程序链接与 ELF 目标文件实验

（苏丰，南京大学计算机学院）

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可重定位目标模块组成的程序，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：数据与 ELF 数据节
阶段 2：指令与 ELF 代码节
阶段 3：符号解析
阶段 4：switch 语句与链接
阶段 5：重定位
阶段 6：位置无关代码

实验中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，链接相关模块生成的可执行程序应能正常运行并输出期望的字符串。

- 实验环境：Linux x86-32/64
- 实验语言：汇编/C

二、实验数据

在本实验中，每位学生可从 Lab 实验服务器下载包含本实验相关文件的一个 tar 文件。可在 Linux 实验环境中使用命令“**tar xvf <tar 文件名>**”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
- outputs.txt：各阶段实验中链接生成的程序在运行时的预期输出字符串

三、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，然后使用如下命令生成可执行程序 linklab：

gcc -no-pie -o linklab main.o phase[n].o（个别阶段还需链接进其它模块）

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

./linklab

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。
- ✓ 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和实验内容。

阶段 1 数据与 ELF 数据节

- 实验目的：1）理解 ELF 目标文件的基本组成与结构；2）熟悉程序中静态区数据的存储与访问机制。
- 实验内容：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）学号：

  gcc -no-pie -o linklab main.o phase1.o

  ./linklab

  学号

  为使程序 linklab 能够输出指定的字符串，需要查看和分析 phase1.o 模块的反汇编结果，从中定位输出函数及其字符串参数，进一步使用 hexedit 等二进制编辑工具修改 phase1.o 文件的数据节中存放的该字符串参数的初始内容，使之与输出目标字符串相同。

阶段 2 指令与 ELF 代码节

- 实验目的：1）理解 ELF 目标文件中指令代码的存储与访问；2）进一步了解和熟悉机器指令的表示方式；3）进一步巩固和掌握过程调用的机器级表示。
- 实验内容：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）学号：

  gcc -no-pie -o linklab main.o phase2.o

  ./linklab

  学号

  为完成实验目标，需要查看 phase2.o 的反汇编代码，获知其中相关函数的功能作用，定位输出字符串的函数，然后将 do_phase()函数中的空操作（nop）指令替换为调用输出字符串函数的机器指令，以输出目标字符串“123456789”。

阶段 3 符号解析

- 实验目的：1）了解程序链接过程中符号解析的作用；2）了解链接器对全局符号的解析规则。
- 实验内容：针对可重定位目标文件“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 模块链接后运行时输出（且仅输出）学号：

  gcc -no-pie -o linklab main.o phase3.o phase3_patch.o

  ./linklab

  学号

  为使程序能够输出指定的字符串，需要查看和分析 phase3.o 模块的反汇编结果，理解字符串内容的输出过程以及所访问数据的结构，进一步设计符合输出要求的新数据结构，并利用链接器对全局符号的解析规则，使程序在执行时实际访问的是新数据结构，从而输出目标字符串。

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 ASCII 编码取值映射为特定输出字符。

阶段 4 switch 语句与链接

- 实验目的：1）理解 switch 语句的机器级表示及其相关链接处理；2）加深对符号引用和重定位基本概念的理解。
- 实验内容：修改二进制可重定位目标文件“phase4.o”中相关节的内容（注意针对 x86-32 不允许修改.text 节和重定位节的内容，针对 x86-64 不允许修改.text 节的内容），使其如下与 main.o 链接后运行时输出（且仅输出）学号：

  gcc -no-pie -o linklab main.o phase4.o

  ./linklab

  学号

  为了完成实验目标，需要分析 phase4.o 模块的机器级代码，了解 do_phase 函数的执行逻辑和其中 switch 语句的机器级表示及其所涉及的跳转表地址引用与重定位等链接相关概念，并通过对模块中特定节内容的修改，使程序输出目标字符串。

  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）中每一字符逐一进行变换处理。

阶段 5 重定位

- 实验目的：1）了解重定位的概念、作用与过程；2）了解常见的重定位类型；3）了解 ELF 目标文件中重定位信息的表示与存储。
- 实验内容：修改二进制可重定位目标文件“phase5.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，可能位于目标文件的不同重定位节中，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学号进行编码处理后得到的一个特定字符串，例如下例中的“**L*`Upt6HZ**”（具体字符串内容见 outputs.txt 文件中对应本阶段的预期输出字符串）：

  gcc -no-pie -o linklab main.o phase5.o

  ./linklab

  L*`Upt6HZ

  为完成实验目标，首先需要分析 phase5.o 模块中各函数的执行逻辑，定位其中包含的各个符号引用并与重定位项进行对照分析，推断被清零的重定位项所对应的符号引用，然后根据重定位记录的组成结构，构造符合规范的重定位项，并用其替代模块中被清零的重定位项。

  为帮助获得构造重定位项所需要的信息，可对照分析 phase5.o 的反汇编结果与主要 C 代码框架，以帮助推断、定位模块中每个重定位项所对应的符号引用。以下是 phase5.o 对应的主要 C 代码框架，其中略去了一些实现细节，有些函数名和变量名及其顺序可能与实际情况不同，宏 MY_ID 随实验数据不同可具有不同的值，在本实验案例中为字符串“123456789”。

```
/* 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:  ......
    case 2:  ......
    case 4:  ......
    case 5:  ......
    case 6:  ......
    case 7:  ......
           code = code + CODE_TRAN_ARRAY[mode];  break;
    ......
    default: ......
    }
    return code;
}

void generate_code( int cookie )
{
    int i;
    CODE = cookie;
    for( i = 0; i < sizeof(CODE_TRAN_ARRAY)/sizeof(int); i++ ) {
        CODE = transform_code( CODE, i );
    }
}

int encode_1( char* str )
{
    int n, i;
    n = strlen(str);
    for(i=0; i<n; i++) {
        str[i] = (DICT[str[i]] ^ CODE) & 0x7F;
        if(str[i]<0x20 || str[i]==0x7F) str[i] = ...;
    }
    return n;
}

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

```
}

typedef int (*CODER)(char*);
CODER encoder[2] = { encode_1, encode_2 };

void do_phase()
{
    generate_code(...);
    encoder[……]( BUFFER );
    puts( BUFFER );
}
```

阶段 6 位置无关代码

- 实验目的：1）了解位置无关代码（PIC）的基本原理；2）了解 PIC 相关重定位类型及相应处理方式。
- 实验内容：修改二进制可重定位目标文件“phase6.o”，恢复其中被**人为清零**的一些**重定位记录**(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学号进行编码处理后得到的一个特定字符串，例如下例中的“**l*]Rmq3EW**”(具体字符串内容见 outputs.txt 文件中对应本阶段的预期输出字符串)：

  gcc -no-pie -o linklab main.o phase6.o

  ./linklab

  l*]Rmq3EW

  本阶段程序采用了与阶段 5 基本相同的源代码（仅个别数据的初始值有所变化，这里不再重复给出）。本阶段**不同于**阶段 5 的主要之处是：本阶段目标文件 phase6.o 采用了**位置无关代码（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和对数据、函数符号的引用方式发生了变化。

  与阶段 5 类似，为完成实验目标，需要分析 phase6.o 模块中各函数的执行逻辑，并对照反汇编操作获得的机器级代码和 C 语言代码，推断被清零的重定位项的内容，然后构造符合规范和要求的重定位数据，用以替代模块中被清零的重定位项。

  提示：

  - ✓ 可以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。

四、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）用 tar 工具打包为一个名为“学号.tar”的文件提交，注意其中不能包含任何目录结构（例如在运行 tar 命令前 cd 到提交文件所在目录，或者使用 tar 的“-C”命令选项）。

可在 tar 文件中只包含已完成阶段的对应文件，如验证正确可获得相应阶段的实验分数。