实验6

班级：191211

姓名：孙鹤轩

学号：20211000156

实验目标：

1 加深对程序链接中符号解析、重定位等基本概念、位置无关代码和 ELF 文件的基本组成

等方面知识的理解和掌握；

2 掌握计算机系统思维，理解高级语言中数据、运算、过程调用和 I/O 操作等在计算机系统

中的实现方法，将程序设计、汇编语言、系统结构、操作系统、编译链接中的重要概念贯穿

起来。能够对计算机系统复杂工程问题制定解决方案

3 掌握各种开源的编译调试工具，能够对分析优化程序设计，提高在代码调试、性能提升、

软件移植和鲁棒性等方面的能力。

实验任务：

1 学习 MOOC 内容

https://www.icourse163.org/learn/NJU-1449521162

第七周 程序的链接

第 1 讲 链接与 ELF 实验：概述

第 2 讲 链接与 ELF 实验：静态数据与 ELF 数据节

第 3 讲 链接与 ELF 实验：指令与 ELF 代码节及课后实验

2 完成实验

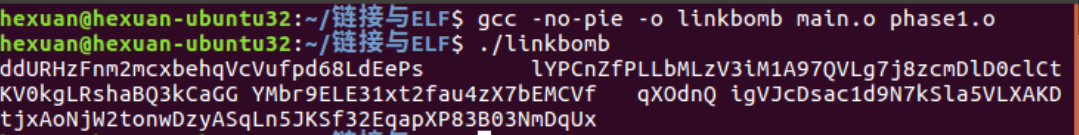
详见链接与 ELF 实验文档

2.1 静态数据对象与 ELF 数据节 phase1

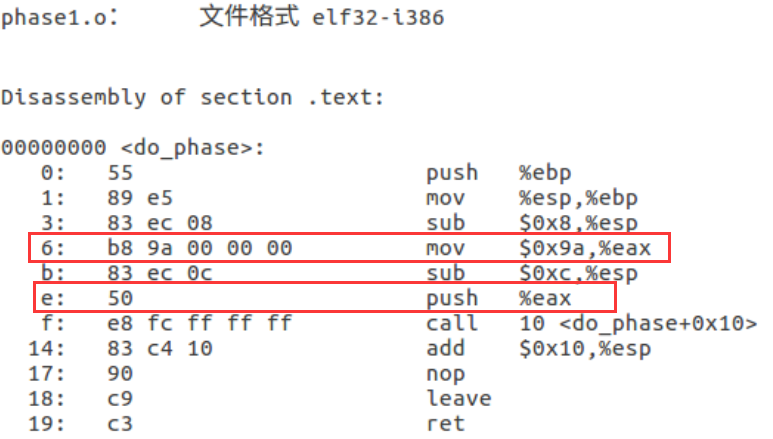
2.1.1实验结果与分析

我们的实验目标是修改二进制可重定位目标文件“phase1.o”的数据节内容（不允许修改 其他节），使其与main.o链接后运行时输出自己的学号

首先对程序进行链接执行产看原本输出：

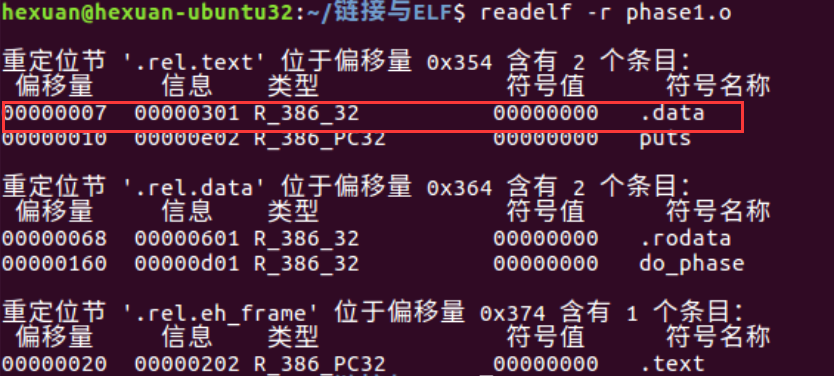


接着对phase1.o进行反汇编，查看汇编代码，如下：



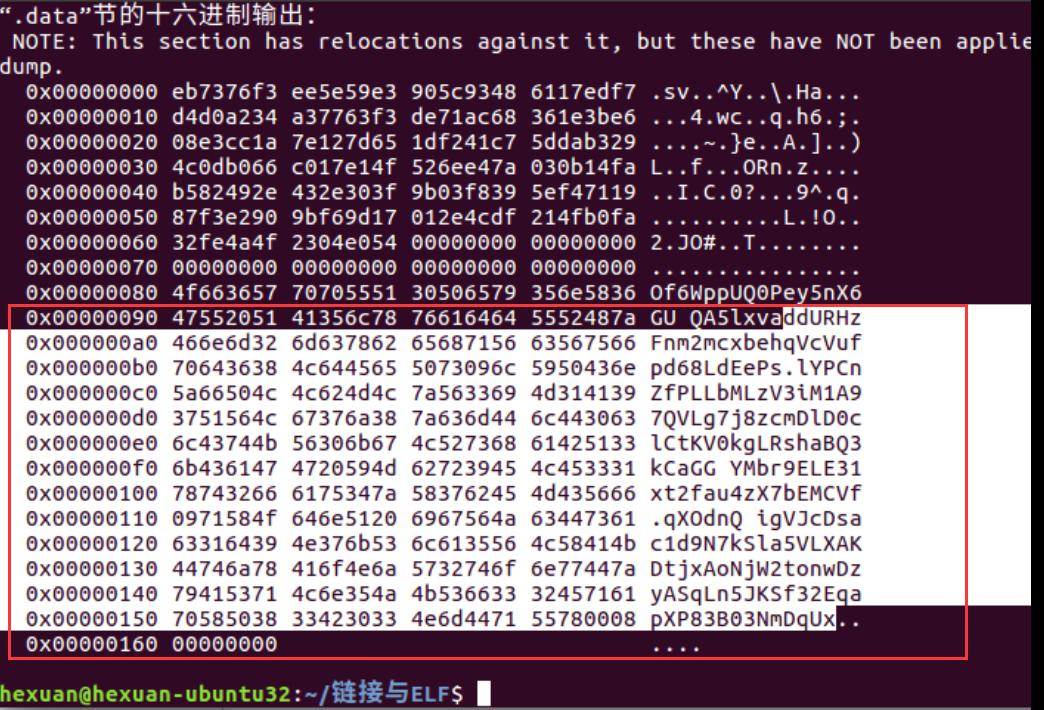
其中标记的两行代码对常熟0x9a进行了压栈

接着使用readelf查看重定位信息，如下：



其中R\_386\_PC32PC绝对地址重定位方式下，重定位后的引用地址=符号定义地址-符号引用所在地址+重定位前引用处的初始值

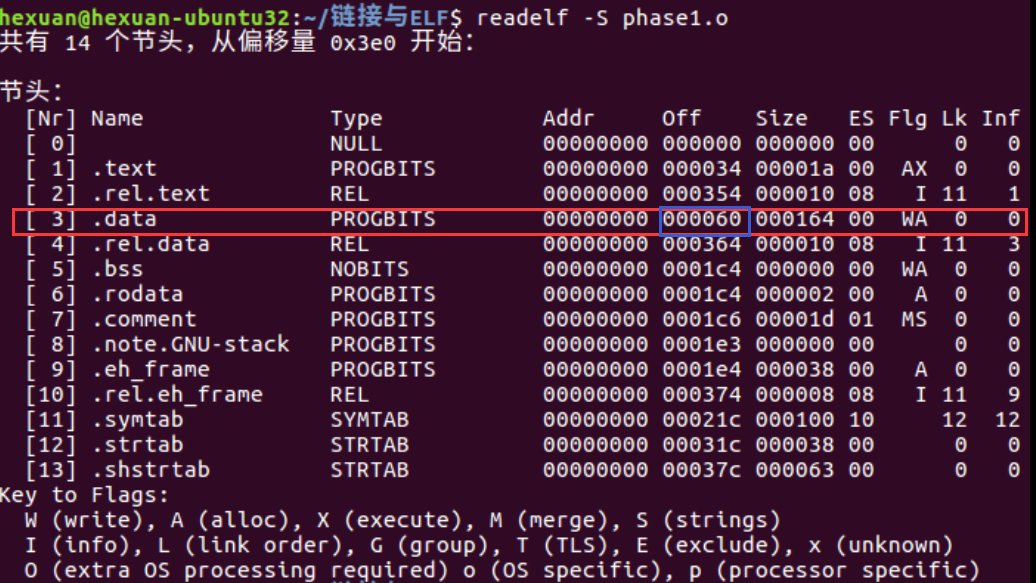
利用readelf -x .data获得模块1指定节的内容，如下：



上述在data节中对应0x9a偏移量地址对应的字符串与前面执行程序输出的字符串一致，可见输出字符串对应的地址确实是data节中的0x9a

然后使用hexedit工具，对phase1.o模块的数据节中相应字节进行修改

首先查看.data节的偏移量：

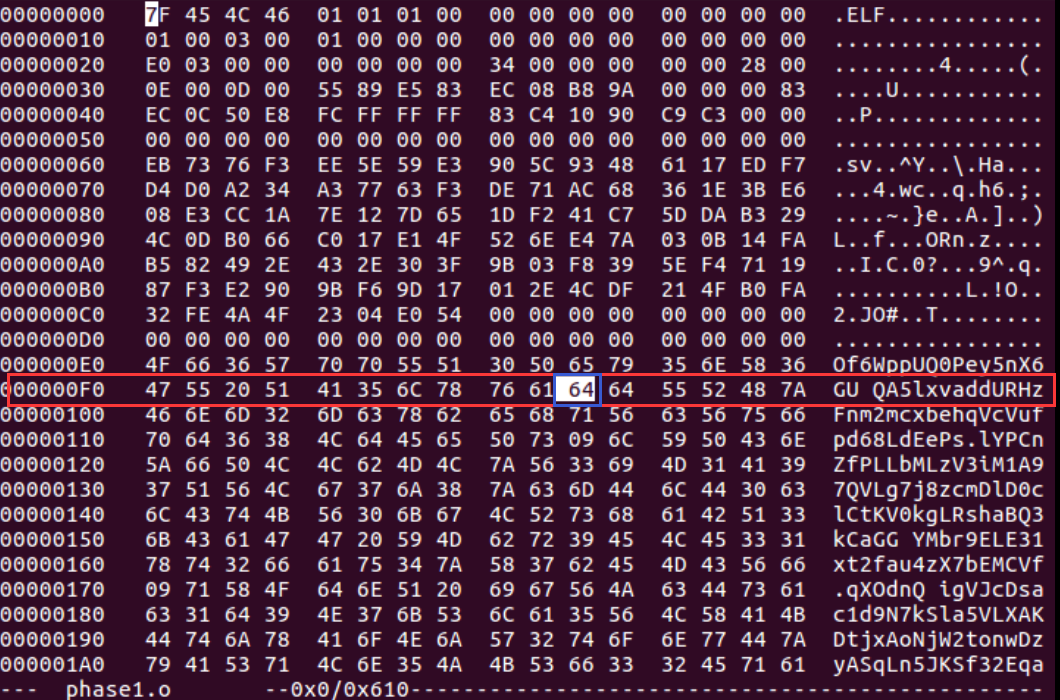


由上图可知，data节在文件中的偏移量为0x60.

所以输出字符串在文件中的偏移量为0x60 + 0x9a = 0xfa

然后利用hexedit工具打开phase1.o模块

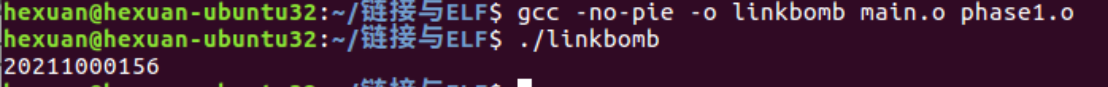
定位到偏移量为0xfa的字节处：



现在把输出字符串从第一个字节开始，依次替换为需要输出的学号对应的字符串，1对应的编码为31后面依次类推，输入完学号后，最后加上00作为字符串结束的标志。

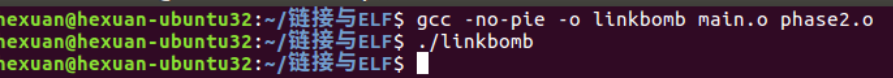


最后重新链接程序，输出结果如下：



2.2 指令与 ELF 代码节 phase2

首先查看原本的输出结果：



什么都没有输出

接着利用objdump对phase2进行反汇编，并得到汇编代码，然后利用readelf查看phase2中出现的重定位记录，结果如下：



接着在反汇编代码中查找call指令看有无函数调用

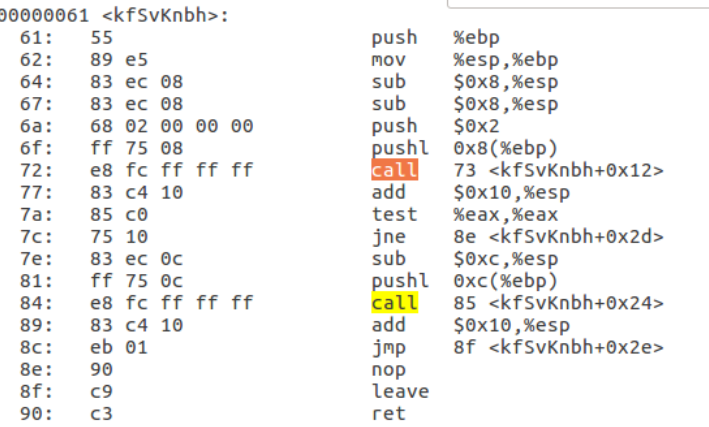
共定位到三处，如下：





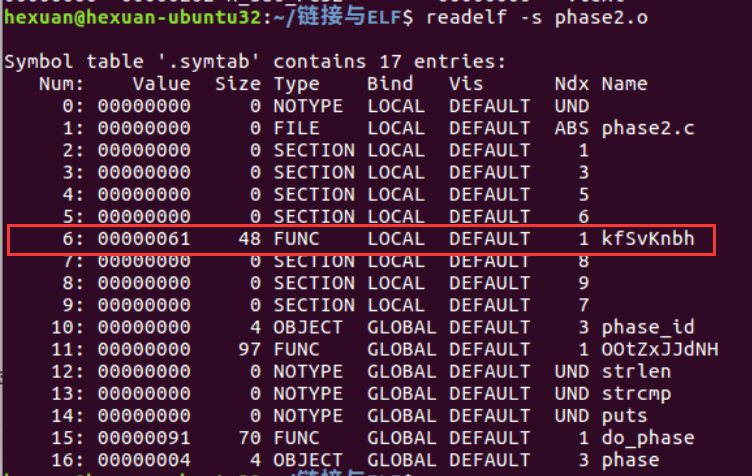


根据重定位信息分析直到其分别对应strlen函数、strcmp函数和puts函数，前两个函数与输出无关，我们只需要关注puts函数即可

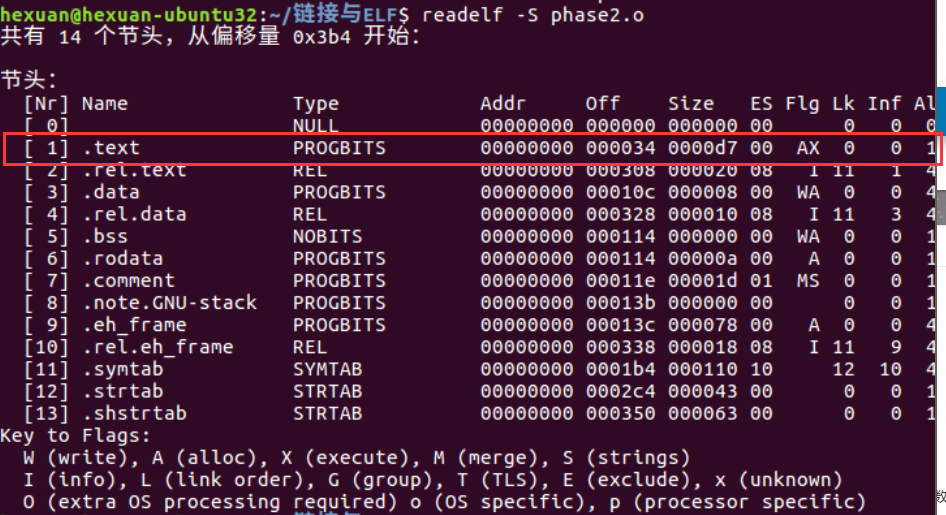


根据上述汇编代码我们知道puts函数在kfsvknbh函数中被调用，所以我们需要在phase2模块的dophase函数中调用kfSvKnbh函数以实现字符串的输出

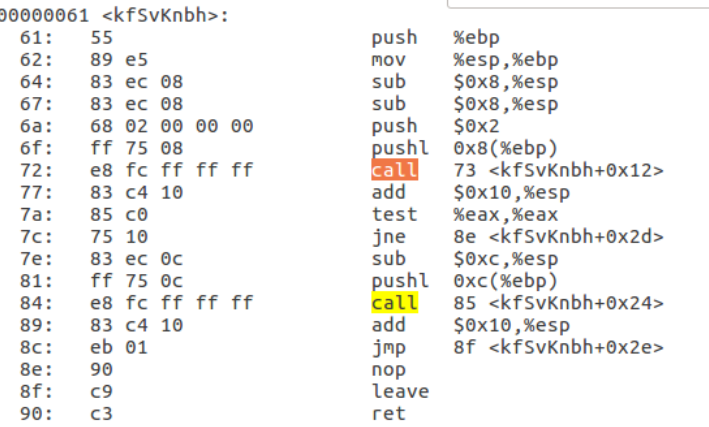
首先从符号表中获得目标函数在.text节中的偏移量



可以看到该函数的偏移量为0x61，并且这个函数对应编号为1的节，大小为48



利用readelf查看得到编号为1的节是text节，所以目标函数位于text节中偏移量为0x61的位置上，kfSvKnbh函数具有LOCAL链接绑定属性，故调用该函数可通过相对于PC值的偏移量来进行跳转，不需要构造相应的重定位记录进行配合



根据上述汇编代码结合重定位表可知第一个call语句调用的是strcmp函数，strcmp函数比较传递给它的两个字符串参数的内容是否相同，如果相同的话将返回0，如不同则返回非0的一个数。

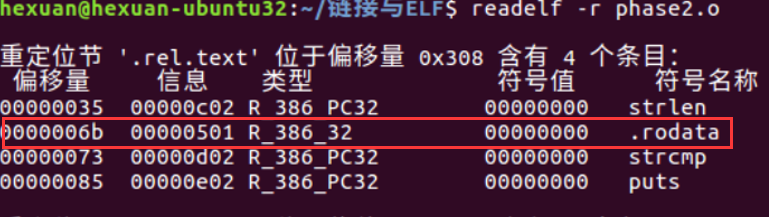
当存放于EAX寄存器中的、strcmp函数的返回值等于0，即两个字符串相同时，将执行jne跳转指令后的参数压栈指令和调用puts函数的另一call指令。这个call对应的就是puts输出函数，被输出的、传递给puts函数的参数字符串地址就是传递给kfSvKnbh函数的第二个参数。

所以为了正确调用ecRIvzPzKN函数完成学号字符串的输出，需要完成两个任务:

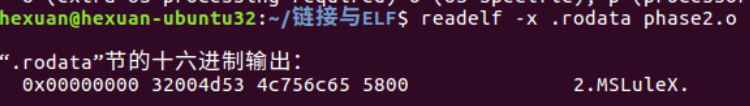
1.确保ecRIvzPzKN函数中传递给strcmp函数的两个参数字符串，在内容上相同。

2.结合重定位记录确定用于strcmp函数进行字符串比较的内置字符串在phase2.o中的存放位置，进一步获得字符串内容。

可以看到先压入栈的偏移量的位置对应一个重定位记录如下：



查看.rodata节的内容如下:



获得其内容为“MSLuleX.” ，内置字符串地址 = .rodata节起始地址 + 引用处的初始值0x2

然后在栈中构造与内置字符串和学号字符串内容相同的局部字符串变量，并将其地址作为实参压入栈中 。

构造在do\_phase函数中，调用kfSvKnbh函数的机器指令代码指令内容:在栈中构造和初始化两个字符串局部变量，使其内容相同于程序内置的比较字符串"MSLuleX"和待输出的学号字符串“123456789" 。可以利用sub指令先分配两个字符串的存储空间，然后使用一组move指令对两个字符串的内容按照前面分析的结果进行设置。然后用push指令将两个字符串的地址作为参数压入栈中。并进一步使用call指令调用目标函数。

编辑如下汇编代码

sub $0x28,%esp

movl $0x566f6c49,-0x10(%ebp)

movl $0x536345,-0xc(%ebp)

movl $0x31323032,-0x1a(%ebp)

movl $0x30303031,-0x16(%ebp)

movl $0x363531,-0x12(%ebp)

sub $0x8,%esp

lea -0x1a(%ebp),%eax

push %eax

lea -0x10(%ebp),%eax

push %eax

call 0x00

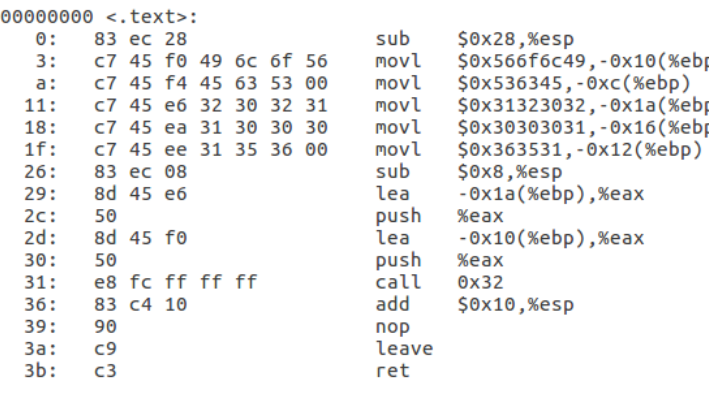
add $0x10,%esp

nop

leave

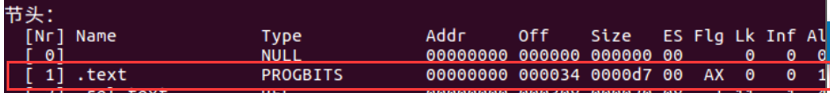
ret

然后利用as工具进行汇编，再利用objdump进行反汇编并查看，如下：



使用上步构造的调用输出函数的指令代码，替换do\_ \_phase()函数体中的nop指令,以实现期望输出。

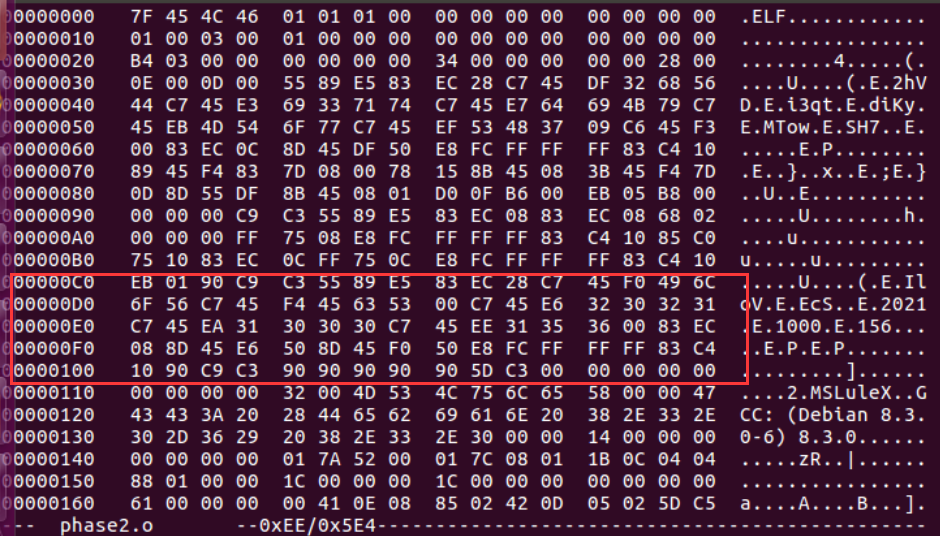
使用readelf工具及其“-S”命令行选项获得节头表，从中获得.text在phase2.o中偏移量为 0x34



进一步得到函数中插入指令代码的位置为：

0x34 + 0xcf = 0xc8 （0xcf是do\_phase函数中第一个被替换的nop指令的偏移量）

使用hexedit将phase2.o文件中上述位置0xc8起始的字节内容（原为nop指令代码0x90） 替换为调用指令代码序列



实验完成