

**课 程 实 验 报 告**

**课程名称： 计算机系统基础实验**

**专业班级：\_\_\_ 计卓202201**

**学 号： U202215322**

**姓 名： 濮澍**

**指导教师： 许 向 阳**

**实验时段： 2024年3月15日~5月12日**

**实验地点： 南一楼 803**

**原创性声明**

  本人郑重声明：本报告的内容由本人独立完成，有关观点、方法、数据和文献等的引用已经在文中指出。除文中已经注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品或成果，不存在剽窃、抄袭行为。

特此声明！

学生签名： *纸质版再签名*

报告日期：2024.5.28

实验报告成绩评定：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 一（30分） | 二（20） | 三（20） | 四（20） | 五（10分） | 合计（100） |
|  |  |  |  |  |  |

指导教师签字：

                    日期：



**课 程 实 验 报 告**

**课程名称： 计算机系统基础**

**实验名称： 数据的表示**

**院 系 ：计算机科学与技术**

**专业班级 ： 计卓202201**

**学 号 ： U202215322**

**姓 名 ： 濮澍**

**指导教师 ： 许向阳**

**2024年 3月 24 日**

**一、实验目的与要求**

⑴ 熟练掌握程序开发平台(VS2019/VS2022/VS2023) 的基本用法，包括程序的编译、链接和调试；

⑵ 熟悉地址的计算方法、地址的内存转换；

⑶ 熟悉数据的表示形式。

**二、实验内容**

**任务1 数据存放的压缩与解压编程**

定义了 结构 student ，以及结构数组变量old\_s[N], new\_s[N]; (N=5)

struct student {

char name[8];

short age;

float score;

char remark[200]; // 备注信息

};

编写程序，输入N个学生的信息到结构数组old\_s中。将 old\_s[N] 中的所有信息依次紧凑(压缩)存放到一个字符数组message中，然后从 message 解压缩到结构数组 new\_s[N]中。打印压缩前(old\_s)、解压后(new\_s)的结果，以及压缩前、压缩后存放数据的长度。

要求：

1. 输入的第0个人姓名(name)为自己的名字，分数为学号的最后两位；
2. 编写指定接口的函数完成数据压缩

压缩函数有两个： int pack\_student\_bytebybyte(student\* s, int sno, char \*buf);

int pack\_student\_whole(student\* s, int sno, char \*buf);

s为待压缩数组的起始地址； sno 为压缩人数； buf 为压缩存储区的首地址；两个函数的返回均是调用函数压缩后的字节数。pack\_student\_bytebybyte要求一个字节一个字节的向buf中写数据；pack\_student\_whole要求对short、float字段都只能用一条语句整体写入，用strcpy实现串的写入。

1. 使用指定方式调用压缩函数

old\_s数组的前N1（N1=2）个记录压缩调用pack\_student\_bytebybyte 完成；后N2（N2==3）个记录压缩调用pack\_student\_whole，两种压缩函数都只调用1次。

（4） 使用指定的函数完成数据的解压

解压函数的格式：int restore\_student(char \*buf, int len, student\* s);

buf 为压缩区域存储区的首地址；len为buf中存放数据的长度；s为存放解压数据的结构数组的起始地址； 返回解压的人数。解压时不允许使用函数接口之外的信息（即不允许定义其他全局变量）

（5）仿照调试时看到的内存数据，以十六进制的形式，输出message的前20个字节的内容，并与调试时在内存窗口观察到的message的前20个字节比较是否一致。

（6）对于第0个学生的score，根据浮点数的编码规则指出其个部分的编码，并与观察到的内存表示比较，验证是否一致。  
 (7) 指出结构数组中个元素的存放规律，指出字符串数组、short类型的数、float型的数的存放规律。

**任务2 编写位运算程序**

按照要求完成给定的功能，并**自动判断程序**的运行结果是否正确。（从逻辑电路与门、或门、非门等等角度，实现CPU的常见功能。所谓自动判断，即用简单的方式实现指定功能，并判断两个函数的输出是否相同。）

1. int absVal(int x); 返回 x 的绝对值

仅使用 !、 ~、 &、 ^、 |、 +、 <<、 >>， 运算次数不超过 10次

判断函数： int absVal\_standard(int x) { return (x < 0) ? -x : x;}

1. int negate(int x); 不使用负号，实现 -x

判断函数： int netgate\_standard(int x) { return -x;}

1. int bitAnd(int x, int y); 仅使用 ~ 和 |，实现 &

判断函数： int bitAnd\_standard(int x, int y) { return x & y;}

1. int bitOr(int x, int y); 仅使用 ~ 和 &，实现 |
2. int bitXor(int x, int y); 仅使用 ~ 和 &，实现 ^
3. int isTmax(int x); 判断x是否为最大的正整数（7FFFFFFF），

只能使用 !、 ~、 &、 ^、 |、 +

1. int bitCount(int x); 统计x的二进制表示中 1 的个数

只能使用，! ~ & ^ | + << >> ，运算次数不超过 40次

1. int bitMask(int highbit, int lowbit); 产生从lowbit 到 highbit 全为1，其他位为0的数。例如bitMask(5,3) = 0x38 ；要求只使用 ! ~ & ^ | + << >> ；运算次数不超过 16次。
2. int addOK(int x, int y); 当x+y 会产生溢出时返回1，否则返回 0

仅使用 !、 ~、 &、 ^、 |、 +、 <<、 >>， 运算次数不超过 20次

1. int byteSwap(int x, int n, int m); 将x的第n个字节与第m个字节交换，返回交换后的结果。 n、m的取值在 0~3之间。  
   例：byteSwap(0x12345678, 1, 3) = 0x56341278

byteSwap(0xDEADBEEF, 0, 2) = 0xDEEFBEAD

仅使用 !、 ~、 &、 ^、 |、 +、 <<、 >>， 运算次数不超过 25次

**三、实验记录及问题回答**

**（1）任务 1 的算法思想、运行结果等记录**

**·函数思想**

bytebybyte在于一个字节一个字节复制，所以只需定义一个char\*指针，每次放入数据都指向下一个数据就行，对float和short需要分别根据二者字节数定义循环

函数中定义一个偏移量，记录当前操作时偏移量，最终返回时返回了用户输入的压缩后的字节数

whole 在于使用str函数和指针运算进行压缩，strcpy自不用说，而这里的float和short则需要将char\*指针强制类型转换为 float\*和short\* 再解引用赋值。Whole中的偏移量now\_add就不在是一个字节一个字节加，而是根据每次加入的数据量大小进行增加。而且在此函数中，buf+now\_add相当于指针用途

restore\_student相当于whole的相反操作

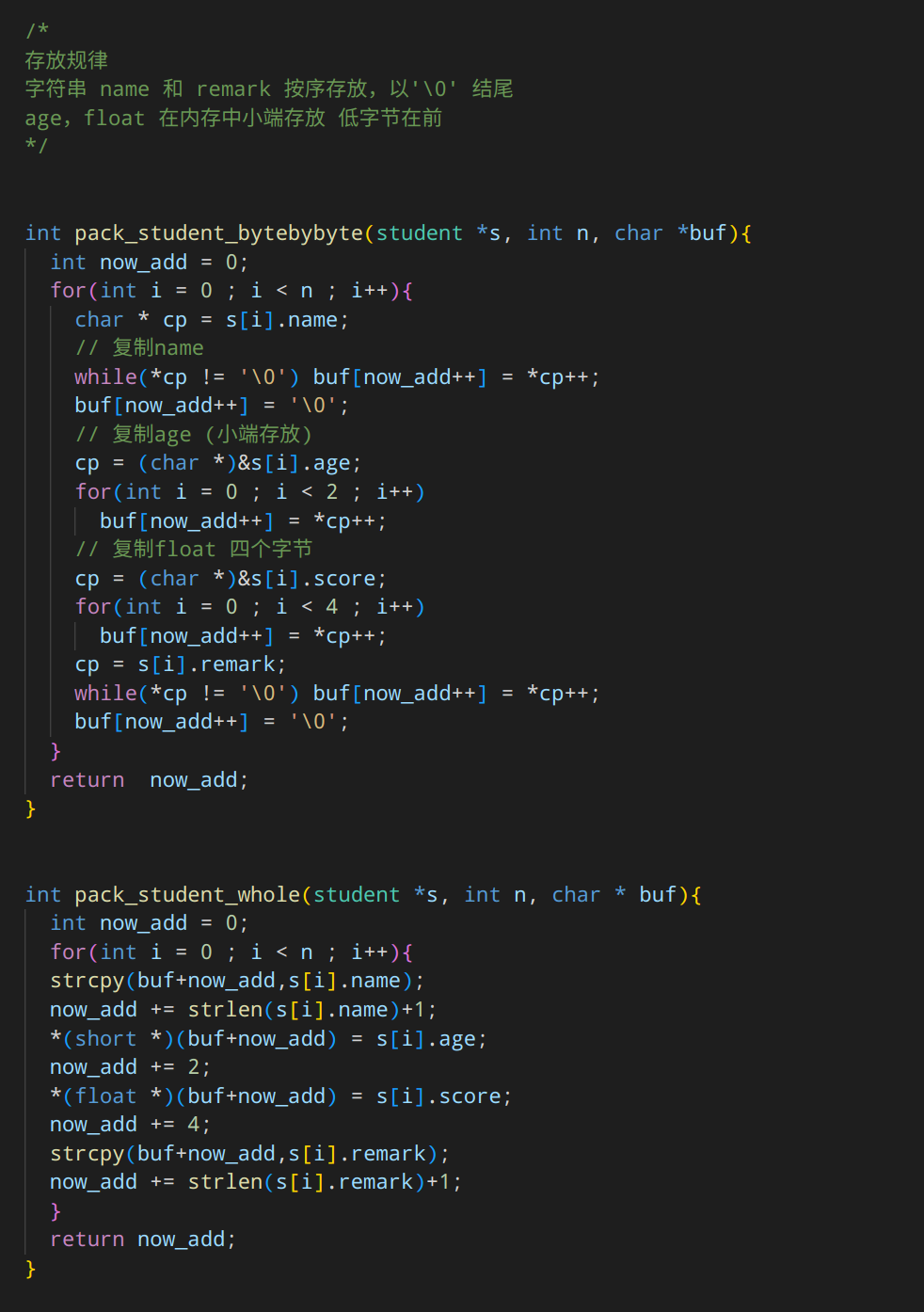


图1 压缩函数

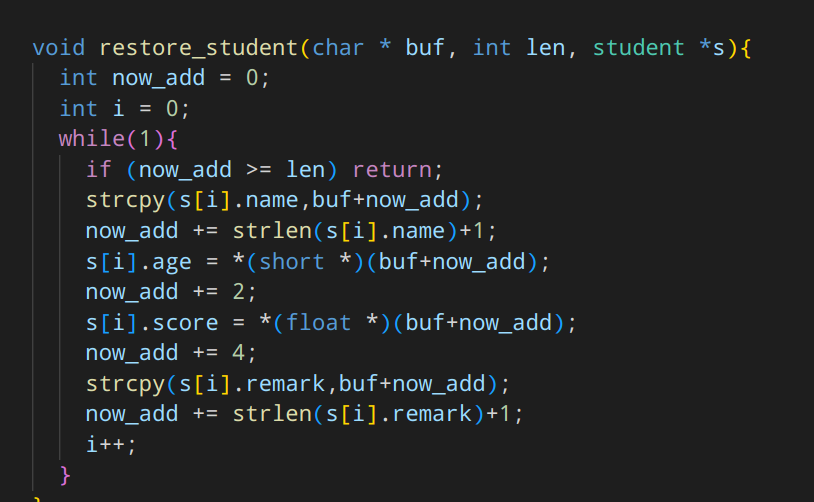


图2 解压缩函数

**·运行结果**

根据任务书，对内存等进行观察

Code总览

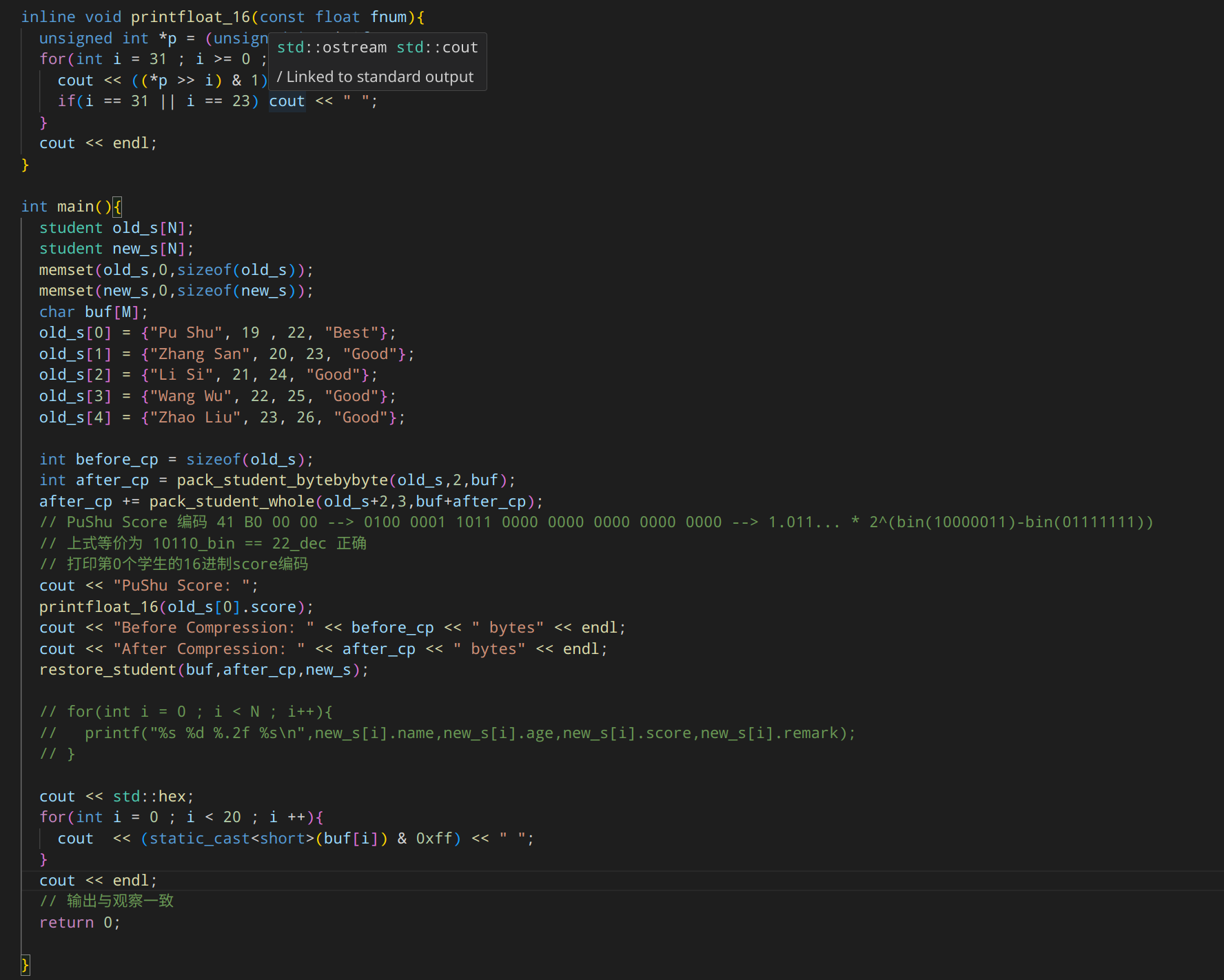


图3 main函数

图3中注释表示出了该学生分数22的float形式，在程序运行时，有下图4运行结果和分析一致

main函数中前两个学生通过bytebybyte压缩，后三个用whole压缩

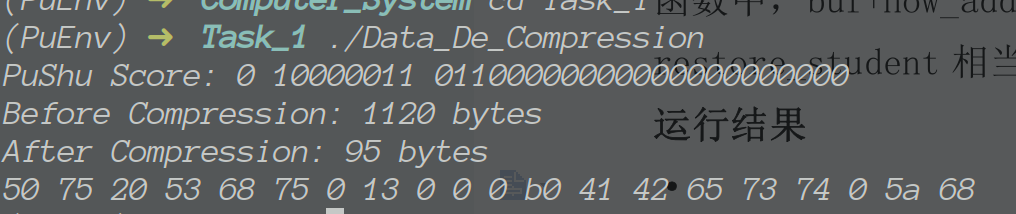


图4 运行结果

根据任务书，我们观察buff原本的前20个字节和输出的20个字节区别 发现二者一致

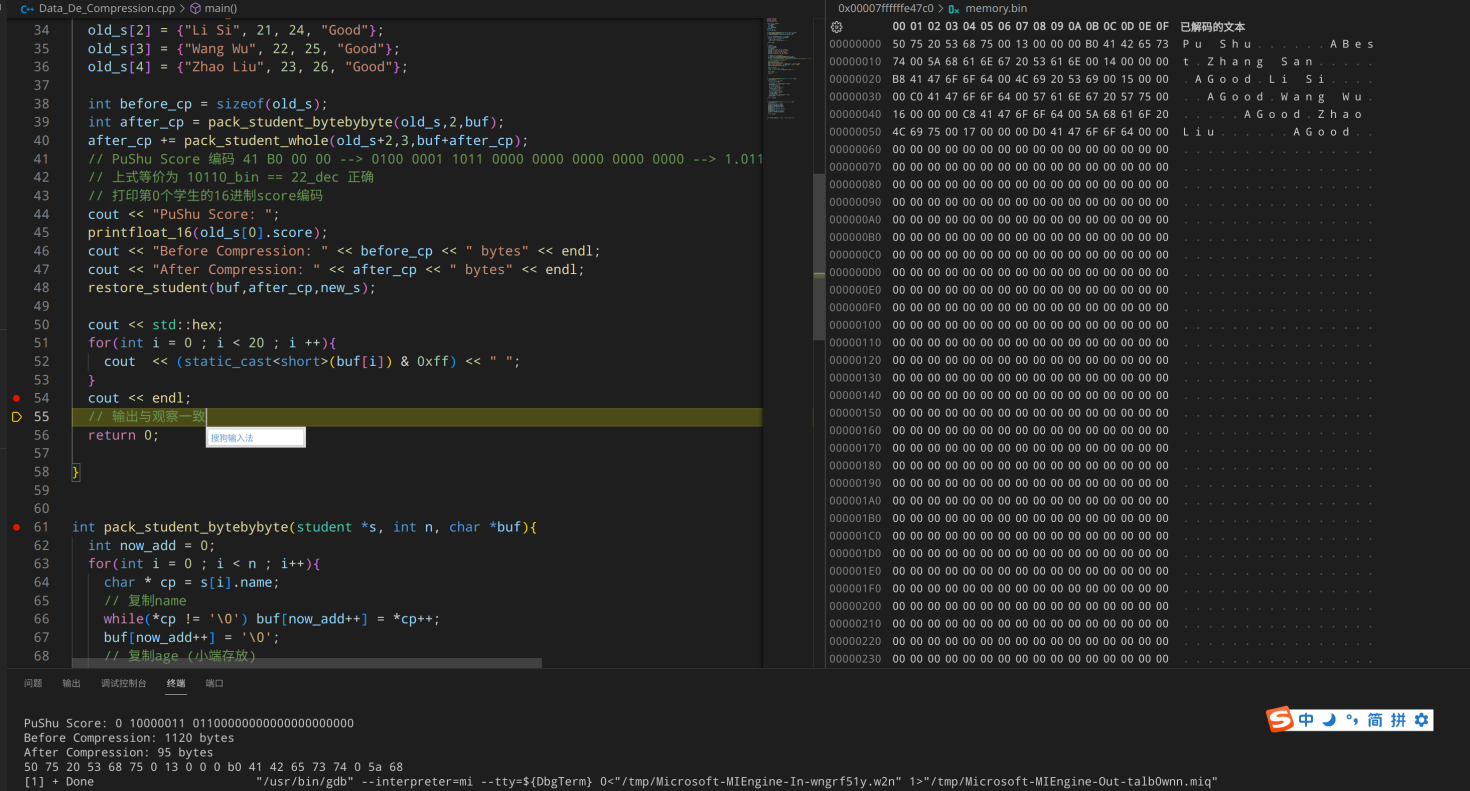


图6 压缩后内存窗口和输出对比

**·数据存放规律**

**存放规律**

**字符串 name 和 remark 按序存放，以'\0' 结尾**

**age，float 在内存中小端存放 低字节在前**

**（2）任务 2 的算法思想、运行结果等记录**

**·算法思想**

absVal与符号位密切相关，所以返回值一定用到’x>>31’ 是1 时-1取反 否则不变

neg 就是简单的取反加一

bitAnd/Or/Xor画出真值表就可求解

isTmax要判断输入是否是0x7FFFFFFF 该数+1即为0x80000000，与Tmax相反，异或为0

又要特判x==-1

bitCount 整体思路还是循环，只是这里通过每次取更多位上的数据后，相加获得结果，比较难 意思是类似合并相邻石子，每次都合并一组，最终合并成最大组

bitMask 需要获取lowbit->highbit 位运算就是取中间部分

addOK 符号位不同 一定不溢出，输出1 ； 符号位相同，相加后符号位相反则溢出

byteswap 通过掩码和异或操作获取n字节和m字节的数据，通过左右移进行交换

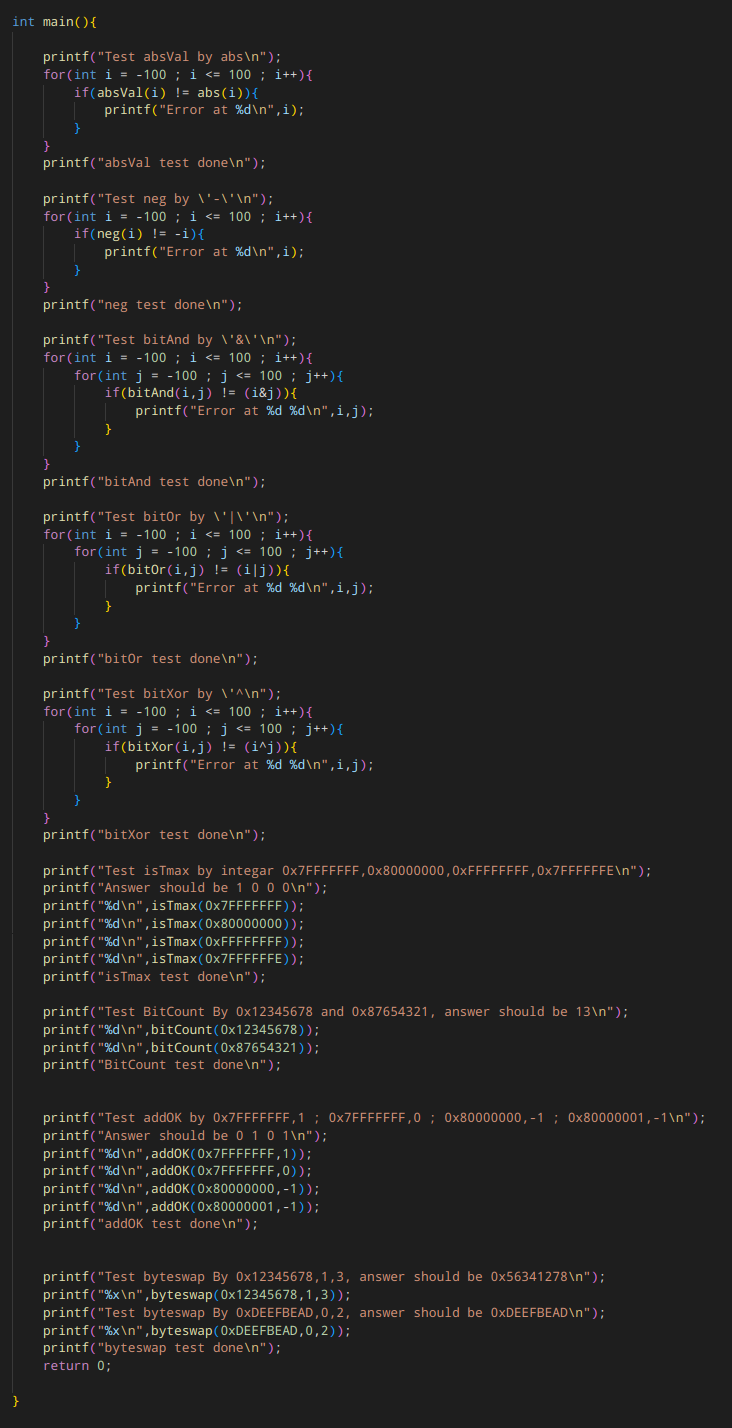


图7 测试函数

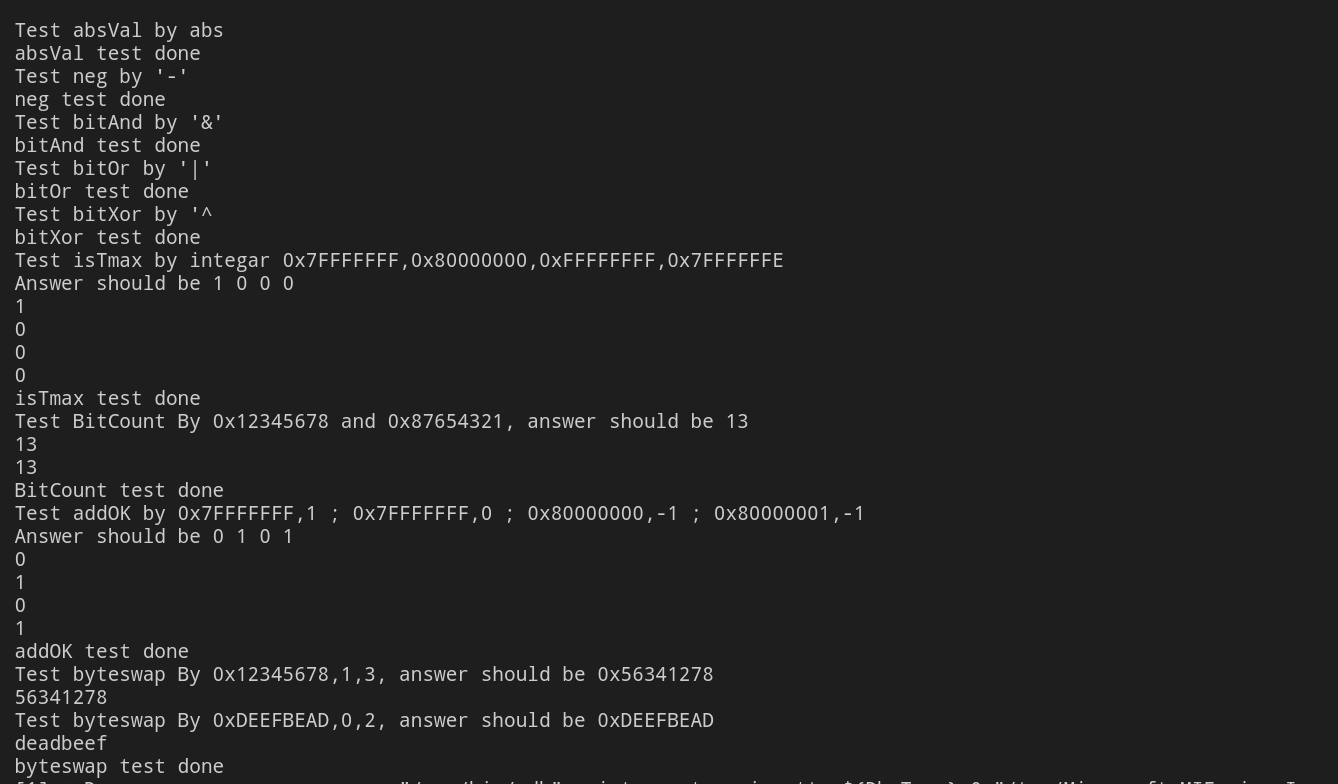


图8 测试结果

1. **体会**

通过任务1熟悉了不同数据类型在操作系统中存放形式，并了解了一系列压缩思想与方法

通过任务2 熟悉并练习了位运算，让系统运行更快捷，并且对思维很有帮助。但有的时候确实很难思考出思路

**五、源码**

实验任务 1、2 的源程序（单倍行距，5号宋体字）

任务1源程序

#include <iostream>

#include <cstring>

#include <iterator>

using namespace std;

typedef struct Student{

char name[15];

short age;

float score;

char remark[200];

} student;

const int N = 5;

const int M = 1e5;

int pack\_student\_bytebybyte(student \*s,int sno, char \*buf);

int pack\_student\_whole(student \*s, int sno, char \*buf);

void restore\_student(char \*buf,int len, student \*s);

inline void printfloat\_16(const float fnum){

unsigned int \*p = (unsigned int \*)&fnum;

for(int i = 31 ; i >= 0 ; i--){

cout << ((\*p >> i) & 1);

if(i == 31 || i == 23) cout << " ";

}

cout << endl;

}

int main(){

student old\_s[N];

student new\_s[N];

memset(old\_s,0,sizeof(old\_s));

memset(new\_s,0,sizeof(new\_s));

char buf[M];

old\_s[0] = {"Pu Shu", 19 , 22, "Best"};

old\_s[1] = {"Zhang San", 20, 23, "Good"};

old\_s[2] = {"Li Si", 21, 24, "Good"};

old\_s[3] = {"Wang Wu", 22, 25, "Good"};

old\_s[4] = {"Zhao Liu", 23, 26, "Good"};

int before\_cp = sizeof(old\_s);

int after\_cp = pack\_student\_bytebybyte(old\_s,2,buf);

after\_cp += pack\_student\_whole(old\_s+2,3,buf+after\_cp);

// PuShu Score 编码 41 B0 00 00 --> 0100 0001 1011 0000 0000 0000 0000 0000 --> 1.011... \* 2^(bin(10000011)-bin(01111111))

// 上式等价为 10110\_bin == 22\_dec 正确

// 打印第0个学生的16进制score编码

cout << "PuShu Score: ";

printfloat\_16(old\_s[0].score);

cout << "Before Compression: " << before\_cp << " bytes" << endl;

cout << "After Compression: " << after\_cp << " bytes" << endl;

restore\_student(buf,after\_cp,new\_s);

cout << std::hex;

for(int i = 0 ; i < 20 ; i ++){

cout << (static\_cast<short>(buf[i]) & 0xff) << " ";

}

cout << endl;

// 输出与观察一致

return 0;

}

/\*

存放规律

字符串 name 和 remark 按序存放，以'\0' 结尾

age，float 在内存中小端存放 低字节在前

\*/

int pack\_student\_bytebybyte(student \*s, int n, char \*buf){

int now\_add = 0;

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

char \* cp = s[i].name;

// 复制name

while(\*cp != '\0') buf[now\_add++] = \*cp++;

buf[now\_add++] = '\0';

// 复制age (小端存放)

cp = (char \*)&s[i].age;

for(int i = 0 ; i < 2 ; i++)

buf[now\_add++] = \*cp++;

// 复制float 四个字节

cp = (char \*)&s[i].score;

for(int i = 0 ; i < 4 ; i++)

buf[now\_add++] = \*cp++;

cp = s[i].remark;

while(\*cp != '\0') buf[now\_add++] = \*cp++;

buf[now\_add++] = '\0';

}

return now\_add;

}

int pack\_student\_whole(student \*s, int n, char \* buf){

int now\_add = 0;

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

strcpy(buf+now\_add,s[i].name);

now\_add += strlen(s[i].name)+1;

\*(short \*)(buf+now\_add) = s[i].age;

now\_add += 2;

\*(float \*)(buf+now\_add) = s[i].score;

now\_add += 4;

strcpy(buf+now\_add,s[i].remark);

now\_add += strlen(s[i].remark)+1;

}

return now\_add;

}

void restore\_student(char \* buf, int len, student \*s){

int now\_add = 0;

int i = 0;

while(1){

if (now\_add >= len) return;

strcpy(s[i].name,buf+now\_add);

now\_add += strlen(s[i].name)+1;

s[i].age = \*(short \*)(buf+now\_add);

now\_add += 2;

s[i].score = \*(float \*)(buf+now\_add);

now\_add += 4;

strcpy(s[i].remark,buf+now\_add);

now\_add += strlen(s[i].remark)+1;

i++;

}

}

任务2源程序

#include <iostream>

#include <cmath>

using namespace std;

int absVal(int x){

return (x+(x>>31)) ^ (x>>31);

}

int neg(int x){

return ~x + 1;

}

int bitAnd(int x,int y){

return ~(~x|~y);

}

int bitOr(int x,int y){

return ~((~x)&(~y));

}

int bitXor(int x, int y){

return ~(~(x&(~y)) & ~((~x)&y));

}

int isTmax(int x){

return !(x^~(x+1)) & !!(x+1); // 特判x == -1

}

int bitCount(int x){

// 将相邻的两位相加，结果存储在这两位中的低位

x = (x & 0x55555555) + ((x >> 1) & 0x55555555); // 0101

// 将相邻的四位相加，结果存储在这四位中的低两位

x = (x & 0x33333333) + ((x >> 2) & 0x33333333); // 0011

// 将相邻的八位相加，结果存储在这八位中的低四位

x = (x & 0x0F0F0F0F) + ((x >> 4) & 0x0F0F0F0F); // 0000 1111

// 将相邻的十六位相加，结果存储在这十六位中的低八位

x = (x & 0x00FF00FF) + ((x >> 8) & 0x00FF00FF); // 00000000 11111111

// 将相邻的三十二位相加，结果存储在这三十二位中的低十六位

x = (x & 0x0000FFFF) + ((x >> 16) & 0x0000FFFF); // 00000000 00000000 11111111 11111111

// 此时，x的低十六位就是原始输入x中1的个数

return x;

}

int bitMask(int highbit, int lowbit){

return ((~0) << lowbit) & ~((~0) << (highbit+1));

}

int addOK(int x,int y){

// return ((x>>31 & 1) ^ (y>>31 & 1)) | (((x>>31 & 1) & (y>>31 & 1)) ^ (x+y) >> 31 & 1;

return (x>>31 & 1) ^ (y>>31 & 1) | !((x>>31&1) & (y>>31&1) ^ (x+y)>>31&1);

}

int byteswap(int x,int n,int m){

int mask = 0x000000FF;

int get\_n = (mask << (n<<3)) & x;

int get\_m = (mask << (m << 3) ) & x;

return (~((mask << (n<<3)) | (mask << (m<<3))) & x) | (((get\_n >> (n<<3)) << (m<<3)) & (mask << (m<<3)) | ((get\_m >> (m<<3)) << (n<<3)) & (mask << (n<<3)));

}

int main(){

printf("Test absVal by abs\n");

for(int i = -100 ; i <= 100 ; i++){

if(absVal(i) != abs(i)){

printf("Error at %d\n",i);

}

}

printf("absVal test done\n");

printf("Test neg by \'-\'\n");

for(int i = -100 ; i <= 100 ; i++){

if(neg(i) != -i){

printf("Error at %d\n",i);

}

}

printf("neg test done\n");

printf("Test bitAnd by \'&\'\n");

for(int i = -100 ; i <= 100 ; i++){

for(int j = -100 ; j <= 100 ; j++){

if(bitAnd(i,j) != (i&j)){

printf("Error at %d %d\n",i,j);

}

}

}

printf("bitAnd test done\n");

printf("Test bitOr by \'|\'\n");

for(int i = -100 ; i <= 100 ; i++){

for(int j = -100 ; j <= 100 ; j++){

if(bitOr(i,j) != (i|j)){

printf("Error at %d %d\n",i,j);

}

}

}

printf("bitOr test done\n");

printf("Test bitXor by \'^\n");

for(int i = -100 ; i <= 100 ; i++){

for(int j = -100 ; j <= 100 ; j++){

if(bitXor(i,j) != (i^j)){

printf("Error at %d %d\n",i,j);

}

}

}

printf("bitXor test done\n");

printf("Test isTmax by integar 0x7FFFFFFF,0x80000000,0xFFFFFFFF,0x7FFFFFFE\n");

printf("Answer should be 1 0 0 0\n");

printf("%d\n",isTmax(0x7FFFFFFF));

printf("%d\n",isTmax(0x80000000));

printf("%d\n",isTmax(0xFFFFFFFF));

printf("%d\n",isTmax(0x7FFFFFFE));

printf("isTmax test done\n");

printf("Test BitCount By 0x12345678 and 0x87654321, answer should be 13\n");

printf("%d\n",bitCount(0x12345678));

printf("%d\n",bitCount(0x87654321));

printf("BitCount test done\n");

printf("Test addOK by 0x7FFFFFFF,1 ; 0x7FFFFFFF,0 ; 0x80000000,-1 ; 0x80000001,-1\n");

printf("Answer should be 0 1 0 1\n");

printf("%d\n",addOK(0x7FFFFFFF,1));

printf("%d\n",addOK(0x7FFFFFFF,0));

printf("%d\n",addOK(0x80000000,-1));

printf("%d\n",addOK(0x80000001,-1));

printf("addOK test done\n");

printf("Test byteswap By 0x12345678,1,3, answer should be 0x56341278\n");

printf("%x\n",byteswap(0x12345678,1,3));

printf("Test byteswap By 0xDEEFBEAD,0,2, answer should be 0xDEEFBEAD\n");

printf("%x\n",byteswap(0xDEEFBEAD,0,2));

printf("byteswap test done\n");

return 0;

}



**课 程 实 验 报 告**

**课程名称： 计算机系统基础**

**实验名称： 机器级语言理解**

**院 系 ：计算机科学与技术**

**专业班级 ： 计卓202201**

**学 号 ： U202215322**

**姓 名 ： 濮澍**

**指导教师 ： 许向阳**

**2024年 4月 7 日**

**一、实验目的与要求**

通过逆向分析一个二进制程序（称为“二进制炸弹”）的构成和运行逻辑，加深对理论课中关于程序的机器级表示各方面知识点的理解，增强反汇编、跟踪、分析、调试等能力。

实验环境：Ubuntu，GCC，GDB等

**二、实验内容**

**任务1** 二进制炸弹拆除

作为实验目标的二进制炸弹（binary bombs）可执行程序由多个“关”组成。每一个“关”（阶段）要求输入一个特定字符串，如果输入满足程序代码的要求，该阶段即通过，否则程序输出失败。实验的目标是设法得到得出解除尽可能多阶段的字符串。

为了完成二进制炸弹的拆除任务，需要通过反汇编和分析跟踪程序每一阶段的机器代码，从中定位和理解程序的主要执行逻辑，包括关键指令、控制结构和相关数据变量等等，进而推断拆除炸弹所需要的目标字符串。

实验源程序及相关文件 bomb.rar

bomb.c 主程序

phases.o 各个阶段的目标程序

support.c 完成辅助功能的目标程序

support.h 公共头文件

**阶段1： 串比较 phase\_1(char \*input);**

要求输出的字符串(input) 与程序中内置的某一特定字符串相同。提示：找到与input串相比较的特定串的地址，查看相应单元中的内容，从而确定input 应输入的串。

**阶段2：循环 phase\_2(char \*input);**

要求在一行上输入 6个整数数据，与程序自动产生的 6个数据进行比较，若一致，则过关。提示：将输入串input拆分成 6个数据由函数 read\_six\_numbers(input, numbers) 完成。之后是各个数据与自动产生的数据的比较，在比较中使用了循环语句。

**阶段3：条件分支 phase\_3(char \*input);**

要求输入两个整数数据，与程序中给定的数据比较，相等则过关。提示：在自动生成数据时，使用了 switch … case 语句。

**阶段4：递归调用和栈 phase\_4(char \*input);**

要求在一行中输入两个数和一个串，第一个数表示在一个有序的数组（或者binary search tree）中需要搜索到的数，该数是在一定范围之内的；第二个数表示找到搜索数的路径（在树的左边搜索编码为二进制位0，在树的有边搜索编码为二进制位1）。

**阶段5：指针和数组访问 phase\_5(char \*input);**

要求在一行中输入一个串，该串与程序自动生成的串相同。在生成串和比较串时，使用了数组和指针。

**阶段6：链表、结构、指针的访问 phase\_6(char \*input);**

要求在一行中输入6个数，这6个数是一个链表中结点的顺序号（从 1 到6）。按照输入的顺序号，将对应链表结点中的值形成一个数组。若该数组是按照降序排列的，则过关。

**三、实验记录及问题回答**

**（1）实验任务1 的实验记录**

实验记录分为六个阶段分别拆弹

1. 串比较

在0x0000000000401853 <+83>: call 0x40154d <strings\_not\_equal> 前打断点 观察rsi中字符串（rdi中是自己输入的字符串）

1. 循环

在反汇编代码显示的几个je机器代码处打断点，每次都观察寄存器rax和rdx的关系，让输入与rdx（有几个是rax）中值相同

1. 条件分支

观察反汇编代码，第一个必须跳过的于输入有关的地址上打断点，观察rax和rdx。可得第一个数3

0x0000000000401997 <+109>: je 0x40199e <phase\_3+116>

函数后半段有一堆跳转到phase\_3+254地址的指令，因此直接看phase\_3+254附近代码 发现第二个弹坑

0x0000000000401a28 <+254>: mov eax,DWORD PTR [rbp-0x14]

0x0000000000401a2b <+257>: cmp DWORD PTR [rbp-0x10],eax

0x0000000000401a2e <+260>: je 0x401a35 <phase\_3+267>

这时需要观察eax和rbp-0x10的值，获得答案

1. 递归调用

0x0000000000401b15 <+79>: mov eax,DWORD PTR [rbp-0x1c]

0x0000000000401b18 <+82>: cmp eax,0xe

0x0000000000401b1b <+85>: jle 0x401b22 <phase\_4+92>

第一个 数与0xE = 14 有关

0x0000000000401b43 <+125>: cmp eax,DWORD PTR [rbp-0x10]

0x0000000000401b46 <+128>: jne 0x401b50 <phase\_4+138>

对比了eax和[rbp-0x10]。因此观察rax和栈中内容，获得第二个答案7

1. 指针和数组

本题中对输入数据做了处理and eax 0xf --> 即输入的字符的ASCII码只有后四位有效。 而根据代码中的循环，发现是通过低四位代表的数字作为索引，向一个特定数组中取值获得一个需要的数组。需要的数组根据观察代码末尾call stringnotequal前的rdi和rsi获得bruins

再观察代码，发现特定数组地址

0x0000000000401bc6 <+90>: lea rdx,[rip+0x27e3] # 0x4043b0 <array.3096>

取出数组内容

(gdb) x/s 0x4043b0

0x4043b0 <array.3096>: "maduiersnfotvbyl8\003"

因此要输入的字符与0xf后应该获得索引 13 6 3 4 8 7

所以查ASCII表后得到solution “MFCDHG”

1. 链表

此题具体代码没分析明白，但取巧了一下，因为反汇编代码给出了node1的地址

0x0000000000401c40 <+37>: lea rax,[rip+0x27b9] # 0x404400 <node1>

发现只需打印3个字 就可获得该结点的 数据域 编号 指针域

记录数据 根据指针域获得下一个node地址 再反复进行此操作五次 进行数据排序

获得结果。

**（2）拆除炸弹的过程中关键操作**

主要的操作就是

1. 反汇编phase\_x
2. 根据反汇编代码的地址打断点(b \*0x...)
3. 在反汇编关键地址打断点（对比，跳转）以观察寄存器和栈(i r ; x/<nxw> address或x/s观察字符串)
4. 根据寄存器和栈值获得常量答案

**四、体会**

了解了gdb的使用，了解了如何对地址打断点，比较浅显的了解了机器代码怎么阅读，函数调用方法。掌握了一定从大段机器代码中阅读重要信息的能力，但对整体代码的理解还是不够。



**课 程 实 验 报 告**

**课程名称： 计算机系统基础**

**实验名称： 缓冲区溢出攻击**

**院 系 ：计算机科学与技术**

**专业班级 ： 计卓 2201**

**学 号 ： U202215322**

**姓 名 ： 濮澍**

**指导教师 ： 许向阳**

**2024年 5月 27日**

**一、实验目的与要求**

通过分析一个程序（称为“缓冲区炸弹”）的构成和运行逻辑，加深对理论课中关于程序的机器级表示、函数调用规则、栈结构等方面知识点的理解，增强反汇编、跟踪、分析、调试等能力，加深对缓冲区溢出攻击原理、方法与防范等方面知识的理解和掌握；

实验环境：Ubuntu，GCC，GDB等

**二、实验内容**

**任务** 缓冲区溢出攻击

**程序运行过程中，需要输入特定的字符串，使得程序达到期望的运行效果。**

对一个可执行程序“bufbomb” 实施一系列缓冲区溢出攻击(buffer overflow attacks)，也就是设法通过造成缓冲区溢出来改变该程序的运行内存映像(例如将专门设计的字节序列插入到栈中特定内存位置)和行为，以实现实验预定的目标。bufbomb 目标程序在运行时使用函数 getbuf读入一个字符串。根据不同的任务，学生生成相应的攻击字符串。

实验中需要针对目标可执行程序bufbomb,分别完成多个难度递增的缓冲区溢出攻击(完成的顺序没有固定要求)。按从易到难的顺序，这些难度级分别命名为smoke (level 0)、fizz (level 1)、bang (level 2)、boom (level 3)和kaboom (level 4)。

**1、第0级 smoke**

正常情况下，getbuf函数运行结束，执行最后的ret指令时，将取出保存于栈帧中的返回（断点）地址并跳转至它继续执行（test函数中调用getbuf处）。要求将返回地址的值改为本级别实验的目标smoke函数的首条指令的地址， getbuf函数返回时，跳转到smoke函数执行，即达到了实验的目标。

**2、第1级 fizz**

要求getbuf函数运行结束后，转到 fizz函数处执行。与smoke的差别是，fizz函数有一个参数。 fizz函数中比较了参数val 与 全局变量cookie的值，只有两者相同（要正确打印val）才能达到目标。

**3、第2级 bang**

要求getbuf函数运行结束后，转到 bang 函数执行，并且让全局变量global\_value 与 cookie相同（要正确打印global\_value）。

**4、第3级 boom**

无感攻击，执行攻击代码后，程序仍然返回到原来的调用函数继续执行，使得调用函数（或者程序用户）感觉不到攻击行为。

构造攻击字符串，让函数 getbuf将cookie值返回给 test函数，而不是返回值 1 。还原被破坏的栈帧状态，将正确的返回地址压入栈中，并且执行 ret 指令，从而返回到 test函数。

**三、实验记录及问题回答**

**（1）实验任务的实验记录**

**通过重写Buf越过当前子程序的堆栈区，smoke通过直接修改保存的EIP**

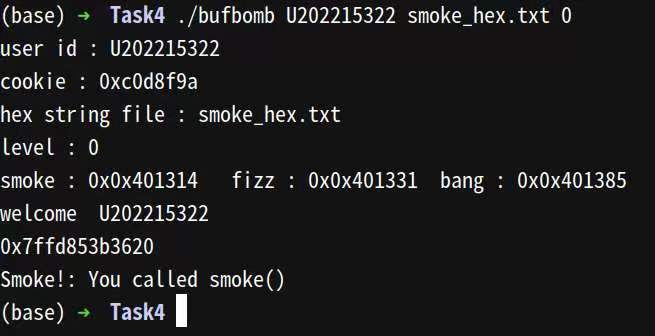
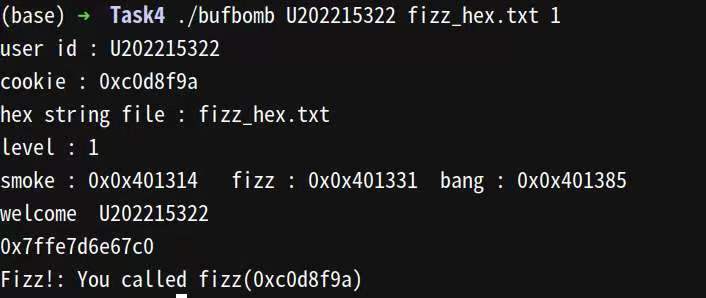


图1 第0级 smoke



**图2 第一级fizz**

**fizz通过修改全局变量值加修改eip为fizz程序地址来攻击**

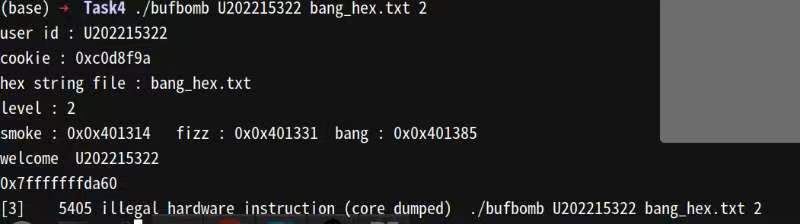


图3 第二级bang

这里重新运行的时候不知道为什么报了个没见过的错误，之前都能完美运行的； 之前遇到了许多 如进入printf错误，gdb运行时和正常运行堆栈段不同等错误，还需通过写两个不同的rbp地址的文件来分别测试

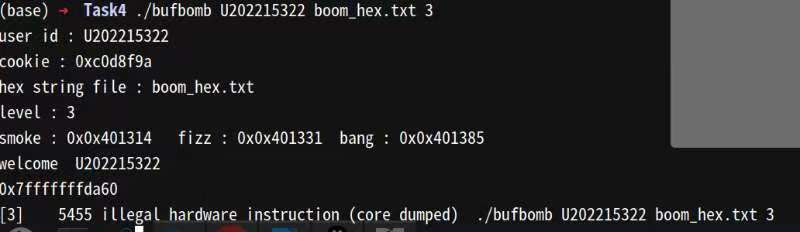


图4 第三级 boom

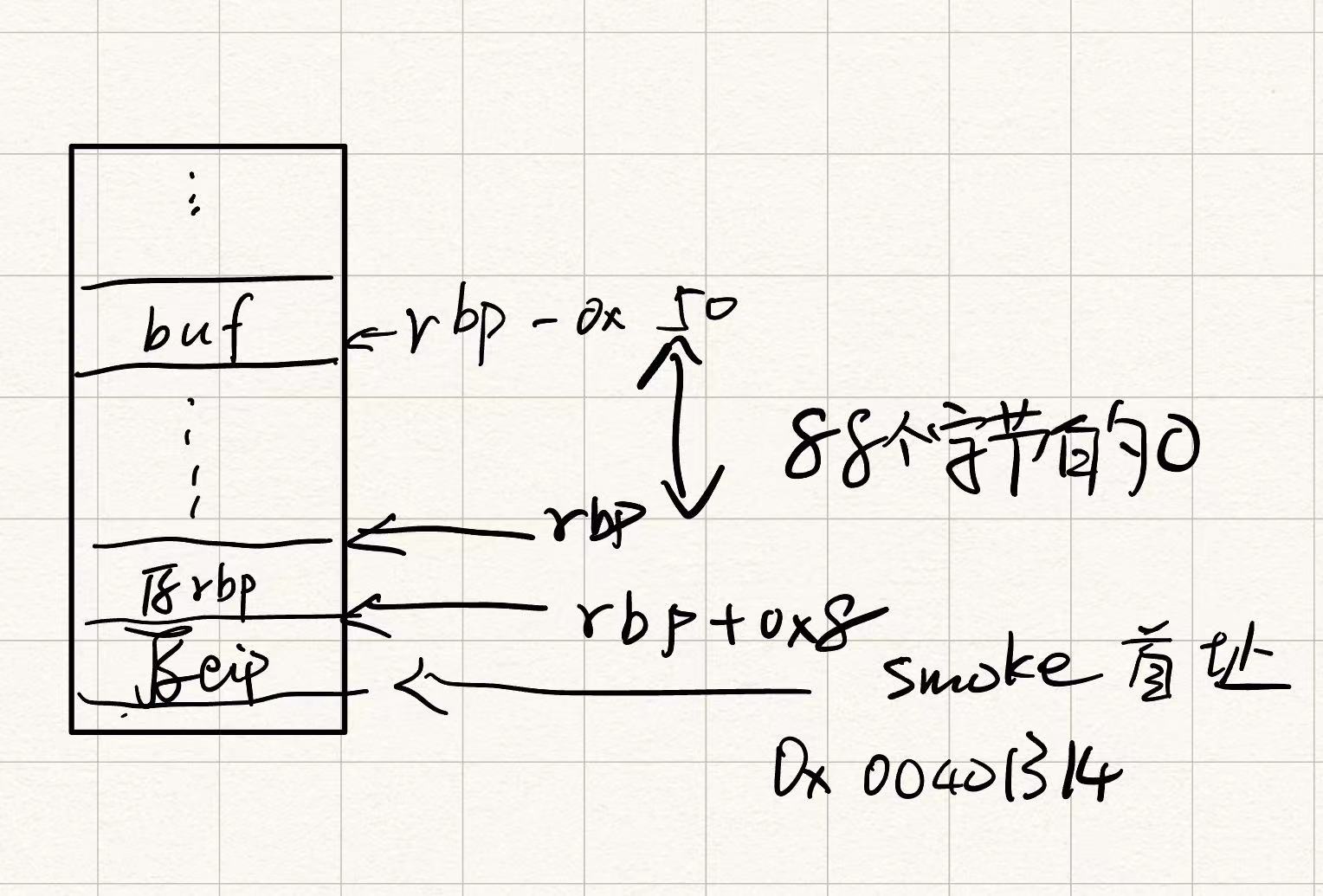
出现了和第二关一样的问题，但我当时做的时候这块是能过的，不知道发生了什么问题

**bang与boom类似 通过获得汇编程序的十六进制代码，对程序代码进行攻击**

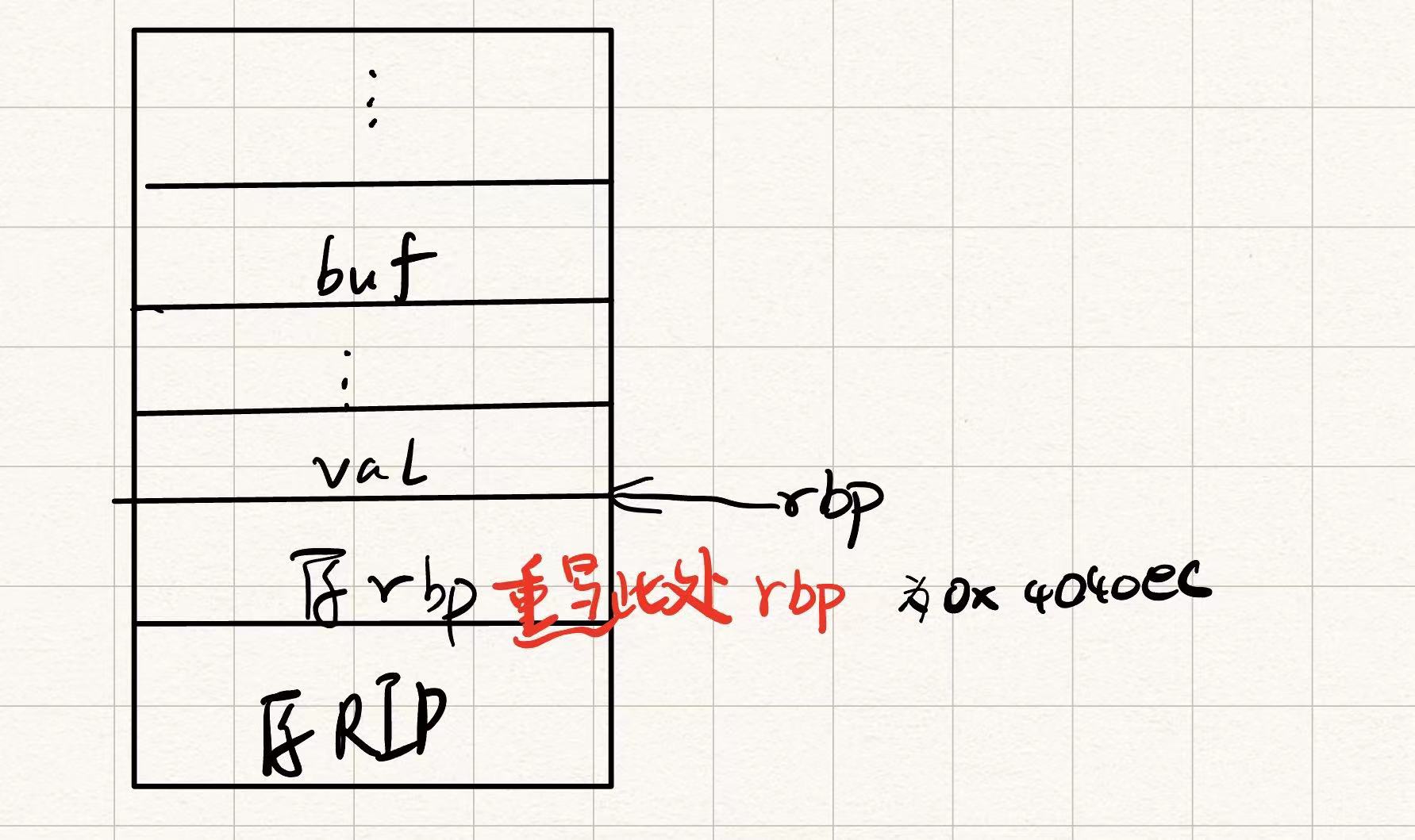
**（2）缓冲区溢出攻击中字符串产生的方法描述**

**要求：一定要画出栈帧结构 （包括断点的存放位置，保存ebp的位置，局部变量的位置等等）**

1. **第0关 smoke**



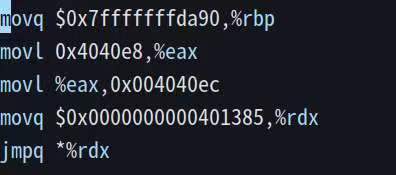
**通过观察运行时寄存器的值，发现buf在栈中距离原ebp的距离为0x50，因此重写buf时，用足够长度00填充，并在原EIP位置上（原ebp＋8bytes）填写smoke函数首地址即可**

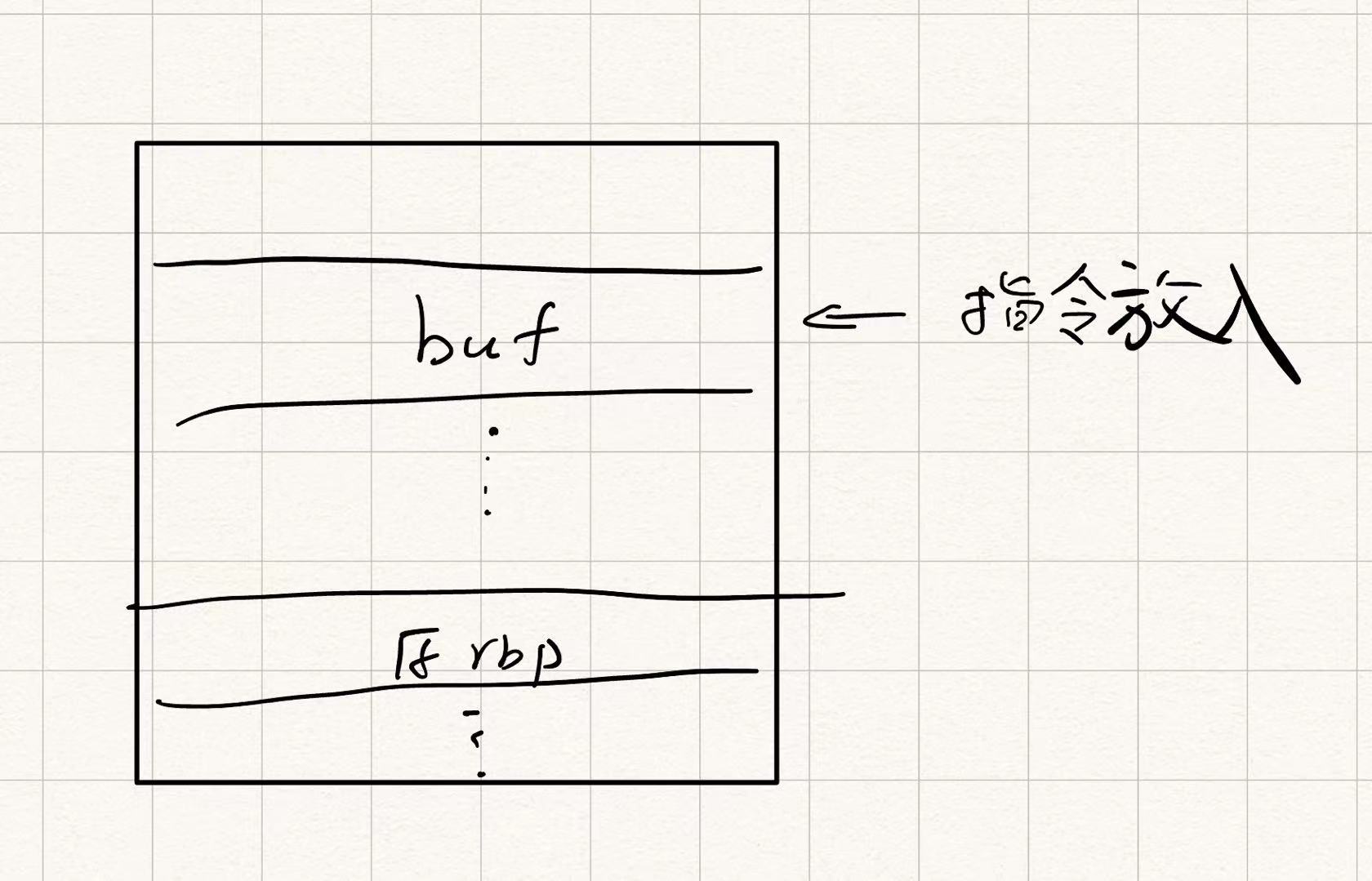
1. **第1关 fizz**

**这关是要通过将val的地址改成和cookie相同，来规避传参工作，即 通过修改rbp地址 使得对应val的地址和cookie相同。 cookie : 0x4040e8; val : rbp-0x4 所以原rbp**

**改为4040ec0x**

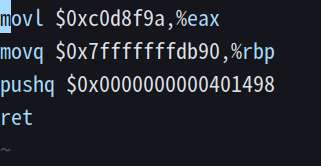
1. **第2关 bang**

**这关需要自己写汇编代码并转化为机器代码，使得可以更改全局变量cookie的值，并通过jmp跳转到对应bang函数地址中去**

第一句是恢复rbp为test原本的rbp地址，避免环境出错，第二句将cookie的地址传入eax,并在第三句通过寄存器寻址修改global\_value的值。 第四句和第五句联合实现了跳转到bang.s的功能

1. **第3关 boom**

**写汇编程序1. 恢复rbp值 2. 修改eax中值为cookie值 3.通过ret跳转到boom.s**



以上程序实现了上述三个功能

1. **体会**

**由于系统环境不同，可参考的东西不够，在debug的时候变得极为复杂，很多时候不知道为什么就报错了。 所以在研究为什么会报seg fault等错误时遇到了很多问题**

**因此在解决这些问题的过程中，增强了我对堆栈段的变化的理解，尤其是rbp和rsp的跳转问题。对电脑安全的防护的意识加深。**

**学习了简单的汇编代码怎么写**



**课 程 实 验 报 告**

**课程名称： 计算机系统基础**

**实验名称： ELF文件与程序链接**

**院 系 ：计算机科学与技术**

**专业班级 ： 计卓202201**

**学 号 ： U202215322**

**姓 名 ： 濮澍**

**指导教师 ： 许向阳**

**2024年 5月 27 日**

**一、实验目的与要求**

通过修改给定的可重定位的目标文件（链接炸弹），加深对可重定位目标文件格式、目标文件的生成、以及链接的理论知识的理解。

实验环境：Ubuntu

工具：GCC、GDB、readelf、hexdump、hexedit、od等。

**二、实验内容**

**任务** 链接炸弹的拆除

**在二进制层面，逐步修改构成目标程序“linkbomb”的多个二进制模块（“.o文件”），然后链接生成可执行程序，要求可执行程序运行能得到指定的效果。修改目标包括可重定位目标文件中的数据、机器指令、重定位记录等。**

**1、第1关 数据节的修改**

修改二进制可重定位目标文件 phase1.o 的数据节中的内容（不允许修改其他节），使其与main.o链接后，生成的执行程序，可以输出自己的学号。

**2、第2关 简单的机器指令修改**

修改二进制可重定位目标文件 phase2.o 的代码节中的内容（不允许修改其他节），使其与main.o链接后，生成的执行程序。在phase\_2.c 中，有一个静态函数 static void myfunc( ) ，要求在 do\_phase 函数中调用myfunc( )，显示信息myfunc is called. Good!。

**3、第3关 有参数的函数调用的机器指令修改**

修改二进制可重定位目标文件 phase3.o 的代码节中的内容（不允许修改其他节），使其与main.o链接后，生成的执行程序。在phase\_3.c 中，有一个静态函数 static void myfunc(int offset) ，要求在 do\_phase函数中调用myfunc(pos )，将do\_phase的参数pos直接传递myfunc，显示相应的信息。

**4、第4关 有局部变量的机器指令修改**

修改二进制可重定位目标文件 phase4.o 的代码节中的内容（不允许修改其他节），使其与main.o链接后，生成的执行程序。在phase\_4.c 中，有一个静态函数 static void myfunc(char \*s) ，要求在 do\_phase 函数中调用myfunc(s )，显示出自己的学号。

**5、第5关 重定位表的修改**

修改二进制可重定位目标文件 phase5.o 的重定位节中的内容（不允许修改代码节和数据节），使其与main.o链接后，生成的执行程序运行时，显示Class Name : Computer Foundation. Teacher Name : Xu Xiangyang。

**6、第6关 强弱符号**

不准修改 main.c 和phase6.o，通过增补一个文件，使得程序链接后，能够输出自己的学号。

#gcc -no-pie -o linkbomb6 main.o phase6.o phase6\_patch.o

**7、第7关 只读数据节的修改**

修改 phase7.o 中只读数据节（不准修改代码节），使其与main.o链接后，能够输出自己的学号。

**三、实验记录及问题回答**

**（1）实验结果及操作过程记录**

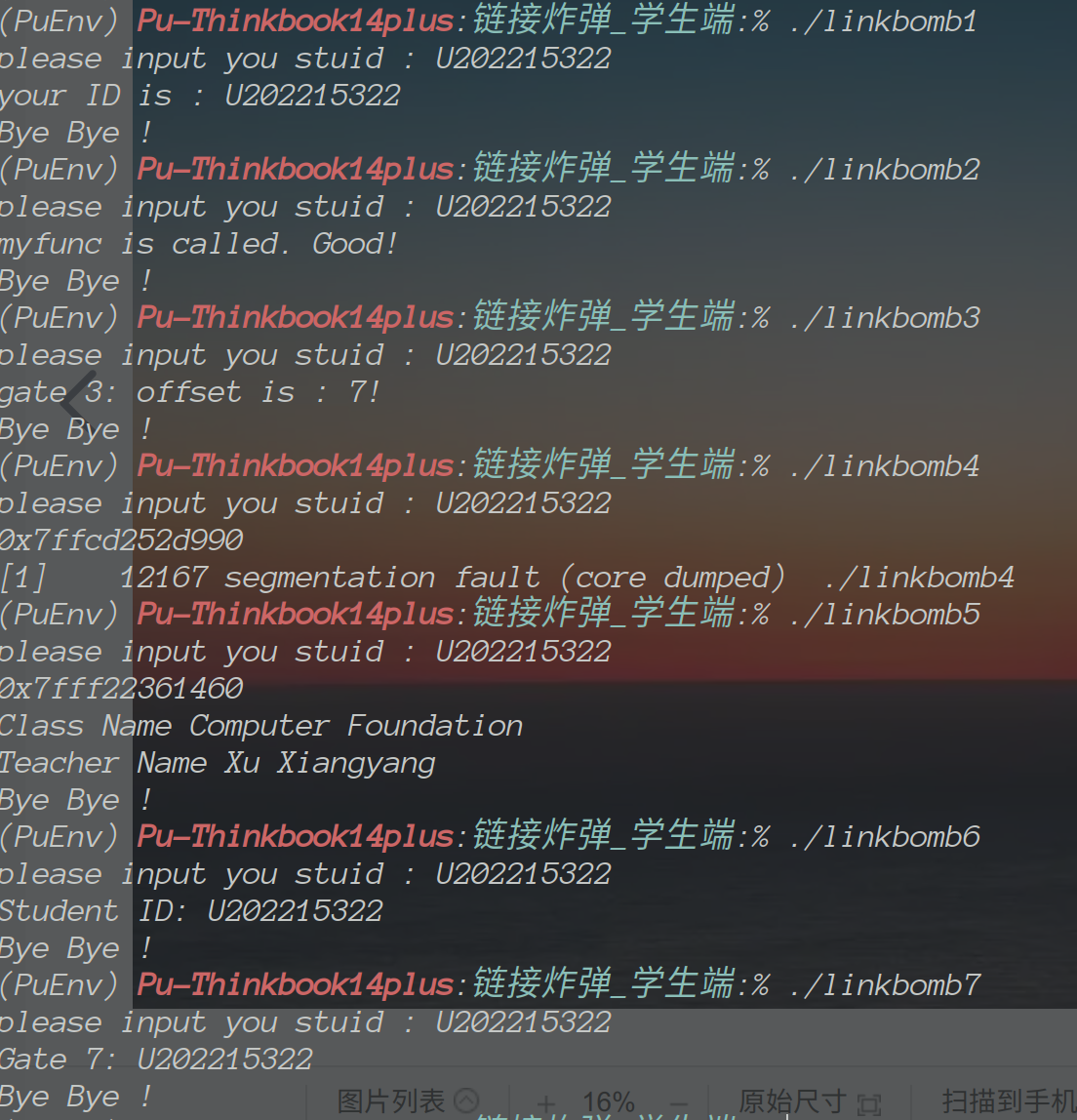


图1 7个任务运行结果

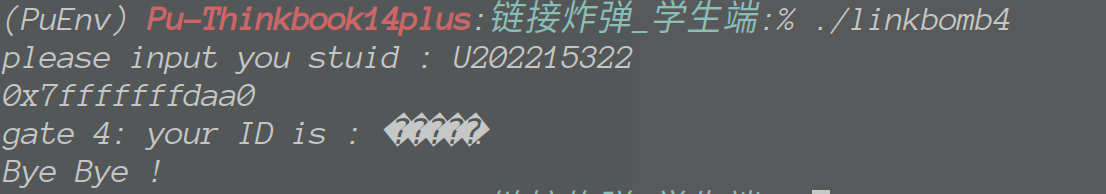


图2 任务4取消地址随机化后的结果

**（2） 描述修改各个文件的基本思想**

1. 第一关 修改文件数据节

未修改前的值为hijklmnopqrs..... 在数据节中定位这段字符串，改为自己的学号

1. 第二关 简单的机器指令修改

找到myfunc首地址，计算相对偏移量，通过写程序确定call的操作码为e8

通过查看phase2.o查看call指令的位置，将其改为e8 d9 ff ff ff

将占位的四个nop删除

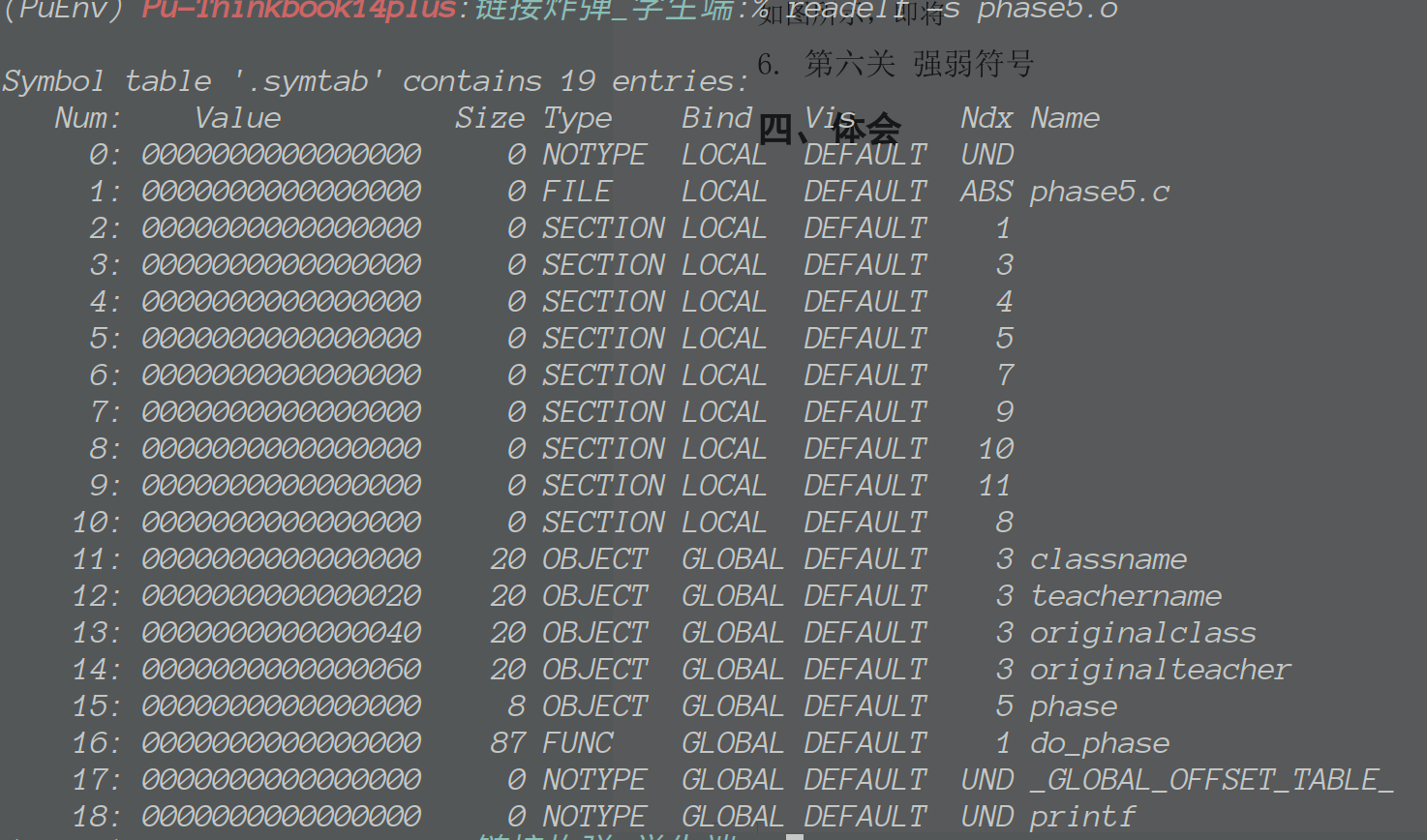
1. 第三关 有参数的函数调用机器指令修改

与第二关类似，函数通过edi传参，与上一题一样 在确定位置将nop改为call即可

1. 第四关 有局部变量机器指令修改

取消地址随机化，固定rbp和局部变量地址 写汇编代码，将stuid的地址传送给edi(movabs $0x7fffffffda90,%rdi)， 剩余部分就是和2，3一样的call部分编写，非常简单

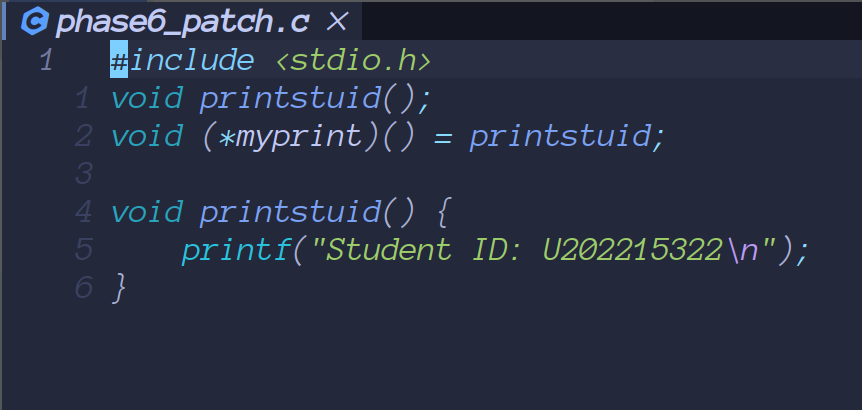
1. 第五关 重定位表的修改
2. 获得符号表
3. 通过观察.rela.text发现 输出不同的原因只与重定位的符号索引不同，因此只需改变originalclass 和 original teacher的需重定位的符号索引为 classname和teachername即可



如图所示，即将0d和0e改为0c和0b索引即可(用hexedit对.rela.text修改)

1. 第六关 强弱符号

phase6.0中有函数指针myprint 为若符号（没有强定义）引用了新写文件中的myprint强符号 所以我们要在patch中实现该程序



**四、体会**

这次实验让我加深了对链接过程的理解，尤其是elf中各个节的作用与意义。更重要的是对符号的重定位有了更直观的理解，在课上没完全理解的基础上，这次实验很好的弥补了我的疑惑。

学会了如何通过objdump readelf, hexedit来实现相关操作



**课 程 实 验 报 告**

**课程名称： 计算机系统基础**

**实验名称： ARM指令系统的理解**

**院 系 ：计算机科学与技术**

**专业班级 ： 计卓202201**

**学 号 ： U202215322**

**姓 名 ： 濮澍**

**指导教师 ： 许向阳**

**2024 年 5 月 27 日**

**一、实验目的与要求**

通过在ARM虚拟环境下调试执行程序，了解 ARM的指令系统。

实验环境：ARM 虚拟实验环境 QEMU

工具：gcc, gdb 等

**二、实验内容**

**任务1、C与汇编的混合编程**

**任务2、内存拷贝及优化实验**

程序及操作方法 见 <ARM实验任务.pdf>

**三、实验记录及问题回答**

**（1）实验任务的实验结果记录**

**任务1、C与汇编的混合编程**

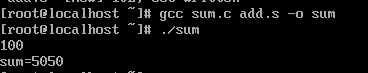


图1 sum程序

**任务2、内存拷贝及优化实验**

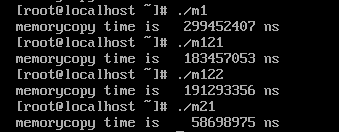


图2 自上而下 无优化 2倍优化 4倍优化 内存突发传输

**（2）ARM 指令及功能说明**

查阅《ISA\_A64\_xml\_A\_profile-2023-09.pdf》，指出10条不同指令（存数、取数、算术运算、转移指令、函数调用等都应覆盖）的功能。

|  |  |
| --- | --- |
| 指令 | 功能 |
| LDP | 一次性读取8个字节或者16个字节数据 |
| STP | 一次性写入8个字节或者16个字节数据 |
| SUB（immediate） | 将一个寄存器的值减去立即数，存在目的寄存器中，该立即数可以选择是否移位 |
| ADD（immediate） | 将一个寄存器的值加上立即数，存在目的寄存器中，该立即数可以选择是否移位 |
| LSL（immediate） | 将一个寄存器的值逻辑左移一个立即数，存在目的寄存器中 |
| B | 依据PC相对偏移量无条件跳转 |
| BL | 依据PC相对偏移量跳转，并且将x30寄存器的值设为PC+4，即指向调用函数指令的下一条指令地址 |
| ADR | 将PC的值加上一个立即数形成PC相对地址 |
| RBIT | 翻转寄存器值的每一位 |
| TBZ | 与0比较，如果相等就跳转到某个标志 |

**四、体会**

通过这次实验，了解了一个新的指令体系，并且通过查阅相关资料，认识了许多只能在ARM中使用的指令，让我对这个指令集产生了不小的兴趣