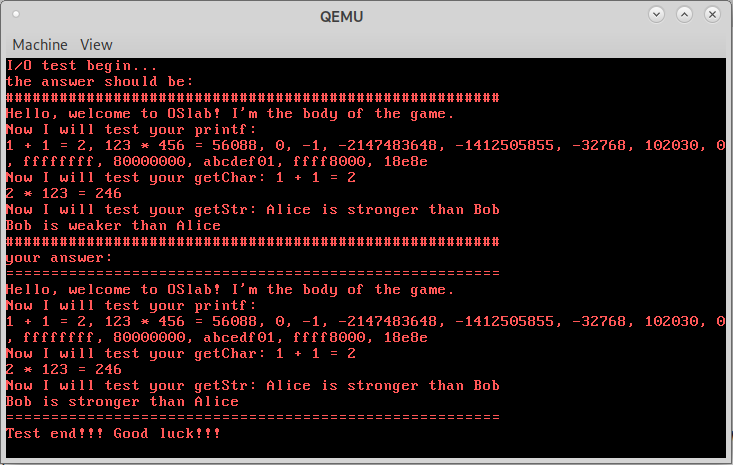
姓名：张涵之 学号：191220154 邮箱：[1683762615@qq.com](mailto:1683762615@qq.com)

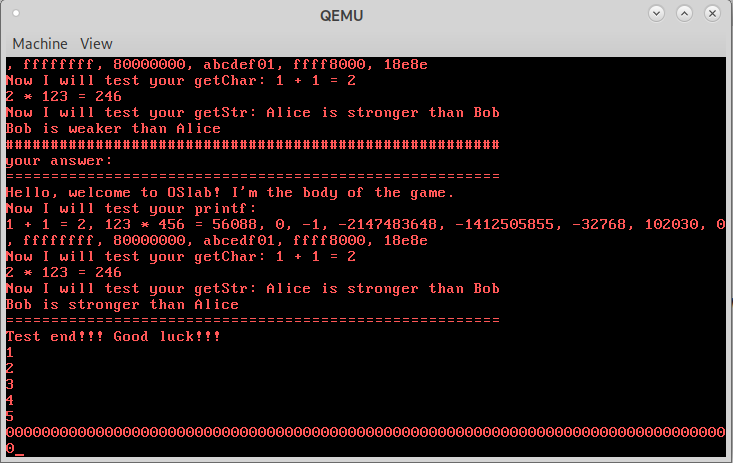
**注意事项：我用虚拟机打包传到课程网站以后，尝试把压缩包下载解压，直接make可能会报错Permission denied，建议先对lab文件夹chmod修改权限以后再make。**

实验进度：我完成了所有内容

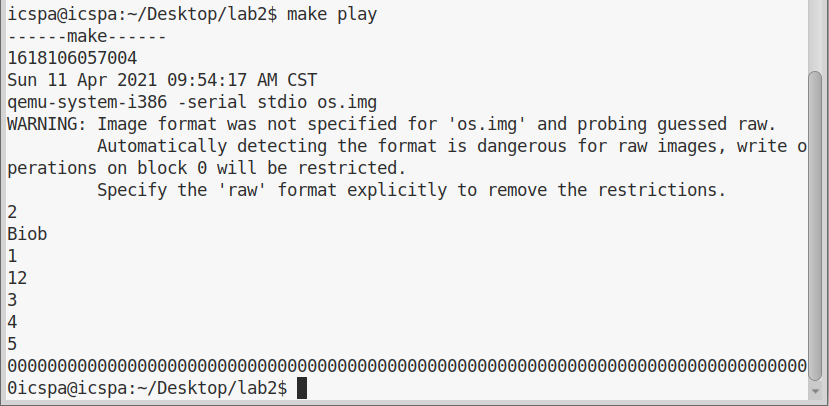
实验结果：下图为官方测试代码运行效果



下图为自动换行、换行符换行及滚屏测试

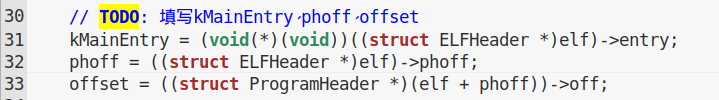


下图为控制台输出结果，其中测试了两次删除，第二行的i和第四行的1



实验修改的代码位置：

bootloader/boot.c //对kernel代码的装载



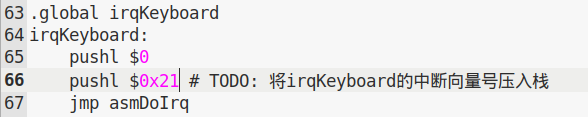
bootloader/start.S //跳转执行bootMain之前设置esp



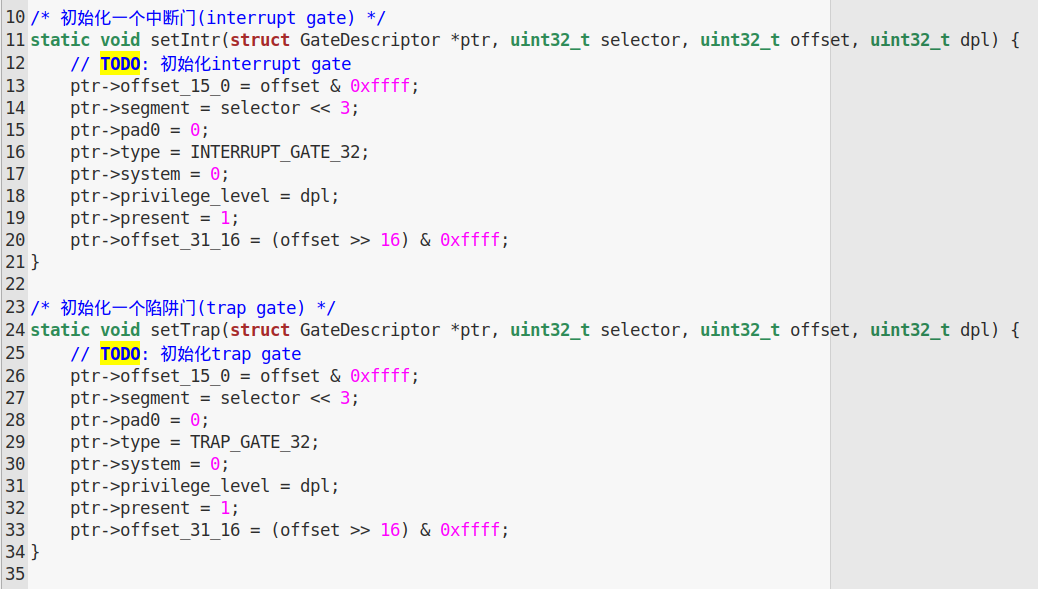
bootloader/Makefile //修改makefile以解决boot block too large问题

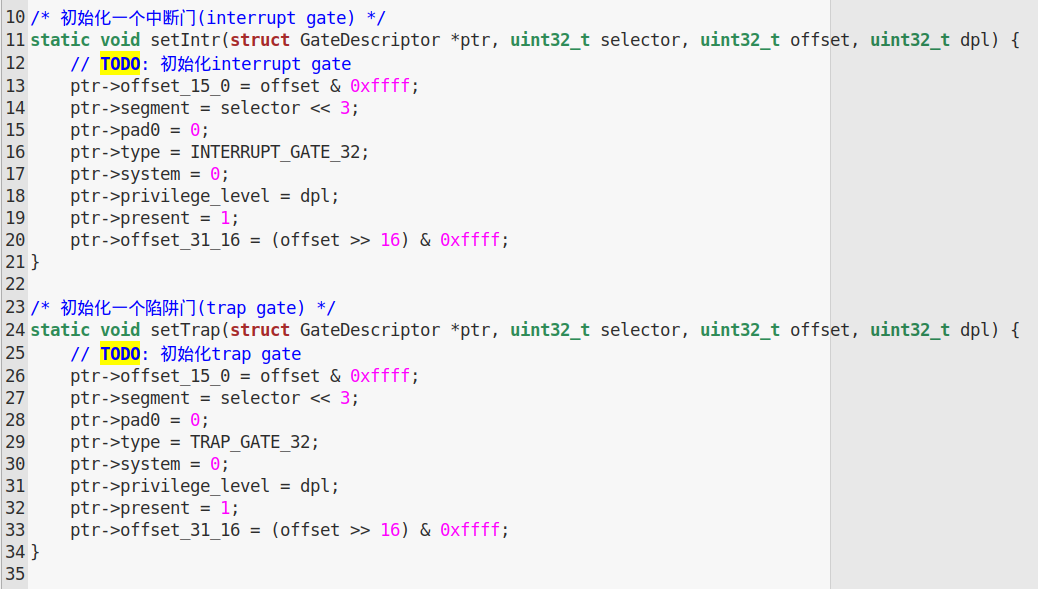


kernel/kernel/doIrq.S //添加irqKeyboard中断向量号的处理

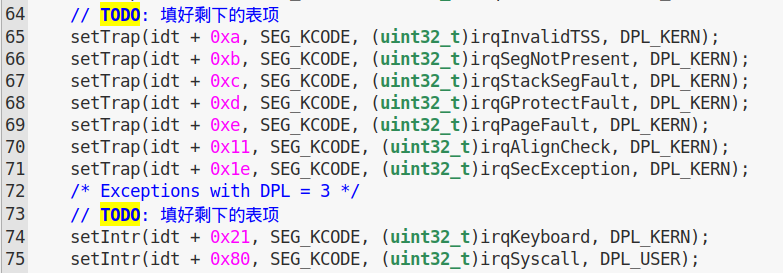


kernel/kernel/idt.c //初始化中断门和陷阱门





//填写陷阱和中断表项，其中除了系统调用是用户态，其他均为内核态



kernel/kernel/kvm.c //仿照内核加载方式加载用户程序

//其中只需修改elf首地址为0x200000，以及扇区的加载从201而非1号开始



//以上代码完成了内核和用户代码的装载，实现了中断和陷阱门的初始化

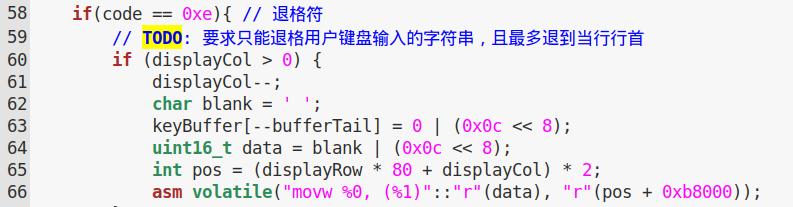
//中断表的加载和调用，一下代码将实现printf、getChar和getStr这些函数

kernel/kernel/irqHandle.c //中断处理程序调用



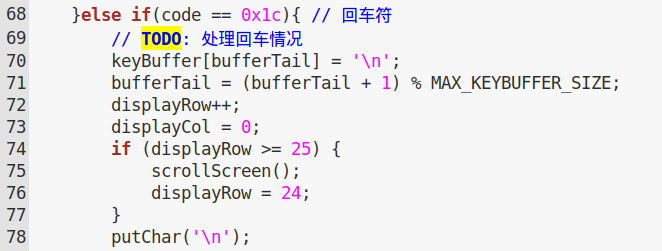
//KeyboardHandle对退格符的处理：若当前列号大于0

//则列号递减，输出空白符覆盖，清空keyBuffer尾部数据并修改尾指针



//KeyboardHandle对换行符的处理：行号递增，列数置0，’\n’存入buffer

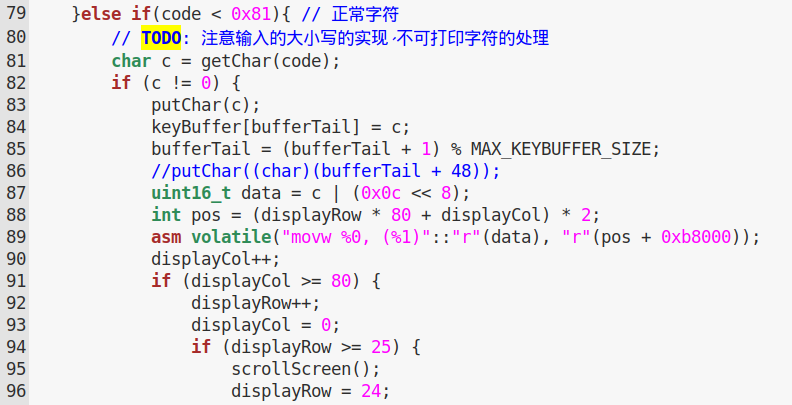
//若当前行数大于25则调用滚屏函数处理，在控制台也输出换行符



//KeyboardHandle对正常字符的处理：不可打印字符按0处理

//大小写的实现键盘模块已提供，此处只需将对应ascii码存入buffer即可

//列号递增，若光标移至行尾（行号80）则换行，仿照换行符处理即可



//syscallPrint需根据参数从用户栈中读出字符数据，这部分框架代码已提供

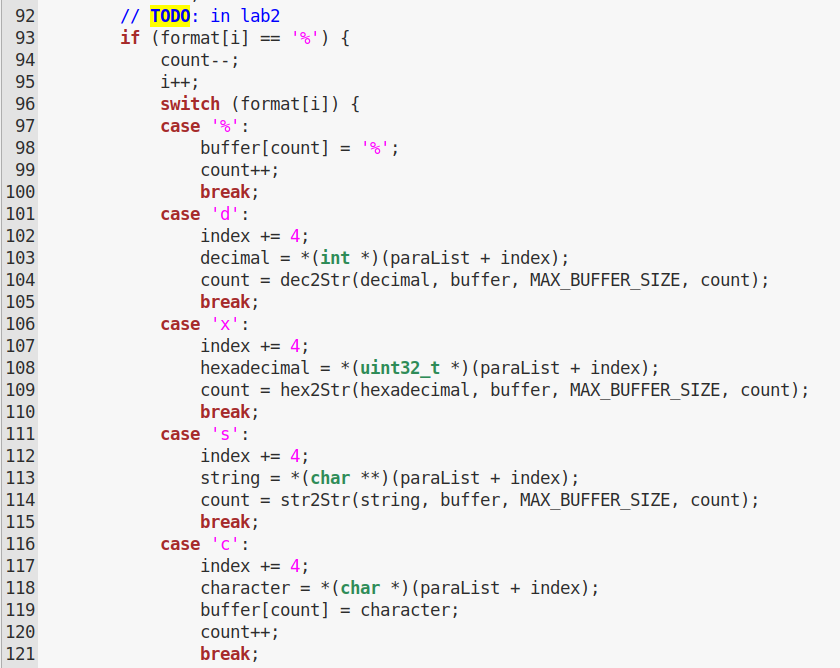
//显示的实现原理、对换行和滚屏的处理与KeyboardHandle类似，不再赘述

lib/syscal.c //printf函数中格式化输出的实现

//对十进制数、十六进制数、字符串和单个字符分别进行了实现

//具体为调用提供的转换函数，将转换得到的字符存入buffer

//需要注意的是此处单个字符也按四字节对其存储，若按1字节会出现问题



//sycallGetChar的实现：查阅资料可知getChar的实现原理是有在用户按下换行符时

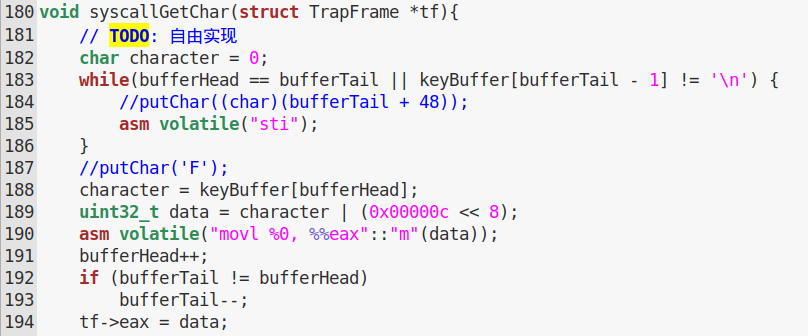
//取当前输入流中第一个字符，其余的字符（如果有的话）仍然留在输入流中

//因此当用户没有输入（buffer头尾指针重合）或没有按下换行符时

//程序反复开中断来等待用户输入，输入完毕后，则取当前buffer的头部

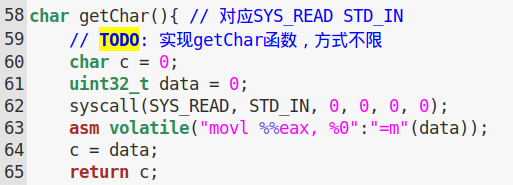
//将返回值存入eax，并将头指针向后移一位，此时若头尾指针没有重合

//说明输入流中还有数据，将尾指针向前移一位，避免将换行符也留在buffer内



//在lib/syscall/c/getChar()中调用syscall，间接调用上面的函数

//由于系统函数返回值存储在eax中，此处用汇编指令将其取出

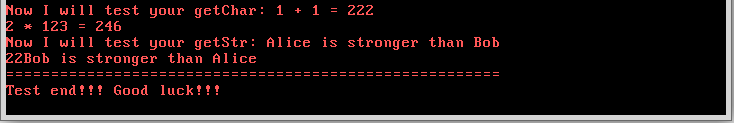
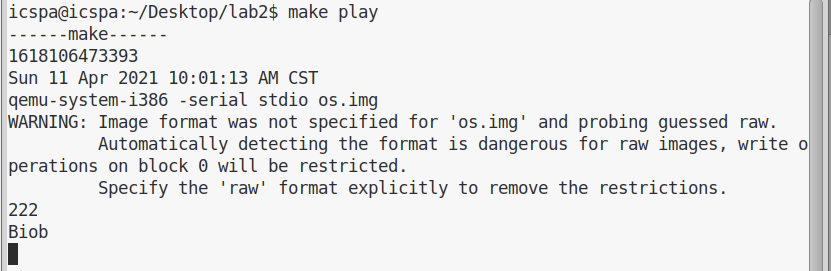


//对于取输入流中第一个字符，其余留在缓冲区中的功能作如下测试

//首先输入222，按下换行符后，再输入Bob，按下换行符

//此处i为打字时的手误，同时再次验证了退格键功能

//可见第一个2被getChar返回了，后面两个则与Bob一同被getStr捕获



//syscallGetStr的实现：仿照syscallPrint对参数进行处理，获得字符串地址和长度

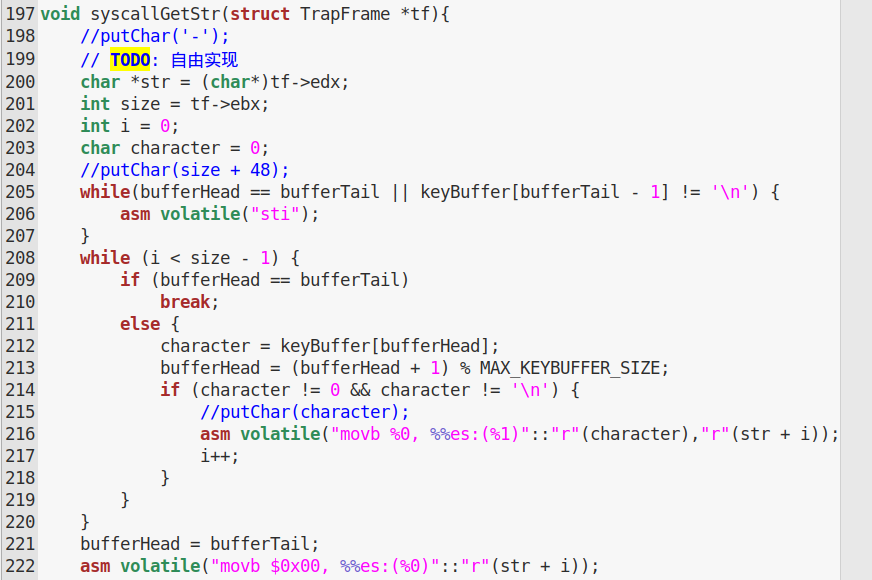
//当没有输入或没有换行时开中断等待用户，原理与syscallGetCar相同

//完毕后从输入流中循环取出size-1个字符存入用户栈（最后一位必须预留给’\0’）

//如果未取满size-1个字符便出现头尾指针相遇，则提前跳出循环

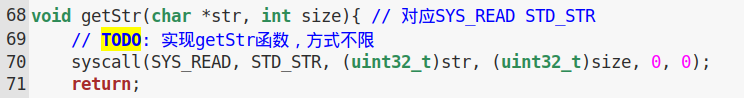
//如果取满size-1个字符后输入流中还有数据，则将这部分数据丢弃

//完成后将头尾指针置于一处，并将’\0’存入字符串末端位置



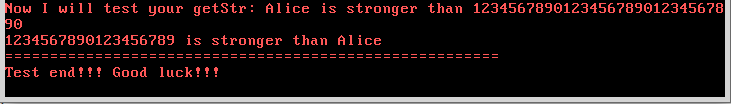
//在lib/syscall/c/getStr()中调用syscall，间接调用上面的函数

//系统函数将字符串存入对应的地址，故此处无需其他操作



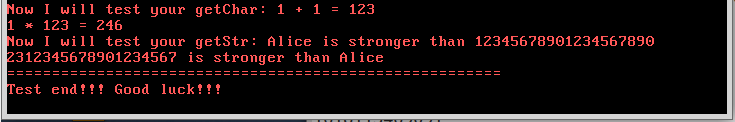
//针对getStr返回字符串长度进行了如下两次测试

//第一次输入了30个字符，输出长度为19，另有一个隐式存在的’\0’



//第二次先输入123，换行，在输入20个字符，可见输出层的仍为19

//其中前两个字符为getChar遗留在缓冲区的，后17个为getStr输入的



思考和总结：

本次实验帮助我更好地理解了ICS中陷阱、中断和函数调用的介绍。其中不同部分（如内核框架、用户程序、装载程序、函数库都分别放在不同的子目录下，不同模块之间的结构和调用关系非常清晰，对我理解操作系统各模块的划分和相互配合有很大的帮助。

在实验过程中，我主要遇到了两次困难。第一次是实现printf并测试成功后，对getChar和getStr的“自主实现，方式不限”感到一筹莫展。首先我观察了printf的代码，认为getStr类似简化版的（不带格式输入的）scanf，可以仿照实现。可是getStr涉及到与用户交互、对键盘模块的调用和对buffer数组的读取，较printf有更多需要考虑的地方。

在syscall系列函数中，该如何隐式地调用KeyboardHandle模块呢？测试很容易得出，显示地调用该模块会造成大量重复乱码的输出。观察发现在用户程序装载之前（在写完这部分代码之前测试printf）或者之后（所有测试用例运行完）的while(1)循环中，KeyboardHandle均可用于从键盘获取输入，并在控制台和屏幕输出。故KeyboardHandle应该不需要显式调用，而是在没有其他中断的情况下受键盘控制自动调用的。因此设置while循环，在获得满足条件的输入（换行符）前反复调用内联函数开中断，每次调用sti都开启下一条指令的硬件中断，KeyboardHandle每次处理一个按键输入，则重复和乱码的问题得到了解决。

syscallGetStr可以仿照syscallPrint，根据输入的参数（字符串地址）将获得的字符依次写入内存中，作为void函数，它不需要考虑返回值的处理。可是getChar没有任何参数，却有返回值char，它对应的系统函数也是void类型。我们知道getChar首先会调用syscall，然后到doIrq里寻找对应的irqHandle，由irqHandle调用syscallHandle，调用sysRead，最后调用syscallGetChar，那么此时返回值该如何经由如此多的步骤传递到getChar中呢？推测由于途径的函数都是void类似，应该对eax没有改动，或改动又复原，可以在系统函数中通过对tf的eax进行修改传递返回值，再到getChar中调用内联汇编指令取出。

此外内联汇编指令的编写也让我有些困惑，如各种mov指令使用的是变量名、寄存器还是地址指针，其中使用的参数，限定的操作数类型都有什么规范。我在网上阅读了一些介绍性的文章，感觉说得都很笼统，例子很少，自己仿照框架代码中的去写，遇见报错或者结果与预期不符的情况再取修改，也能实现想要的功能，但总有一种不太满意的感觉。希望在本学期的其他实验和以后其他科目的学习中，我能够比较熟练地了解和使用内联汇编。

另外，官方版本的getChar似乎会把换行符也留在输入流中，但如果这样的话，根据提供的测试用例，测试getChar以后继续测试getStr，此时会直接读到之前调用getChar遗留的换行符，而不会给用户额外输入任何内容的机会，为了折衷，这个版本的getChar保留了输入流缓冲区的特性，但把换行符排除在外，从而能够通过提供的测试代码。