|  |
| --- |
| **实验报告** |

|  |  |  |  |
| --- | --- | --- | --- |
| **实验名称：** | **Lab4** | **日期：** | **2024.12.22** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **班级：** | **U10M21003.05** | **学号：** | **2024302589** | **姓名：** | **苗清皓** |

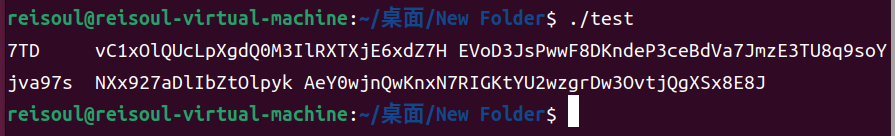
1. **实验概述：**

Lab4:ELF与链接。

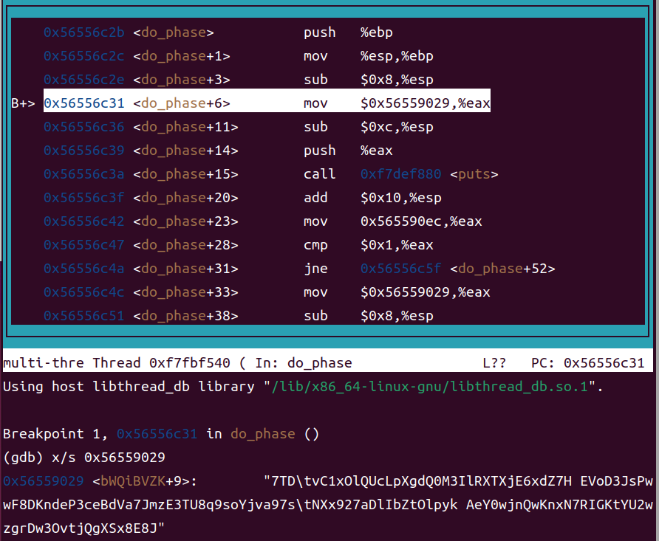
实验原理：给出一个main.o文件和五个phase.o文件，要求修改.o文件，使链接生成的可执行文件实现指定的行为。

实验目的：掌握ELF文件的基本组成，加深对程序链接的理解，熟练掌握查看符号表等重定位信息的方法。

1. **实验分析：**

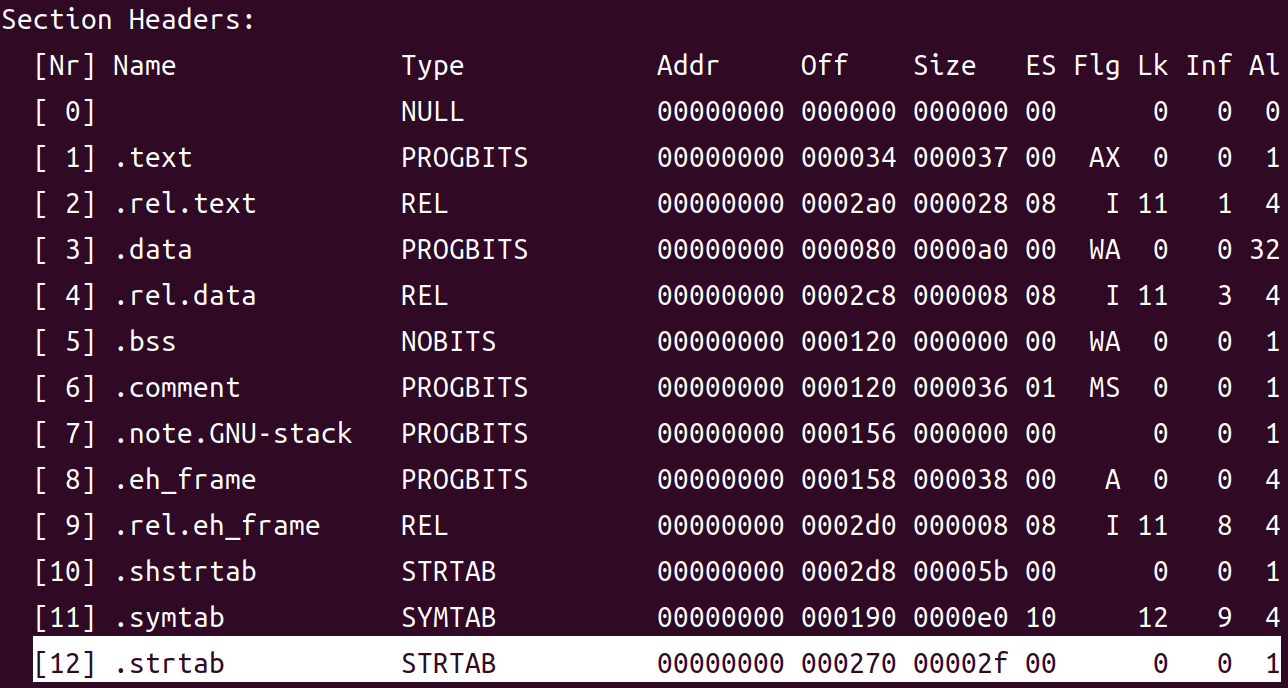
阶段一：

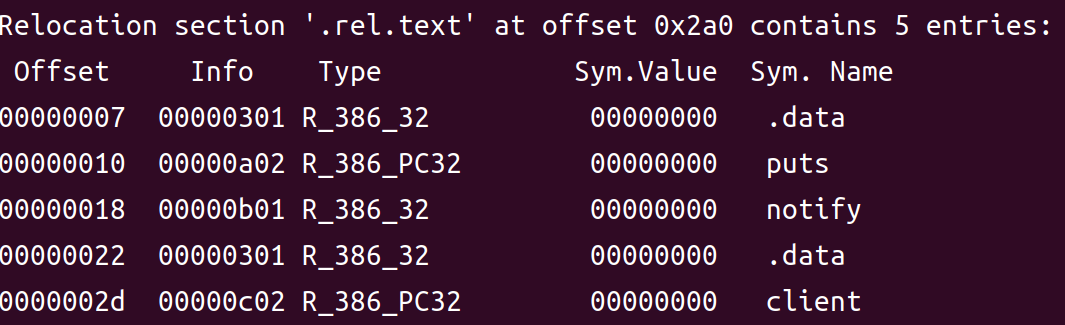
首先链接main.o和phase1.o，生成test可执行文件，直接运行一下：

发现输出是一串无意义的字符串。反汇编一下发现main函数调用了do\_phase函数进行输出，打开gdb进行调试，在do\_phase函数处设置断点运行。

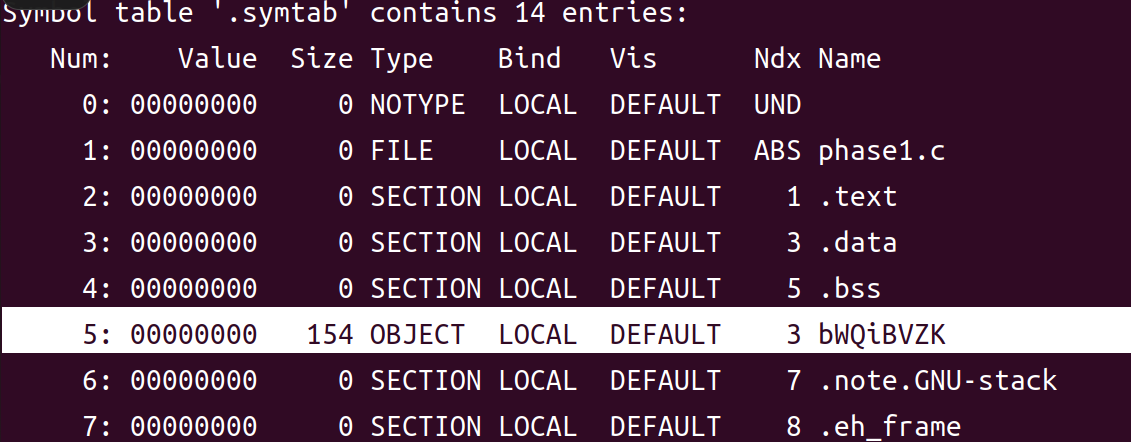
查看0x56559029处的值发现这就是输出的字符串。我们要做的就是把这串字符串修改为学号。

首先要定位字符串的位置。

输入readelf -S phase1.o查看节头表：

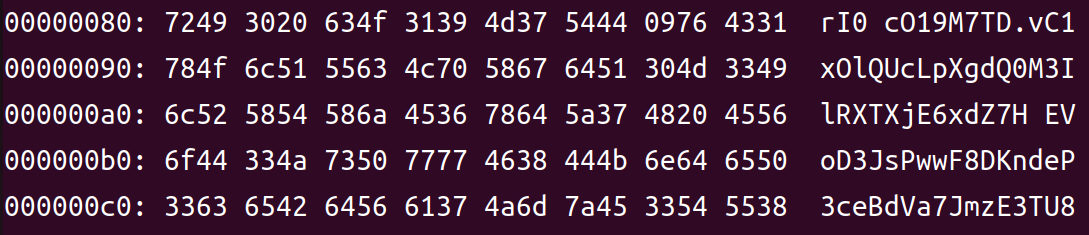
输入readelf -r phase1.o查看可重定位条目：

Offset说明是从代码第几个字节处开始出现重定位条目c。

输入readelf -s phase1.o查看符号表：

这里出现了一个可疑的名称，推测这就是字符串的名字。

从Ndx处发现它属于第三节，即数据节，而数据节的起始地址从节头表的off一栏得出是80字节。

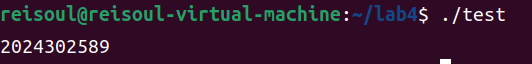
通过vim打开phase1.o,输入：%！xxd将二进制转换为十六进制表示，找到80字节处：

可以看到我们输出的字符串就在这里存放，那我们把从7TD开始的字节修改为学号即可。但是要注意，这里是用十六进制表示的ASCII码，我们也需要用这种格式进行修改。

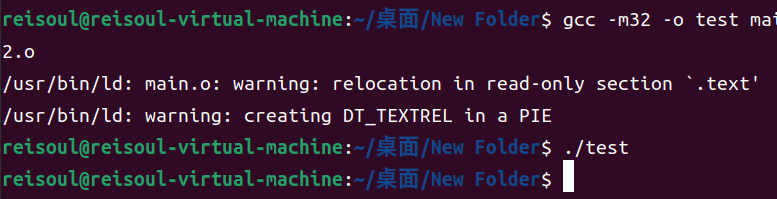
我的学号2024302589就需要改为3230 3234 3330 3235 3839 0000

还要注意，因为是字符串，所以要加\0，防止输出后面的内容。

在vim界面，按ESC键再按i键，进入插入模式，进行修改。随后输入：%！xxd -r，转换回二进制，再输入：wq保存并退出。

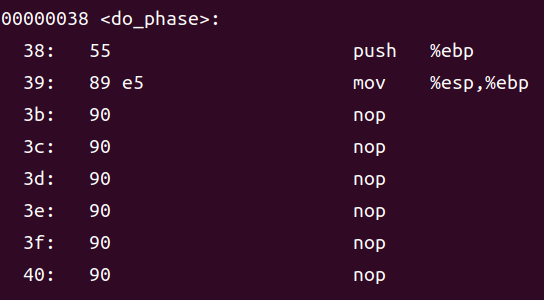
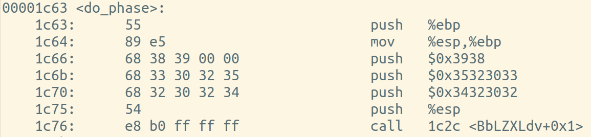
最后重新链接main.o和phase1.o，运行test程序。

第一阶段完成！

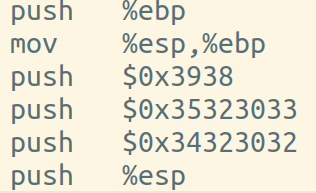
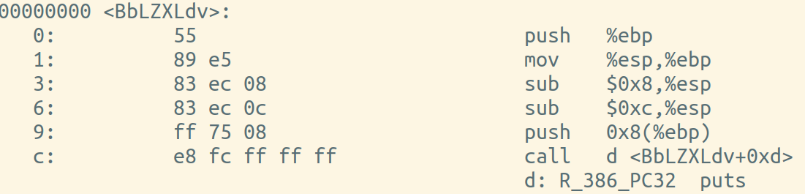
第二阶段：修改phase2.o的代码段内容，使其与main.o链接后能够输出学号。

还是先链接，直接运行看看效果，发现什么也没有输出。查看phase2.o的反汇编代码，发现do\_phase后面执行的都是空指令NOP.

这就需要我们自己补充代码，通过vim写入phase2. o了。

再查看程序，发现了一个函数，它里面调用了puts函数。那么我们的目标就是通过do\_phase调用这个函数，并把学号当作参数传给这个函数，让它输出学号。

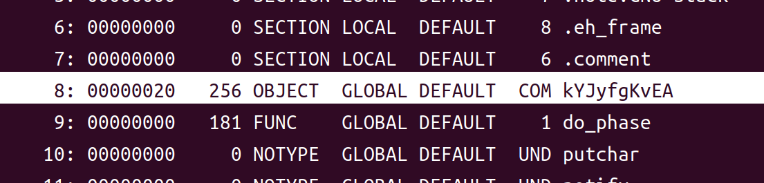
编写汇编代码，将学号压栈。编译为机器语言后，加入do\_phase中。

之后要调用函数，需要我们计算跳转地址。即目标函数起始地址-调用函数语句下一条语句的起始地址，再这里就是

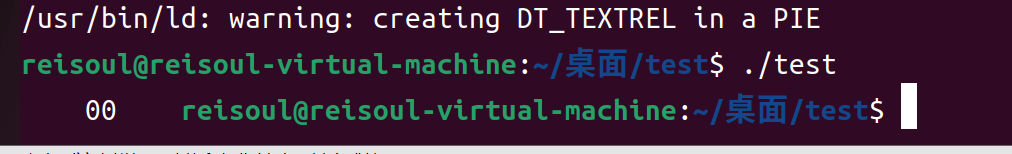
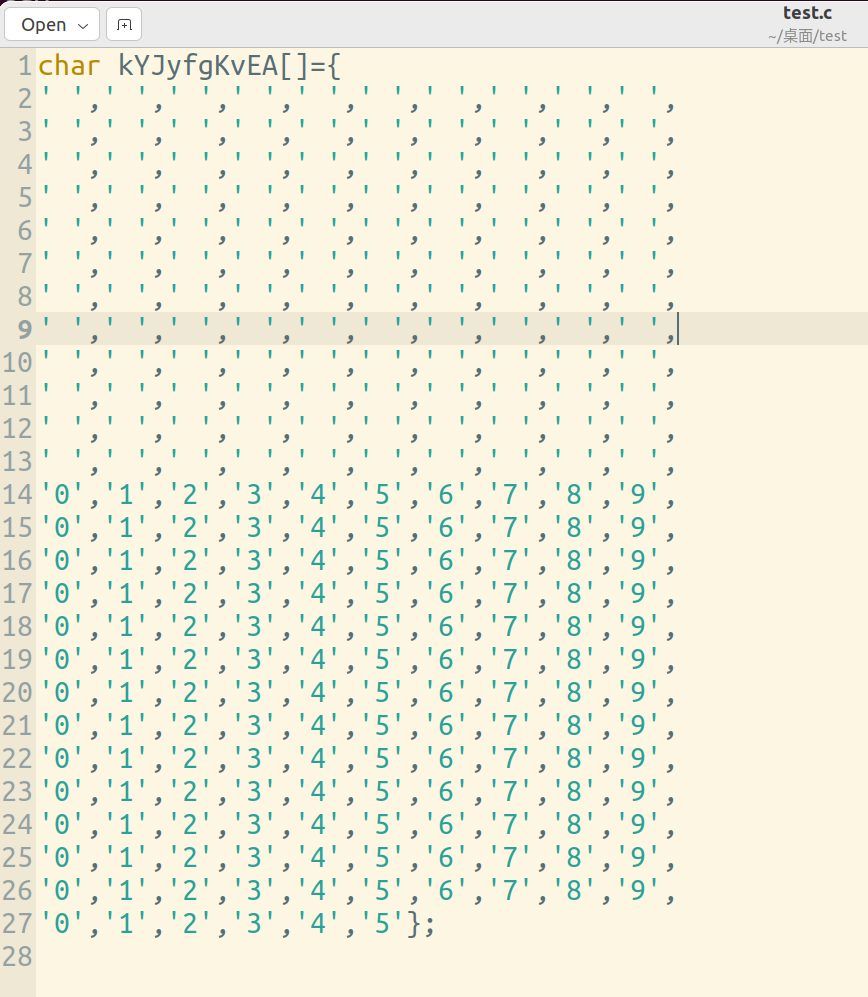
1c2b-1c7b=ffff ffb0

按上述修改后在链接，反汇编，发现已经正确调用了目标函数。运行成功完成阶段二。

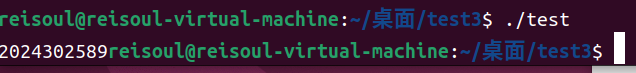
阶段三：新建一个phase3\_patch.o，使其与main.o和phase3.o链接后运行时能输出自己的学号。

查看链接main.o和phase3.o后的符号表：

COM表示未定义，是弱符号。说明我们需要在phase3\_patch.o里定义一个新的同名数组，并把它初始化为学号，成为强符号。这样链接后就会用强符号覆盖弱符号，输出学号。

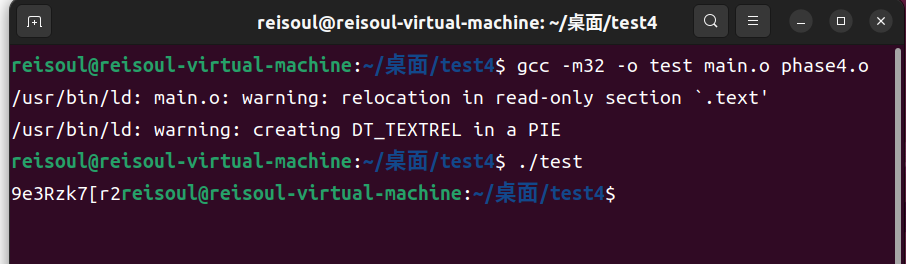
已知数组大小为256，我们先随意定义一个同名数组。

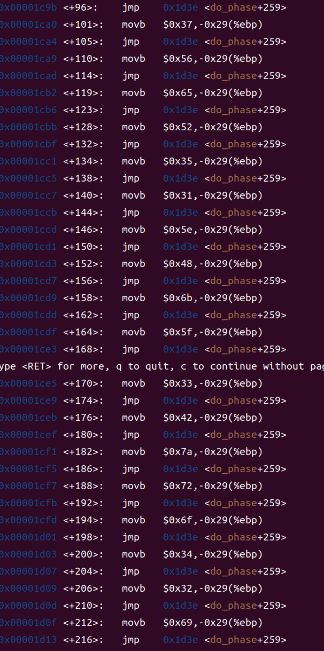
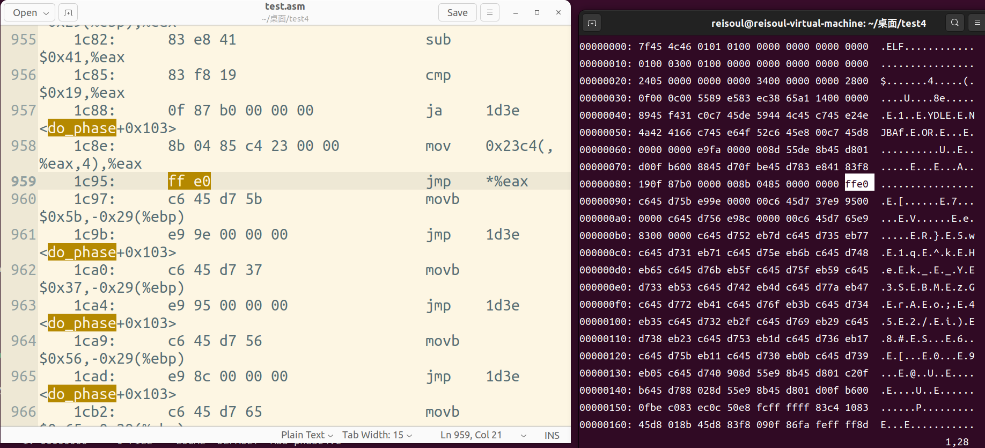
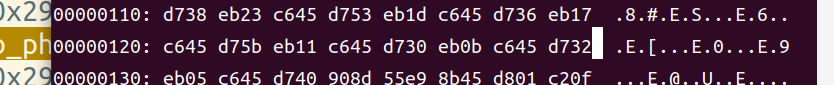
然后链接运行。发现数字是乱序的，那么我们需要找出对应关系，替换掉特定的字符。这里我采用试错的方法，将数组定义为42个大写英文字母和小写英文字母的重复组合，这样可以较快地定位哪里需要替换。

我们有恒心有毅力，细心查找，终于完成了字符的替换。

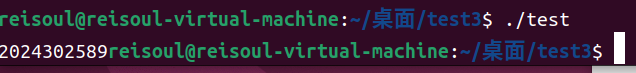
第三阶段成功！

第四阶段：修改phase4.o相应节中的内容，（不能修改.text节），使其与main.o链接后输出学号。

还是先直接链接并运行：

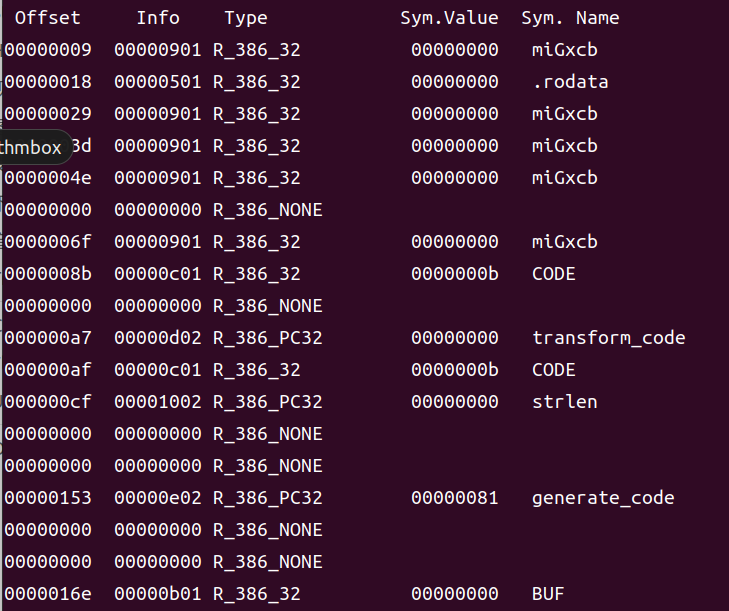
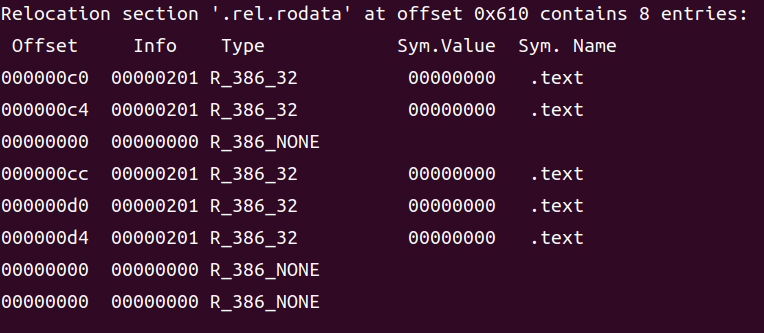
还是输出乱序字符串。

查看反汇编代码可得，这是一个switch语句，有多次跳转。而switch语句的格式都是相同的，如右图，都是c6 45 d7（也就是E.）后面跟上要打印的字符。那么我们仍然可以在vim里编辑。通过试错法修改数据，把乱码字符串变成我们想要的结果。

我们有恒心有毅力而且细心，终于完成了所有字符的替换，输出了正确的学号，完成了第四阶段。

第五阶段：补充phase5.o重定位节中被清零的重定位记录，使其与main.o链接后能够正确输出学号编码后的字符串。

我们用readelf -r phase5.o查看一下.rel.text节，发现有些重定位条目出现了空缺：

重定位的条目的相对位置不变，只是把有的重定位条目清零了。我们要做的就是修改重定位节内容，能够正确链接。

经过理论学习，我们知道重定位就是把符号引用和符号定位关联起来的过程。重定位节包含重定位过程需要的各种信息。重定位节的每个条目对应一个需要重定位的位置。

从表中我们可以得知一下信息：

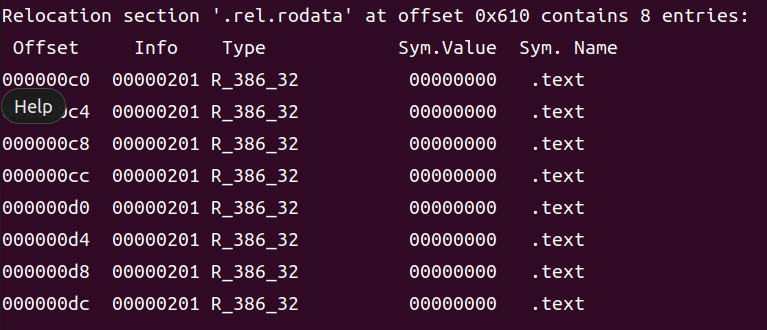
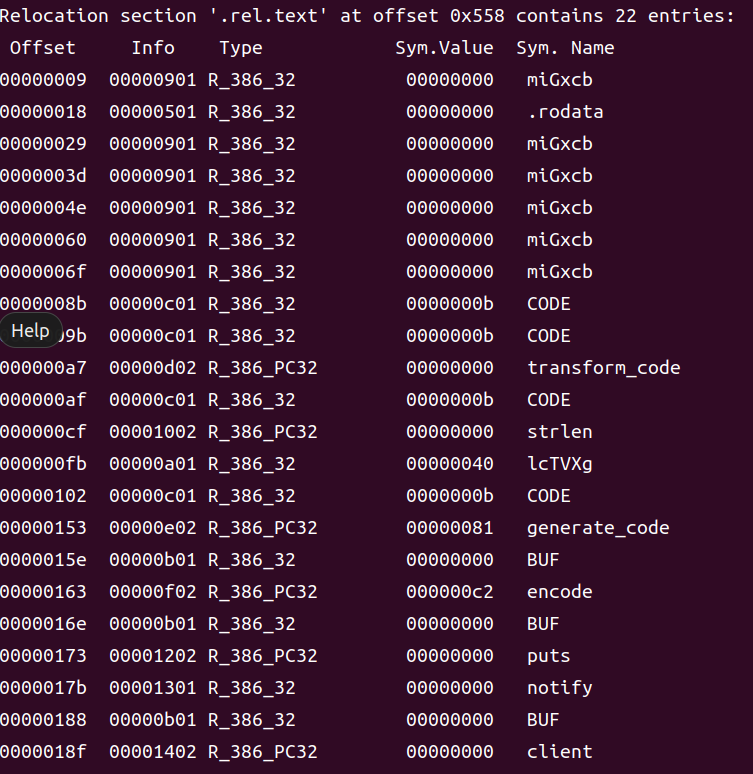
r\_offset: 需要重定位的位置（节内的偏移）

r\_info: 包括两部分：sym(占24位)和type(占8位)）

sym：重定位到哪个符号（符号表索引）

type：重定位类型（如何计算重定位地址）

那么我们的思路就已经确定了，首先要确定offset，知道哪些地方的重定位条目被清零了；然后确定需要重定位到哪个符号；最后确定type of relocation，即重定位的类型。

我们打开vim，经过以上步骤，把缺失部分的代码补齐可得：

成功输出，阶段五完成！

1. **实验总结:**

通过lab4，我对linux的操作系统有了更加深入的了解，掌握了readelf等指令和vim的操作方式。通过五个阶段的实验，我对程序的链接有了更加深入的理解。虽然几个阶段的实验难度都很大，但是分别让我掌握了查看修改数据，编写汇编代码和替换指令，强弱符号的原理，switch语句的重定位和重定位条目的分析，让我对计算机系统基础理论的理解更上一层楼，极大地锻炼了我的动手能力和探索能力。