操作系统Lab2实验报告

1. 实验流程
2. bootloader从实模式进入保护模式，加载内核至内存，并跳转执行。

a)从实模式进入保护模式：

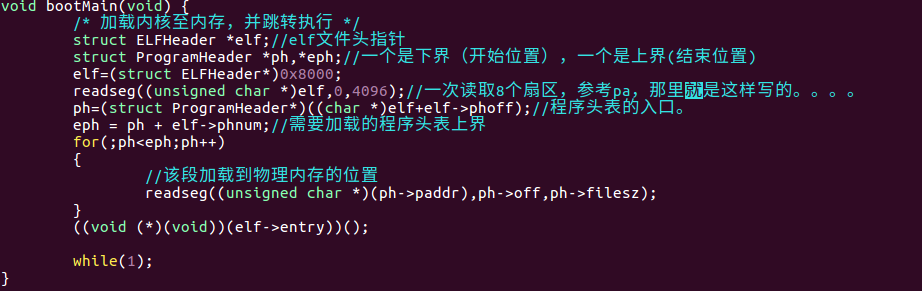
在bootloader/start.S里进行操作：(本操作在lab1中已经实现过)

b)加载内核至内存并跳转执行：

在bootloader/boot.c中进行修改

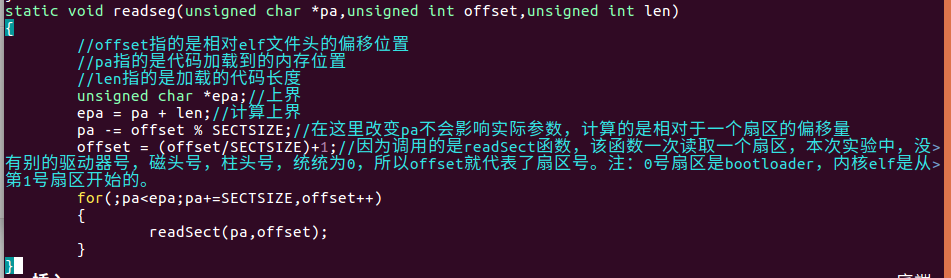
涉及到读取elf文件，自己定义实现一个读取段的函数

i)bootMain



加载elf头表到内存中，根据elf头表中phoff得知程序头表的偏移量，根据phnum得知程序段的个数，接下来就可以按段加载程序段，程序段加载到内存的位置根据ph->paddr获得，程序段的偏移量为ph->off，程序段的大小位ph->filesz,读取程序段完成后elf->entry即为内核的入口地址,跳转到kernel的kEntry函数运行。

Ii)bootMain中调用的readseg函数:



具体解释看注释。

1. 内核初始化IDT,初始化GDT，初始化TSS
2. 内核初始化IDT:

在kernel/kernel/idt.c的initIdt中已经实现。

b)初始化GDT：

①在kernel/kernel/kvm.c的initSeg中已经实现

②因为要在之后打印字符，我用的是写显存打印字符，所以增加了一个段 描述符,在kernel/include/x86/memory.h中#define 了一个 SEG\_VEDIO将其值设置为6

在initSeg函数中增加

gdt[SEG\_VEDIO] = SEG(STA\_W,0xb8000,0xffffffff,DPL\_KERN);

初始化视频段。

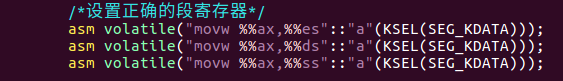
c)初始化TSS+设置正确的段寄存器:

①还是在initSeg函数中，初始化tss,esp0和tss.ss0，添加内敛汇编：

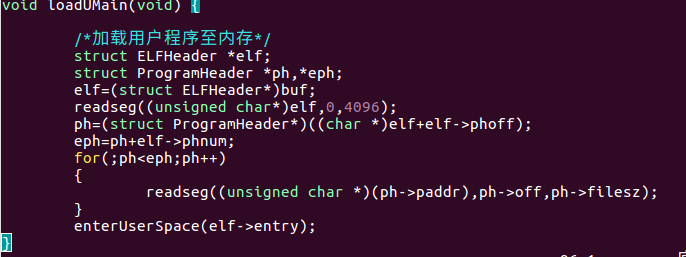
初始化TSS

分别初始化了tss.esp和tss.ss0

②设置正确的段寄存器:

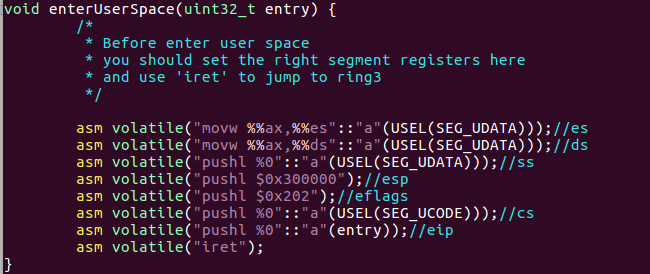


1. 内核加载用户程序至内存，对内核堆栈进行设置，通过iret切换至用户空间执 行用户程序
2. 加载用户程序至内存：



与加载内核至内存同理，不过需要注意用户程序是从201号扇区开始， 所以在readseg函数中计算offset时需要加上201而不是1

b)对内核堆栈进行设置，瞳孔iret切换至用户空间执行



①首先设置正确的段寄存器，因为切换至用户空间，所以段寄存器加载用 户空间段。所以使用USEL(SEG\_UDATA);

②依次将接下来ss,esp,eflags,cs,eip的值入栈，通过iret之后，这些 值会更新到相应寄存器中。

1. 实现0x80系统中断，实现printf函数
2. 实现0x80系统中断

①在kernel/kernel/irqHandle.c文件中，irqHandle函数判断中断号，

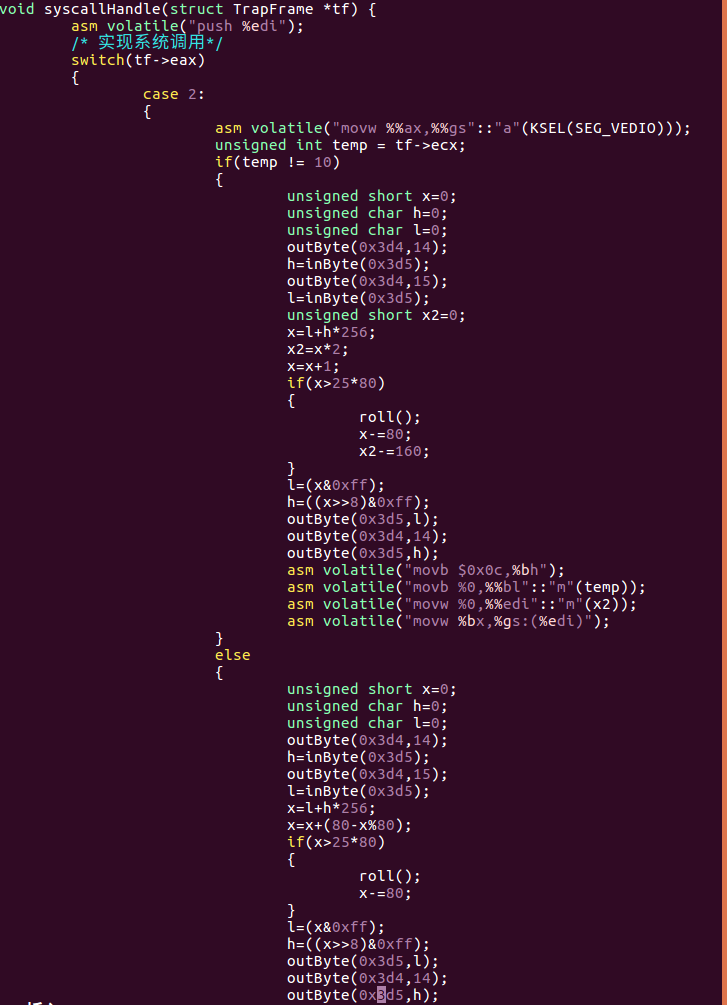
为0x80则跳转至同一文件下的syscallHandle函数中

②syscallHandle中使用0x3d4，0x345端口读取显卡的14,15号寄 存器获取当前光标所在位置，并在打印字符后更新。

详解：

0x3d4号端口写入要操作的寄存器编号，使用0x3d5端口读写寄 存 器，14号寄存器存储光标所在位置高8位，15号寄存器存储光标所 在位置低八位，两个合起来为一个16位整数。

函数使用outByte和inByte函数读写端口



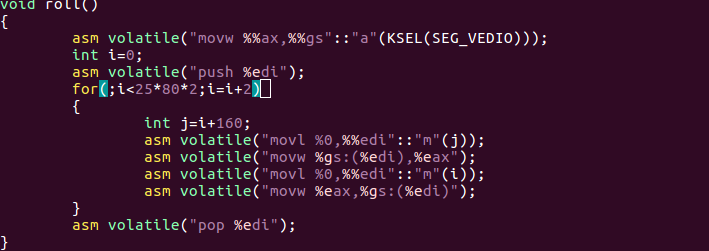


目前对于特殊字符(\\*形式)的字符，只对\n进行了判断，更新行号。

输出为了醒目，采用了黑底红字的形式。

③因为在实际输出时，输出会超出qemu的屏幕，所以自定义实现了roll 函数，该函数的作用是将屏幕中所有行上移一行（第一行丢弃）,并在最 后一行继续输出，缺点是无法再查看被舍弃的行。

Roll函数实现:



b)实现printf函数

①printf多参数实现机制：

Printf中参数个数不确定，但是参数的地址是确定的，且printf第 一个参数总是一个字符串的首地址，一个参数占用一个int字节大小 的位置，所以根据第一个字符串指针的地址依次加上sizeof(int)就 可以访问第二个第三个......参数。

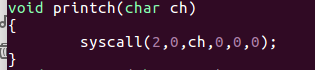
本实验中，printf的参数需要实现%d,%c,%s,%x形式的 输出，所 以分别实现了四个函数，为

printdec,printch,printstr,printhex.

②printf总函数：

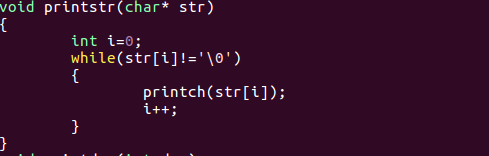


③：printch：



在这里调用了syscall，因为本人愚笨，不知道如何实现一次打印整个字 字符串，所以一个字符调用一次系统中断。

④printstr



对字符串中的每个字符调用一次printch打印。

⑤printdec



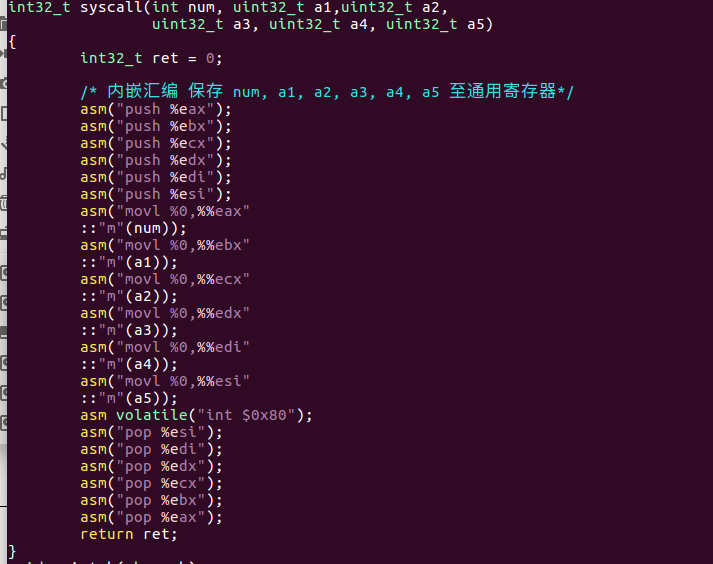
⑥printhex



printdec和printhex均将整数化为了字符串然后调用printstr进行打 印。

c)syscall函数的实现

该函数将参数保存在寄存器中调用0x80中断



1. 程序运行过程:
2. 装载bootloader并跳转执行
3. 装载内核程序并跳转执行，初始化IDT，GDT，TSS。
4. 装载用户程序，设置用户空间并跳转执行。
5. Printf调用四个print子函数，最终调用syscall打印字符。Syscall设置寄存 器，并调用0x80号中断。进入kernel/kernel/doIrq.S中的irqSyscall函数将 error code和中断号压栈，并跳转到asmDoIrq函数，保存当前进程状态至内核 栈中，并调用irqHandle函数，irqHandle判断调用号后，调用syscallHandle

函数实现打印字符。

1. 程序执行结果：

