**操作系统实验**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Lab2：

内核及用户程序的加载、idt的初始化、printf参数处理、

中断处理程序的分发(irqHandle)、键盘中断、

getChar及getStr实现

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

院系：人工智能学院

姓名：石睿

学号：211300024

班级：操作系统-2023春季学期

邮箱：[211300024@smail.nju.edu.cn](mailto:211300024@smail.nju.edu.cn)

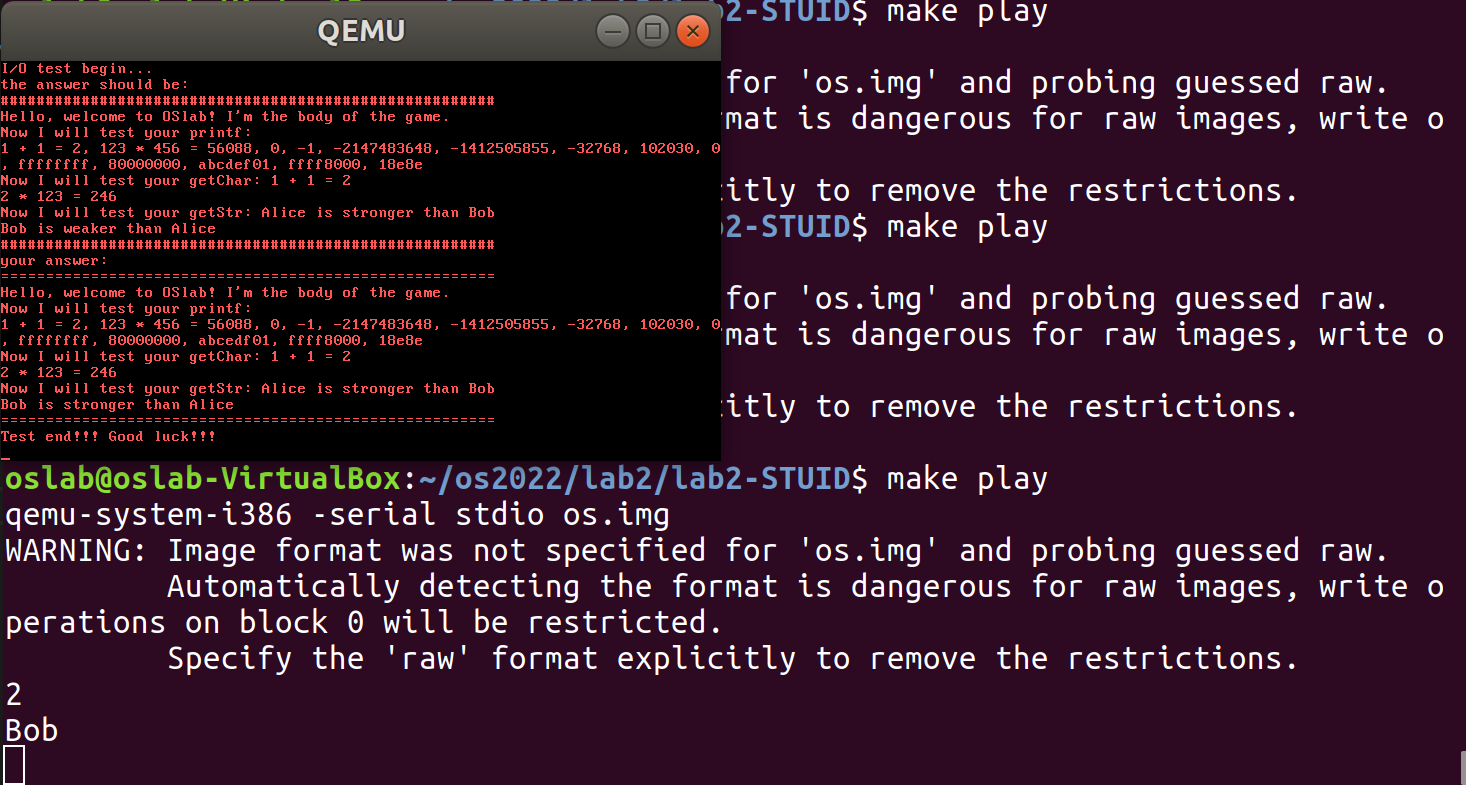
实验时间：2022.4.4

1. **实验进度**

我已经完成了Lab2的所有内容。

1. **实验结果**

结果如下图所示，具体实现过程在第三部分介绍。



**可检测的Task1：printf参数处理、系统调用、字符串的显存打印**

**可检测的Tak2：getChar和getStr实现 - 键盘中断、两者的系统调用**

1. **实验修改的代码位置**
   1. **内核的加载**

正确装载elf文件以及内核的步骤有以下3步：

Step1：通过readSect函数把ELF文件整体读入正确的固定位置

Step2：找到ELF头、程序头表的位置，并得知表项的数量

Step3：通过程序头表中的type信息，逐个判断当前段是不是需要被装入。

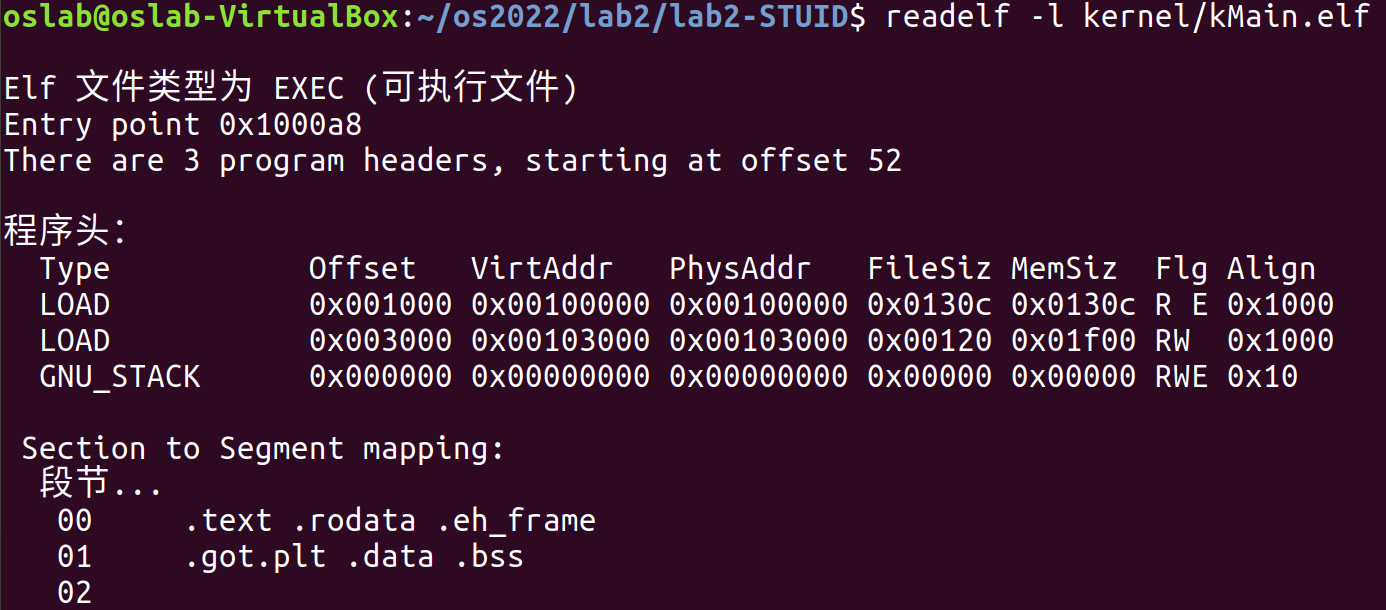
3.1 段的装入。如果当前段的type是PL\_load,需要知道相较于elf起始位置的偏移量ph->off、被加载到的物理内存中的位置ph->paddr。

【具体实现】\*(unsigned char\*)(ph->paddr+j) = \*(unsigned char\*)(elf+ph->off+j);

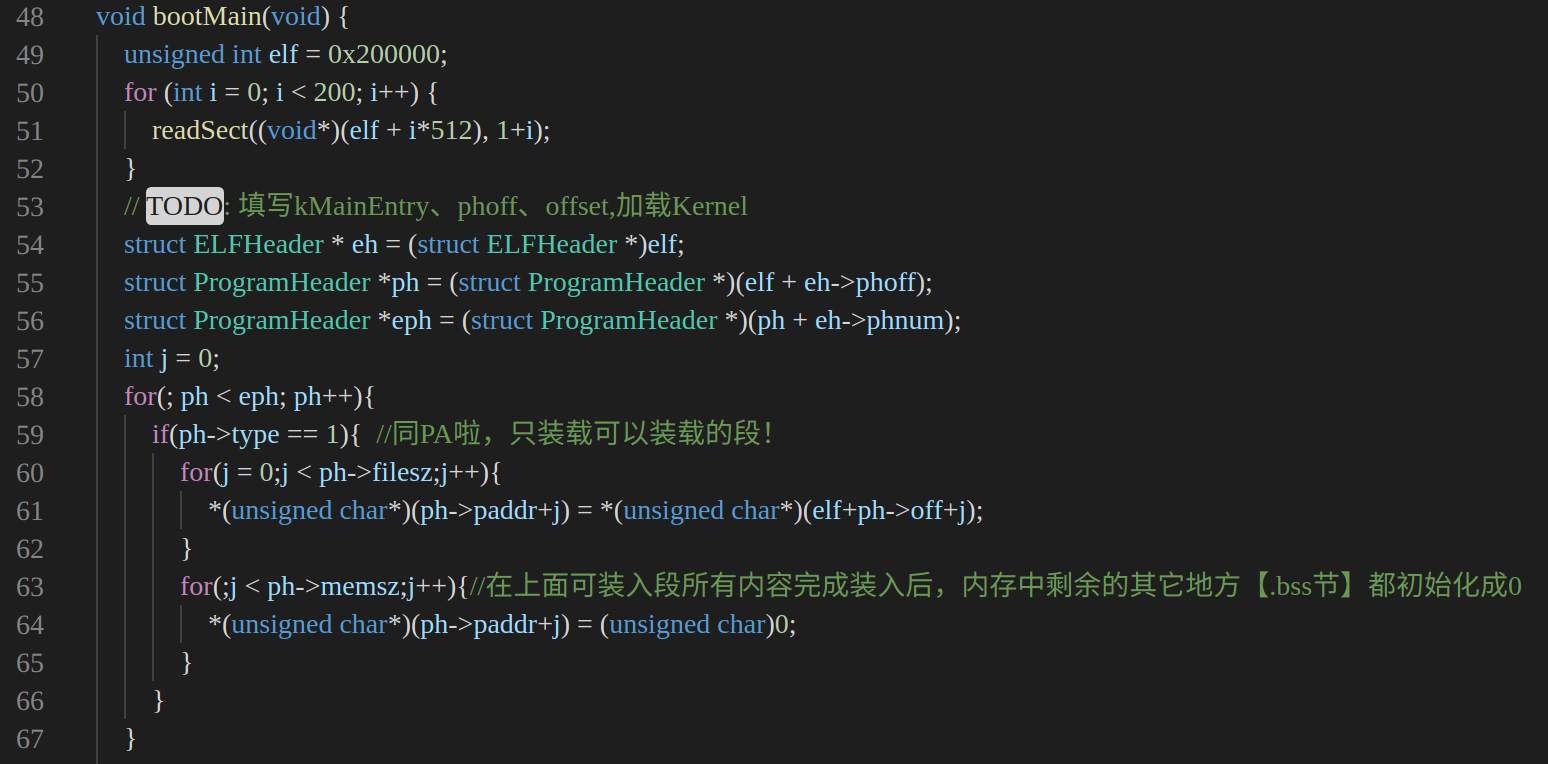
3.2 bss的初始化。.bss段在进程的虚拟存储映象中没有被分配空间，而在真实内存空间中是被分配空间的了。其大小是memsz-filesz。

【具体实现】\*(unsigned char\*)(ph->paddr+j) = (unsigned char)0;

而本次框架代码中给出的代码未实现step2和step3，没有根据程序头表中的内容进行装载，导致数据段的装载是错误的 。



**本次实验编译后elf文件中程序头表的信息**



**修正后我的实现如上**

* 1. **完善中断机制、提供系统服务函数（idt的初始化）**

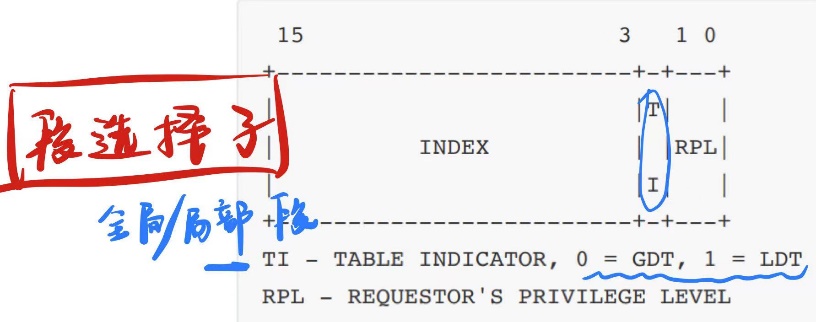
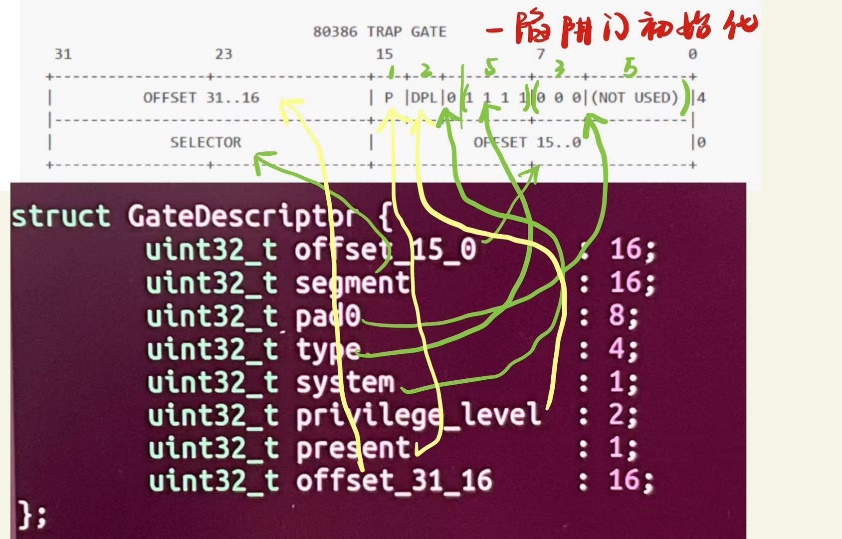
本阶段需要做一些列的初始化：idt、8259a、gdt、tss、vga、keyboard device的初始化，

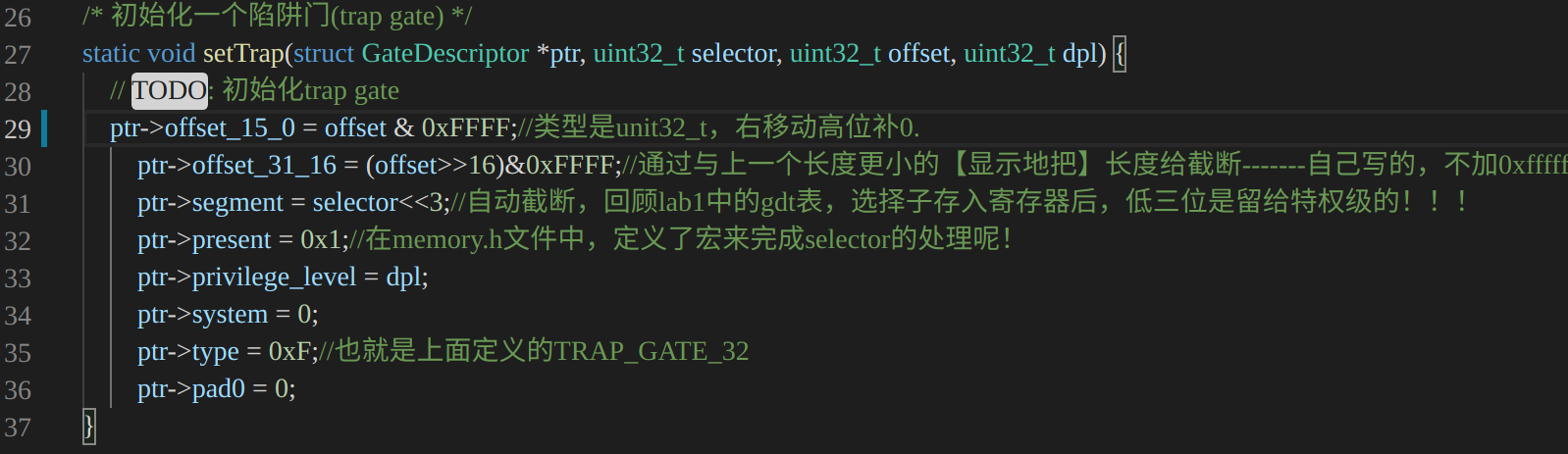
其中只需要补全idt初始化函数initIdt()即可。

限于篇幅，此处仅以陷阱门（Trap Gate）为例进行分析，具体GateDescriptor中的各个元素 和陷阱门描述符的对应关系如下图所示。其中需要注意的是段选择子selector，一共16位寄存器

中保存段选择子的下标+第三位的特权级，而门描述符中只保存它的下标（idt的什么位置）。所

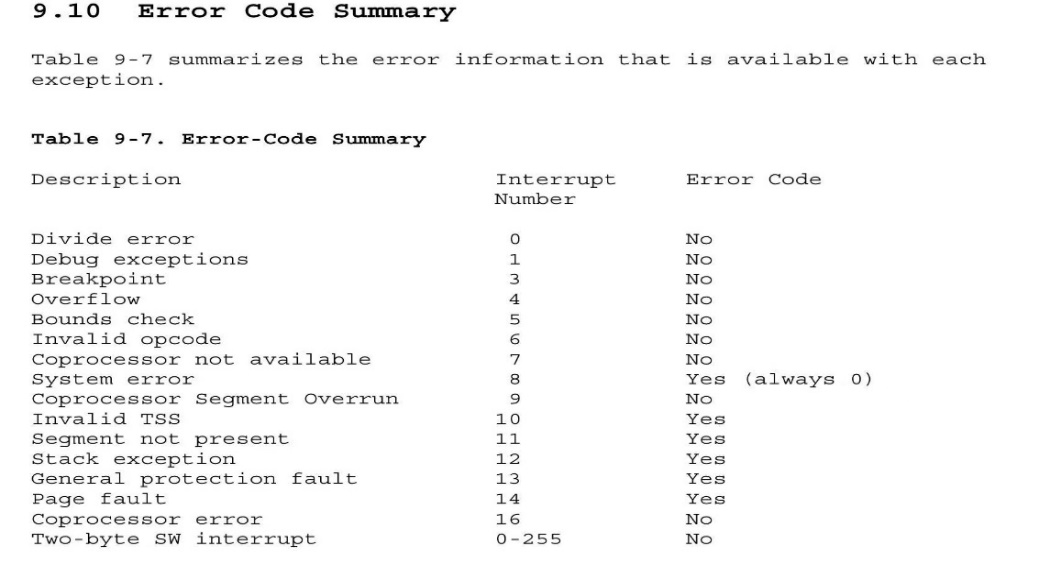
以在实现的时候需要把selector右移动3位后才可以赋值给segment！



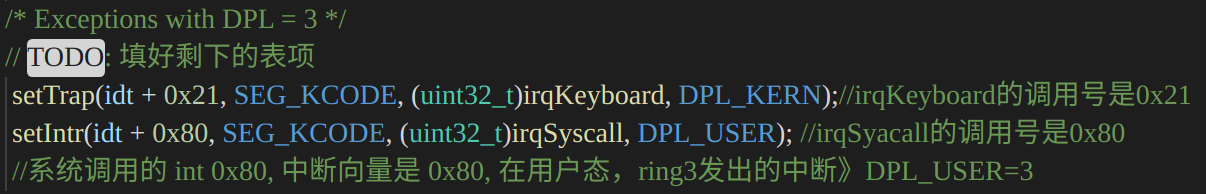


**陷阱门trap gate的初始化如上所示**

之后需要对idt表项中设置中断处理函数，首先要完成对特定中断向量的设置。



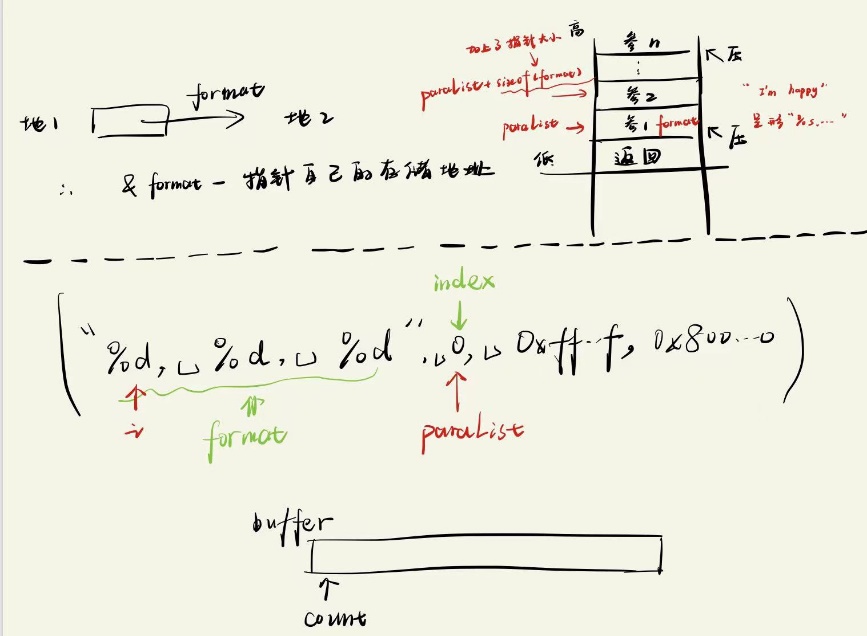
在查阅了Intel 80386手册后，找到了对于特定的中断向量的中断号，以及他们对应的出错码（它们的出错码会在doIrq.S中被压入栈中）。其次，完成本次实验中后续中断处理程序irqKeyboard,irqSyscall的们描述符的初始化。它们分别是陷阱门和中段门，实验手册中提到了它们的中断向量号分别为0x21和0x80，并且特权级（DPL）分别是内核态（ring0）和用户态（ring3），具体实现如下所示。



* 1. **加载用户程序、printf解析参数**

加载用户程序和加载内核几乎完全相同，只需要注意一下程序入口偏移量需要减去0x200000.

Printf解析参数和PA中实现的略有不同，但大体一致。PA中采用了va\_list，是C语言中解决函数读入不定参数所定义的宏，可以解决printf参数不定的问题。本Lab中除了采用了C语言调用惯例中实参从右向左逐步压栈的方式查找参数，其他和PA中的实现相同，代码中相关变量的图示如下。



* 1. **中断处理程序分发、键盘处理与打印到显存、getChar、getStr**

中断处理程序的分发和PA中也类似，在irqHandle根据中断向量号调用不同处理函数即可

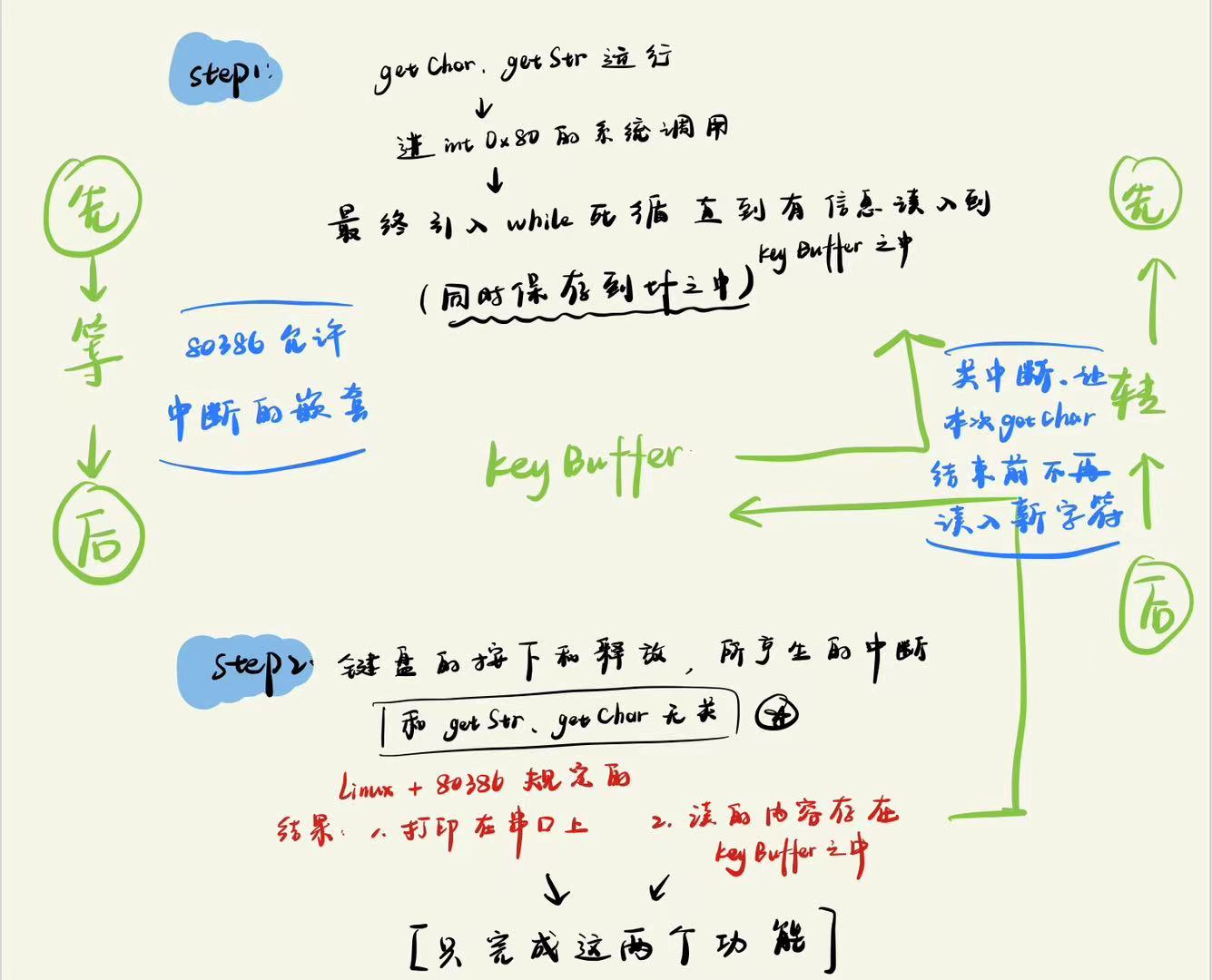
键盘的处理根据读入的键盘码分别对读入是退格符、回车符、正常字符分三种情况进行解析。

在解析键盘之前，首先要明白getChar和键盘中断之间的关系。也就是app/main.c中char num = getChar();的处理流程（此处仅仅以getChar为例）。

Step1：getChar处理。syscall->int 0x80->irqHandle->syscallGetChar。此时如果键盘没有输入就一直等待，直到有信息读入keyBuffer之中。这一步可以work的基础是Intel 80386支持中断的嵌套，在syscall（一个用户级ring3的系统调用）中可以优先处理中间插入的键盘中断（一个内核级的中断），并在键盘中断处理完成后返回继续处理syscall！

Step2：键盘中断。在getChar的while(1)循环中敲击键盘，硬件产生了键盘中断，由于优先级会先处理这个键盘中断。在个人实现的时候，把键盘读入的信息打印到串口上，并且把读入的内容放到了keyBuffer之中，中断处理完毕，iret返回。

Step1和step2用流程图说明可能会更加清楚这个过程。



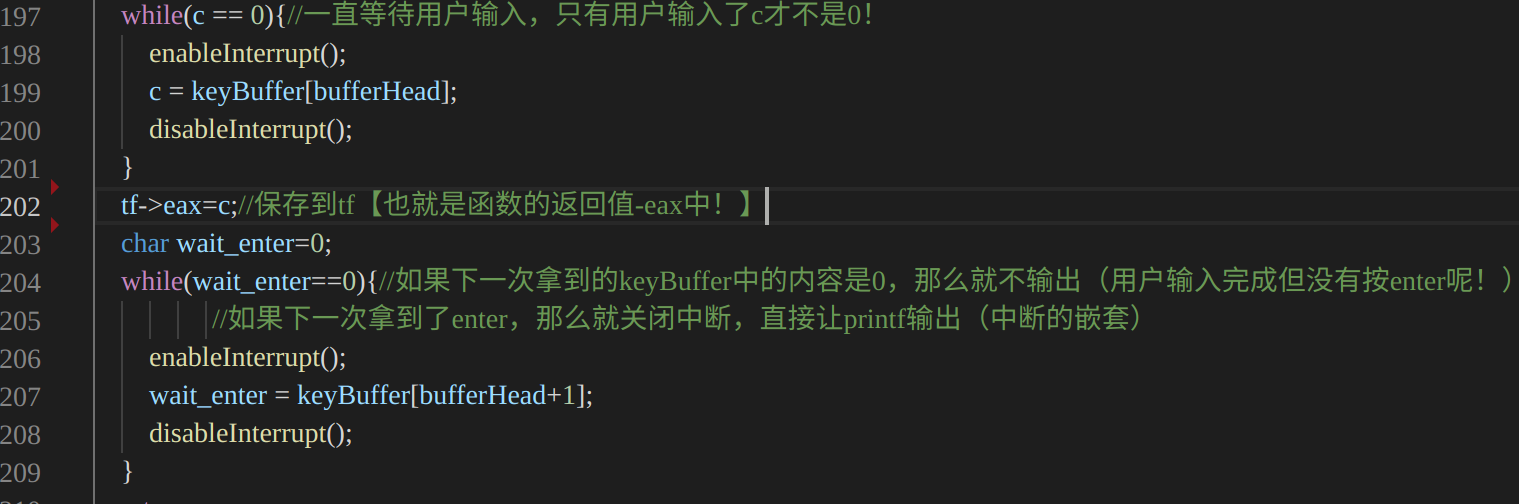
此外还有两点需要格外注意：

1. putChar只有在内核级（Kernel文件夹下）中才可以使用（因为框架的声明）。

getChar、getStr在用户级和内核级都可以使用，也就是在app/main.c中使用的。

1. 为了避免在输入完成的时候，由于在键盘按下到松开的时候有一段时间，会读入很多个相同按键的内容到keyBuffer之中。为了避免这个问题，根据两个函数完成输入状态的方法：getChar以读入一个字符判完成，getStr以读入\n判完成，以及框架代码中提供的开中断函数enableInterrupt()和disableInterrupt()两个函数。以getChar为例，让在【没输入的时候】开中断->关中断->开中断->关中断…，保证了一直可以输入进来。

在【输入完成的时候】保持关中断出while循环，这样就无法再继续读入键盘仍然按下时候的信息啦！syscallGetChar的部分实现实现如下图所示。



**四、思考题的看法**

EX1：计算机系统的中断机制在内核处理硬件外设的I/O这一过程发挥了什么作用？

A1：因为外设的I/O的速度会比CPU速度慢很多很多。引入中断机制有以下两个作用：

1. 让外设来做I/O的同时,CPU可以在这个过程中同时完成其他的任务，而不必一直等待I/O的完成.
2. 在硬件完成I/O的时候再使用中断让CPU不做现在的工作而来处理原来的进程I/O之后的工作呢！

并且，相比较 轮询和DMA的方式，中断是一种比较灵活高效的方式

EX2：IA-32提供了四个特权级，但TSS中只有3个堆栈位置信息，为什么没有ring3的信息？

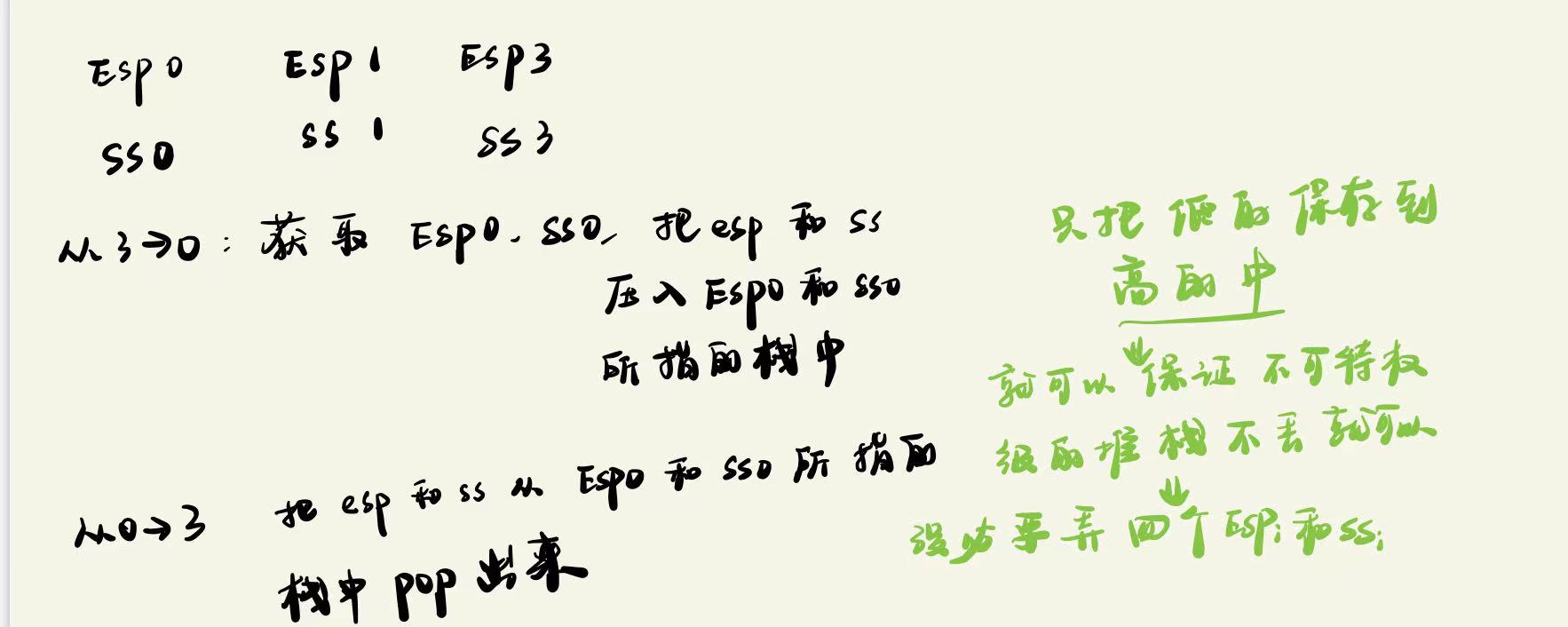
A2：特权级变化的时候栈的变化

Situation1：低-高（如ring3->ring0）：从TSS获取高特权级目标栈段，将低特权级的信息压入高特权级栈中（ss和esp）

Situation2：高-低（如ring0->ring3）：将低特权级栈中取出并恢复到ss和esp。

所以在TSS中，从高-低的时候，CPU是不需要在TSS中寻找低特权级目标栈的，所以最低特权级ring3的栈段是没有必要保存在TSS中的呢！（没有人比他级别更低，所以没有人会把它的栈段保存到“ss3”和“esp3”之中呢！）。

下图通过ring0<->ring3的栈转换过程详细说明一下。



EX3：系统调用前后若不保存被调用者保存寄存器会不会产生不可恢复的错误？

A3：会的！TSS的功能只能把栈段恢复，但是并没有恢复原来级别进程中的全部状态，如在计算过程中的寄存器，它们的值还是错误的数值，这样就会出现不可恢复的错误。

EX4：printf的%d、%x、%s、%c转换说明符的含义。

A4：

%d – 转换成有符号的十进制整数

%x – 转换成无符号十六进制整数

%s – 转换成字符串

%c – 转换成一个字符