171491410

宝文庆

硬件3班

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。

int count=1;

int value=1;

int buf[10];//三个变量

void main()

{

asm(

"cld \n\t"//将标志寄存器Flag的方向标志位DF清零。

"rep \n\t"//重复前缀指令

"stosl"//将EAX中的值保存到ES:EDI指向的地址中

:

: "c" (count), "a" (value) , "D" (buf[0])

:

);

}

汇编：

.file "lab0\_ex1.c"

.globl count

.data

.align 4

.type count, @object

.size count, 4

count:

.long 1

.globl value

.align 4

.type value, @object

.size value, 4

value:

.long 1

.comm buf,40,32

.text

.globl main

.type main, @function

main:

.LFB0:

.cfi\_startproc

pushl %ebp

.cfi\_def\_cfa\_offset 8

.cfi\_offset 5, -8

movl %esp, %ebp

.cfi\_def\_cfa\_register 5

pushl %edi

pushl %ebx

.cfi\_offset 7, -12

.cfi\_offset 3, -16

movl count, %edx

movl value, %eax

movl buf, %ebx

movl %edx, %ecx

movl %ebx, %edi

#APP

# 6 "lab0\_ex1.c" 1

cld

rep

stosl

# 0 "" 2

#NO\_APP

popl %ebx

.cfi\_restore 3

popl %edi

.cfi\_restore 7

popl %ebp

.cfi\_restore 5

.cfi\_def\_cfa 4, 4

ret

.cfi\_endproc

.LFE0:

.size main, .-main

.ident "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"

.section .note.GNU-stack,"",@progbits

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。

1 #include <stdio.h>

2 int

3 main(void)

4 {

5 printf("Hello, world!\n");

6 return 0;

7 }(gdb)

gdb) info breakpoints

Num Type Disp Enb Address What

1 breakpoint keep y 0x08048426 in main at lab0\_ex2.c:1

breakpoint already hit 1 time

2 breakpoint keep y 0x08048426 in main at lab0\_ex2.c:2

3 breakpoint keep y 0x08048426 in main at lab0\_ex2.c:3

4 breakpoint keep y 0x08048426 in main at lab0\_ex2.c:4

5 breakpoint keep y 0x08048426 in main at lab0\_ex2.c:5

6 breakpoint keep y 0x08048432 in main at lab0\_ex2.c:6

7 breakpoint keep y 0x08048437 in main at lab0\_ex2.c:7

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，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。

题目的意思应该是输出更改后的gintr这个结构体变量的内容 ，参照上面处理方法我把它强制类型转换成llu型： \* ( long long unsigned \*) & gintr 就OK了。

解释：首先是取gintr这个变量的地址，强制转换为指向llu型变量的指针，再引用这个地址就得到了llu型的变量。

输出结果为：

intr is 0x10002

gintr is 0xee0000010002

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。

struct list\_entry {

struct list\_entry \*prev, \*next;

};

typedef struct list\_entry list\_entry\_t;

struct entry {

list\_entry\_t node;

int num;

};

int main() {

struct entry head;

list\_entry\_t\* p = &head.node;

list\_init(p);

//p->prev = p-> next = p

head.num = 0;

int i;

for (i = 1; i != 27; i ++) {

struct entry \* e = (struct entry \*)malloc(sizeof(struct entry));

e->num = i;

list\_add(p, &(e->node));

/\* list\_add\_after ( p , &(e -> node) ) =>>

\_\_list\_add (&(e -> node ) , p , p -> next) =>>

{

p->next = p->next->prev = &(e -> node );

&(e -> node )->next = p->next;

&(e -> node )->prev = p;

}

\*/

p = list\_next(p);//p = p->next

}

//reverse list all node

while ((p = list\_prev(p)) != &head.node)

printf("%d\n", ((struct entry \*)p)->num);

return 0;

}