## 实验4 复杂结构实验

一、实验目的

理解函数调用过程中堆栈的变化情况

理解数组、链表在内存中的组织形式

理解struct 和 union 结构数据在内存中的组织形式

二、实验内容

1、给定如下array\_init.c文件，使用命令gcc -fstack-protector-all -ggdb array\_init.c –o array\_init编译代码，使用命令objdump –d array\_init > array\_init.s反汇编二进制文件，分析反汇编后代码，并完成以下要求

#include <stdio.h>

#define M 2

#define N 10

void init(int a[N]){

int i;

char temp[N];

printf("input student id : \n");

fgets(temp,N,stdin);

for(i=0;i<N;i++){

a[i]=temp[i]-'0';

}

}

void g(){

int a[N];

init(a);

}

void print(int b[M]){

int i;

for(i=0;i<M;i++){

printf("%d ",b[i]);

}

printf("\n");

}

void f(){

int b[M];

print(b);

}

int main(){

g();

f();

return 0;

}

实验要求：

1. 查看函数g和f的反汇编代码，分别给出函数g和f中数组a，b在栈上的分布，在下图中给出a[0]-a[9]以及b[0]、b[1]位置。

函数g 函数f

|  |
| --- |
| old ebp |
|  |
|  |
| %gs(14) |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

|  |
| --- |
| old ebp |
|  |
|  |
| %gs(14) |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

(2) 运行程序，程序的输入为9位学号，观察输出。请详细解释为什么b[0]和b[1]是这两个值。说明使用未初始化的程序局部变量的危害。

2给定如下三维数组A的定义以及store\_ele函数，其中R,S,T是用#define定义的常量。

又给定3\_d\_array这个可执行文件，在3\_d\_array的main函数中仅调用了一次store\_ele函数，使用命令objdump –d 3\_d\_array > 3\_d\_array.s反汇编二进制文件,观察store\_ele函数。

int A[R][S][T];

int store\_ele(int i,int j,int k,int dest){

A[i][j][k] = dest;

return sizeof(A);

}

1 push %ebp

2 mov %esp,%ebp

3 mov 0xc(%ebp),%eax

4 mov 0x8(%ebp),%ecx

5 mov %eax,%edx

6 lea (%edx,%edx,1),%eax

7 mov %eax,%edx

8 lea 0x0(,%edx,8),%eax

9 sub %edx,%eax

10 imul $0xb6,%ecx,%edx

11 add %eax,%edx

12 mov 0x10(%ebp),%eax

13 add %eax,%edx

14 mov 0x14(%ebp),%eax

15 mov %eax,0x804a060(,%edx,4)

16 mov $0x5c6c0,%eax

17 pop %ebp

18 ret

实验要求

（1）将数组地址计算扩展到三维，给出A[i][j][k]地址的表达式。(A的定义为int A[R][S][T]，sizeof(int)=4,起始地址设为addr(A))

（2）使用命令gdb ./3\_d\_array启动gdb调试。在store\_ele函数入口设置断点，以自己的9位学号为输入，运行程序。在store\_ele函数中，单步执行，并打印出每步汇编指令执行后寄存器eax、ecx、edx的值。上面给出了store\_ele函数的汇编指令及其指令编号，根据自己的实验结果填写每条指令运行后的结果。

（3）根据以上内容确定R、S、T的取值

|  |  |  |  |
| --- | --- | --- | --- |
|  | %eax | %ecx | %edx |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |
| 6 |  |  |  |
| 7 |  |  |  |
| 8 |  |  |  |
| 9 |  |  |  |
| 10 |  |  |  |
| 11 |  |  |  |
| 12 |  |  |  |
| 13 |  |  |  |
| 14 |  |  |  |
| 15 |  |  |  |
| 16 |  |  |  |

3、函数recursion是一个递归调用函数。其原函数存在缺失，试根据其汇编代码确定原函数，保存为recursion.c

int recursion (int x){

if(x<2 )

return 1 ;

else

return recursion(x-1) + recursion(x-2);

}

00000000 <recursion>:

0: 55 push %ebp

1: 89 e5 mov %esp,%ebp

3: 53 push %ebx

4: 83 ec 04 sub $0x4,%esp

7: 83 7d 08 02 cmpl $0x2,0x8(%ebp)

b: 7f 07 jg 14 <recursion+0x14>

d: b8 01 00 00 00 mov $0x1,%eax

12: eb 28 jmp 3c <recursion+0x3c>

14: 8b 45 08 mov 0x8(%ebp),%eax

17: 83 e8 01 sub $0x1,%eax

1a: 83 ec 0c sub $0xc,%esp

1d: 50 push %eax

1e: e8 fc ff ff ff call 1f <recursion+0x1f>

23: 83 c4 10 add $0x10,%esp

26: 89 c3 mov %eax,%ebx

28: 8b 45 08 mov 0x8(%ebp),%eax

2b: 83 e8 02 sub $0x2,%eax

2e: 83 ec 0c sub $0xc,%esp

31: 50 push %eax

32: e8 fc ff ff ff call 33 <recursion+0x33>

37: 83 c4 10 add $0x10,%esp

3a: 01 d8 add %ebx,%eax

3c: 8b 5d fc mov -0x4(%ebp),%ebx

3f: c9 leave

40: c3 ret

4、给定以下结构定义

struct ele{

union {

struct{

int\* p;

int x;

}e1;

int y[3];

};

struct ele \*next;

};

实验要求

（1）确定下列字节的偏移量。

e1.p

e1.x

y

y[0]

y[1]

y[2]

next

（2）下面的过程（省略一些表达式）是对链表进行操作，链表是以上述结构作为元素的。现有proc函数主体的汇编码，查看汇编代码，并根据汇编代码补全proc函数中缺失的表达式，并保存为proc.c。（不需要进行强制类型转换）

void proc(struct ele \*up){

up->\_\_next\_\_\_\_\_= \*(up->\_\_y[0]\_\_\_\_)+up->\_\_\_y[2]\_\_\_\_\_;

}

00000000 <proc>:

0: 55 push %ebp

1: 89 e5 mov %esp,%ebp

3: 8b 45 08 mov 0x8(%ebp),%eax

6: 8b 40 0c mov 0xc(%eax),%eax

9: 8b 55 08 mov 0x8(%ebp),%edx

c: 8b 12 mov (%edx),%edx

e: 8b 0a mov (%edx),%ecx

10: 8b 55 08 mov 0x8(%ebp),%edx

13: 8b 52 08 mov 0x8(%edx),%edx

16: 01 ca add %ecx,%edx

18: 89 10 mov %edx,(%eax)

1a: 90 nop

1b: 5d pop %ebp

1c: c3 ret

（3）有以下main函数，该main函数中声明了一数组和一链表并打印了每个元素的地址，查看地址，并解释产生原因，体会数组与链表分别使用静态内存和动态内存的差异。

int main(){

struct ele a[5];

struct ele \* head, \* p;

head=NULL;

for(int i=0;i<5;i++){

p=(struct ele \*)malloc(sizeof(struct ele));

p->next=NULL;

if(head==NULL){

head=p;

}else{

p->next=head;

head=p;

}

for(int j=0;j<i;j++){

malloc(sizeof(int));

}

}

printf("array address:\n");

printf("%x\t%x\t%x\n",(unsigned int)&a[0],(unsigned int)&a[1],(unsigned int)&a[2]);

printf("\nlist address:\n");

p=head;

while(p!=NULL){

printf("%x\t",(unsigned int)p);

p=p->next;

}

printf("\n");

}

**实验报告要求：**

见之前实验