操作系统实验一

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

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。

c代码：

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])

:

);

}

汇编代码：

.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

汇编语言是比c更低级的一种语言，能直接操作硬件，c代码在编译的时候先编译成汇编代码它们都是让人与机器交流的语言  
C语言更接近自然语言，代码比较容易阅读  
汇编更接近机器，掌握它可以从更深层次去理解机器  
 它们经过编译程序处理后得到的都是二进制指令数据，对机器来说无区别

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。

输入gdb a.out进入调试

先用 gdb <1> 查看代码，输出helloworld

然后给每行设置断点 break <行数>

输入info breakpoints查看所有断点

按r运行 到第一个断点停止

按q退出

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。

错误原因：由于%llx是指输出long long unsingned int但是gintr是定义的64位结构体。

更改方法：

把printf("intr is 0x%llx\n", gintr);改为printf("intr is 0x%llx\n", \*(long long unsigned \*)&gintr);

由于定义的结构体是64位的,long long unsigned 也是64 位的所以直接强制转换后就可以了

改变后编译结果：

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。

#include <stdio.h>

#include <stdlib.h>

#include "list.h"

struct entry {

list\_entry\_t node;

int num;

};

int main() {

struct entry head;

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

list\_init(p);

head.num = 0;

int i;

for (i = 1; i != 25; i ++) {

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

e->num = getchar();

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

p = list\_next(p);

}

//reverse list all node

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

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

return 0;

}