#### bug01描述：



undefined reference to `\_\_stack\_chk\_fail'

解决方法：在编译时，在CFLAGS后面加上-fno-stack-protector

-fno-stack-protector

#### bug02描述：

gcc: fatal error: stdio.h: No such file or directory

解决方法：可能是不小心吧环境污染了，导致stdio.h都不能包括进来，根据https://stackoverflow.com/questions/19580758/gcc-fatal-error-stdio-h-no-such-file-or-directory解决。

sudo apt-get install libc6-dev

https://www.ppkao.com/shiti/7403389/

#### bug03描述：

在qemu虚拟机中没问题，按一个键打印一个字符，但到了实体机按一次打印两个相同字符

原理：

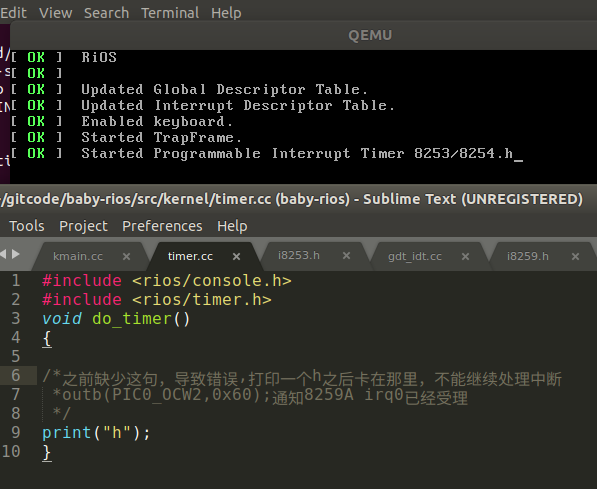
​ 按一个键打印两个字符，因为一次是Make code，一次是Break Code.键盘的扫描码Scan Code，通码Make code，断码Break Code 用户按键盘上的字母，硬件底层会产生对应的Scan Code，而且是按下那一刻产生一个通码Make code，释放的时候产生一个断码Break code。即你从按下一个键盘上的字母，到手松开，实际上对应着一个通码Make Code和一个断码Break Code，两者概念上都属于扫描码Scan Code。

解决方法：判断键盘状态

#### bug04描述：

中断只能执行一次

解决方案：在中断处理函数中通过in out指令告知8259A中断已经被处理，不然将会一直卡死在那里，不会受理下一次中断



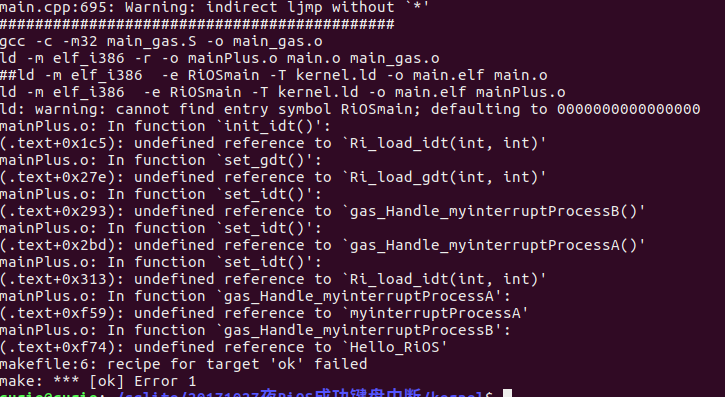
解决方案：方法一是给8259A发EOI信号，即为outb(PIC0\_OCW2,0x60),即outb(0x20,0x60);方法二是采用自动EOI方式，目前RiOS采用后者，一下这两句分别给主片和从片设为自动EOI模式．

outb\_wait(0x20 + 1, 0x3); // Auto EOI in 8086/88 mode  
outb\_wait(0xa0 + 1, 0x3); // Auto EOI in 8086/88 mode

#### bug05描述：

​ 这个问题可以追溯到上学期做第一次操作系统实验的时候，当时找到了原因，但没有找到解决问题的方法，导致我一直只能用汇编和Ｃ，无法使用C++,现在这个问题已经解决．

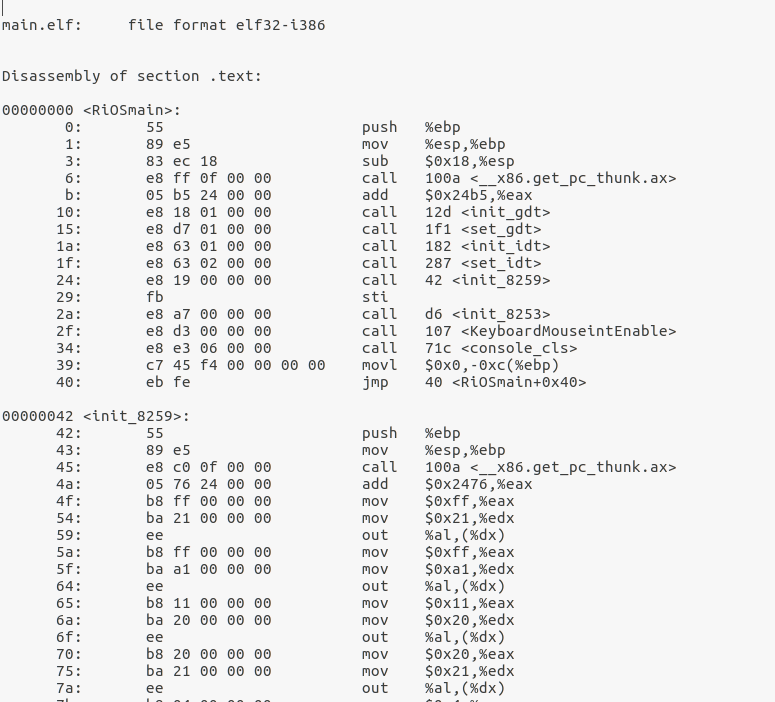
​ 问题的背景是我采用C语言开发本系统的过程中，代码没有问题，编译通过可以正常运行的情况下，我将main.c改为main.cpp.当然Makefile里也作相应变化比如用g++来编译main.cpp什么的。但是，却发现一些通过标号调用汇编代码的地方全不行了，报错undefined.



一直找不到原因，我将编译后的机器码反汇编成.asm文件找原因

Linux命令：

objdump -S -D main.elf >main.asm



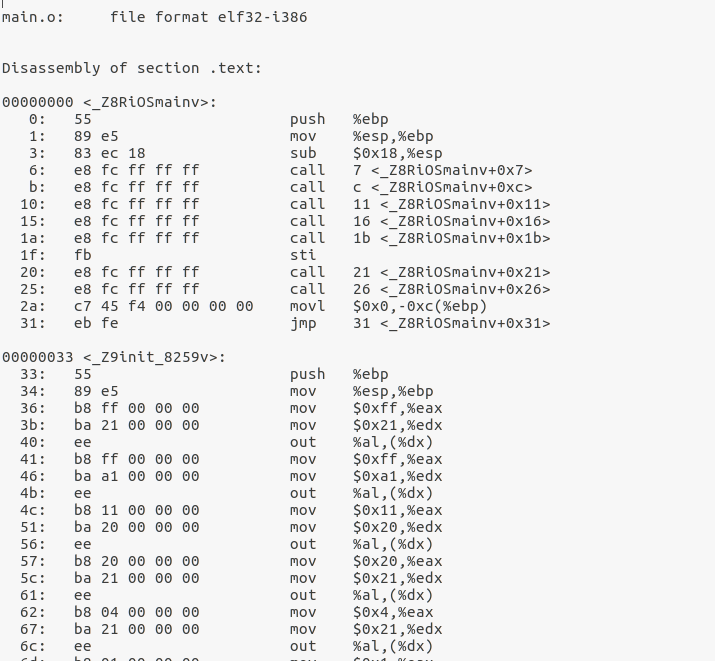
之前编译正常.c文件时的编译结果

可以看到，我的函数名RiOSmain和init\_8259在编译之后是保留原来的字的。

Linux命令：

g++ -nostdinc -I. -fpermissive -fno-stack-protector -c main.cpp -m32 -o main.o

objdump -S main.o >main.asm



编译.cpp文件时的结果

此时我惊讶地发现，在gcc或g++对.cpp文件编译时，原来RiOSmain的函数名变成了\_Z数字RiOSmainv ，即gcc是这样处理的函数到标号的映射是标号：\_Z数字函数名v　这样根据名字去链接必然失败．

解决方法：将C++源程序保存为.cc文件，在头文件的把函数声明包在extern "c"{}之中，这样Ｃ++的函数编译后就能和C编译后的函数兼容

/\*this is hello.h\*/  
#ifdef \_\_cplusplus  
extern "C" {  
#endif /\* \_\_cplusplus \*/  
  
void hello\_cplusplus();  
  
#ifdef \_\_cplusplus  
}  
#endif /\* \_\_cplusplus \*/

extern "C"的主要作用就是为了能够正确实现C++代码调用其他C语言代码。加上extern "C"后，会指示编译器这部分代码按C语言的进行编译，而不是C++的。由于C++支持函数重载，因此编译器编译函数的过程中会将函数的参数类型也加到编译后的代码中，而不仅仅是函数名；而C语言并不支持函数重载，因此编译C语言代码的函数时不会带上函数的参数类型，一般只包括函数名。

C++ undefined 调用不了函数的关键

extern "C"的意思，是让C++编译器（不是C编译器，而且是编译阶段，不是链接阶段）在编译C++代码时，为被extern “C”所修饰的函数在符号表中按C语言方式产生符号名（比如前面的add），而不是按C++那样的增加了参数类型和数目信息的名称（\_Z3addii）。

展开来细说，就是：

如果是C调用C++函数，在C++一侧对函数声明加了extern "C"后符号表内就是add这样的名称，C程序就能正常找到add来调用；如果是C++调用C函数，在C++一侧在声明这个外部函数时，加上extern "C"后，C++产生的obj文件符号表内就也是标记为它需要一个名为add的外部函数，这样配合C库，就一切都好。

#### bug06描述：

不断重启

原因：把ＧＤＴ的地方覆盖了，引发triple fault

解决方案：先设置好ＧＤＴ再去设置ＩＤＴ

#### bug07描述：

/\*这里是修改好后的正确写法\*/  
u8 two\_sectors[1024]={0};  
IDE\_read\_sector((void \*)two\_sectors, DATA\_BLK\_NR\_TO\_SECTOR\_NR(p\_ft->f\_inode->i\_zone[7]));  
/\*之前这里错误地写成了,IDE\_read\_sector((void \*)&two\_sectors,多了个&造成大错\*/  
union free\_space\_grouping\_head g\_head;  
u8 \* p1 = (u8 \*)&g\_head ;IDE\_write\_sector((void \*)p1,sector\_num );

联合体相当于是个实体,而数组int a[10]中的a就是个指针,若在指针前面再加上取地址符&就成了指针的地址,在联合体前面加取地址符就是指向联合体的指针.我之前错误地在指针前面加了取地址符,导致间接寻址都无效.



特地写了一个简单的程序来验证一下加深印象

#include <stdio.h>  
int main()  
{  
 int a[100]={0};  
 for(int i=0;i<100;i++)a[i]=i;  
 int \*p=(int \*)&a;  
 int \*q=(int \*)&a;  
 printf("X &p:%d,&q:%d\n",(int)&p,(int)&q);  
 printf("OK p:%d,q:%d\n",(int)p,(int)q);  
 printf("p[13]%d,q[13]:%d\n",p[13],q[13]);  
  
 int \*pointer =(int \*)&a;/\*指针指向数组的方法\*/  
 printf("pointer[13]:%d\n",pointer[13]);  
 return 0;  
}

#### bug08描述

write函数里面不光要更新硬盘上的数据,还要把内存中活动inode表也同步更新,之前由于这个导致文件虽然已经写入磁盘,但通过文件描述符来读取的时候文件大小始终为0

#### bug09描述

在open 函数中我在最后漏写了这一句

current->filp[fd]->f\_inode = &active\_inode\_table.inode\_table[active\_inode\_table\_nr];

文件系统中,我们有三个表,这三个表联系紧密,他们之间的关系类似于索引,使用时要确保这种索引关系要时刻保持, 不然这个链就断了,之前没有current->filp[fd]->f\_inode 指向活动inode表,导致虽然已经正确写入磁盘, 活动inode表里也有,但是 甲=>乙=>丙 两个指针中有一个没有指向正确的地方,这就导致最终指不到正确的inode, 造成错误.

#### bug10描述

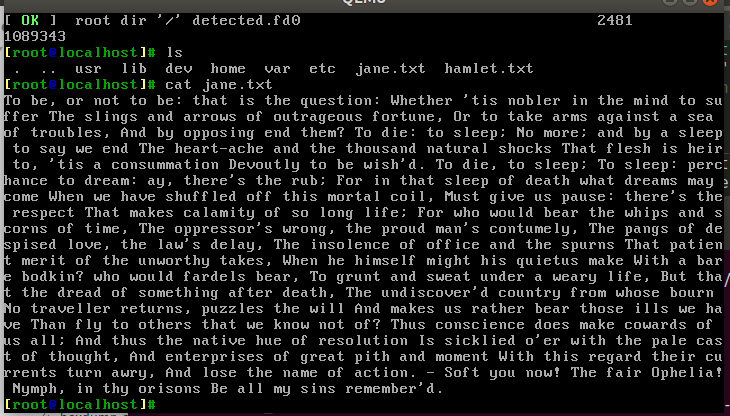
close 函数也存在和bug09中open函数类似的问题,那就是没有保持那三个表的同步性

/\*更正时,在close新添加的语句\*/  
for(j=0;j<MAX\_ACTIVE\_INODE;j++)  
 if(active\_inode\_table.inode\_table[j].i\_ino==filp->f\_inode->i\_ino)  
 break;  
 memset(&active\_inode\_table.inode\_table[j],0x00,sizeof(m\_inode));

这样在close时也把活动inode清理,这就不会出差错了

#### bug11描述

i\_size当时设的值太小了,当时没有考虑到后来文件变得很大的时候



当时开始设的文件大小i\_size类型是u8,也就是8位二进制数,但可以看到我的jane.txt内容相当多,是简爱一本书的内容,大小有1089343,用十六进制表示是0x109f3f,明显8位二进制数早已不能表示它,发生溢出.当我的文件大小没有超过8位时,cat jane.txt和cat hamlet.txt都能正常工作,但是此时文件大小比较大,i\_size这一项错误了,导致显示jane.txt时错误地显示了hamlet.txt的内容.故需要修改i\_size类型,使其为32位,同时,我的inode大小是固定好的,还要调整其他项,使inode大小不发生改变

这只是其中原因之一,之二是我还没有支持二级间地址访问,所以一个文件最大容量受限,容纳几kB还行,但尚不能容纳几MB大小的单个文件.

#### bug12描述

这是个非常简单的错误,但找起来却不轻松我本要一个数除以(80*25),结果错误地写成int times = len/80* 25; 这样实际上变成了(len/80)*25,导致错误.应该写成int times = len/(80*25);