链接与 ELF 实验

注意：由于慕课学期较短，本学期只要求完成本实验中的 3 个阶段（即 Phase 1、2、3）。实验数据中包含的剩余阶段可供进一步练习，以加深对相关理论课程知识的理解和掌握。

一、实验介绍

本实验的目的在于加深对程序生成与运行过程中链接的基本概念、作用和 ELF 文件的基本格式组成的理解。实验的主要内容是在二进制层面上逐步修改一个由多个 C 模块组成的程序（称为 “linkbomb”），使其在运行时满足实验指定的行为要求。

本实验包含以下阶段，各阶段考察程序链接与 ELF 文件的不同方面知识：

阶段 1：静态数据与 ELF 数据节

阶段 2：指令与 ELF 代码节

阶段 3：switch 语句与重定位

阶段 4：可重定位目标文件

实验中，你需要完成对二进制可重定位目标文件（.o 文件）特定内容的修改，包括数据内容、机器指令、重定位记录等。在完成每个阶段的修改后，你应该能够链接相关模块得到一个二进制可执行程序 “linkbomb”，该程序应能正常运行并输出期望的正确结果。

- 实验环境：Linux i386
- 实验语言：汇编

二、实验数据

在本实验中，每位学生应从下列链接下载包含本实验相关文件的一个 tar 文件：

http://cs.nju.edu.cn/sufeng/course/mooc/0809NJU064_linklab.tar

可在 Linux 实验环境中使用命令 “**tar xvf 0809NJU064_linklab.tar**” 将其中包含的文件提取到当前目录中。该 tar 文件中包含如下实验所需文件：

- main.o：主程序的二进制可重定位目标模块，实验中无需且不应该修改
- phase1.o, phase2.o, …：各阶段实验所针对的二进制可重定位目标模块

三、实验结果提交

因慕课平台对文件提交和处理的功能限制，**本实验无法采取课程视频最后说明的上传实验结果文件的提交与评分形式，因而改为通过相应单元测验的题目对实验完成情况进行考查**。

四、实验内容

实验的每一阶段 n 中，按照阶段的目标要求修改二进制可重定位目标模块 phase[n].o 后，可使用如下命令生成可执行程序 linkbomb：

```
linux> gcc -no-pie -o linkbomb main.o phase[n].o
```

并且，如下运行链接生成的可执行程序 linkbomb，应输出符合该阶段目标的字符串。

```
linux> ./linkbomb
```

提示：

- ✓ main.o 中会调用相应模块 phase[n].o 中如下定义的一个全局函数指针变量 phase：

```
void (*phase)() = do_phase;
```

- ✓ 其中，作为其初始值的"do_phase"是各阶段模块中实现的一个全局函数，用来完成该阶段的具体功能。
- ✓ 注意：在设计上，各阶段目标模块 phase[n].o 中的程序链接后只会引用本模块中（和 C 标准库中）定义的符号，但注意本模块中相应符号未必定义为 local 的。因此，在分析模块中的符号引用时（除对标准库函数如 puts 的引用以外），可只在本模块中寻找对应的符号定义（例如参考本模块中的符号表信息和重定位记录）。
- ✓ 各阶段之间没有相互依赖关系，可按任意阶段顺序进行实验。

下面针对具体每个实验阶段，分别说明实验需要达到的目标。

Phase 1

修改二进制可重定位目标文件 “phase1.o” 的**.data 节**的内容（注意不允许修改其它节的内容），使其如下与 main.o 模块链接后运行时输出（且仅输出）本慕课号 “**0809NJU064**”：

```
linux> gcc -no-pie -o linkbomb main.o phase1.o
linux> ./linkbomb
linux> 0809NJU064
```

提示：

- ✓ 可查看反汇编代码，获得输出函数的调用参数的地址，按照目标输出内容，修改该参数在 phase1.o 文件的数据节中的相应内容
- ✓ 可使用 hexedit 等工具或自己编写程序实现对二进制 ELF 文件的编辑和修改

Phase 2

修改二进制可重定位目标文件 “phase2.o” 的**.text 节**的内容（注意不允许修改其它节的内容），使其如下与 main.o 模块链接后运行时输出（且仅输出）本慕课号 “**0809NJU064**”：

```
linux> gcc -no-pie -o linkbomb main.o phase2.o
linux> ./linkbomb
linux> 0809NJU064
```

提示：

- ✓ 可查看反汇编代码，定位模块中的各组成函数并推断其功能作用，进一步修改入口函数 do_phase()中的机器指令（可用自己指令替换函数体中的 nop 指令）以实现所需输出功能
- ✓ 注意：模块中的函数并不都是完成该阶段目标所必需的——有些函数是不相关的、不必修改

Phase 3

修改二进制可重定位目标文件 “phase3.o” 中相关节的内容（注意不允许修改.text 节和重定位节的内容），使其如下与 main.o 链接后运行时输出（且仅输出）本慕课号 “**0809NJU064**”：

```
linux> gcc -no-pie -o linkbomb main.o phase3.o
linux> ./linkbomb
linux> 0809NJU064
```

提示：

- ✓ 该模块针对一个 COOKIE 字符串（由大写英文字母组成，每个字符互不相同，且总长度与慕课号字符串相同）中每一字符逐一进行变换处理
- ✓ 要完成本阶段，需熟悉 switch 语句的机器表示和特定重定位类型的处理算法

Phase 4

修改二进制可重定位目标文件 "phase4.o"，恢复其中被**人为清零**的一些**重定位记录**（分别对应于本模块中需要重定位的符号引用，注意不允许修改除重定位节以外的内容），使其如下与 main.o 链接后，运行所生成程序时输出对本慕课号进行编码处理后得到的一个特定字符串 "**?K?b[Sl?Q@**" ：

```
linux> gcc -no-pie -o linkbomb main.o phase4.o
linux> ./linkbomb
linux> ?K?b[Sl?Q@
```

注意：如果你对重定位信息的修改不正确或不完整的话，如上链接（往往不会报错）后运行生成的 `linkbomb` 程序可能会得到以下结果之一：

- ■ `"Segmentation fault"`出错信息 —— 可能原因？请对照查看机器代码思考"如果未对相关引用进行必要的重定位会发生什么？"
- ■ 类似`"Welcome to this small lab of linking. To begin lab, please link the relevant object module(s) with the main module."`的输出信息，提示模块未正确链接 —— 原因：虽然按上述步骤在生成 `linkbomb` 程序时实际已链接进本模块，但某些重要的重定位记录未正确恢复
- ■ 输出不正确编码结果，即输出的字符串不同于前面给出的特定字符串 —— 原因：未能完整恢复所有重定位记录

提示：

- ✓ 实验中共需要对模块中的 **7** 个被置为全零的重定位记录进行恢复，注意这些重定位记录可能位于目标文件的不同重定位节中
- ✓ 查看课程内容中相关介绍，了解重定位记录的组成结构及其在 ELF 文件中的位置，以便构造符合规范的二进制重定位记录数据，并用其替代 ELF 模块 phase4.o 中合适位置上的被清零的重定位记录
- ✓ 为获得生成重定位记录所必须的信息，可对照模块的反汇编结果和如下本模块的主要代码框架（略去主要实现，注意函数和变量的实际顺序可能不同），推断并找到每个重定位记录对应的符号引用：

```
/* NOTE: Those capitalized variable names in following code will
be substituted by different actual names in the module. */

char BUFFER[] = ......;
const int CODE_TRAN_ARRAY[] = ......;
int CODE = ......;
const char DICT[] = ......;
...... // Definitions of other variables
```

```
int transform_code( int code, int mode ) {
    switch ( CODE_TRAN_ARRAY[mode] ... )
    {
    case 0:  ......
    case 1:  ......
    ......
    default: ......
    }
    return code;
}

void generate_code( int cookie ) {
    ... = cookie;
    for( i=0; i<...; i++ ) {
        ... = transform_code( ..., i );
    }
}

int encode_1( char* str ) {
    ......
    for(i=0; i<n; i++) {
        str[i] = DICT[str[i]] ...;
        if(str[i]<0x20 || str[i]>0x7E) str[i] = ...;
    }
    ......
}

int encode_2( char* str ) {
    ......
    // Similar to encode_1
    ......
}

typedef int (*CODER)(char*);
CODER encoder[] = { ...... };

void do_phase() {
    generate_code(...);
    ......; // Call one encoder here
    printf("%s\n", ...);
}

......
```