**题目：第7题：在Linux中下修改一个现有的elf可执行程序**

**摘要：**

本文实现了如下功能：让某elf程序运行后先执行一个特别的附件功能（附加功能是：创建一个指定文件写入hello，world的字符串）后再继续运行该elf程序，否则程序结束。并且附加功能嵌入到了原来的elf程序中。

实现思路是：

1. 先生成一个 elf 程序，并通过 FILE \*fp = fopen("test", "rb+") 读取该 elf 文件
2. 接着获取 elf 文件的入口地址，程序头，节区头等信息
3. 然后写出要添加的附加功能的汇编程序并编译获取机器码，通过 readelf -e elf文件名 查看 elf 文件的程序入口地址等信息，并根据 elf 文件的程序地址等信息修改生成的机器码中相应的地址信息
4. 将修改好的机器码添加进elf的text段的最后（为了方便，此处我将数据也一并添加进了text段）
5. 修改elf 的入口地址为嵌入的程序首地址，当嵌入程序执行结束再跳转到原 elf 文件的首地址，使程序继续执行原本的 elf 文件程序，再修改 elf 的程序头，节区头
6. 通过 fclose(fp) 关闭 elf 文件

**关键词：**

生成机器码，读取elf文件，修改elf文件，elf代码嵌入

**姓名--学号--班级：**

1. 刘俊傲--U201617047(软工1603)
2. 任抒怀--U201617052(软工1603)
3. 周振宇--U201617064(软工1603)

**作业概述：**

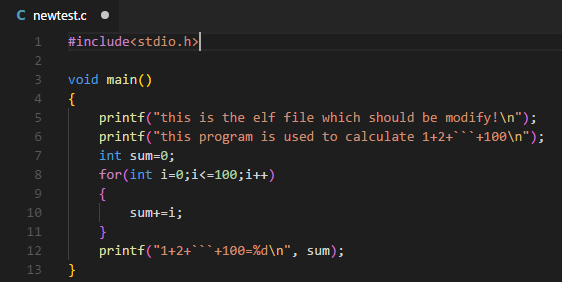
让一个elf程序运行后先执行一个特别的附件功能（附加功能是：创建或打开一个指定文件写入helloworld的字符串）后再继续运行该elf程序，否则程序结束。并且附加功能嵌入到了原来的elf程序中。

实现思路是：

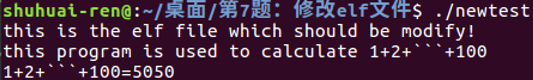
1. 先生成一个 elf 程序，并通过 FILE \*fp = fopen("test", "rb+") 读取该 elf 文件。
2. 接着获取 elf 文件的入口地址，程序头，节区头等信息。
3. 然后写出要添加的附加功能的汇编程序并编译获取机器码，通过 readelf -e elf文件名 查看 elf 文件的程序入口地址等信息，并根据 elf 文件的程序地址等信息修改生成的机器码中相应的地址信息。
4. 将修改好的机器码添加进elf的text段的最后（为了方便，此处我将数据也一并添加进了text段）。
5. 修改elf 的入口地址为嵌入的程序首地址，当嵌入程序执行结束再跳转到原 elf 文件的首地址，使程序继续执行原本的 elf 文件程序，再修改 elf 的程序头，节区头。
6. 通过 fclose(fp) 关闭 elf 文件
7. **功能需求：**

在linux系统下修改一个现有的elf可执行程序。

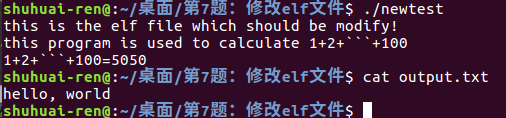
本次实验要修改的elf程序为newtest：



其原功能为打印相关字符串，并对1-100进行求和。若直接运行效果应该如下所示：



现要求对该elf程序注入代码，使之运行后首先执行一个特别的附加功能。这里的附加功能是指创建一个指定文件output.txt，向其写入“hello，world”字符串。注入代码成功后运行newtest程序应该看到如下结果：



在写入“hello，world”字符串之后应继续运行newtest程序，否则程序结束。

要求附加功能要嵌入到原来程序中，不能是独立程序。

1. **设计思路：**

（1）要在elf程序中嵌入程序，首先编写newtest.c代码，并生成一个elf程序（这里是newtest），并打开该elf文件。

（2）打开后获取文件开始位置的elf头。

（3）通过获取的elf头部e\_header，进而获取到elf头内确定的程序入口地址entry、程序头部的偏移量phdr、节区头部的偏移量shdr。

（4）通过获得的偏移量，记录在elf头的程序信息，表项数量和表项大小，计算出程序头部与节区头部的偏移地址。

（5）通过偏移地址获取到程序头p\_header，节区头s\_header。

（6）用汇编编写要添加的附加功能（hello\_world.s文件），因为之后要编译成机器码。

（7）在附加功能的最后用跳转语句跳转到elf程序的原首地址，以实现执行完附加功能后，原elf程序正常执行。

（8）为了将代码要嵌入到elf程序中，需要把附加功能代码转换成机器码，存在code[]中。

（9）计算嵌入程序的代码长度，嵌入数据的长度，便于之后修改信息。

（10）找到可运行的段，（elf程序的text段，为了方便把数据也一起加进去了）把要嵌入的代码写入到段的末尾，因为对应着未写满的节区，留有可写空间。

（11）修改程序的入口地址，以实现程序运行时首先执行我们嵌入的附加功能。

（12）嵌入了代码之后，要对程序头部，节区头部的一些信息进行改动：

对于程序头部，要修改对应嵌入段的节区总长，和该段对应的内存大小。增加的长度通过之前计算的机器码长度得到。

对于节区头部，要修改节区头记录的节区大小。

（13）嵌入和相关修改完成，关闭elf程序。

（14）以上elf文件修改及代码注入工作由optional2.c文件实现。

1. **开发环境下载，安装和配置：**
2. 使用原生Ubuntu 16.04系统。

请到<https://cn.ubuntu.com/download/> 下载64位原生Ubuntu 16.04系统



本小组成员的实验环境：

处理器：Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz 2.40GHz

内存：4.00GB

操作系统：Ubuntu16.04LTS

（2）剩下的文件编译执行工作使用系统终端中的gcc等命令即可完成。

1. **程序的难点和核心技术分析：**

**难点一：**

了解elf程序的结构特点，elf程序是如何运行的，各部分的功能是什么，各部分的如何相互作用影响，以及如何达到我们的目的。

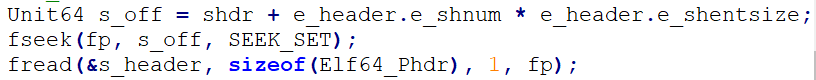
****

我们了解到，elf头部标明程序头部和节区头部的信息，位于elf文件的开始，要获得进一步的信息，首先要获得elf头部：

**1**

然后可以从获得的elf头部读取我们需要的信息：程序入口地址，程序头偏移量，节区头偏移量。

要注意的是偏移量不代表就是偏移地址了，这也是难点之一。为了计算偏移地址，我们使用偏移量加上elf头部的大小。

****

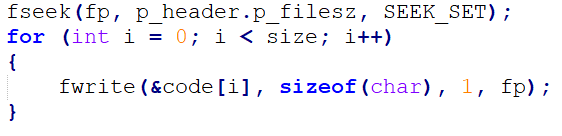
得到偏移地址后就可以得到需要的程序头部，节区头部了。

程序头部和节区头部都是结构数组。程序头部的每个单向指示了可执行的目标文件在运行时的一个段，因此是描述目标文件动态状态下的内部结构。

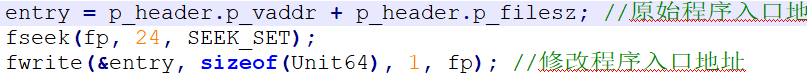
一个段可包含多个节区，节区占用文件中的一个连续字节区域，包含了目标文件的所有信息。节区头部就是描述节区的位置以及属性的。

因此，我们获得了程序头和节区头后，就可以开始嵌入代码了。

定位到可执行段的末尾，写入我们已经写好的嵌入代码：

****

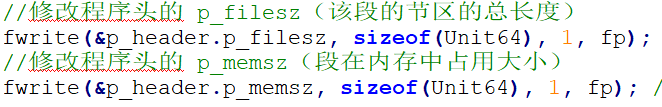
写入过程完成，接下来就是修改elf程序记录的相关信息了。

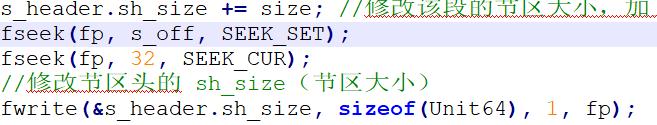
修改程序入口地址使elf程序运行时首先执行附加功能：

因为节区长度增加，因此节区头部偏移量改变，需要修改：

5

紧接着修改程序头，修改该该段的节区总长度，内存大小，都加上嵌入代码的长度：  
6

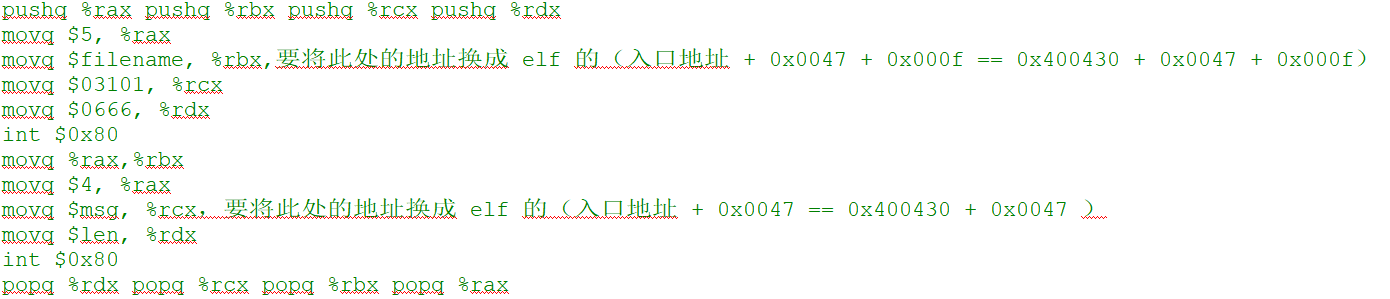


然后修改节区头部，修改对应的节区大小：  


至此，嵌入工作和修改工作都已完成。

**难点二：**

被注入的代码hello\_world.s需要使用汇编编程。汇编语言在linux下的不同语法标准对我们而言有些陌生，我们本身对于汇编的熟悉也不够。以下代码来自hello\_world.s。

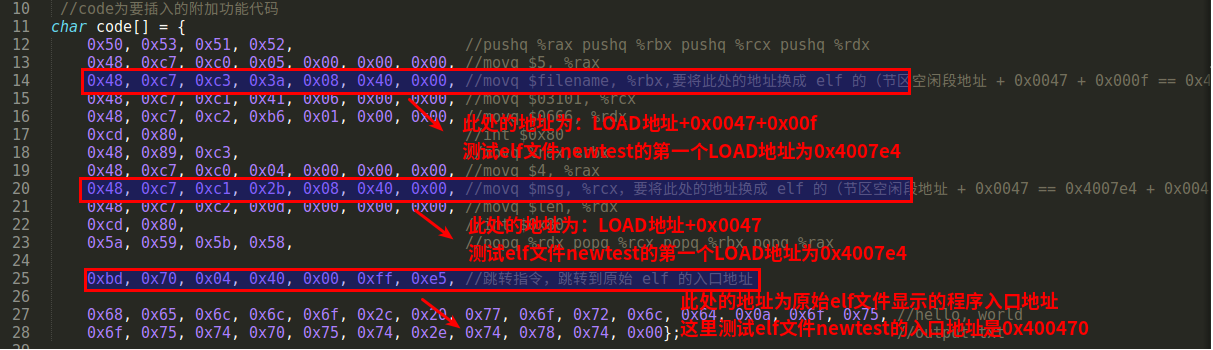


结尾要跳转到原elf程序的位置，这就需要事先readelf-e elf文件名，查看elf文件的程序入口地址等信息，填入到嵌入代码中

同样要嵌入的还有程序所需的数据：  
10

**难点三：**

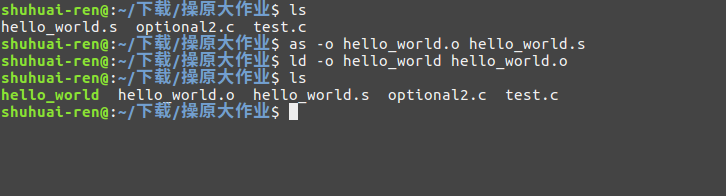
注意被注入的机器码要根据测试 elf 文件的信息修改相关地址信息，以使附加功能代码能被正确添加到测试 elf 文件之中：



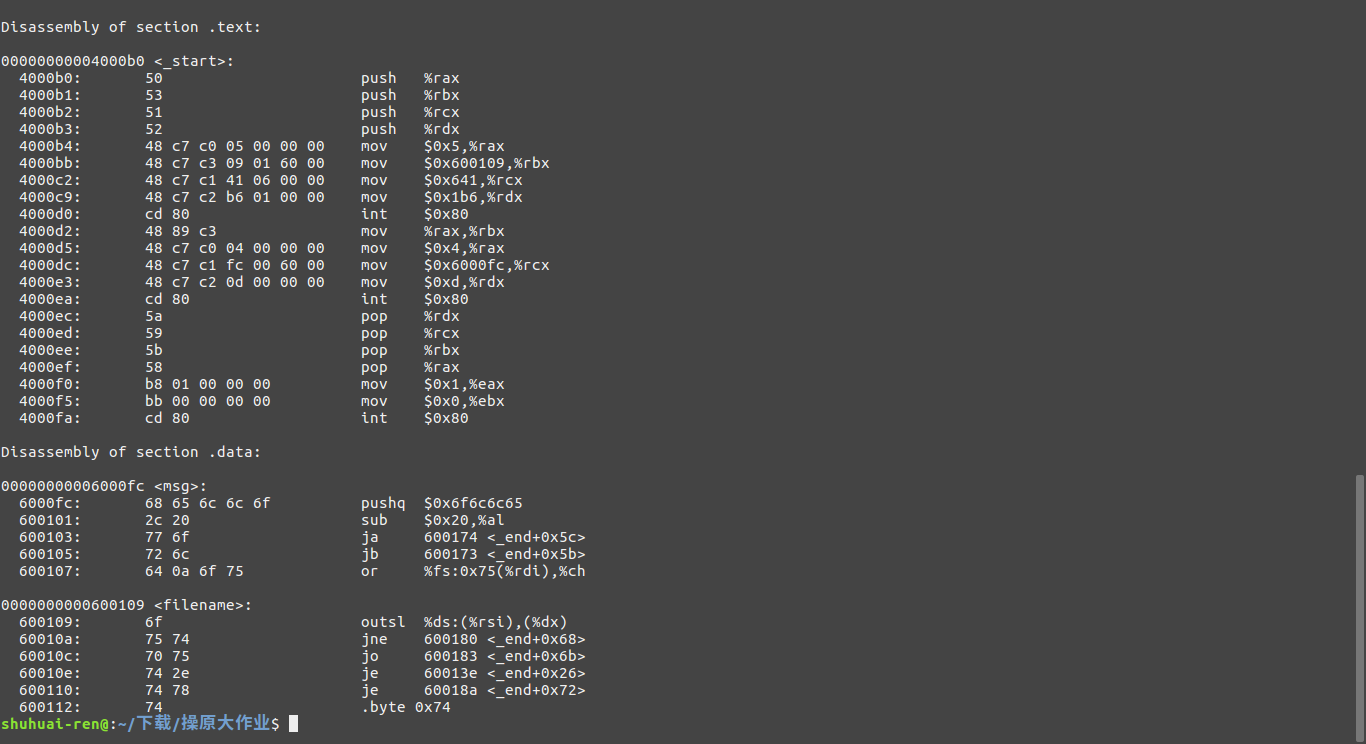
1. **运行和测试过程：**

1.首先通过下面两条指令编译，链接 hello\_world.s 汇编代码

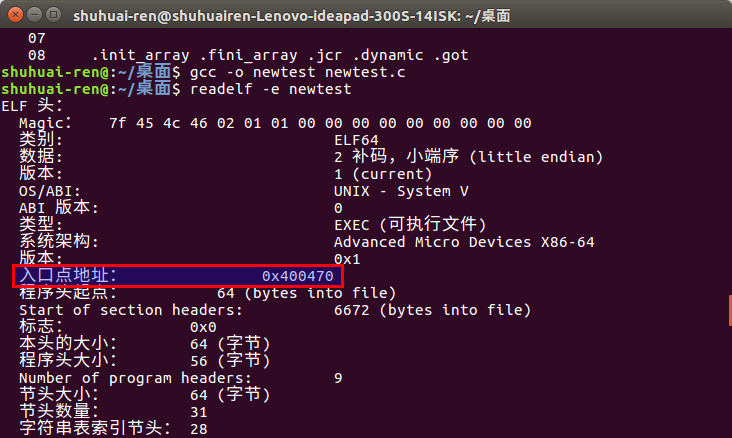
|  |
| --- |
| as -o hello\_world.o hello\_world.s |
| ld -o hello\_world hello\_world.o |

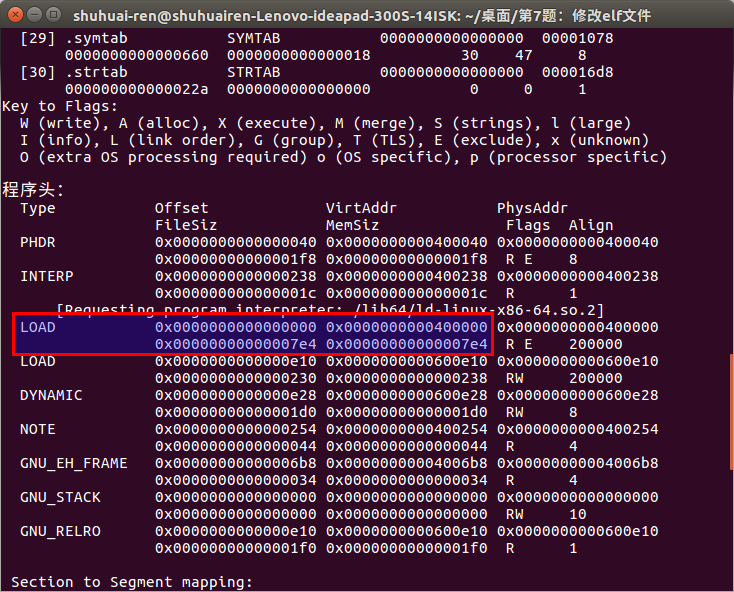


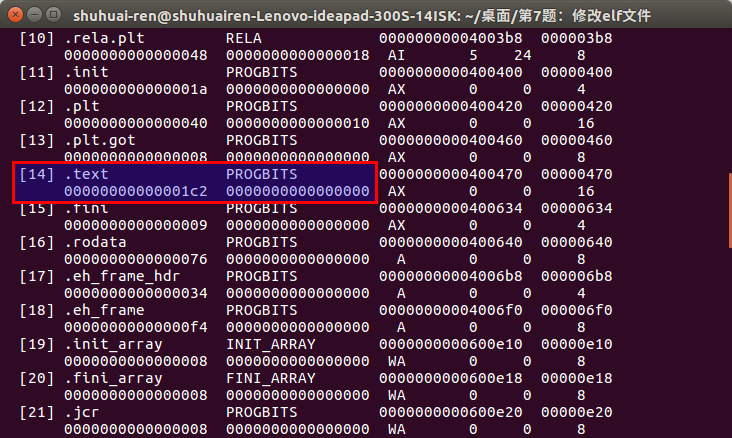
2.使用 objdump -D hello\_world 查看生成的 hello\_world 的机器码等详细信息（也可自己手写机器码，这里是通过汇编生成的机器码）



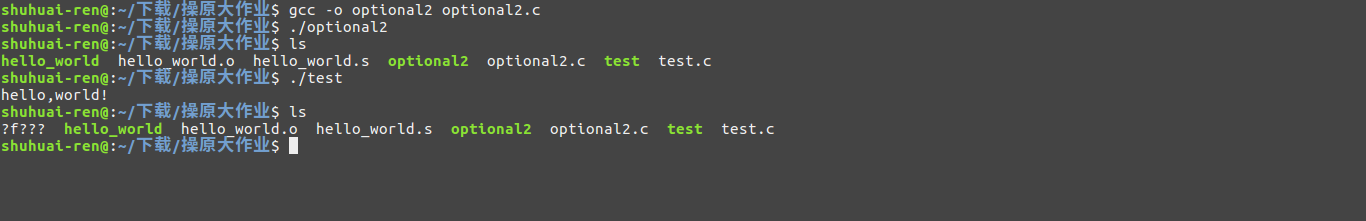
3.使用 gcc -o newtest newtest.c 生成用来测试的elf可执行文件，并通过 readelf -e newtest 来查看程序入口地址、LOAD段地址、text段地址等详细信息

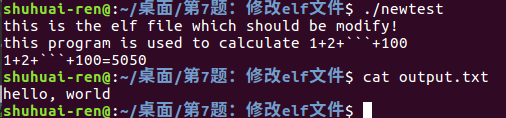


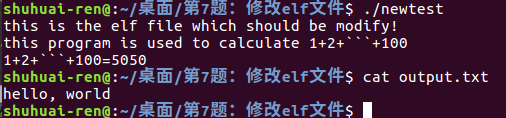




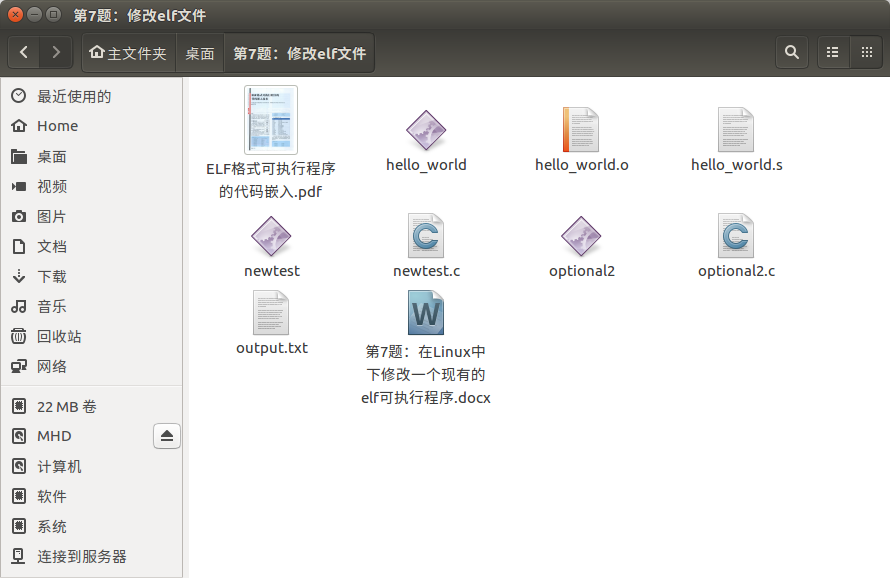
4.使用 gcc -o optional2 optional2.c 生成修改 elf 文件的程序optional2，然后先运行 ./optional2完成对newtest的修改，再运行 ./newtest ，就会发现同级目录下生成了一个文件output.txt（由于第一版的机器码地址计算有误，导致生成的文件乱码，改正后如第二张图片所示）



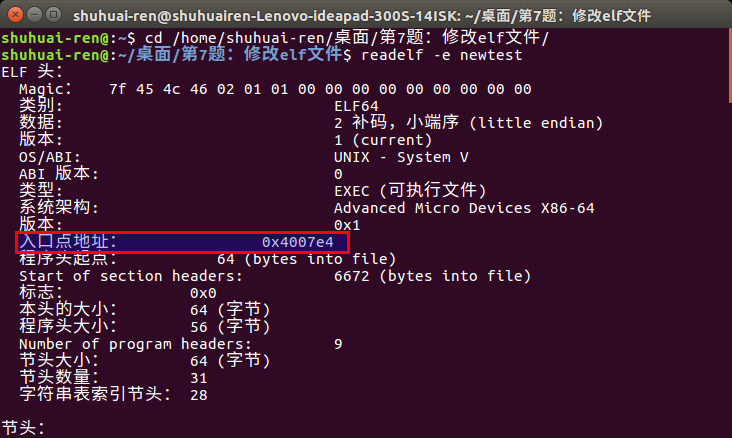
运行完./newtest后的结果如下：

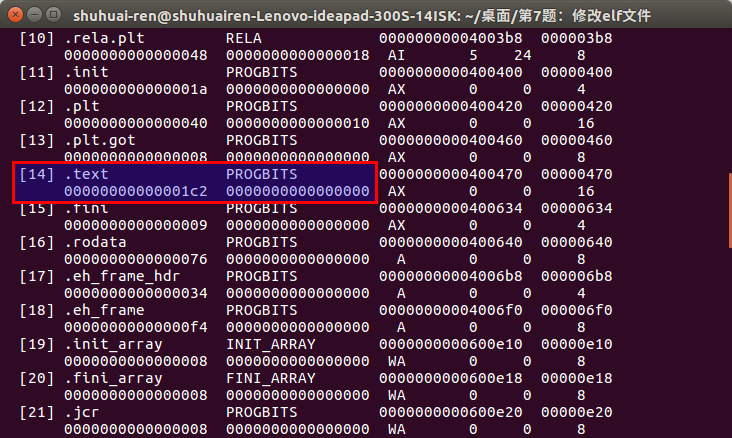


同级目录下生成了output.txt的文件，里面是写入的“hello,world”字符串（从上面的cat output.txt命令中可以看出）。



5.接着，再使用readelf -e newtest 命令查看此时的newtest可执行文件，会发现此时的程序入口地址已经被修改，但 text 段的入口地址并未被修改

程序入口地址由原来的0x400470变为现在的0x4007e4：

text段地址不变：

至此，对elf文件newtest的修改全部成功！

1. **软件使用说明：**

运行环境：**原生**Ubuntu 16.04系统。

**注意**：运行该程序必须要在原生的 ubuntu 系统上进行，因为系统的改变，可能导致指令的机器码也会相应变化，导致出现段错误，就比如跳转指令，在 ubuntu/kali 就会报错

optional2为修改elf文件的程序，newtest为被修改的elf文件。

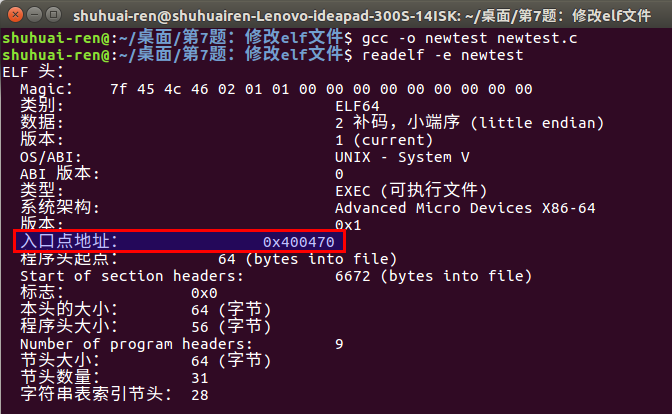
程序运行前请先进行地址信息检查：

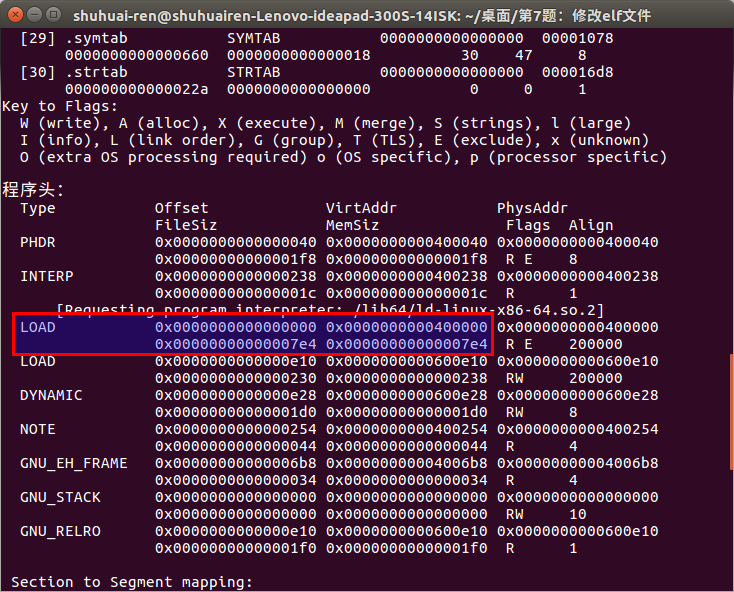
1. 首先通过

gcc -o newtest newtest.c

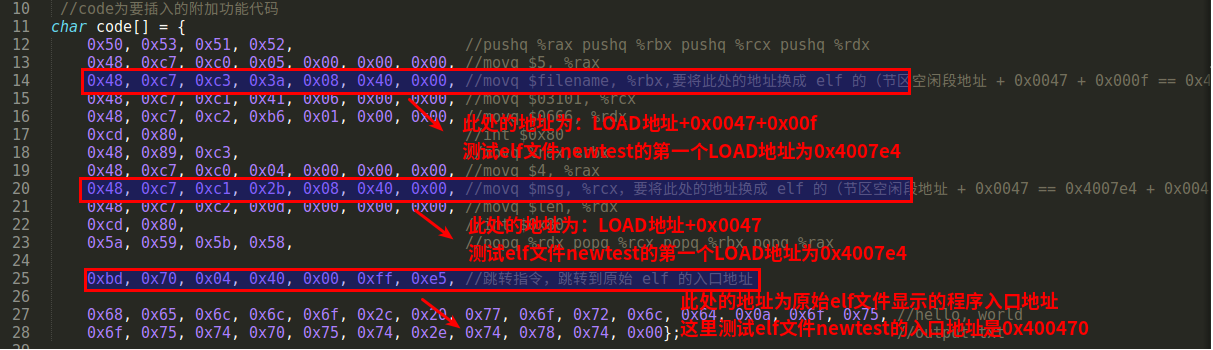
readelf -e newtest

查看测试elf文件newtest的详细信息，看看程序入口地址是否是 0x400470，第一个 LOAD 地址是否是 0x4007e4。：





1. 如果程序入口地址和第一个 LOAD 地址不如上所述，则需要修改 optional2.c中机器码code[]中的相关地址信息，相应的计算方法如下所示：



1. 如果 newtest文件已经生成，且optional2也执行过一次，则代表 test 已经被修改过，如果想再重新测试一遍，则必须 gcc -o newtest newtest.c 再重新生成 newtest 文件。
2. **编程参考网址：**
3. [Linux 汇编语言开发指南](https://www.ibm.com/developerworks/cn/linux/l-assembly/)
4. [linux下汇编语言开发总结](https://blog.csdn.net/q_l_s/article/details/52722667)
5. [Linux 汇编器：对比 GAS 和 NASM](https://www.ibm.com/developerworks/cn/linux/l-gas-nasm.html)
6. [自己动手写病毒—ELF文件病毒](https://blog.csdn.net/luojiafei/article/details/7225818)
7. [Linux及安全实践四——ELF文件格式分析](https://www.cnblogs.com/lxq20135309/p/5551658.html)
8. [ELF文件病毒的分析和编写](https://blog.csdn.net/luojiafei/article/details/7206063)
9. [ELF格式可执行程序的代码嵌入](ELF格式可执行程序的代码嵌入.pdf)