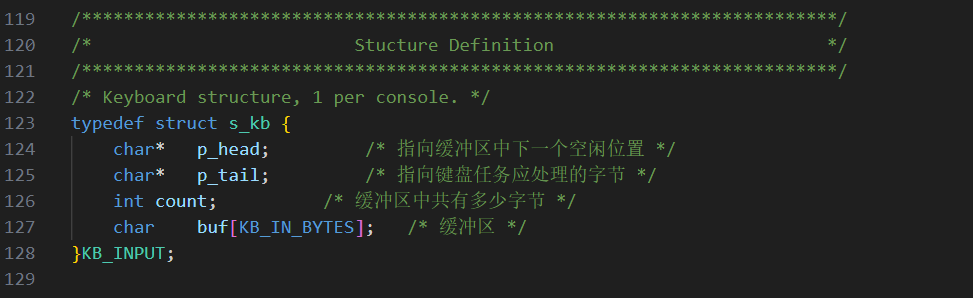
（4）处理输入数据：处理器读取缓冲区中的数据，解析扫描码并将其转换为相应的字符或操作。这些数据可能会触发操作系统级别的事件，如字符输入到文本框、执行特定操作等。

（5）清空缓冲区：一旦处理器读取了缓冲区中的数据，通常会清空相应的缓冲区位置，以便接受新的输入。如果处理速度太慢，缓冲区可能会被填满，新的键盘输入会被丢弃或临时存储，直到处理器能够处理为止。

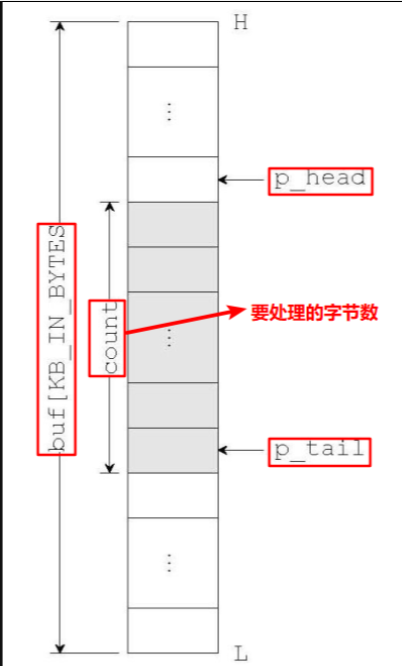
键盘输入缓冲区的设计有助于系统处理异步输入，并确保按键操作能够被及时响应，即使系统处理速度有限。

**添加键盘缓冲区过程**：

先来建立一个缓冲区，用以放置中断例程收到的扫描码：

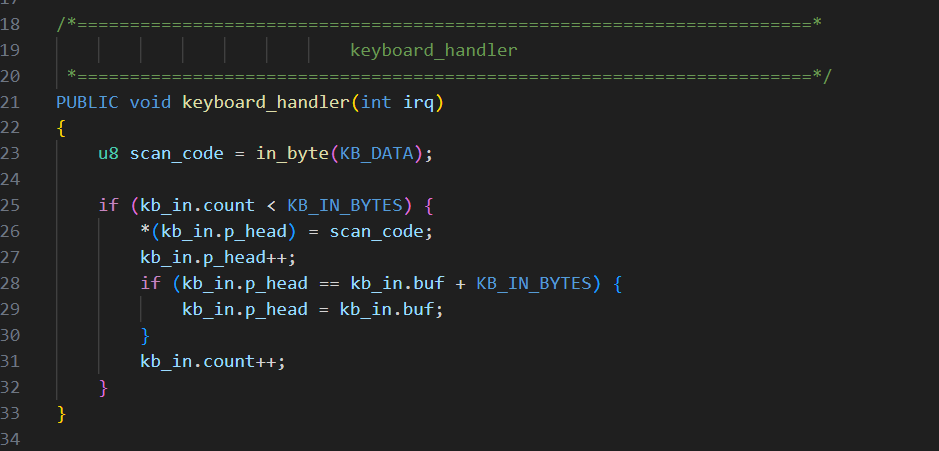


这个缓冲区的用法如下图所示，白色框表示空闲字节，灰色框表示已用字节。在执行写操作的时候要注意，如果已经到达缓冲区末尾，则应将指针移到开头。

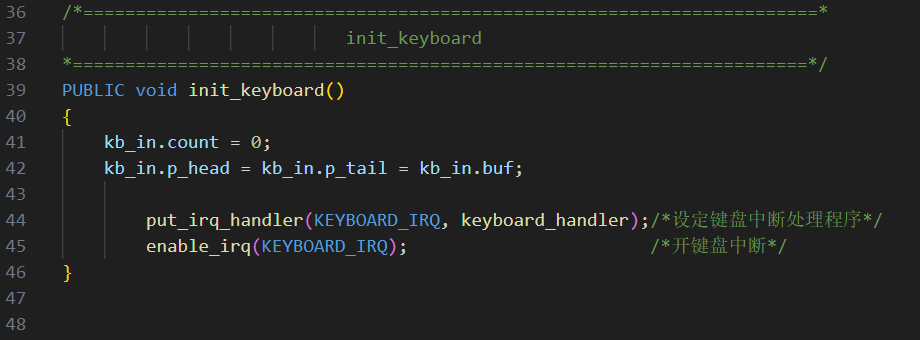


对照上图，我们可以容易地**对缓冲区进行添加操作**：

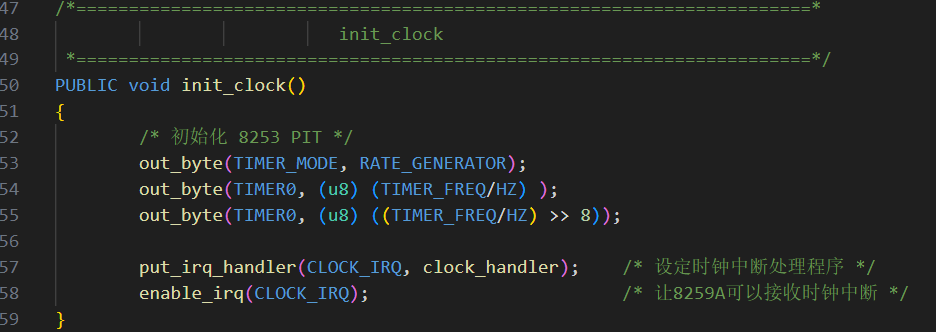




代码很简单，但是，如果缓冲区已满，这里使用的策略是直接就把收到的字节丢弃。其中的 kb\_in ，由于我们只在 keyboard.c 中使用，于是把它声明成一个 PRIVATE 变量（ PRIVATE 定义位于 const.h 中，被定义成了 static ）。注意，kb\_in 的成员需要初始化，初始化的代码放在 init\_keyboard( ) 中。



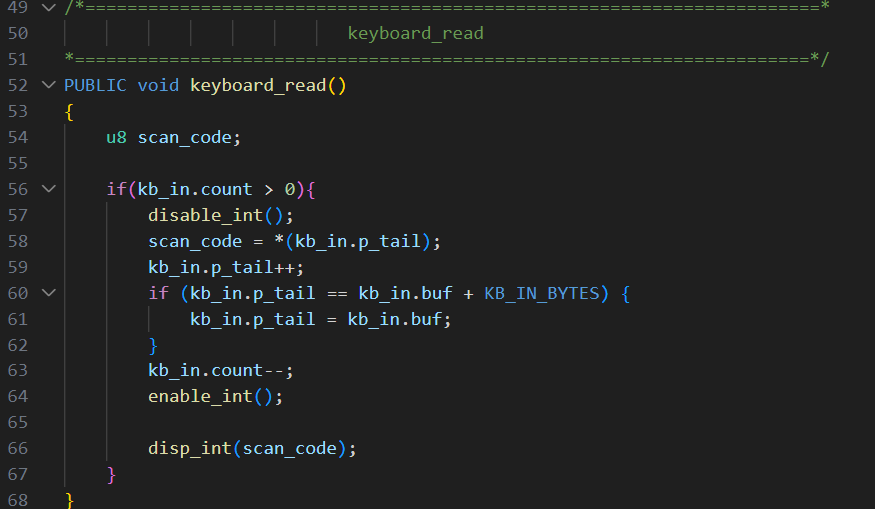
为了保持 kernel\_main 的整洁，我们把时钟中断的设定和开启也放到单独的函数 init\_clock 中。



这样，在 kernel\_main 中调用这两个函数就可以了。

### 为什么在/c/的kliba.asm，代码7.12中，需要disable\_int, enable\_int？。

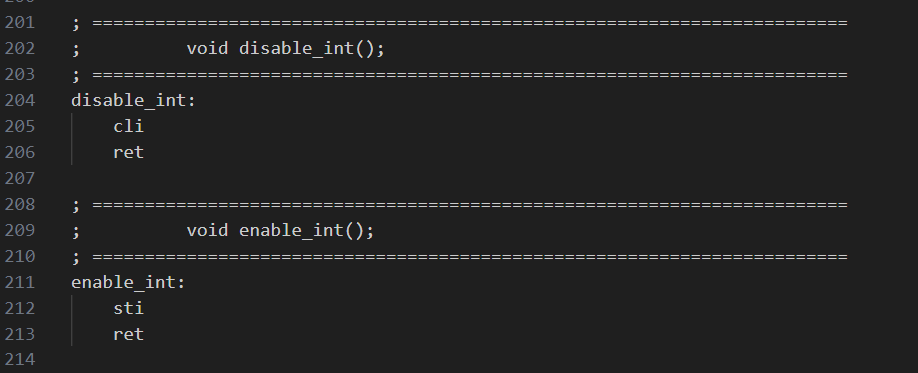
在一个将来不仅会处理键盘操作，还将处理诸如屏幕输出等内容终端任务运行过程中，一般会不停地调用 keyboard.c 中的函数 keyboard\_read ：



函数首先判断 kb\_in.count 是否为 0 ，如不为 0 ，表明缓冲区中有扫描码，就开始读。读缓冲区开始时关闭了中断，到结束时才打开。这是因为 kb\_in 作为一个整体，对其中的成员的操作应该是不受打扰的。读操作相当于写操作的反过程。

其中，disable\_int( ) 和 enable\_int( ) 的定义很简单。就是关中断和开中断。

所以在/c/的kliba.asm，代码7.12中，需要定义disable\_int, enable\_int。定义如下：



## 解释一下，重新设置显示开始地址的原理。

重新设置开始地址这一过程涉及到对VGA系统相关寄存器的修改。在微型计算机系统中，VGA（Video Graphics Array，视频图形阵列）是一种视频传输标准，由IBM在1987年推出。它传输的是模拟信号，在当时具有分辨率高、显示速率快、颜色丰富等优点。本实验默认实验显示终端采用了VGA系统。

对于VGA系统，它的寄存器信息如下：



其中，VGA某一类寄存器可能不止只有一个，比如Data Registers，它对应一组寄存器。然而，这一组寄存器只对应一个端口0x3D5，因此需要Address Register这一个寄存器来表示寄存器组的索引。Address Register的端口号为0x3D4。当想要访问寄存器组的某个寄存器时，先将对应索引该写入0x3D4的Address Register，然后写入0x3D5，这样可以实现对对应寄存器的操作。

举个例子，Cursor Location High Register和Cursor Location Low Register是用来设置光标位置的，索引号分别是0Eh，0FH，修改它可以修改光标的位置，比如让光标跟随输入的字符位置。这也是示例代码中提供的功能。

回到本问题，我们得知，Start Address High Register和Start Address Low Register两个寄存器用来重新显示开始地址，那么，通过修改这两个寄存器即可实现重新设置显示开始地址。

因此，我们编写对应的函数：

PUBLIC void in\_process(u32 key){

char output[2] = {'\0', '\0'};

if (!(key & FLAG\_EXT)) {

......

}

else {

int raw\_code = key & MASK\_RAW;

switch(raw\_code) {

case UP:

if ((key & FLAG\_SHIFT\_L) || (key & FLAG\_SHIFT\_R)) {

disable\_int();

out\_byte(CRTC\_ADDR\_REG, START\_ADDR\_H);

out\_byte(CRTC\_DATA\_REG, ((80\*15) >>8) & 0xFF);

out\_byte(CRTC\_ADDR\_REG, START\_ADDR\_L);

out\_byte(CRTC\_DATA\_REG, (80\*15) & 0xFF);

enable\_int();

}

break;

case DOWN:

if ((key & FLAG\_SHIFT\_L) || (key & FLAG\_SHIFT\_R)) {

......

}

break;

default:

break;

}

}

}

该函数会检查当前的按键是否为Shift+UP，如果是，则卷动屏幕至80\*15处，也就是向上卷动15行。从上文可知，Start Address High Register和Start Address Low Register两个寄存器可以用来设置从屏幕的任何位置显示，这允许我们把显存划分为不同的部分，只需要简单的寄存器设置就可以显示相应位置的内容。

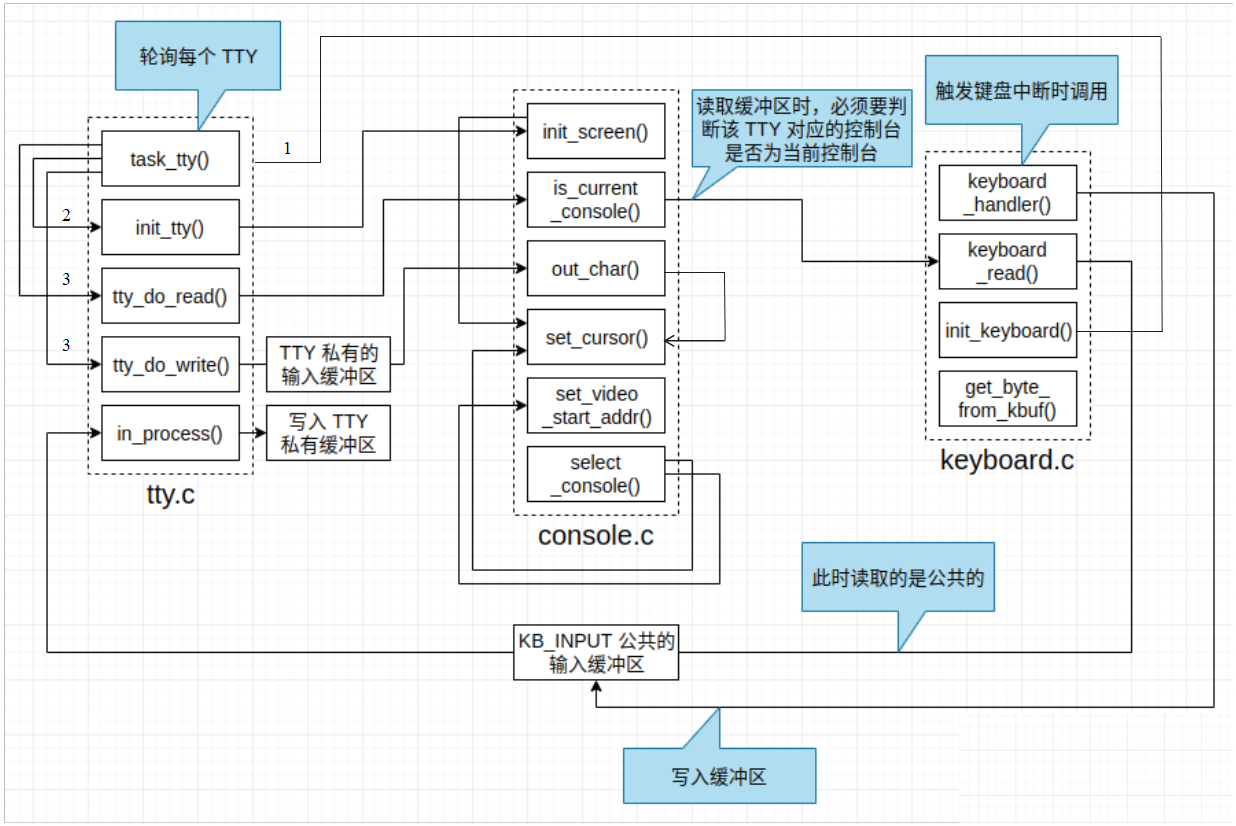
## 解释多个tty，能够并发处理输入输出的原理，结合代码给出一个书上框图更为细致的流程图。

### 原理

TTY 是一种终端设备，能够使用户通过用户界面与系统进行交互。系统可以处理多个 TTY，允许多个用户或进程同时与操作系统进行交互。每个 TTY 都与自己的输入和输出流相关联，这种隔离可确保用于一个 TTY 的数据不会干扰另一个 TTY。当用户在终端上输入时，输入会被发送到内核中相应的 TTY 设备驱动程序。

操作系统会为每个TTY分配一个独立的进程或线程来处理输入输出操作，这些进程或线程会根据输入输出请求的优先级和时间片轮转等算法来进行并发处理，从而实现多个TTY的并发输入输出操作。此外，操作系统还会通过信号量、互斥锁等机制来确保多个TTY之间的输入输出操作不会相互干扰，从而保证并发处理的正确性和稳定性。

### TTY工作流程图



### 流程图文字说明

* 在执行TTY任务之前，用户在终端输入字符（如键盘输入），之后输入的字符被传递给操作系统的TTY设备驱动程序，TTY设备驱动程序将输入字符存储在相应的输入缓冲区中
* 当TTY任务开始运行时，所有TTY都将被初始化，并且全局变量nr\_current\_console会被赋值为0。然后循环开始并一直进行下去。
* 对于每一个TTY，首先执行tty\_do\_read()
* 若当前TTY的控制台是当前控制台，它将调用keyboard\_read()，读取缓冲区数据，并将读入的字符交给函数in\_process()来处理
* 如果是需要输出的字符，会被in\_process()放人当前接受处理的TTY的缓冲区中。
* 然后tty\_do\_write()会接着执行，如果缓冲区中有数据，就被送入out\_char显示出来。
* 每个TTY具有独立的缓冲区和状态，多个TTY可以同时由不同的用户或进程使用；操作系统内核通过进程调度、中断处理和TTY管理来处理并发访问。

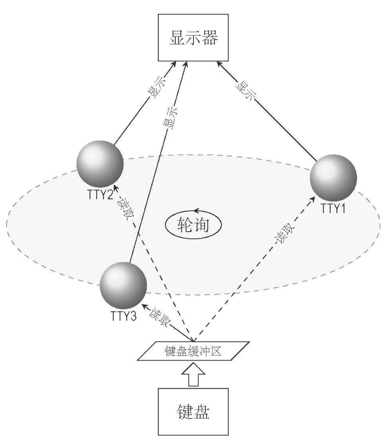
1. tty.c中：
   1. task\_tty()：轮询每个 TTY，对每个 TTY 都进程初始化，并完成对键盘缓冲区的读取和字符显示
   2. init\_tty()：初始化缓冲区、初始化屏幕；之所以初始化TTY的既因为其中的缓冲区需要设置初值，也因为要为每个TTY指定对应的CONSOLE
   3. tty\_do\_read()：若当前TTY的控制台是当前控制台，则读取缓冲区数据
   4. tty\_do\_write()：若缓冲区中有数据，则进行写入；TTY缓冲区中取出键值，然后用out\_char显示在对应的CONSOLE中
   5. in\_process()：根据输入的按键值来更新输入缓冲区，并根据不同的按键值执行相应的操作，如滚动屏幕或选择控制台。
2. console.c中：
   1. scroll\_screen()：滚动屏幕
   2. is\_current\_console()：判断是否为当前控制台
   3. out\_char()：向指定控制台输出字符
   4. set\_cursor()：设置光标位置
   5. set\_video\_start\_addr()：设置屏幕的显示位置
   6. select\_console()：切换控制台
   7. init\_screen()：往控制台输出字符
3. keyboard.c中：
   1. keyboard\_handle()：触发键盘中断时调用
   2. keyboard\_read()：读取键盘缓冲区
   3. init\_keyboard()：初始化键盘
   4. get\_byte\_from\_kbuf()：从键盘缓冲区中读取下一个字节

## 基础题：请添加一个你自己个性化的TTY，在这个TTY上，你可以根据键盘输入或者移动光标的某种规律，运行一个你的彩蛋 程序，而在其他TTY中不会有这个效果。

本题实现的是在对应的TTY上，每次输入字母Z，都会接着输出字母os

多个tty能够并发处理输入输出的原理是通过多进程或多线程实现的。每个tty都对应着一个进程或线程，它们可以同时运行，互不干扰。当用户在一个tty中输入命令时，该tty对应的进程或线程会读取用户输入的内容，并进行相应的处理。处理完成后，该进程或线程会将结果输出到对应的tty中，供用户查看。

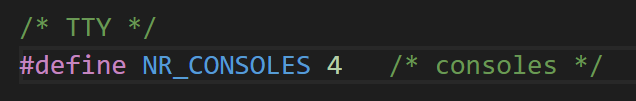
在Linux系统中，每个tty都对应着一个虚拟终端设备文件，当用户登录系统时，系统会为其分配一个tty，并将其连接到用户的终端设备上。用户可以在该tty中输入命令，系统会将命令传递给对应的进程或线程进行处理，并将处理结果输出到该tty中



实现的具体步骤如下：

首先，修改const.h文件，添加1个TTY；

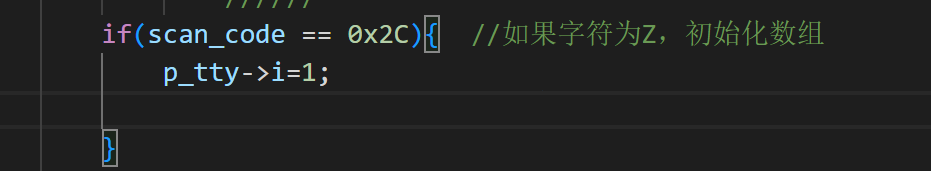
我们将在这个TTY内实现个性化功能；



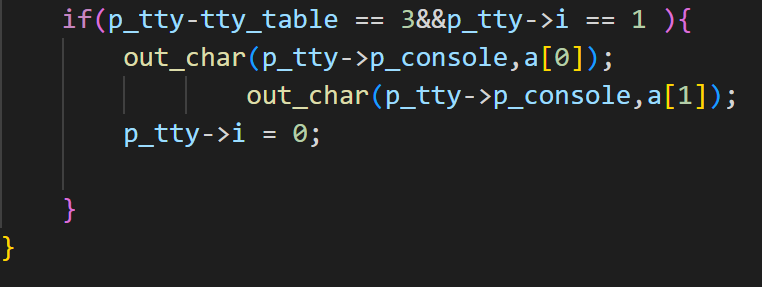
接着在头文件tty.h中定义循环变量i以及存储字符的字符数组zhang[]



更改keyboard.c文件，检测如果输入的字母为字母Z，则将标志位变为1；



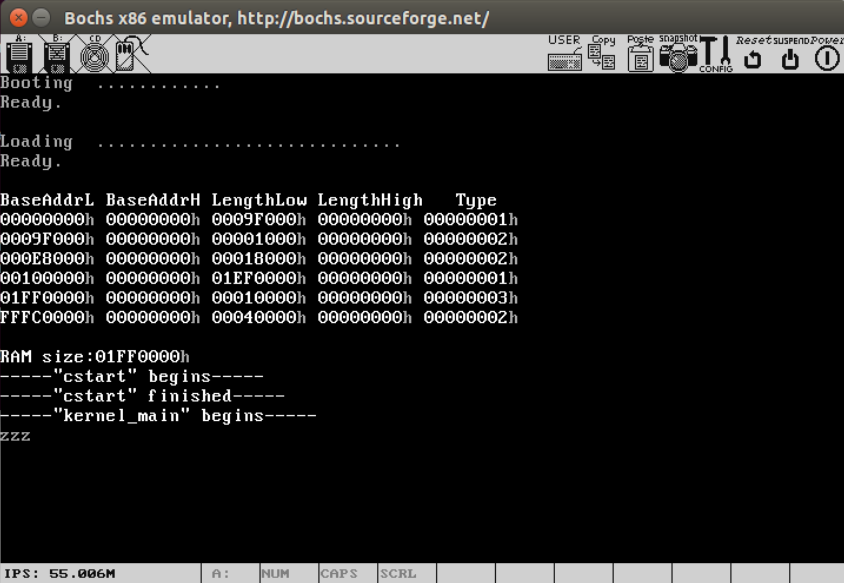
在tty.c文件中，增加一个判断，即在当p\_tty-tty\_table=3，即在新添加的TTY中时，运行自己的彩蛋：



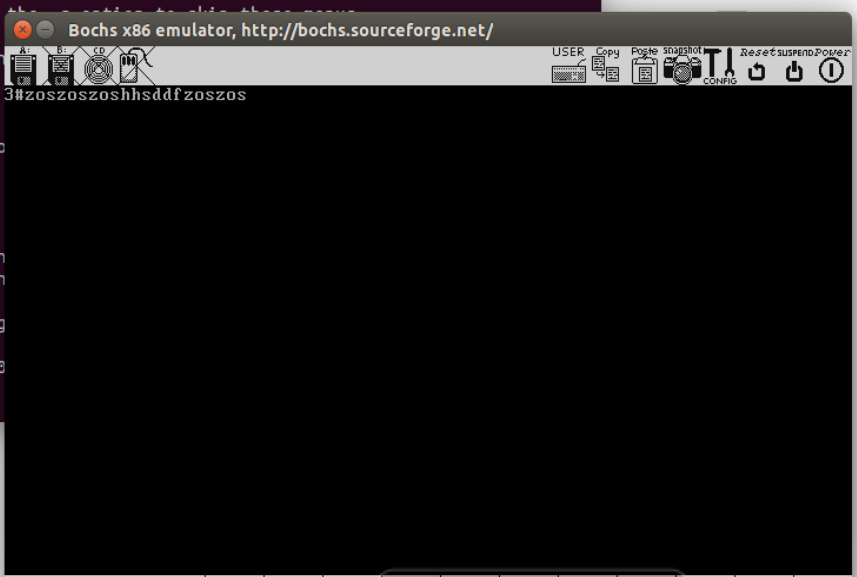
即当我们的标志位为1时，暑促字符串“os“

最终的实验结果如图所示：

当不在有彩蛋的TTY时，输入字母Z将不会有变化；



当在有彩蛋的TTY内输入z时，会接着出现os的字样；

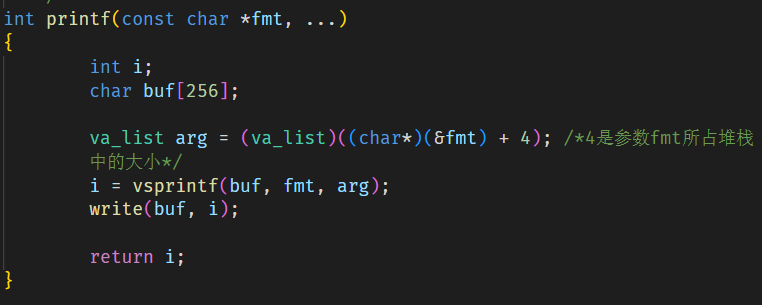


## 选做题：尝试扩展一下printf，让它支持%s，想想目前的printf实现是否有什么安全漏洞？可以怎么解决。

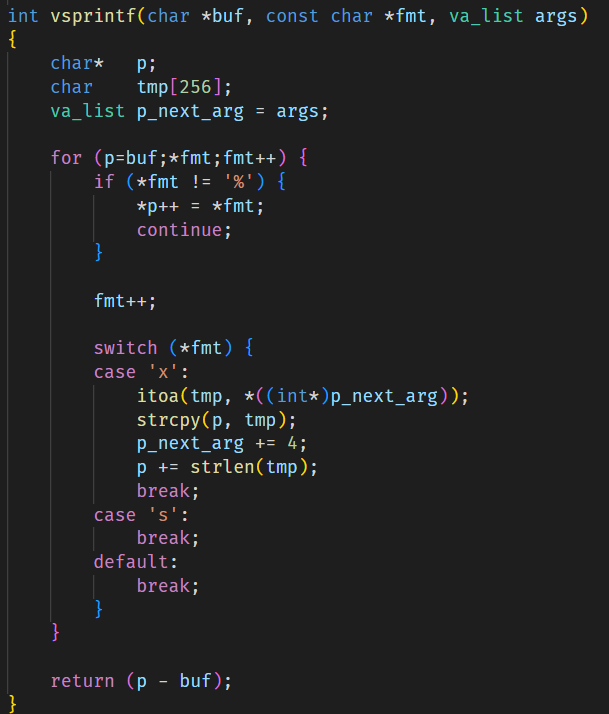
该题目可以分为两部分：扩展printf与printf漏洞发现。之后的报告内容也会分为这两部分。

### printf扩展

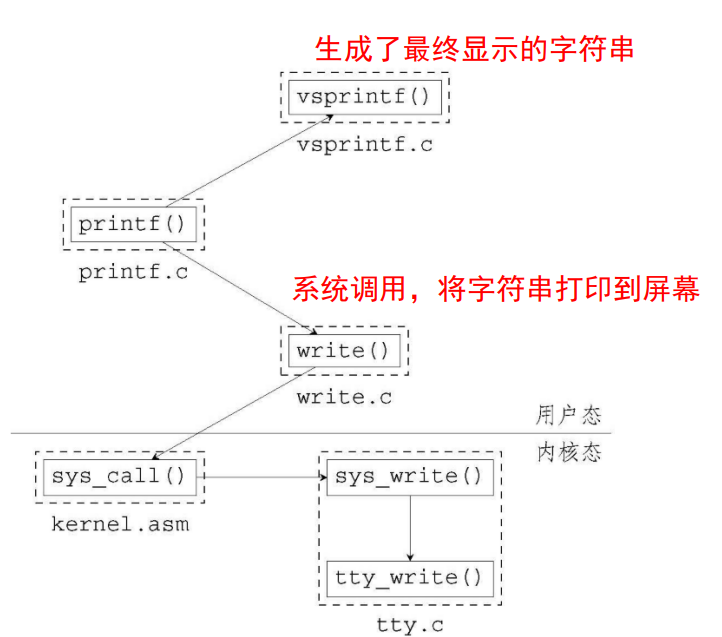
实验给出的printf实现如下：



printf又调用了vsprintf，其代码如下：



printf具体打印流程如下图：



根据上述代码，我们只需要修改case ‘s’这一分支的代码内容即可。具体代码实现为：

**case ‘s’:**

**strcpy(tmp, \*((char\*\*)p\_next\_arg)); //按照字符串形式拷贝**

**strcpy(p, tmp);**

**p\_next\_arg += 4;**

**p += strlen(tmp);**

**break;**

整体实现并不复杂。在主体函数中，我们设置了三个调用printf的进程，具体代码见下图：



其中，TestA和TestB分别调用printf的%s方法进行打印。编译运行该项目，bochs页面显示如下：

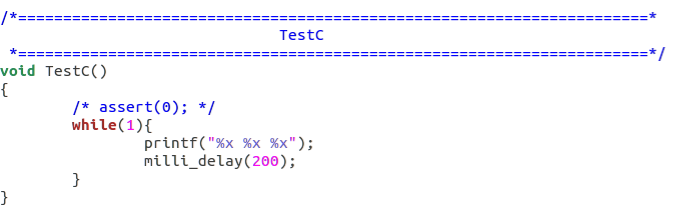


可以看到，程序成功输出了对应的字符串信息。

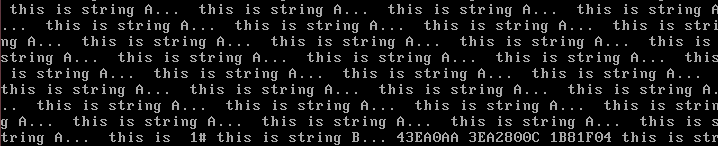
### printf漏洞发现

该实验的printf存在格式化字符串漏洞。首先，我们通过实验展示这一漏洞：

在前文中，我们设置了三个调用printf函数的进程。最后一个进程，我们编写的printf语句为printf(“%x %x %x”)，缺少了占位符后的实际数字。



屏幕输出如下：

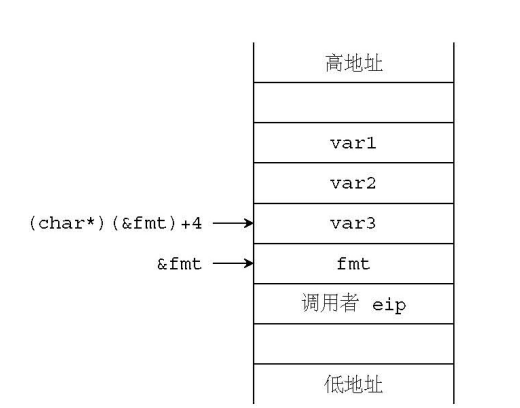


其中，printf(“%x %x %x”)输出的值为最后一行的43EA0AA、3EA2800C、1B81F04。虽然代码中我们没有设定占位符的实际数字，但它仍然打印出了一些信息。这些信息是内存中的信息，借助该漏洞，攻击者可以非法获取内存信息。

接下来，我们介绍格式化字符串漏洞出现的原因：

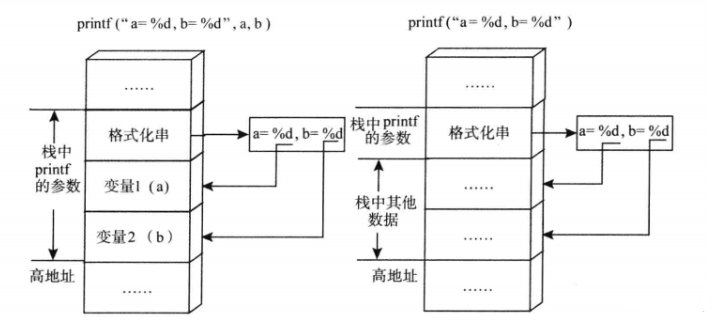
格式化字符串是由普通字符（包括%）和转换规则构成的字符序列。普通字符被原封不动地复制到输出流中。转换规则根据与实参对应的转换指示符对其进行转换，然后将结果写入到输出流中。

转换规则由可选的部分和必选部分组成。其中只有转换指示符type是必选部分，用来表示转换类型，如%x、%d、%s等。本实验中，格式化字符串的参数fmt是通过栈传递的，具体分布如下：



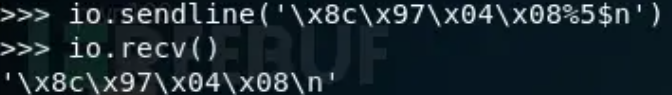
由于格式化字符串变长参数的特性，在实际运行中，如果fmt的符号说明个数超过待匹配的参数个数，即有更多的符号说明需要被匹配时，printf会根据解析结果和调用约定去取栈上相应的值并输出，程序就会打印出栈中本来不应当被访问的信息。

之后，我们就能够通过这种方式构造特定的payload，获取到内存相关的信息，例如通过泄露的栈变量计算参数偏移量、通过栈上的信息计算程序基地址以及libc基地址等。



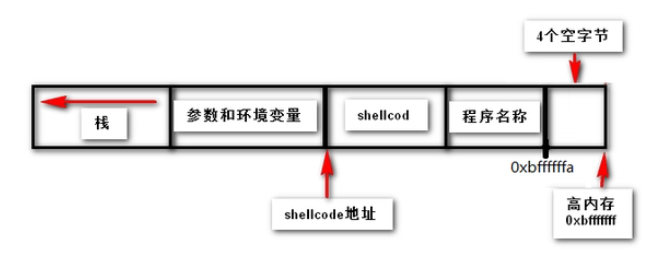
当然，上述内容只能达成读栈上任意信息的操作，利用格式化字符串漏洞，我们还可以向栈上的任意位置写信息。printf有一个特殊的格式化控制符%n，和其他控制输出格式和内容的格式化字符不同的是，这个格式化字符会将已输出的字符数写入到对应参数的内存中。举个例子，假设构造一个payload为“\x8c\x97\x04\x08%5$n”，其中0804978c是.bss段的首地址，一个可写地址，那么，执行之后该地址的内容就从0变为4。







既然可以通过printf对栈上的信息进行读写，那么，我们就可以进一步进行攻击，比如get shell等。



至于具体的防御措施，可以有以下几种方案：

* 静态代码分析与检查：在程序真正运行之前，检查printf的输入是否合理。如果符号说明个数少于待匹配的参数，或者参数实际值为NULL，那么直接报错并终止程序运行。
* 堆栈保护：堆栈保护是一种可以使程序在堆栈溢出时退出并停止执行的技术。通过在启用堆栈保护技术，则可以在程序运行时对堆栈进行动态监测和保护，防止对栈信息的非法读写。
* ASLR：我们可以通过地址空间布局随机化（ASLR，Address Space Layout Randomization）技术来打乱地址空间中的布局。ASLR将内存空间分为多个区域，每个区域都有不同的访问权限，且相邻的两个区域之间有一个空隙，用于避免两个区域的数据互相溢出。这可以避免攻击者以确定的方式访问系统内存。

# 实验结果总结

（对实验结果进行分析。并理论联系实际，思考并列出本实验对应的OS原理的知识点，并说明本实验中的实现部分如何对应和体现了原理中的基本概念和关键知识点。）

TTY任务的运行方式：

在TTY任务中执行一个循环，这个循环将轮询每一个TTY，处理它的事件，包括从键盘缓冲区读取数据、显示字符等内容。

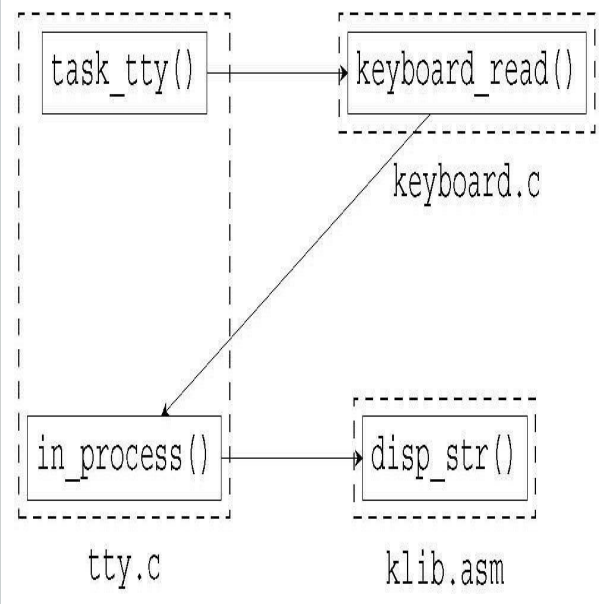
并非每轮询到某个TTY时，箭头所对应的全部事件都会发生，只有当某个TTY对应的控制台是当前控制台时，它才可以读取键盘缓冲区，键盘和显示器算做每一个TTY的一部分，它们是公用的

轮询到每一个TTY：

1. 处理输入——查看是不是当前TTY，如果是则从键盘缓冲区读取数据。

2. 处理输出——如果有要显示的内容则显示它。

如下图所示task\_tty( )是一个循环，它不断调用keyboard\_read( )，而keyboard\_read( )从键盘缓冲区得到数据后会调用in\_process( )，将字符直接显示出来。



Make code和break code：

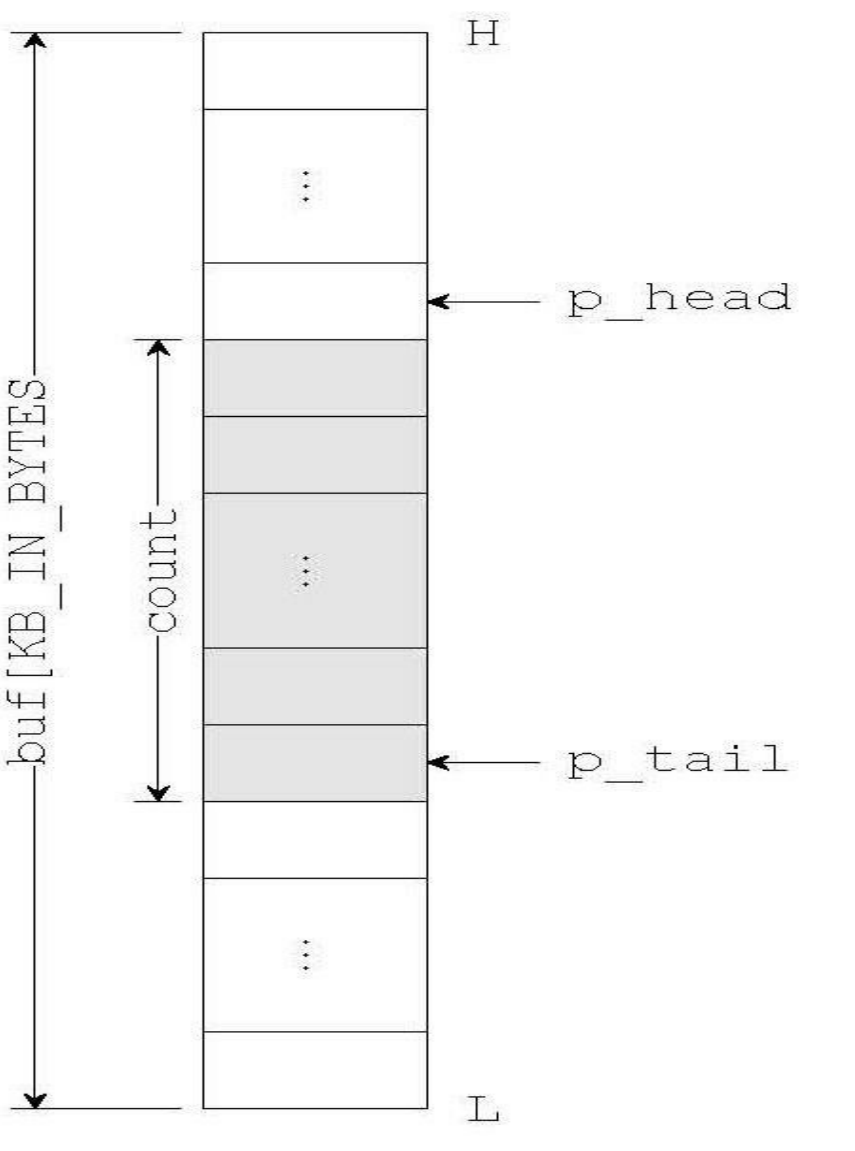
（1）敲击键盘所产生的编码被称作扫描码（Scan Code），它分为Make Code和Break Code两类。当一个键被按下或者保持住按下时，将会产生Make Code，当键弹起时，产生Break Code。除了Pause键之外，每一个按键都对应一个MakeCode和一个Break Code。

（2）在敲击键盘时，首先产生扫描码。操作系统获取扫描码，然后把扫描码解析成ASCII码。接收到的扫描码类型是Make Code，例如假设我们按下s，那么键值就是keymap[0x1F\*MAP\_COLS],然后在根据其在显示器上打出对应得字符

（3）作用是用以放置中断例程收到的扫描码，键盘的输入缓冲区由键盘驱动或键盘控制器实现，是内存的一块区域。缓冲区作用如下图所示

多个tty能够并发处理输入输出的原理是通过多进程或多线程实现的。每个tty都对应着一个进程或线程，它们可以同时运行，互不干扰。当用户在一个tty中输入命令时，该tty对应的进程或线程会读取用户输入的内容，并进行相应的处理。处理完成后，该进程或线程会将结果输出到对应的tty中，供用户查看。

在Linux系统中，每个tty都对应着一个虚拟终端设备文件，当用户登录系统时，系统会为其分配一个tty，并将其连接到用户的终端设备上。用户可以在该tty中输入命令，系统会将命令传递给对应的进程或线程进行处理，并将处理结果输出到该tty中。



多个tty能够并发处理输入输出的原理：

多个tty能够并发处理输入输出是通过多进程或多线程实现的。每个tty都对应着一个进程或线程，它们可以同时运行，互不干扰。当用户在一个tty中输入命令时，该tty对应的进程或线程会读取用户输入的内容，并进行相应的处理。处理完成后，该进程或线程会将结果输出到对应的tty中，供用户查看

在Linux系统中，每个tty都对应着一个虚拟终端设备文件，当用户登录系统时，系统会为其分配一个tty，并将其连接到用户的终端设备上。用户可以在该tty中输入命令，系统会将命令传递给对应的进程或线程进行处理，并将处理结果输出到该tty中

# 个人分工及心得体会

（每个人分别填写自己在本次实验中的分工，并总结实验的心得体会。）

## 张子航

完成了本次实验的基础题，并撰写实验报告的对应部分

在本次实验中，我学习到了如何实现键盘输入，掌握了显示器输出的基本方法，并对于tty任务子系统的基本功能和方法有了更深的理解掌握；此外，我还基本掌握了如何在os中实现读取键盘输入的扫描码加载到缓冲区并通过tty读取并进行解码，来实现对已读键盘输入的正确处理及输出，以及通过address register对指定的CRT controller data register进行修改实现指定光标位置的改变，实现上下翻页，还学会了tty实现并发处理输入输出的基本原理，并通过实验设计了个性化的tty

## 辜汝曦

完成本次实验要解决的问题的第2题、第5题，并撰写实验报告的对应部分、以及实验目的及实验内容部分。

在本次实验中，我主要通过编写代码扩展了printf函数处理%s占位符的功能。通过本次实验，我对printf的工作机制更加熟悉，也了解了格式化字符串漏洞的攻击流程。目前我学习的漏洞很大一部分都是溢出漏洞，也就是在程序运行时，输入数据超出了内存空间的范围，导致数据溢出，并覆盖程序中的其他数据，从而导致程序崩溃或执行其它意外的行为。这次实验使我对溢出漏洞的理解更加深刻。

## 杨馨悦

完成本次实验要解决的问题的第3题，并撰写实验报告的对应部分。

通过本次实验，我了解了输入/输出系统的基本结构，了解了显示器的输出概念，了解了和键盘输入原理了解了TTY（TeleTYpe）的基本架构以及相关扩展功能。通过详细阅读代码及教材解释，我明白了多个TTY能够并发处理输入输出的原理，并搞清楚了TTY具体的工作流程，进一步加深了我对操作系统中I/O系统的了解。

## 赵敏

完成本次实验要解决的问题的第1题，并撰写实验报告的对应部分。

在本次实验中，我学习了键盘中断的添加过程，学会了键盘输入原理和显示器的输出概念，意识到键盘缓冲区存在的必要性以及对其的处理方式，了解TTY的基本架构以及相关扩展功能，学会如何设置不同程序处于不同环级，最后了解printf的实现机理，熟悉了printf的实现机制，键盘和终端的显示和连接在我这里终于不是一个理所当然的magic过程，而是每一步都是精心的设计和处理，受益匪浅。