**关于实验五的一些说明**

1. **安装 hexedit : apt install hexedit, 虚拟机环境下也是可以安装成功的。本次实验需要使用这个工具修改.o文件**

**最简用法：hexedit filename**

**在 hexedit 中，F1 ：显示帮助信息； 保存修改结果：F2； 退出不保存：ctrl+c**

**其他：查看帮助：hexedit –help**

**2. 实验操作说明**

**提供的实验包含有 main.c、phase1.o、phase2.o、phase3.o 、phase4.o、phase5.o、phase6.o、phase7.o,其中只有phase6.o是未加-m32生成的64位目标文件，其余全部为32位目标文件。以生成32为可执行程序 linkbomb 的方法为例：**

**#gcc -c -g -m32 main.c -o main.o**

**#gcc -no-pie -m32 -o linkbomb[n] main.o phase[n]\*.o n=1、2、3……**

**例 第二关： gcc -no-pie -m32 -o linkbomb2 main.o phase2.o**

**但是第六关：gcc -no-pie -o linkbomb6 main.o phase6.o phase6\_patch.o（phase6\_patch.o为自己编写的代码生成）**

**使用 readelf、hexdump、od、hexedit 等工具阅读和修改目标文件。**

**另外，实验包中有 phase0.c ，通过该程序，可了解 phase[n].c 的大致写法。**

**3. 实验五的7关提示**

**第二关到第四关要调用函数，由于是直接改写的.o文件加入了对函数的调用指令，但是这个函数的调用在.o文件中并没有重定位信息，因此链接时链接程序不会改写call指令中的位移量，需要自己去计算位移量并写进call指令。**

**第五关很简单，只要修改重定位信息中的符号编号即可，具体如何查看符号编号，见本文件第4(4)。**

**第六关，只能在64位环境下完成,此时需要把main.c重新编译一下（不带 -m32），然后和phase6.o链接。注意分析观察main.o中phase变量的代码和phase6.o中myprint代码在具体使用中的差异，判断myprint该给个什么样的定义和声明。**

**4. 如何查看.o文件的相关信息。**

**下面的程序保存在exp1.c文件中，采用命令：**

**gcc -c -m32 exp1.c -o exp1.o**

**编译后生成exp1.o.下面查看expl.o文件的有关信息。**

**#include <stdio.h>**

**int global\_value = 100, global\_value1 = 200;**

**int main()**

**{**

**int x, y;**

**x = global\_value;**

**y = global\_value1;**

**printf("Sum of global\_value and global\_value1 is %d\n", x+y);**

**return 0;**

**}**

**（1）查看节头表命令：readelf -S exp1.o**

电脑屏幕的照片上有文字

中度可信度描述已自动生成

**图1 exp1.o的节头表**

**从中可以看到数据节在exp1.o中的起始位置为：偏移值为0x98，占用4个字节，代码节从exp1.o的起始位置为：偏移值为0x3c，占用5a字节。没有数据节的重定位节，也就是没有 .real.data。因为数据节中没有需要重定位的内容。换句话说，需要重定位的是 global的地址，而不是地址对应的单元中的内容。用hexedit exp1.o查看如图2所示。其他节的信息也可以用相同的方法查看。**

电脑萤幕

中度可信度描述已自动生成

**图2:exp1.o文件中.text节和.data节的起始位置（其他节也可参考节头表来看），最左边一列为到文件起始位置的偏移值**

**(2)查看数据节中的内容：readelf -x .data exp1.o 显示信息如图3所示。**

文本

描述已自动生成

**图3 数据节的信息，最左边一列是相对数据节偏移量，即数据节存放了两个变量的初始值，第一个变量是global\_value，存放在数据节的开始。**

**(3) 查看代码节中的内容：readelf -x .text exp1.o，显示结果如图4所示。**

图片包含 日历

描述已自动生成

**图4 代码节的内容**

**可以直接看反汇编代码来佐证。可以看到上图的机器码串，与下图5的机器码串是完全对应的。同时从图的提示信息中看到“This section has relocations”,即该节有重定位节。**

日历

描述已自动生成

**图5 将exp1.o反汇编得到的结果：objdump -d exp1.o**

**(4) 查看符号表节 （readelf -s homework.o）  
看节头表是大写的 S，看符号表节，用的是 小写 s**

文本

描述已自动生成

**图6 查看符号表节，global\_value和global\_value1是符号表节中的第5和第6个符号。**

**(5) 查看重定位节: （readelf -r exp1.o）**

**显示结果如图7所示**

电脑屏幕的照片上有文字

中度可信度描述已自动生成

**图7 exp1.o中的重定位信息**

**该命令会列出 exp1.o 中所有的重定位节。从中可以看到 有代码节的重定位节 (.rela.text) , 而没有 数据节的重定位节。  
 在该重定位节中，表名， 在地址 为 0000001e 处，需要重定位，（可以在 objdump -d -r exp1.o 的界面中，看到 从 1e 处开始 有00 00 00 00； 这四个字节的00 链接后会替换为符号表（图6）中编号为5的符号的相关寻址方式（不同的编译开关重定位方式不同）去寻找global\_value 的地址。**

**核心信息：什么位置要重定位? （0000001e）  
 由哪个符号来重定位? ( 符号的索引号是 0005, 从上面的符号表节，看到5号符号就是 global\_value )   
 用什么方式来定位 ？ （09， 即 R\_386\_GOTOFF, 即是一个 32位（四个字节）的地址，相对于GOT表的查找转变成形为x(%ebx)的寻址方式，如图8所示,这种符号的重定位类似于参考教材“动态链接——模块内数据的引用”的做法（但不等价），有兴趣的同学可以参考一下。**

**链接时，连接程序会根据重定位节的信息，以及给符号分配的地址（代码、数据都合并抄好后，也就可以确定符号的地址了，动态库函数除外），来确定 text 节中相应位置应该填写什么值，即重定位了！**

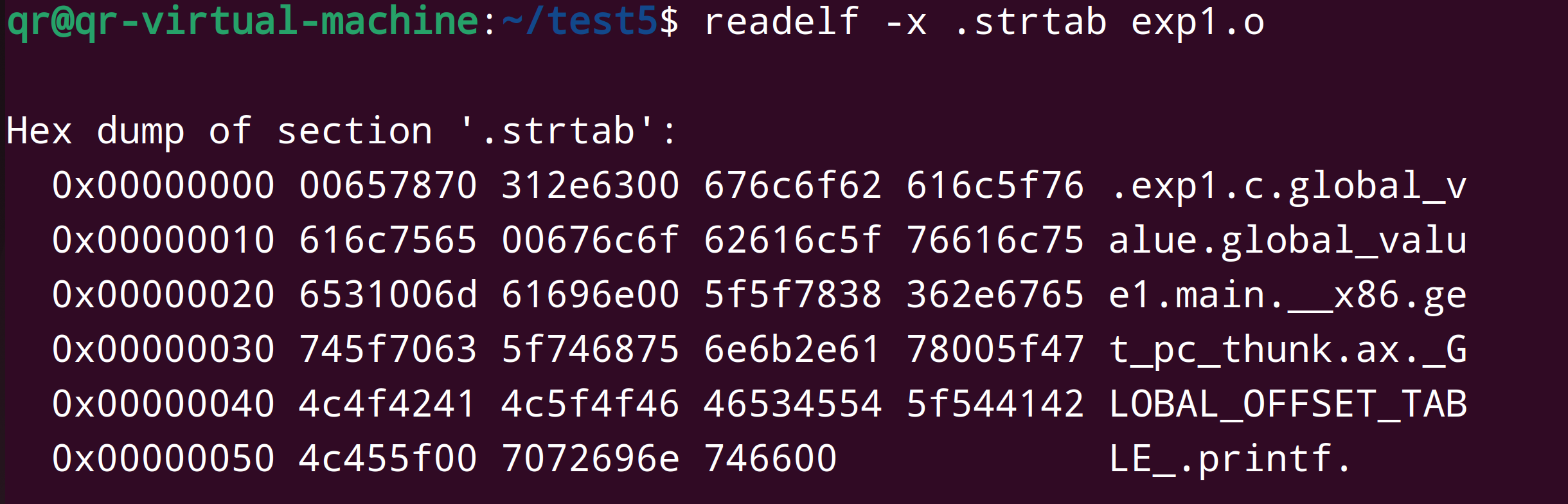
电脑屏幕的照片上有文字

描述已自动生成

**图8 链接后的信息**

**(6)字符串节 （readelf -x .strtab exp1.o）**

**从表中，可以看到 符号的名字串。**



**图9 符号的名字串.strtab节**

**通过细致的观察，同学们对地址、地址对应的存储单元中的内容是否有更好的理解？理解编译器（或者汇编程序）是如何工作的核心要点：在生成 .o文件时，哪些信息是确定的？哪些信息是不能定的？不能确定的是符号的地址（不能确定全局变量的地址，不能确定静态局部变量的地址；不能确定函数的地址。这些不能确定地址的名字，称之为符号）。凡是要用到符号地址的地方，由于编译时不能确定符号的地址值（或者地址相对值），也就导致相应的引用之处无法填上准确的值，只好先用对应长度的 0来占位，并且要记录下来，哪些地方是要重定位的。只有在链接时，把代码、数据都合并好后，才能确定了这些符号的地址，才能确定引用这些符号的地方要填什么值，并且在链接时填上，这就是重定位！**

**最后，同学们要注意的是不同的编译开关看到的重定位信息是不同的。课堂上讲的例子是采用-no-pie -fno\_pic这样的编译开关生成与位置有关代码查看到的。**