# 实验题目：lab0，操作系统的编程基础

1. **了解汇编**

尝试理解下面的命令

$gcc -S -m32 lab0\_ex1.c

-S表示仅仅编译，不进行链接或汇编

-m32表示生成32位机器的汇编代码



接着我们将得到lab0\_ex1.s文件:

1. **int** count=1;
2. **int** value=1;
3. **int** buf[10];//三个变量
4. **void** main()
5. {
6. asm(
7. "cld \n\t"//将标志寄存器Flag的方向标志位DF清零。
8. "rep \n\t"//重复前缀指令
9. "stosl"//将EAX中的值保存到ES:EDI指向的地址中
10. :
11. : "c" (count), "a" (value) , "D" (buf[0])
12. :
13. );
14. }

请写出汇编代码与c代码之间的关系:

1. .file   "lab0\_ex1.c"
2. .globl  count
3. .data
4. .align 4
5. .type   count, @object
6. .size   count, 4
7. count:
8. .**long**   1
9. .globl  value
10. .align 4
11. .type   value, @object
12. .size   value, 4
13. value:
14. .**long**   1
15. .comm   buf,40,32
16. .text
17. .globl  main
18. .type   main, @function
19. main:
20. .LFB0:
21. .cfi\_startproc
22. pushl   %ebp
23. .cfi\_def\_cfa\_offset 8
24. .cfi\_offset 5, -8
25. movl    %esp, %ebp
26. .cfi\_def\_cfa\_register 5
27. pushl   %edi
28. pushl   %ebx
29. .cfi\_offset 7, -12
30. .cfi\_offset 3, -16
31. movl    count, %edx
32. movl    value, %eax
33. movl    buf, %ebx
34. movl    %edx, %ecx
35. movl    %ebx, %edi
36. #APP
37. # 6 "lab0\_ex1.c" 1
38. cld
39. rep
40. stosl
41. # 0 "" 2
42. #NO\_APP
43. popl    %ebx
44. .cfi\_restore 3
45. popl    %edi
46. .cfi\_restore 7
47. popl    %ebp
48. .cfi\_restore 5
49. .cfi\_def\_cfa 4, 4
50. ret
51. .cfi\_endproc
52. .LFE0:
53. .size   main, .-main
54. .ident  "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
55. .section    .note.GNU-stack,"",@progbits
56. **用gdb调试**

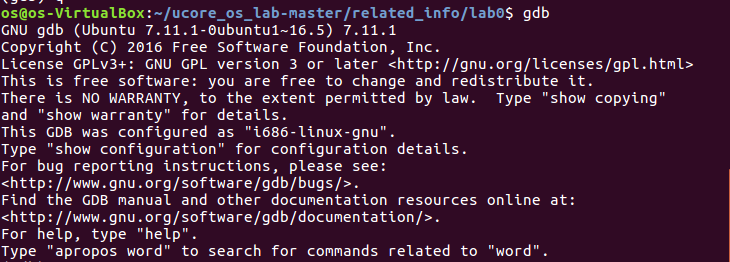
尝试下面的命令，

$gcc -g -m32 lab0\_ex2.c

**进行编译**



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。

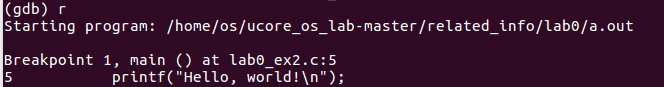


**进入gdb**

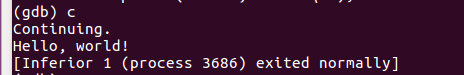


**载入a.out文件**



**设置断点**

**运行**



**输出结果**



**退出**

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 \*/

struct gatedesc {

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

unsigned gd\_ss : 16; // segment selector

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

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

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

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

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

unsigned gd\_p : 1; // Present

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

};

int

main(void)

{

unsigned before;

unsigned intr;

unsigned after;

struct gatedesc gintr;

intr=8;

before=after=0;

gintr=\*((struct gatedesc \*)&intr);

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

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

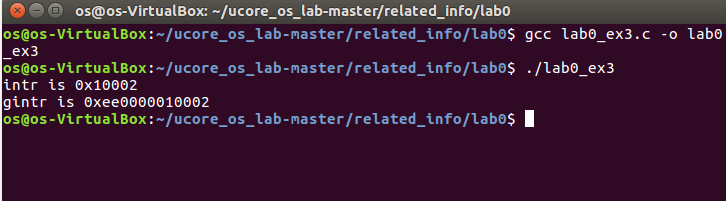
printf("intr is 0x%x\n",intr);

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

return 0;

}

**写出gintr和intr的结果**：

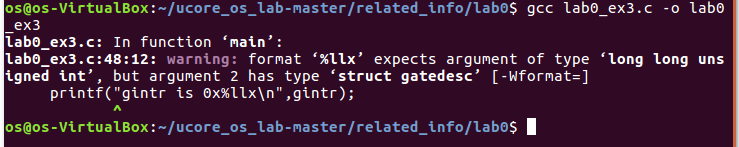


intr is 0x4013d6

gintr is 0xee0000010002

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。

运行结果



**错误分析**

Warning:format ‘%llx’ expects argument of type ‘long long unsigned int’,but argument 2 has type ‘struct gatedesc’[-Wformat]

Ptintf(“gintr is 0x%llx\n”,gintr)

警告：格式'%llx'需要'long long unsigned int'类型的参数，但参数2的类型为'struct gatedesc'[-wformat]

ptintf（“gintr是x%llx\\n”，gintr）

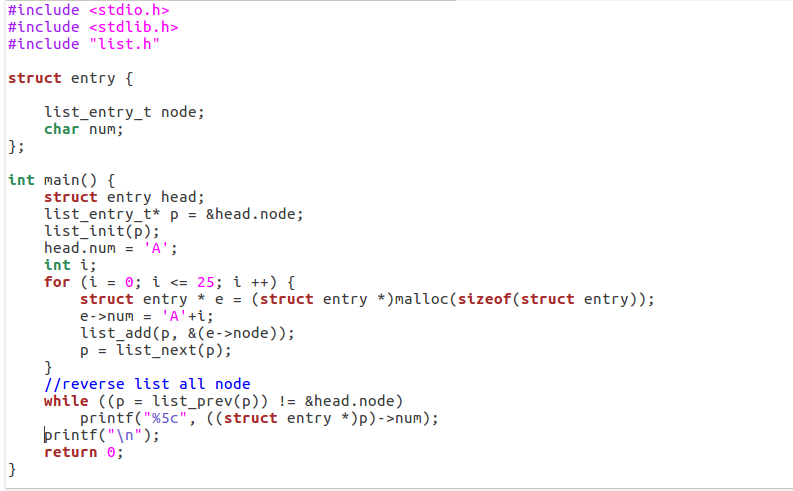
**修改**



**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。

**程序**



**运行结果**

