# OS-Lab1 Documentation

515030910309 张然

**一、设计与实现**

1、八进制输出：

参照十进制，首先从参数中获得数字，然后设置base为8，在输出（putch）一个0作为八进制数的标志，然后调用printnum输出数字。

1. 强制显示’+’

首先设置标记是否强制显示正数’+’得flag，设置为0。然后读取参数，如果遇到了’+’，就将flag设置为1。之后显示数字的时候，如果是正数，则首先显示（putch）一个’+’，其余按照原样输出。

1. 左对齐右侧补充空格

首先按照右对齐的方法正常输出数字，但是不会在数字左侧添加pacd。

所有数字输出后，在最外层递归中，计算数字长度，结合目标width，用’ ’补足剩余的位数。（数字输出部分已经完成，即不断除以base，然后从最高位开始取余输出。

区别是否是最外层递归的方法是，如果padc遇到了’-’，则将’+’传递下去，每次右对齐补足的时候如果不是’+’或’-’，就正常在左侧补足。否则不补足。然后递归结束后，判断如果padc是’-’，就向右补足。

之所以选择’+’，是因为如果出现了’%+’的情况，会首先按照强制显示数字符号处理’+’，而不会出现用’+’补足的情况。

1. Back trace

首先获得当前的%ebp值（readebp），利用%ebp获得储存在栈上的保存的%ebp、返回地址、五个参数（分别将%ebp转为uinit32，然后加1~6获得）。将如上信息输出。之后递归的读取%ebp的值，获得相应的其他值，直到%ebp值为0。

为了获得文件名、函数、行数等信息，在debuginfo\_eip中添加stab\_binsearch，在text段找到指定地址对应的信息。然后将文件名、行数、函数名、相对行数输出。

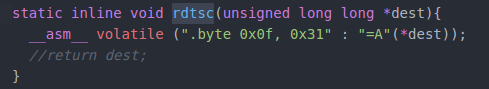
1. %n

为了统计cprint输出了多少个字符，首先获得输出变量的地址。然后判断空指针和越界，如果合法的话，就将putdat储存在该地址处。

1. Time

首先利用“grep -r”找到所有需要修改的的位置，包括函数本身和指令集。

然后readcmd调用time后，利用汇编指令rdtsc，记录执行相应指令前后的时间。



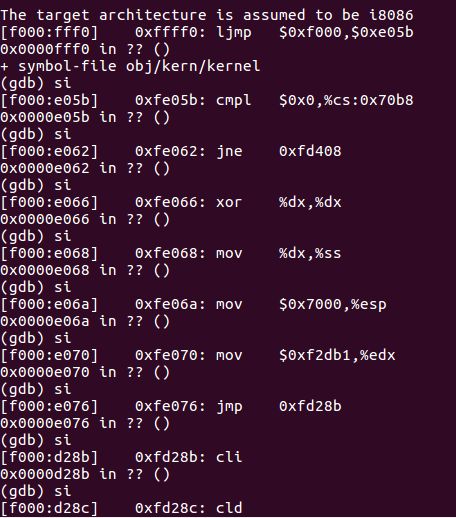
调用方法与readcmd内部相同，遍历所有指令，找到对应的指令，然后执行，但是需要argc减1，argv加一，即去掉time的部分。还需要进行一些错误输入检查。

1. **文档问题与exercise部分解答**

**Part1 PC Bootstrap**

Exe2

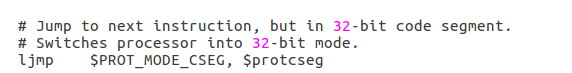
{ 跳转 ss赋为0 栈顶指向0x7000 关中断 清除方向标志



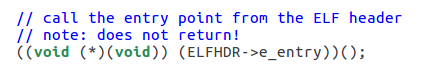
}

**Part2 The Boot Loader**

At what point does the processor start executing 32-bit code? What exactly causes the switch from 16- to 32-bit mode?

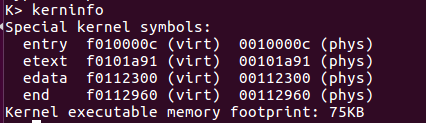
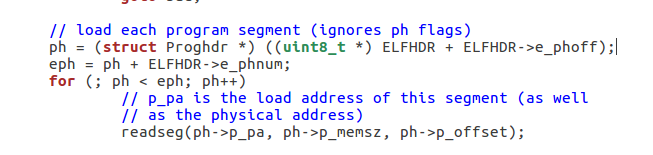
IMG_256

What is the last instruction of the boot loader executed, and what is the first instruction of the kernel it just loaded?

The last inst:

The first inst:

How does the boot loader decide how many sectors it must read in order to fetch the entire kernel from disk? Where does it find this information?



}

**Loading the kernel**

Exe4{

1. a被正常赋值，但是c和a指向同一位置，修改c即修改a

2. 3[c] = \*（3 + c）

3. c+=1 向后移sizeof(int)

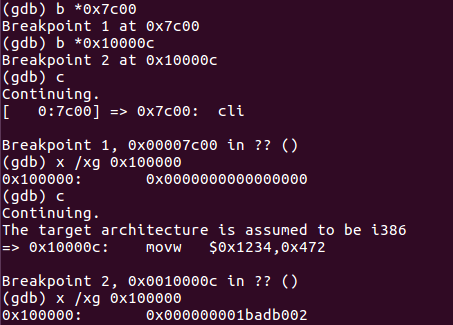
4. c + 1 向后移sizeof（char）然后写入sizeof（int）长度的数字

5. 思路同上

}

Exe5{

0x100000处存储的信息发生了变化



}

**Link vs. Load Address**

Exe6{

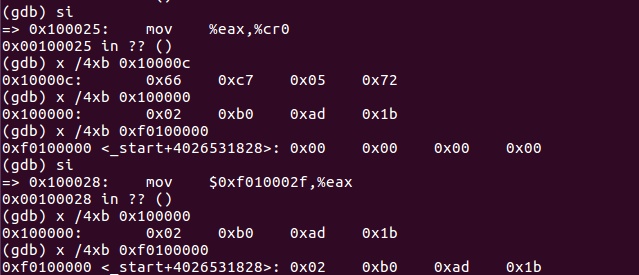
修改链接地址，loader仍从0x7c00开始，gdtr读取错误，ljmp跟着错误

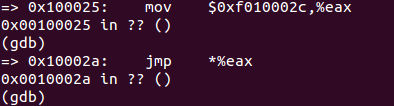
}

**Part 3: The Kernel**

EXE7{

修改CR0的时候切换至protected mode （movl %eax，%cr0）

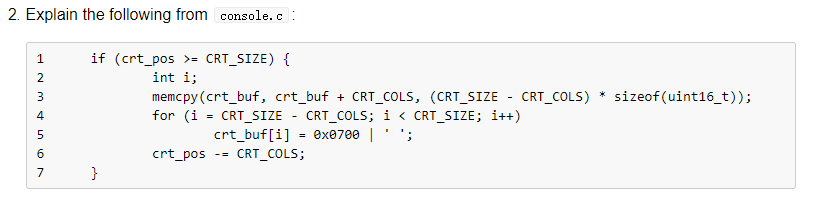


注释后在跳转的时候出现错误

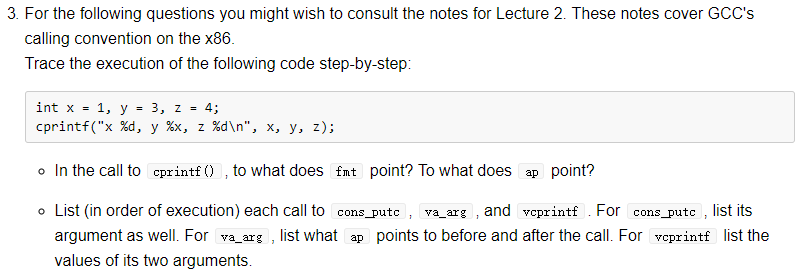
}

**Formatted Printing to the Console**

Console.c定义了如何把字符显示在显示器上，提供了cons\_init, cputchar, getchar, iscons等函数接口，其中cputchar被readline和cprintf使用（即printf.c）。Printf.c调用该函数将字符传到显示器。



当当前已输入的字符位置大于最大size的时候，需要将屏幕向上滚动一行。方法是，将buf中下一行的数据（从crt\_buf + CRT\_COLS开始，大小为(CRT\_SIZE - CRT\_COLS)\*sizeof(uint16\_t)）全部向上一行赋值，然后在最后一行都变成空格，最后修改当前写入的位置。



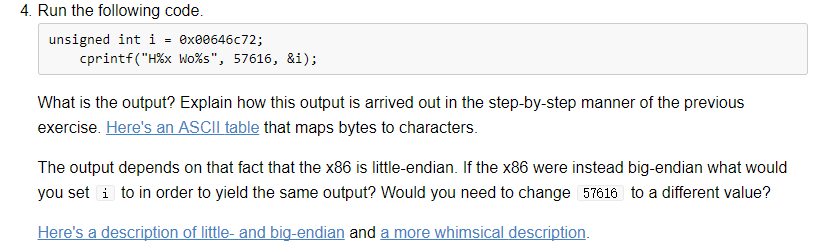
1. fmt指向字符串，ap指向参数列表
2. vcprintf，fmt包括了输入的字符串"x %d, y %x, z %d\n"，ap指向变量x，值为1.

cons\_putc （int）’x’ cons\_putc （int）’ ’

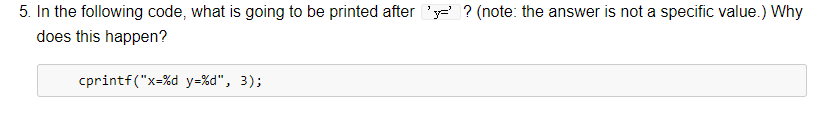
va\_arg 运行前指向x，运行结束后指向y。cons\_putc （int）’1’ cons\_putc （int）’,’ cons\_putc （int）’ ’ cons\_putc （int）’y’ cons\_putc （int）’ ’

va\_arg 运行前指向y，运行结束后指向z。cons\_putc （int）’3’ cons\_putc （int）’,’ cons\_putc （int）’ ’ cons\_putc （int）’z’ cons\_putc （int）’ ’

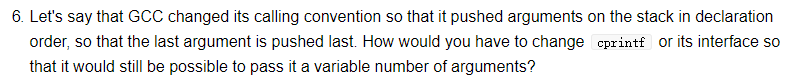
va\_arg 运行前指向z，运行结束后指向z之后。cons\_putc （int）’4’ cons\_putc （int）’\n’



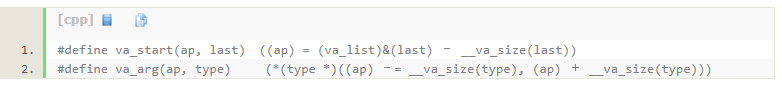
He110 World（57616-> 0xe110, 72->r 6c->l 64->d 00->’\0’)



Ap指向垃圾地址，对应的值不能确定

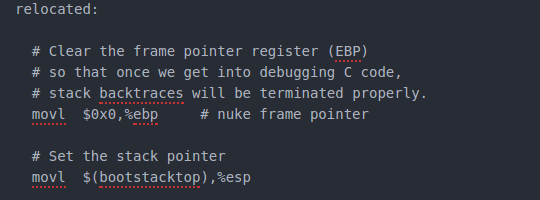


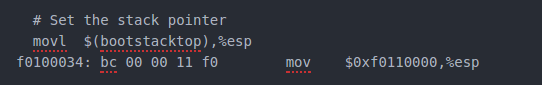
修改两个宏，使va\_start指向最后一个变量开始的位置，va\_arg将ap向前移动



**The Stack**

Exe11{

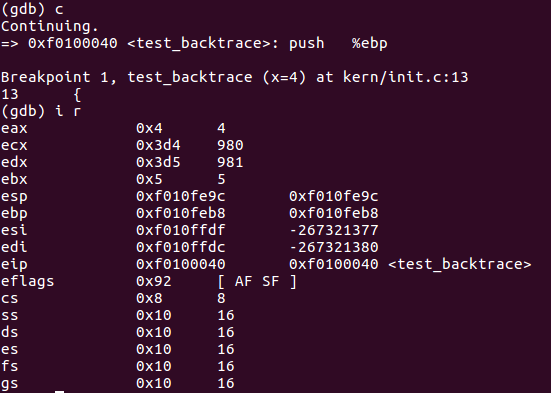
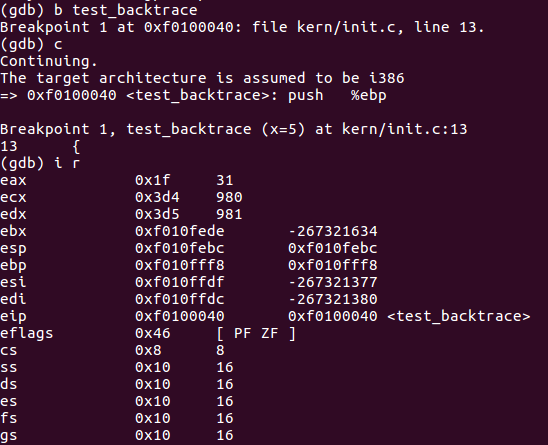
（entry.S)

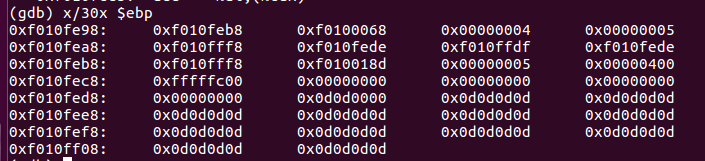
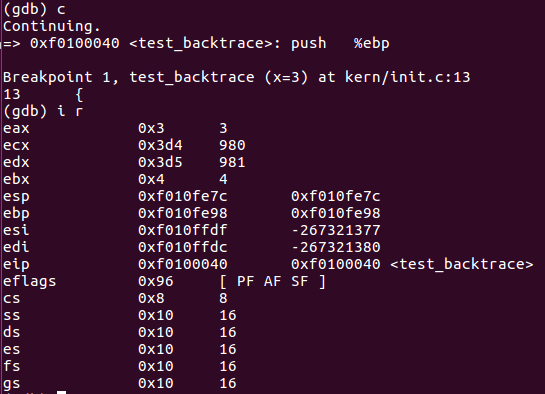
(kernel.asm)

Ebp初始为0，esp指向栈底

}

Exe12{



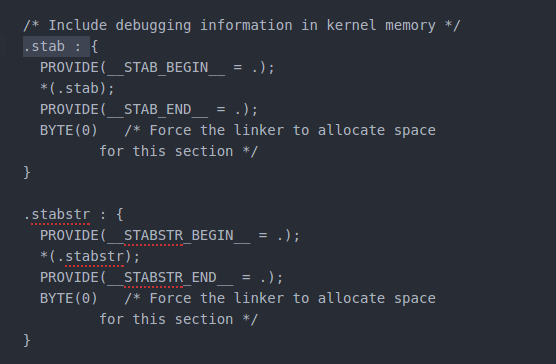


返回地址 保存ebp 保存ebx

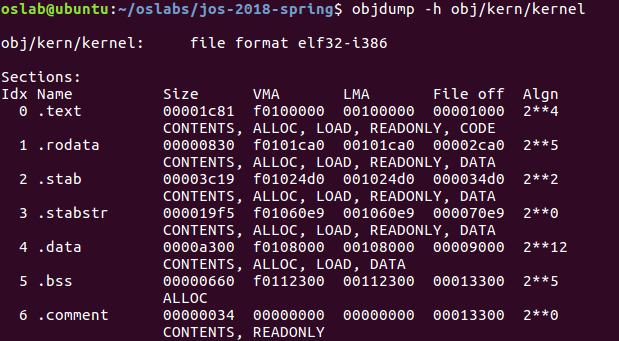
}

Exe14{

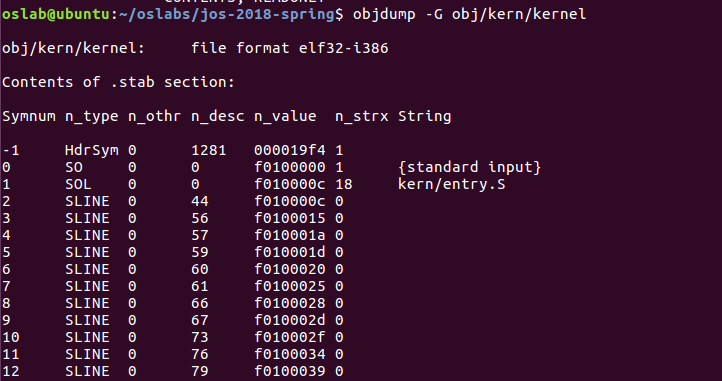
In kernel.ld:



run i386-jos-elf-objdump -h obj/kern/kernel



run i386-jos-elf-objdump -G obj/kern/kernel



}