LAB4 - ELF与链接

请你在实验截止前务必确认你提交的内容符合要求*(*格式、相关内容等*)*，建议你在提交后下载你提交的内容进行确认。如果由于你的提交格式错误造成评分程序扣分，责任自负。

实验介绍

本实验的目的在于加深对ELF文件的基本格式组成和程序生成与运行过程中链接的基本概念的理解。实验的主要内容是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。你需要完成对二进制可重定位目标文件（.o文件）内容的特定修改，包括数据内容、机器指令、重定位记录等。

实验中你需要完成6个阶段对特定二进制目标文件的修改工作（可不按顺序）。在每一阶段的最后，你应该能够链接相关模块得到一个可执行二进制程序“linkbomb”，该程序应能正常运行并输出期望的正确结果。

实验语言：C；实验环境：Linux (Debian 9 i386)

实验数据

在本实验中，每位学生会得到一个包含以下内容的TAR文件（可通过实验发布页面中的链接下载），将该文件下载到本地目录中并使用“tar xf xxxxxxxx.tar”命令将其中包含的文件提取出来：

- 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, phase3.o, phase4.o, phase5.o, phase6.o：各阶段实验所针对的二进制可重定位目标模块。

实验内容

在实验中的每一阶段n，在按照阶段的目标要求修改可重定位二进制目标模块phase[n].o后，通常（特例情况见具体阶段说明）可使用如下命令生成可执行程序linkbomb：

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

注意：上述命令行中加入了链接选项“-no-pie”，以指示链接器生成传统的二进制可执行文件“linkbomb”。虽然本实验要求修改的是各可重定位二进制目标文件（而非可执行文件），但在本实验所有类似链接操作中使用该选项可避免引入不必要的实验复杂性，使理解二进制程序更为简单，因此建议使用该链接选项。
随后，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。

```
$ ./linkbomb
```

提示：

- main.o中会调用相应模块phase[n].o中采用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。
- 各阶段目标模块phase[n].o中的程序只引用本模块中（和C标准库中）定义的符号。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重定位记录和符号表信息）。
- 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”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链接后能够运行输出（且仅输出）自己的学号：

```
$ gcc -no-pie -o linkbomb main.o phase1.o
$ ./linkbomb
你的学号
```

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链接后能够运行输出（且仅输出）自己的学号：

```
$ gcc -no-pie -o linkbomb main.o phase2.o
$ ./linkbomb
你的学号
```

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，进一步修改其中入口函数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

针对给定的可重定位目标文件“phase3.o”，创建生成一个名为“phase3_patch.o”的二进制可重定位目标文件（注意不允许修改phase3.o模块），使其如下与main.o、phase3.o链接后能够运行和输出（且仅输出）自己的学号：

```
$ gcc -no-pie -o linkbomb main.o phase3.o phase3_patch.o
$ ./linkbomb
你的学号
```

提示：

- phase3.o模块的入口函数do_phase()依次遍历一个COOKIE字符串（由一组互不相同的英文字母组成，且总长度与学号字符串相同）中的每一字符，并将其不同可能ASCII编码取值映射为特定输出字符。
- 需熟悉并利用链接的符号解析规则。

Phase 4

修改二进制可重定位目标文件“phase4.o”中相应节的内容（注意不允许修改.text节和重定位节的内容），使其如下与main.o链接后能够运行输出（且仅输出）自己的学号：

```
$ gcc -no-pie -o linkbomb main.o phase4.o
$ ./linkbomb
你的学号
```

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，且总长度与学号字符串相同）进行变换处理。 模块的入口函数do_phase()依次遍历COOKIE字符串中的所有字符，对其中每一字符，使用一个switch语句将该字符的不同可能取值（即各大写英文字母的ASCII编码值）映射为输出字符串中相应位置上的字符的具体取值。
- 完成本阶段需熟悉switch语句的机器表示的各个组成部分和特定重定位类型的处理算法。

Phase 5

修改二进制可重定位目标文件“phase5.o”，补充完成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链接后可生成正常执行的程序：

```
$ gcc -no-pie -o linkbomb main.o phase5.o
$ ./linkbomb
xxxxxxxxx
```

运行程序输出的字符串是对你的学号进行编码处理后的结果（其中可能包含空格、制表等空白字符），因此应该不同于你的学号。
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）后运行所得**linkbomb**程序可能会得到以下结果之一：

- "Segmentation fault"出错信息——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？”。
- "Illegal instruction"出错信息——原因类似上面（但存在细节差异）。
- "Welcome to this small lab of linking. To begin lab, please link the relevant object module(s) with the main module. "提示模块未链接的信息——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，但某个重要的重定位记录未正确设置。
- 输出不正确的编码结果——为检查输出结果的正确性（以确定自己是否已正确完成本阶段），可将当前输出结果提交至实验数据服务器（链接见实验发布页面）进行验证。

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。
- 请参考课程中对重定位内容的介绍，在模块的（多个）重定位节中定位出需要恢复的重定位记录。
- 为获得构造重定位记录所需要的信息，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，再按照重定位记录的格式，构造重定位记录的相应二进制表示，最后写入模块中相应重定位节的特定位置。

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

const int CODE_TRAN_ARRAY[] = ......;
int CODE = ......;
const char FDICT[] = ......;
......

int encode_1( char* str )
{
        ......
        for(i=0; i<n; i++)
        {
                str[i] = FDICT[str[i]] ...;
                ......
        }
        ......
}

int encode_2( char* str )
{
        ......
        for(i=0; i<n; i++)
        {
                str[i] = FDICT[str[i]] ...;
                ......
```

```
        }
        ......
}

typedef int (*CODER)(char*);
CODER encoder[] = { encode_1, encode_2 };

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

void generate_code( ... )
{
        ......
        SOME_GLOBAL_VARIABLE = ...;
        for( i=0; i<...; i++ )
                SOME_GLOBAL_VARIABLE = transform_code( ... );
}

void do_phase()
{
        generate_code(...);

        encoder[...](...);

        printf("%s\n", ...);
}
```

Phase 6

修改二进制可重定位目标文件“phase6.o”，补充完成其中被人为清空的一些重定位记录（类似前一阶段可能位于目标文件的多个重定位节中）和部分机器指令（注意不允许修改除这些以外的内容），使其如下与main.o模块链接后可生成正常执行的程序：

```
$ gcc -no-pie -o linkbomb main.o phase6.o
$ ./linkbomb
xxxxxxxxxxxxxx
```

phase6采用了与phase5基本相同的源代码（部分数据的初始值有所变化）。 phase6不同于phase5的主要之处是：phase6.o采用了Position Independent Code (PIC)的编译方式（即编译生成可重定位目标模块时使用了GCC的"-fPIC"选项），因此生成的指令代码对数据和函数对象的引用方式发生了变化。

提示：

- 实验中共需要对phase6.o模块中的**8**个随机被置为全零的重定位记录和**1-2**个被置为nop指令序列并与PIC机制相关的函数的机器指令进行恢复。
- 建议自己编写小的样例程序并通过编译再反汇编，了解同样C源代码的PIC与Non-PIC机器代码之间的差异，进一步对照本模块的机器代码，推断其中所缺失的信息。
- 如果对本模块中缺失的重定位信息和函数指令代码的补充不正确或不完整的话，运行链接所得的linkbomb程序可能会返回类似于前一阶段的错误信息。
- 可使用前阶段说明中的方法检查输出结果的正确性（以确定自己是否已正确完成本阶段）。

实验结果提交

将修改完成的各阶段模块（phase1.o, phase2.o, phase3_patch.o, phase4.o, phase5.o, phase6.o）和未改动的main.o、phase3.o模块用tar工具打包（注意其中不能包含任何目录结构），并命名为“学号**.tar**”的单一文件提交。

注意：可只提交已完成阶段的对应模块文件，如验证正确可以获得相应阶段的实验分数。