实验题目：lab0，操作系统的编程基础

实验环境

<http://www.shiyanlou.com/courses/221>

安装linux环境，并安装gcc和gdb。

1. 了解汇编

尝试理解下面的命令

$gcc -S -m32 lab0\_ex1.c

接着我们将得到lab0\_ex1.s文件，请写出汇编代码与c代码之间的关系：

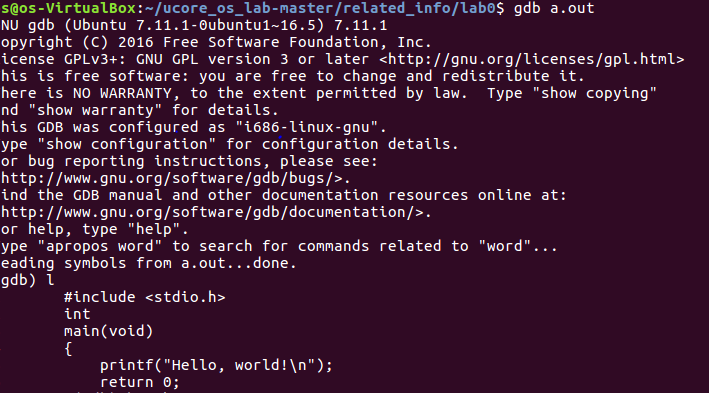
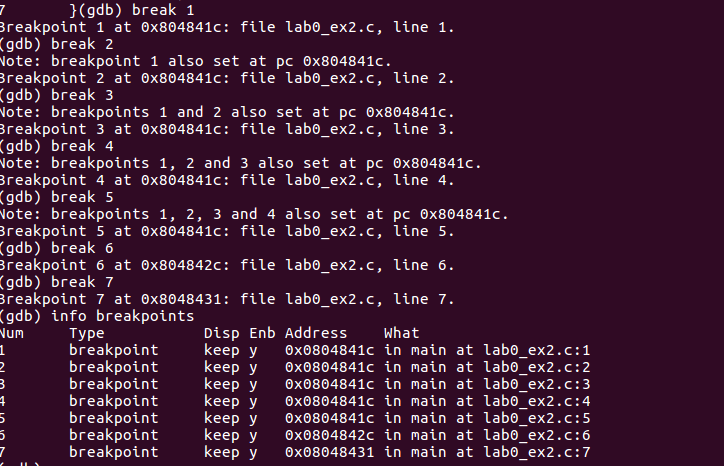
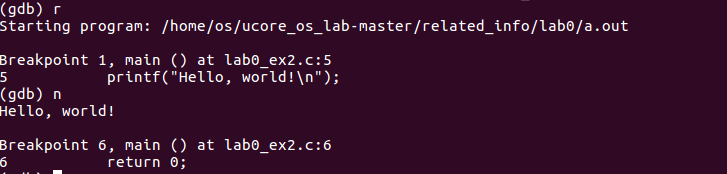
汇编语言是采用助记符号来编写程序的，用辅助符号来代替机器语言的二进制码，就把机器语言变成了汇编语言，可以直接同计算机的底层软件甚至硬件进行交互。C是吧高级语言的基本结构和语句与低级语言的实用性相结合起来了。

1. 用gdb调试

尝试下面的命令，

$gcc -g -m32 lab0\_ex2.c

接着我们会得到a.out文件，请用gdb调试，并写出设置断点、单步执行及查看变量的过程。

 查看变量： 断点： 单步执行：

1. 掌握指针和类型转换相关的Ｃ编程

分析如下代码段，

#include <stdio.h>

#define STS\_IG32 0xE // 32-bit Interrupt Gate

#define STS\_TG32 0xF // 32-bit Trap Gate

typedef unsigned uint32\_t;

#define SETGATE(gate, istrap, sel, off, dpl) { \

(gate).gd\_off\_15\_0 = (uint32\_t)(off) & 0xffff; \

(gate).gd\_ss = (sel); \

(gate).gd\_args = 0; \

(gate).gd\_rsv1 = 0; \

(gate).gd\_type = (istrap) ? STS\_TG32 : STS\_IG32; \

(gate).gd\_s = 0; \

(gate).gd\_dpl = (dpl); \

(gate).gd\_p = 1; \

(gate).gd\_off\_31\_16 = (uint32\_t)(off) >> 16; \

}

/\* Gate descriptors for interrupts and traps \*/

structgatedesc {

unsigned gd\_off\_15\_0 : 16; // low 16 bits of offset in segment

unsignedgd\_ss : 16; // segment selector

unsignedgd\_args : 5; // # args, 0 for interrupt/trap gates

unsigned gd\_rsv1 : 3; // reserved(should be zero I guess)

unsignedgd\_type : 4; // type(STS\_{TG,IG32,TG32})

unsignedgd\_s : 1; // must be 0 (system)

unsignedgd\_dpl : 2; // descriptor(meaning new) privilege level

unsignedgd\_p : 1; // Present

unsigned gd\_off\_31\_16 : 16; // high bits of offset in segment

};

int

main(void)

{

unsigned before;

unsignedintr;

unsigned after;

structgatedescgintr;

intr=8;

before=after=0;

gintr=\*((structgatedesc \*)&intr);

SETGATE(gintr, 0,1,2,3);

intr=\*(unsigned \*)&(gintr);

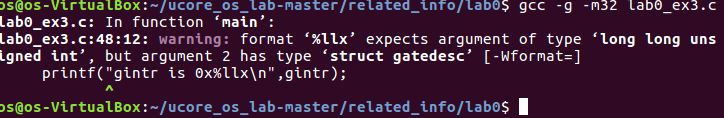
printf("intr is 0x%x\n",intr);

printf("intr is 0x%llx\n", gintr);

return 0;

}

写出gintr和intr的结果，试着编译这段代码，如果遇到错误进行改正，并分析错误原因。





4. 掌握通用链表结构相关的Ｃ编程

查看list.h和lab0\_ex4.c，编写一个程序，利用list.h中的链表结构，将26个英文字母存入链表中，并逆序打印出来。

int main() {  
    struct entry head;  
    list\_entry\_t\* p = &head.node;  
    list\_init(p);  
    head.num = 0;  
    int i;  
    for (i = 'A'; i != '['; i ++) {  
        struct entry \* e = (struct entry \*)malloc(sizeof(struct entry));  
        e->num = i;  
        list\_add(p, &(e->node));  
        p = list\_next(p);  
    }  
    //reverse list all node  
    printf("%c\n",((struct entry \*)p)->num);  
    while ((p = list\_prev(p)) != &head.node)  
        printf("%c\n", ((struct entry \*)p)->num);  
    return 0;  
}

