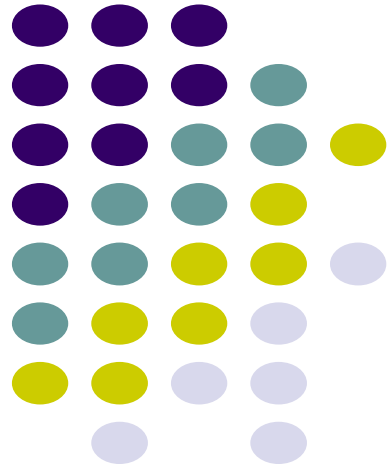
《计算机系统基础（四）：编程与调试实践》

链接与ELF实验



链接与ELF实验：概述

实验概述

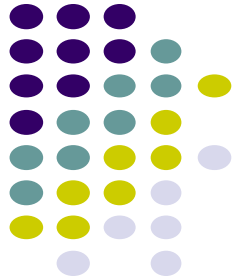
- **实验目的**

  加深对程序生成与运行过程中链接的基本概念和ELF文件的基本结构组成的理解

- **实验内容**
  - 在二进制层面上，逐步修改构成目标程序“linkbomb”的多个二进制模块（.o文件），使其在链接后运行时实现实验指定的行为要求
  - 修改方面包括：二进制可重定位目标文件中的数据、机器指令、重定位记录等

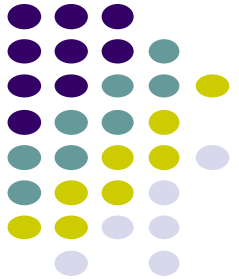
- **实验环境：Linux 32-bit i386，C/汇编语言**

实验数据与文件



- 实验数据包： **linklab_学号.tar**
- 解压命令：**tar xf linklab_学号.tar**
- 数据包中包含下列文件：
  - main.o：主程序的二进制可重定位目标模块（实验中无需修改）
  - phase1.o, phase2.o, ……：各阶段实验需要修改的二进制可重定位目标模块（本学期实验所包含的具体阶段模块以实验文档中的说明为准）

实验内容



- 实验包含6个阶段（本学期实验中实际包含的阶段以实验文档中的说明为准）
- 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：符号解析
  - 阶段4：switch语句与重定位
  - 阶段5：可重定位目标文件
  - 阶段6：位置无关代码（Position-Independent Code，PIC）

实验内容

- 在实验的每一阶段n（n=1,2,3,4,5…），修改相应可重定位二进制目标模块phase[n].o后，使用如下命令链接生成可执行程序linkbomb：

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

  （有些阶段还要求输入额外的.o模块）

- 实验要求：如下运行生成的可执行程序linkbomb，应输出符合各阶段要求的字符串：

  $ ./linkbomb

  $ 123456789　仅供示例，具体应输出的字符串见每阶段说明）

- 各阶段phase[n].c中包含一个do_phase函数完成相应阶段的具体功能
- 一全局函数指针初始化为指向该do_phase函数

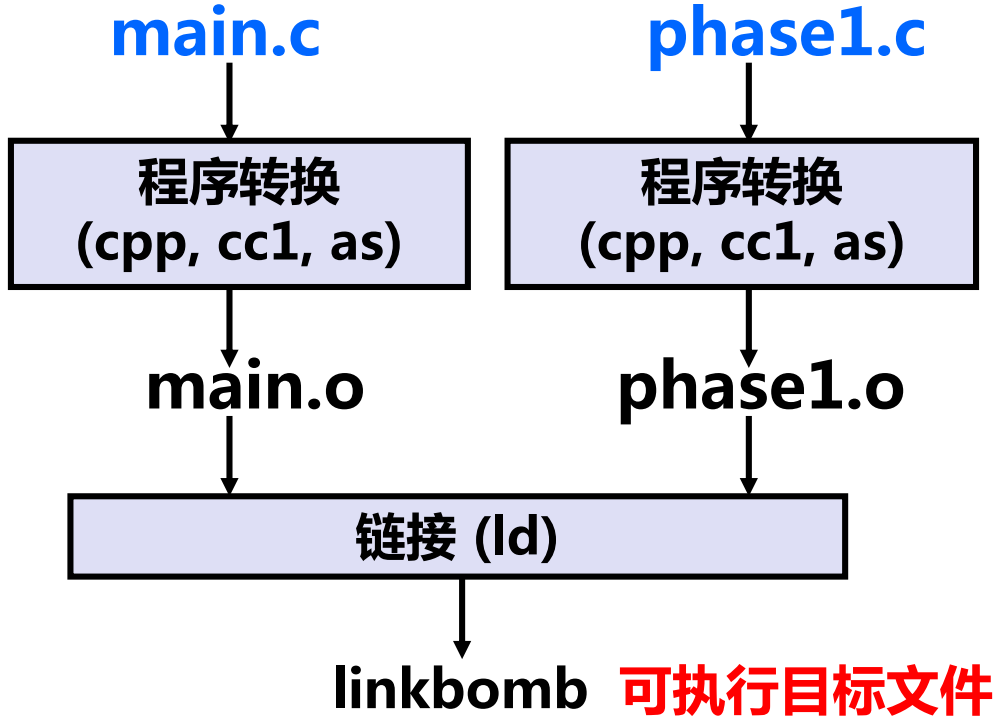
```
// phase[n].c
…
void do_phase() {
        … // 该阶段具体工作
}

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

链接基本概念：可执行文件的生成

- 使用GCC编译器编译并链接生成可执行程序P:
  - $ gcc -o linkbomb main.c phase1.c
  - $ ./linkbomb

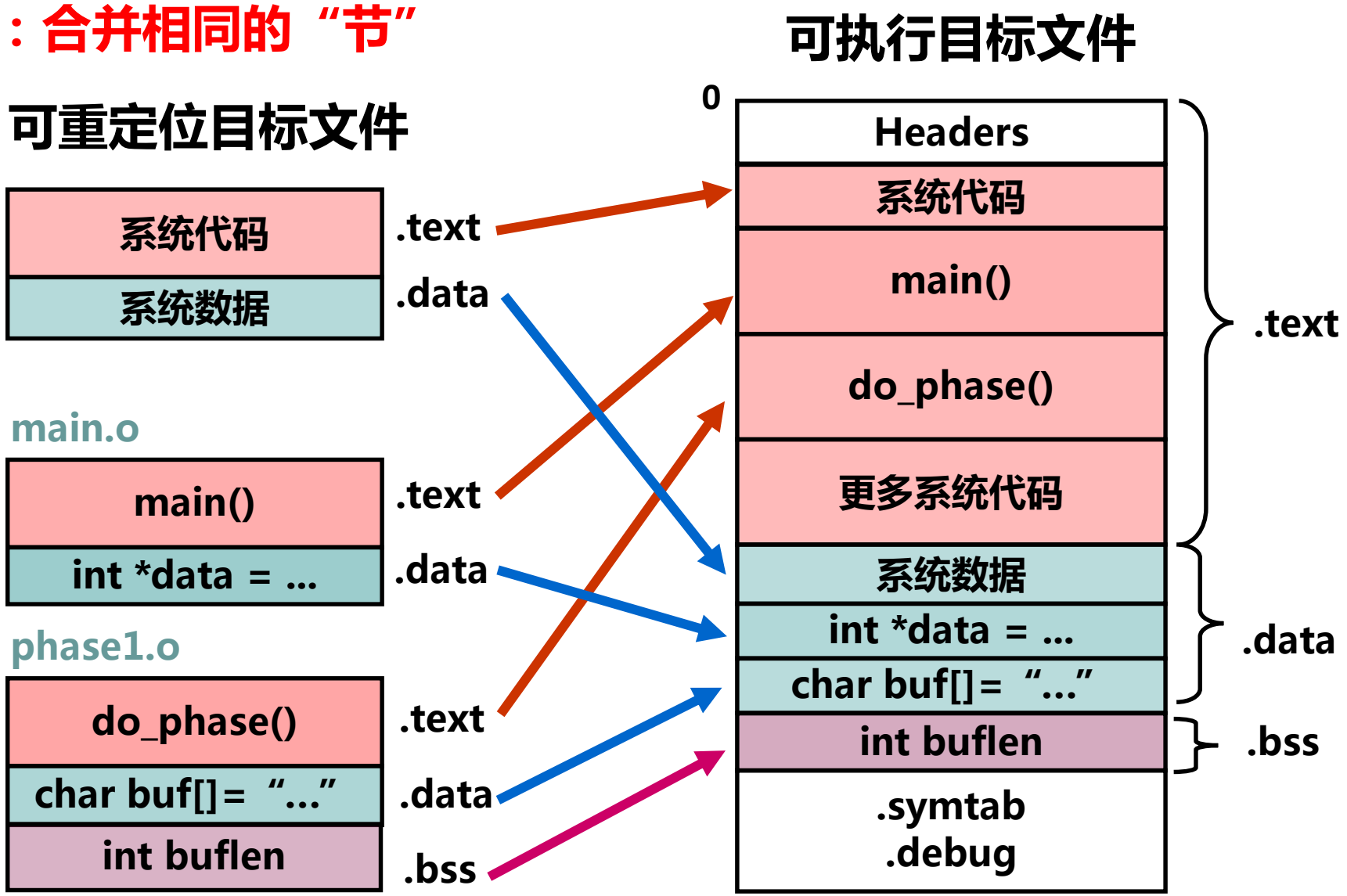
-o：输出目标文件名

GCC编译器的链接过程

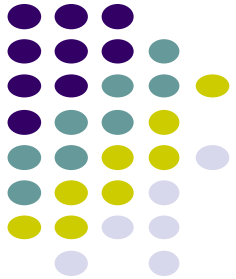


链接过程的本质

链接本质：合并相同的“节”



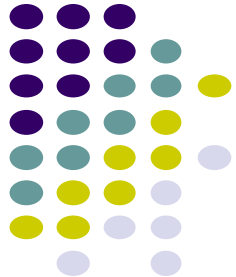
链接操作的步骤



- Step 1. 符号解析（symbol resolution）
  - 程序中包含符号 (包括变量和函数等)的定义和引用
  - 编译器将模块中出现的符号存放在符号表（symbol table）中
    - 符号表是一个结构数组
    - 每个表项包含符号名、长度和位置等信息
  - 链接器将每个符号的引用都与一个确定的符号定义建立关联

- Step 2. 重定位
  - 将多个相同类型的节合并为一个单独的节
  - 计算每个符号定义在虚拟地址空间中的绝对地址
  - 将可执行文件中符号引用处的地址修改为重定位后的符号定义地址

三类目标文件



- 可重定位目标文件（relocatable object files）
  - 可与其他可重定位文件链接生成可执行文件或共享库
- 可执行目标文件（executable object files）
  - 包含的代码和数据可以被直接装载到内存并被执行
- 共享库（shared Libraries）
  - 特殊的可重定位目标模块，能在运行时装载到任意内存地址并与程序动态链接

可重定位目标文件

- 可被链接生成可执行文件或共享库
- 包含代码、数据，其地址未最终确定（通常表示为相对所在节起始的偏移）
- 包含重定位信息，指出哪些符号引用处需要链接器进行重定位——即确定最终地址

ELF文件：Linux平台上常用的一种可重定位或可执行目标文件类型

- ✓ **由不同的节（section）组成**
- ✓ **节是 ELF 文件中具有相同特征的最小可处理单位**

  .text节: 代码　.rodata: 只读数据　.data节:初始化数据　.bss: 未初始化数据
- ✓ **为支持链接、调试等操作，还可能包含许多其他类型的节，如符号表节、字符串表节 、重定位记录节等**

ELF可重定位目标文件格式

ELF 头

- ✓ 定义了ELF魔数、版本、小端/大端、操作系统平台、目标文件的类型、机器结构类型、节头表的起始位置和长度等

.text 节

- ✓ 编译后的代码部分

.rodata 节

- ✓ 只读数据，如 printf 格式串、switch 跳转表等

.data 节

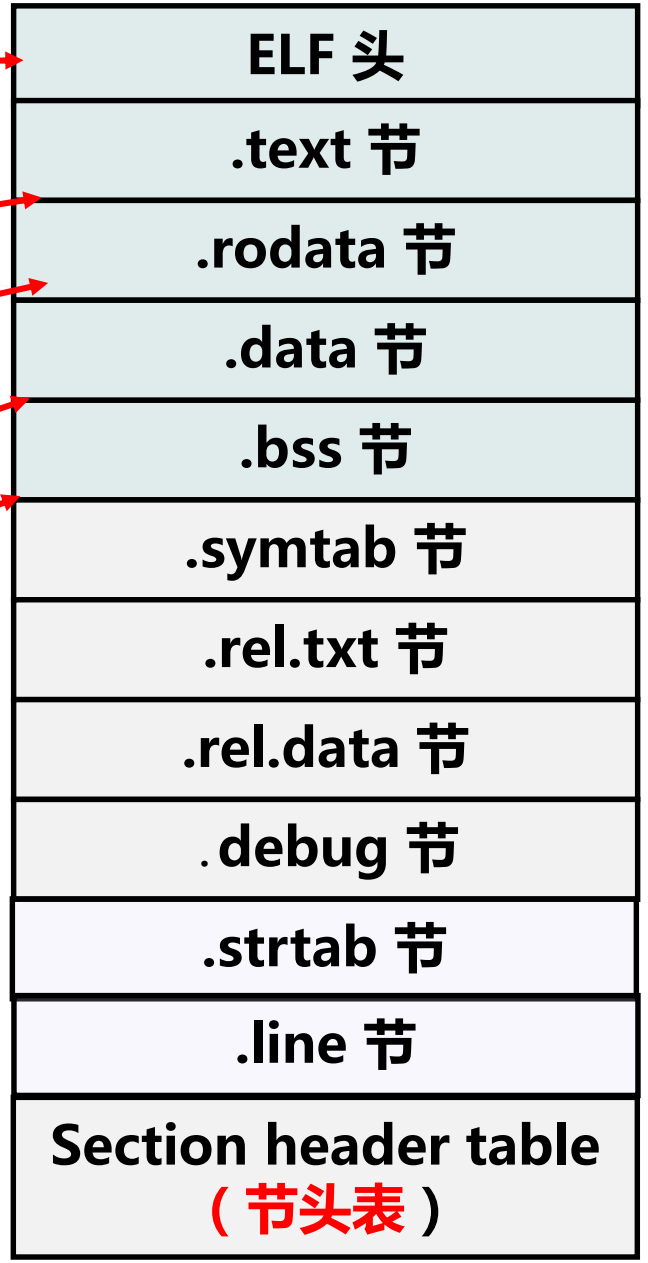
- ✓ 已初始化的静态数据

.bss 节

- ✓ 未初始化的静态数据，仅是占位符，不占据任何实际磁盘空间。区分初始化和非初始化是为了提高存储空间利用效率

节头表

- ✓ 给出了各节的节名、文件偏移、大小、访问属性、对齐方式等重要信息

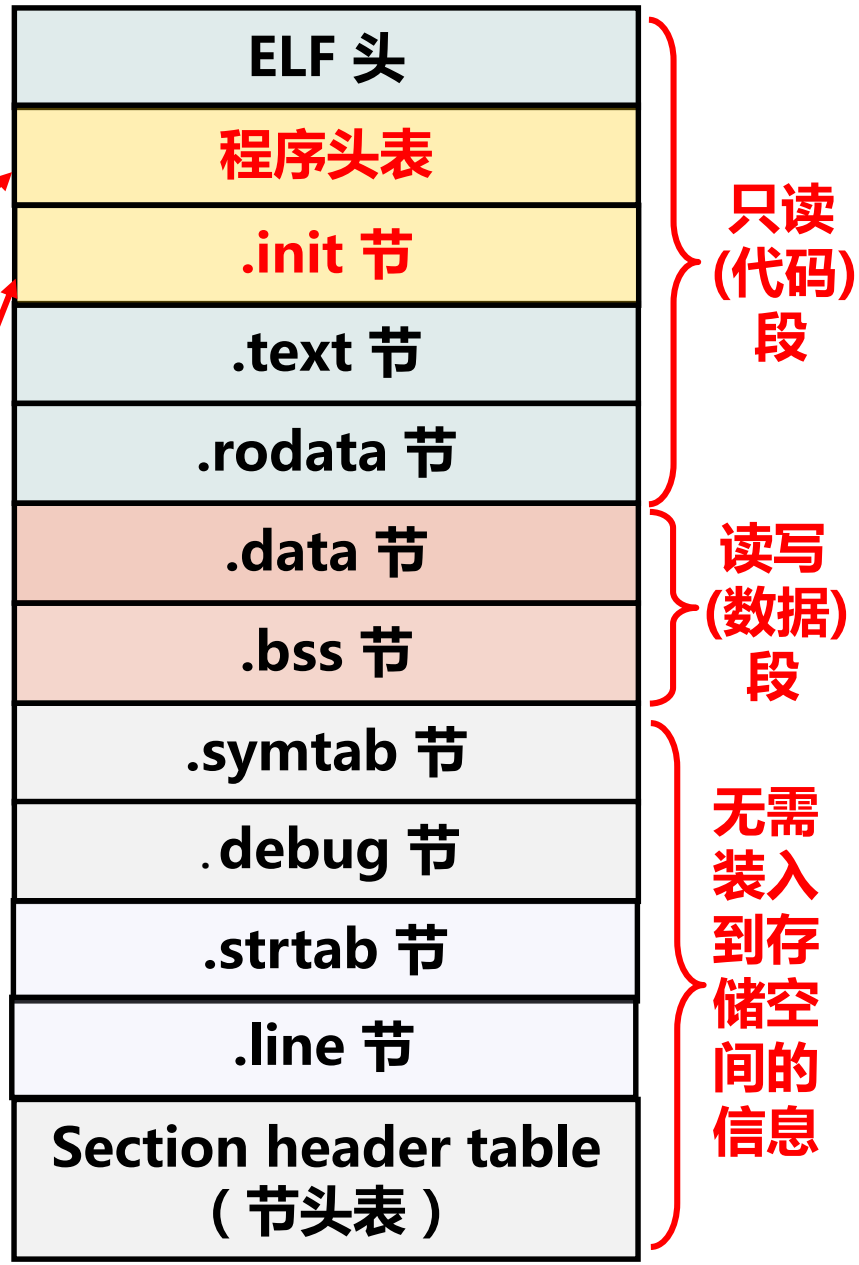


可执行目标文件

- 由不同的段（segment）组成
  - 具相同访问属性的多个节合并映射成段（Segment），如：.data节和.bss节合并映射到一个可读可写数据段
  - 每个段具有相应属性，如：在可执行文件中的位移、大小、在虚拟空间中的位置、对齐方式、访问属性等，记录于程序头表中
- 所有变量和函数已有确定地址（虚拟地址空间中的地址）
- 符号引用处已被重定位，指向所引用的符号的定义所在存储位置
- 可被CPU直接执行，指令地址和指令给出的操作数地址都是虚拟地址

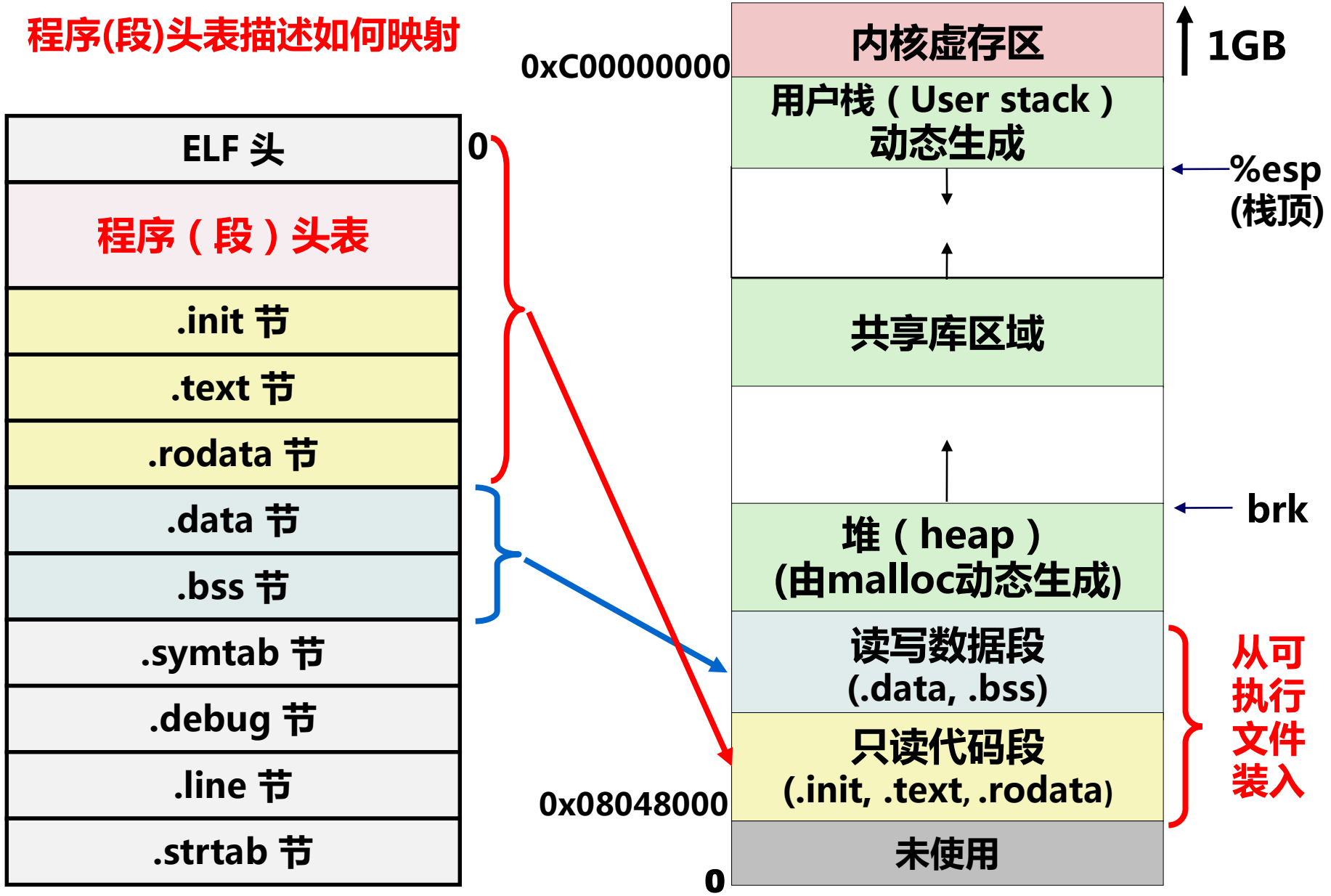
可执行目标文件格式

- 与可重定位文件稍有不同：
  - ELF头中字段e_entry给出执行程序时第一条指令的地址，而在可重定位文件中，此字段为0
  - 多一个程序头表，也称段头表（segment header table），是一个结构数组
  - 多一个.init节，用于定义_init函数，该函数用来进行可执行目标文件开始执行时的初始化工作
  - 少若干.rel节（无需相应的重定位）

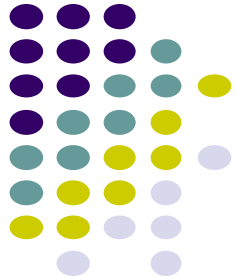


可执行文件的存储器映像

程序(段)头表描述如何映射



实验工具readelf

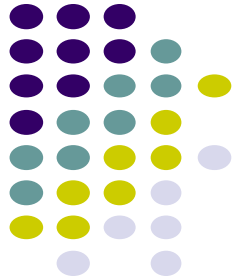


- 读取ELF格式二进制目标文件中的各方面信息并打印输出，如节（节名、偏移量及其中数据等）、符号表、字符串表、重定位记录等
- 命令行格式：readelf <options> elf-file(s)

Options（部分）：

-a –all	等同于同时使用：-h -l -S -s -r -d -V -A -I
-h --file-header	显示ELF文件头
-l --program-headers	显示程序头
-S --section-headers	显示节头
-t --section-details	显示节详细信息
-s --syms	显示符号表
-r --relocs	显示重定位信息
-x --hex-dump=<number|name>	以字节形式显示输出<number|name>指定节的内容
-p --string-dump=<number|name>	以字符串形式显示输出<number|name>指定节的内容
-R --relocated-dump=<number|name>	以重定位后的字节形式显示输出<number|name>指定节内容

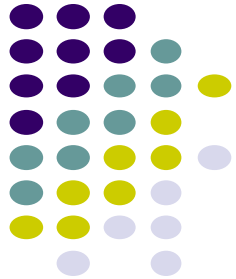
实验工具



- **objdump**：反汇编二进制目标文件中包含的机器指令，获得对应的汇编指令供分析
  - objdump -d bomb　输出bomb程序的反汇编结果
  - objdump -d bomb > bomb.s　获得bomb程序的反汇编结果并保存于文本文件bomb.s中
  - objdump -t bomb　打印bomb程序的符号表，其中包含bomb中所有函数、全局变量的名称和存储地址

- **hexedit**：二进制文件编辑工具

小结



- 本节课介绍了链接与ELF实验的主要内容、实验所包含的各个阶段、实验数据的组成和实验用到的主要工具，并概要介绍了开展实验所需的程序链接与ELF文件等方面的背景知识。

链接与ELF实验：静态数据与ELF数据节

实验阶段1

- 实验内容：修改二进制可重定位目标文件“**phase1.o**”的数据节内容（不允许修改其他节），使其与**main.o**链接后运行时输出自己的学号：

  $ gcc -no-pie -o linkbomb main.o phase1.o

  $ ./linkbomb

  学号

- 实验步骤：

  1. 使用**objdump**工具获得目标文件的汇编代码，使用**readelf**工具获得其重定位记录
  2. 结合汇编代码和重定位信息，定位输出函数的调用参数在目标文件中的存储地址
  3. 使用**hexedit**工具，对**phase1.o**文件的数据节中上述存储地址处的相应字节进行修改

实验阶段1

- 实验步骤1：使用**objdump**工具获得目标文件的汇编代码，使用**readelf**工具获得其重定位记录

$ objdump -d phase1.o

```
00000000 <do_phase>:
   0:    55                   push   %ebp
   1:    89 e5                mov    %esp,%ebp
   3:    83 ec 08             sub    $0x8,%esp
   6:    83 ec 0c             sub    $0xc,%esp
   9:    68 29 00 00 00       push   $0x29
   e:    e8 fc ff ff ff       call   f <do_phase+0xf>
  13:    83 c4 10             add    $0x10,%esp
  16:    c9                   leave
  17:    c3                   ret
```

$ readelf -r phase1.o

```
Relocation section '.rel.text' at offset 0x2fc contains 2 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
0000000a  00000301 R_386_32        00000000   .data
0000000f  00000a02 R_386_PC32      00000000   puts

Relocation section '.rel.data' at offset 0x30c contains 1 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
000000dc  00000901 R_386_32        00000000   do_phase

Relocation section '.rel.eh_frame' at offset 0x314 contains 1
entries:
 Offset     Info    Type            Sym.Value  Sym. Name
00000020  00000202 R_386_PC32      00000000   .text
```

实验阶段1

- 实验步骤2：结合并分析汇编代码与重定位信息，定位输出函数的调用参数在目标文件中的存储地址
  - 汇编器遇到对符号（变量或过程）的引用时，生成一个重定位条目
    - 数据节中引用对应的重定位条目在.rel_data节中，代码节中引用对应的重定位条目在.rel_text节中
  - ELF中重定位条目格式如右侧所示：

```
typedef  struct {
        int  offset;        /*节内偏移*/
        int  symbol:24,  /*所绑定符号*/
             type: 8;      /*重定位类型*/
        } Elf32_Rel;
```

```
00000000 <do_phase>:
  0:    55                   push   %ebp
  1:    89 e5                mov    %esp,%ebp
  3:    83 ec 08             sub    $0x8,%esp
  6:    83 ec 0c             sub    $0xc,%esp
  9:    68 29 00 00 00       push   $0x29
  e:    e8 fc ff ff ff       call   f <do_phase+0xf>
 13:    83 c4 10             add    $0x10,%esp
 16:    c9                   leave
 17:    c3                   ret
```

```
Relocation section '.rel.text' at offset 0x2fc contains 2 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
0000000a  00000301 R_386_32        00000000   .data
0000000f  00000a02 R_386_PC32      00000000   puts
```

重定位信息

- IA-32两种基本重定位类型

  R_386_PC32PC相对地址重定位方式下，重定位后的引用地址 =

  符号定义地址 – 符号引用所在地址 + 重定位前引用处的初始值

R_386_32绝对地址重定位方式下，重定位后的引用地址 =

符号定义地址 + 重定位前引用处的初始值

```
00000000 <do_phase>:
   0:    55                   push   %ebp
   1:    89 e5                mov    %esp,%ebp
   3:    83 ec 08             sub    $0x8,%esp
   6:    83 ec 0c             sub    $0xc,%esp
   9:    68 29 00 00 00       push   $0x29
   e:    e8 fc ff ff ff       call   f <do_phase+0xf>
  13:    83 c4 10             add    $0x10,%esp
  16:    c9                   leave
  17:    c3                   ret
```

+

```
Relocation section '.rel.text' at offset 0x2fc contains 2 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
0000000a  00000301 R_386_32        00000000   .data
0000000f  00000a02 R_386_PC32      00000000   puts
```

可知输出字符串起始地址在.data节地址加上偏移量0x29的位置

实验阶段1

- 实验步骤3：使用hexedit工具，对phase1.o文件的数据节中相应地址处字节进行修改

1. 使用readelf工具查看phase1.o文件的节头表，确定.data节在二进制文件中的偏移量，从而定位出需要修改的目标字符串在文件中的数据内容

➢ readelf -S phase1.o

Section Headers:

[Nr] Name	Type	Addr	Off	Size	ES	Flg	Lk	Inf	Al
[0]	NULL	00000000	000000	000000	00		0	0	0
[1] .text	PROGBITS	00000000	000034	000018	00	AX	0	0	1
[2] .rel.text	REL	00000000	0002fc	000010	08	I	11	1	4
[3] .data	PROGBITS	00000000	000080	0000e0	00	WA	0	0	64
[4] .rel.data	REL	00000000	00030c	000008	08	I	11	3	4
[5] .bss	NOBITS	00000000	000160	000000	00	WA	0	0	1

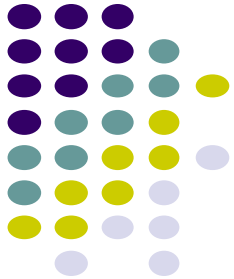
……

由上可知：

✓ .data节在文件中的起始偏移量为0x80

✓ 输出字符串在文件中的起始地址为0x80 + 0x29 = 0xA9

节头表（Section Header Table）



- 节头表是ELF可重定位目标文件中的重要组成部分
- 描述每个节的节名、在文件中的偏移、大小、访问属性、对齐方式等
- 以下是Linux i386系统中ELF文件的节头数据结构

```
typedef struct {
    Elf32_Word      sh_name;        节名字符串在.shstrtab节中的偏移
    Elf32_Word      sh_type;        节类型：无效/代码或数据/符号表/字符串表/...
    Elf32_Word      sh_flags;       节标志：该节在虚拟空间中的访问属性
    Elf32_Addr      sh_addr;        虚拟地址：若可被加载，则对应虚拟地址
    Elf32_Off       sh_offset;      在文件中的偏移地址，对.bss节无意义
    Elf32_Word      sh_size;        节在文件中所占的长度
    Elf32_Word      sh_link;        sh_link和sh_info用于与链接相关的节（如
    Elf32_Word      sh_info;        .rel.text节、.rel.data节、.symtab节等）
    Elf32_Word      sh_addralign;       节的对齐要求
    Elf32_Word      sh_entsize;     如节内容是表项大小固定的一个表（如符号表），
} Elf32_Shdr;                       则给出每个表项的长度，0表示无固定长度表项
```

实验阶段1

- **实验步骤3：使用hexedit工具，对phase1.o文件数据节中相应字节进行修改**

2. 使用**readelf**（或**objdump**工具），查看**.data**节中相应偏移量（**0x29**）处的字符串内容，并与未修改的**phase1.o**链接后程序输出的字符串比较，确定该字符串为修改的目标

➢ **readelf –x .data phase1.o**

```
Hex dump of section '.data':
 NOTE: This section has relocations against it, but these have NOT been applied
to this dump.
  0x00000000 424a4478 364c3137 73677a68 54636263 BJDx6L17sgzhTcbc
  0x00000010 564e5454 53557839 4a20394b 546a4f78 VNTTSUx9J 9KTjOx
  0x00000020 42684649 32514166 75524b48 4b326349 BhFI2QAfuRKHK2cI
  0x00000030 62386371 4c656445 38552055 6b6f3656 b8cqLedE8U Uko6V
  0x00000040 7a317a73 50523364 51694d62 6b586871 z1zsPR3dQiMbkXhq
  0x00000050 76483633 6c717064 62617309 68356272 vH63lqpdbas.h5br
  0x00000060 64645173 4c543146 516d4636 0956574a ddQsLT1FQmF6.VWJ
  0x00000070 48527342 4b76744d 4e726c73 20647359 HRsBKvtMNrls dsY
  0x00000080 76207462 7563684f 576a6a67 6b696832 v tbuchOWjjgkih2
  0x00000090 67764154 3676376e 616c344e 66433153 gvAT6v7nal4NfC1S
  0x000000a0 57624d58 4f676c33 53207920 6a47734a WbMXOgl3S y jGsJ
  0x000000b0 0959337a 4850556d 73440909 75785771 .Y3zHPUmsD..uxWq
  0x000000c0 44736751 6351654c 624b734f 73576679 DsgQcQeLbKsOsWfy
  0x000000d0 424b6c49 3047676e 754b4b00 00000000 BKlI0GgnuKK.....
```

```
ics@debian:~/linklab$ gcc -no-pie -o linkbomb main.o phase1.o
ics@debian:~/linklab$ ./linkbomb
RKHK2cIb8cqLedE8U Uko6Vz1zsPR3dQiMbkXhqvH63lqpdbas    h5brddQsLT1FQmF6    VWJHRsBKvtMNrls dsYv
tbuchOWjjgkih2gvAT6v7nal4NfC1SWbMXOgl3S y jGsJ    Y3zHPUmsD
uxWqDsgQcQeLbKsOsWfyBKlI0GgnuKK
```

实验阶段1

- 实验步骤3：使用hexedit工具，对phase1.o文件数据节中相应字节进行修改

3. 使用hexedit工具将目标文件中输出字符串存储位置的起始若干字符替换为学号中的字符（其后应有一个0x00字节以表示字符串结束） —— 完成编辑后按Ctrl-X键保存并退出hexedit

修改前

```
000000A0   42 68 46 49  32 51 41 66  75 52 4B 48  4B 32 63 49  BhFI2QAfuRKHK2cI
000000B0   62 38 63 71  4C 65 64 45  38 55 20 55  6B 6F 36 56  b8cqLedE8U Uko6V
000000C0   7A 31 7A 73  50 52 33 64  51 69 4D 62  6B 58 68 71  z1zsPR3dQiMbkXhq
000000D0   76 48 36 33  6C 71 70 64  62 61 73 09  68 35 62 72  vH63lqpdbas.h5br
000000E0   64 64 51 73  4C 54 31 46  51 6D 46 36  09 56 57 4A  ddQsLT1FQmF6.VWJ
000000F0   48 52 73 42  4B 76 74 4D  4E 72 6C 73  20 64 73 59  HRsBKvtMNrls dsY
00000100   76 20 74 62  75 63 68 4F  57 6A 6A 67  6B 69 68 32  v tbuchOWjjgkih2
00000110   67 76 41 54  36 76 37 6E  61 6C 34 4E  66 43 31 53  gvAT6v7nal4NfC1S
00000120   57 62 4D 58  4F 67 6C 33  53 20 79 20  6A 47 73 4A  WbMXOgl3S y jGsJ
00000130   09 59 33 7A  48 50 55 6D  73 44 09 09  75 78 57 71  .Y3zHPUmsD..uxWq
00000140   44 73 67 51  63 51 65 4C  62 4B 73 4F  73 57 66 79  DsgQcQeLbKsOsWfy
00000150   42 4B 6C 49  30 47 67 6E  75 4B 4B 00  00 00 00 00  BKlI0GgnuKK.....
00000160   00 47 43 43  3A 20 28 44  65 62 69 61  6E 20 34 2E  .GCC: (Debian 4.
00000170   39 2E 32 2D  31 30 29 20  34 2E 39 2E  32 00 00 00  9.2-10) 4.9.2...
00000180   14 00 00 00  00 00 00 00  01 7A 52 00  01 7C 08 01  .........zR..|..
00000190   1B 0C 04 04  88 01 00 00  1C 00 00 00  1C 00 00 00  ................
000001A0   00 00 00 00  18 00 00 00  00 41 0E 08  85 02 42 0D  .........A....B.
---  phase1.o        --0xA9/0x524-----------------------------------------
```

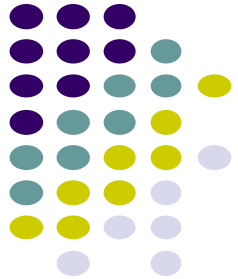
```
000000A0   42 68 46 49  32 51 41 66  75 31 32 33  34 35 36 37  BhFI2QAfu1234567
000000B0   38 39 00 71  4C 65 64 45  38 55 20 55  6B 6F 36 56  89.qLedE8U Uko6V
```

修改后

4. 重新链接、生成和运行程序，验证修改有效

```
ics@debian:~/linklab$ gcc -no-pie -o linkbomb main.o phase1.o
ics@debian:~/linklab$ ./linkbomb
123456789
```

小结



- 本节课介绍了链接与ELF实验第1阶段“静态数据与ELF数据节” 的基本过程和方法，并且演示了其中的主要操作步骤。从中，我们了解了ELF文件结构与组成、二进制程序中静态数据的存储、使用readelf、hexedit等工具查看或修改二进制ELF程序文件各方面内容的基本操作。

链接与ELF实验：指令与ELF代码节

实验阶段2

- 实验内容：修改二进制可重定位目标文件“**phase2.o**”的代码节内容（不允许修改其它节），使其与**main.o**链接后运行输出自己的学号：

  $ gcc -no-pie -o linkbomb main.o phase2.o

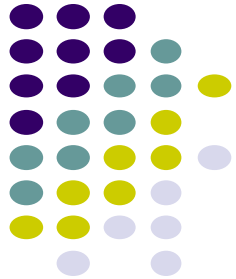
  $ ./linkbomb

  学号

- 实验步骤：

  1. 使用**objdump**工具获得目标文件的汇编代码，使用**readelf**工具获得其重定位记录和符号表
  2. 分析汇编代码并结合重定位信息、符号表，推断模块中各函数的功能作用，找出其中负责输出的函数，在目标文件中定位其存储地址
  3. 构造调用输出函数的指令代码
  4. 使用上步构造的调用输出函数的指令代码，替换**do_phase()**函数体中的**nop**指令，以实现期望输出

实验阶段2



- 实验步骤1：使用**objdump**工具获得目标文件的汇编代码，使用**readelf**工具获得其重定位记录和符号表

$ objdump -d phase2.o

```
0000009c <ecRlvzPzKN>:
  9c:    55                   push   %ebp
  9d:    89 e5                mov    %esp,%ebp
  9f:    83 ec 08             sub    $0x8,%esp
  a2:    83 ec 08             sub    $0x8,%esp
  a5:    68 02 00 00 00       push   $0x2
  aa:    ff 75 08             pushl  0x8(%ebp)
  ad:    e8 fc ff ff ff       call   ae <ecRlvzPzKN+0x12>
  b2:    83 c4 10             add    $0x10,%esp
  b5:    85 c0                test   %eax,%eax
  b7:    75 10                jne    c9 <ecRlvzPzKN+0x2d>
  b9:    83 ec 0c             sub    $0xc,%esp
  bc:    ff 75 0c             pushl  0xc(%ebp)
  bf:    e8 fc ff ff ff       call   c0 <ecRlvzPzKN+0x24>
  c4:    83 c4 10             add    $0x10,%esp
  c7:    eb 01                jmp    ca <ecRlvzPzKN+0x2e>
  c9:    90                   nop
  ca:    c9                   leave
  cb:    c3                   ret
```

$ readelf -r phase2.o

```
Relocation section '.rel.text' at offset 0x390
contains 4 entries:
Offset     Info     Type            Sym.Value  Sym. Name
00000070  00000c02 R_386_PC32      00000000   strlen
000000a6  00000501 R_386_32        00000000   .rodata
000000ae  00000d02 R_386_PC32      00000000   strcmp
000000c0  00000e02 R_386_PC32      00000000   puts
......
```

实验阶段2

- 实验步骤2：分析汇编代码并结合重定位信息，推断模块中各函数的功能作用，找出其中负责输出的函数，在目标文件中定位其存储地址

1. 在phase2.o代码节中找到包含有puts输出函数调用的函数（这里是ecRlvzPzKN函数）

```
0000009c <ecRlvzPzKN>:
  9c:    55                   push   %ebp
  9d:    89 e5                mov    %esp,%ebp
  9f:    83 ec 08             sub    $0x8,%esp
  a2:    83 ec 08             sub    $0x8,%esp
  a5:    68 02 00 00 00       push   $0x2
  aa:    ff 75 08             pushl  0x8(%ebp)
  ad:    e8 fc ff ff ff       call   ae <ecRlvzPzKN+0x12>
  b2:    83 c4 10             add    $0x10,%esp
  b5:    85 c0                test   %eax,%eax
  b7:    75 10                jne    c9 <ecRlvzPzKN+0x2d>
  b9:    83 ec 0c             sub    $0xc,%esp
  bc:    ff 75 0c             pushl  0xc(%ebp)
  bf:    e8 fc ff ff ff       call   c0 <ecRlvzPzKN+0x24>
  c4:    83 c4 10             add    $0x10,%esp
  c7:    eb 01                jmp    ca <ecRlvzPzKN+0x2e>
  c9:    90                   nop
  ca:    c9                   leave
  cb:    c3                   ret
```

R_386_PC32PC相对地址重定位方式下，重定位后的引用地址 =

ADDR(sym) - ADDR(sym引用) + 重定位前引用处的初始值

sym定义地址　　sym引用所在地址

```
Relocation section '.rel.text' at offset 0x390
contains 4 entries:
Offset     Info     Type            Sym.Value  Sym. Name
00000070   00000c02 R_386_PC32      00000000   strlen
000000a6   00000501 R_386_32       00000000   .rodata
000000ae   00000d02 R_386_PC32      00000000   strcmp
000000c0   00000e02 R_386_PC32      00000000   puts
```

实验阶段2

- 实验步骤2：分析汇编代码并结合重定位信息，推断模块中各函数的功能作用，找出其中负责输出的函数，在目标文件中定位其存储地址

2. 从符号表中获得目标函数在.text节中的偏移量（这里是0x9c）

readelf -s phase2.o

```
Symbol table '.symtab' contains 17 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS phase2.c
     2: 00000000     0 SECTION LOCAL  DEFAULT    1
     3: 00000000     0 SECTION LOCAL  DEFAULT    3
     4: 00000000     0 SECTION LOCAL  DEFAULT    5
     5: 00000000     0 SECTION LOCAL  DEFAULT    6
     6: 0000009c    48 FUNC    LOCAL  DEFAULT    1 ecRlvzPzKN
     7: 00000000     0 SECTION LOCAL  DEFAULT    8
     8: 00000000     0 SECTION LOCAL  DEFAULT    9
     9: 00000000     0 SECTION LOCAL  DEFAULT    7
    10: 00000000     4 OBJECT  GLOBAL DEFAULT    3 phase_id
    11: 00000000   156 FUNC    GLOBAL DEFAULT    1 TfnmvTLuYT
    12: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND strlen
    13: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND strcmp
    14: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND puts
    15: 000000cc    70 FUNC    GLOBAL DEFAULT    1 do_phase
    ......
```

实验阶段2

- **实验步骤3**：构造调用输出函数的指令代码

1. 分析目标函数**ecRlvzPzKN**的执行逻辑：需要输入两个参数，第一个参数（位于`0x8(%ebp)`）需与某内置字符串相同（**strcmp**返回**0**）；第二个参数（位于`0xc(%ebp)`）被传递给**puts()**输出函数，因此内容应与学号字符串相同
2. 结合重定位记录确定内置字符串在**phase2.o**中的存放位置，进一步使用**readelf**工具获得其内容为“`IloVEcS`”
   - 内置字符串地址 = `.rodata`节起始地址 + 引用处的初始值`0x2`

```
0000009c <ecRlvzPzKN>:
  9c:    55                   push   %ebp
  9d:    89 e5                mov    %esp,%ebp
  9f:    83 ec 08             sub    $0x8,%esp
  a2:    83 ec 08             sub    $0x8,%esp
  a5:    68 02 00 00 00       push   $0x2
  aa:    ff 75 08             pushl  0x8(%ebp)
  ad:    e8 fc ff ff ff       call   ae <ecRlvzPzKN+0x12>
  b2:    83 c4 10             add    $0x10,%esp
  b5:    85 c0                test   %eax,%eax
  b7:    75 10                jne    c9 <ecRlvzPzKN+0x2d>
  b9:    83 ec 0c             sub    $0xc,%esp
  bc:    ff 75 0c             pushl  0xc(%ebp)
  bf:    e8 fc ff ff ff       call   c0 <ecRlvzPzKN+0x24>
  ……
```

```
Relocation section '.rel.text' at offset 0x390
contains 4 entries:
Offset     Info     Type            Sym.Value  Sym. Name
00000070   00000c02 R_386_PC32      00000000   strlen
000000a6   00000501 R_386_32       00000000   .rodata
000000ae   00000d02 R_386_PC32     00000000   strcmp
000000c0   00000e02 R_386_PC32     00000000   puts
```

```
ics@debian:~/linklab$ readelf -x .rodata phase2.o

Hex dump of section '.rodata':
  0x00000000 3200496c 6f564563 5300      2.IloVEcS.
```

实验阶段2

call指令的操作数应为目标地址相对PC的偏移量，计算如下：

- 偏移量 = 目标地址 – PC值（即call指令的下一条指令的起始地址）

  = 0x9c - 0x104 = - 0x68 = ff ff ff 98

实验步骤3：构造调用输出函数的指令代码

3. 在栈中构造与内置字符串和学号字符串内容相同的局部字符串变量，并将其地址作为实参压入栈中
4. 通过相对PC偏移量的方式跳转到目标函数

提示：可将完成上述功能的汇编源程序（下列红框中）保存于一文件中，再用as汇编器将其汇编为二进制目标文件（.o），从中获得所需机器指令序列

提示：须保证do_phase函数的栈帧大小为16字节的倍数

```
0000009c <ecRlvzPzKN>:
  9c: 55                   push   %ebp
  9d: 89 e5                mov    %esp,%ebp
  9f: 83 ec 08             sub    $0x8,%esp
  a2: 83 ec 08             sub    $0x8,%esp
  a5: 68 02 00 00 00       push   $0x2
  aa: ff 75 08             pushl  0x8(%ebp)
  ad: e8 fc ff ff ff       call   ae
……
```

注释：从符号表中可发现输出函数ecRlvzPzKN具有LOCAL链接绑定（Bind）属性，因此可通过相对PC的偏移量直接调用跳转，无需重定位

```
000000cc <do_phase>:
  cc: 55                                  push   %ebp
  cd: 89 e5                               mov    %esp,%ebp
  cf: 83 ec 28                            sub    $0x28,%esp
  d2: c7 45 f0 49 6c 6f 56                movl   $0x566f6c49,-0x10(%ebp)
  d9: c7 45 f4 45 63 53 00                movl   $0x536345,-0xc(%ebp)
  e0: c7 45 e6 31 32 33 34                movl   $0x34333231,-0x1a(%ebp)
  e7: c7 45 ea 35 36 37 38                movl   $0x38373635,-0x16(%ebp)
  ee: 66 c7 45 ee 39 00                   movw   $0x39,-0x12(%ebp)
  f4: 83 ec 08                            sub    $0x8,%esp
  f7: 8d 45 e6                            lea    -0x1a(%ebp),%eax
  fa: 50                                  push   %eax
  fb: 8d 45 f0                            lea    -0x10(%ebp),%eax
  fe: 50                                  push   %eax
  ff: e8 98 ff ff ff                      call   9c <ecRlvzPzKN>
 104: 83 c4 10                            add    $0x10,%esp
 107: 90                                  nop
 108: c9                                  leave
 109: c3                                  ret
```

实验阶段2

- 实验步骤4：使用调用输出函数的指令代码，替换do_phase()函数体中的nop指令，以获得期望输出

```
Section Headers:
  [Nr] Name    Type        Addr     Off    Size   ES Flg Lk Inf Al
  [ 1] .text   PROGBITS    00000000 000034 000112 00 AX  0   0   1
```

1. 使用readelf工具及其“-S”命令行选项获得节头表，从中获得.text在phase2.o中偏移量为0x34，进一步得到函数中插入指令代码的位置为：
   - 0x34 + 0xcf = 0x103

     （0xcf是do_phase函数中第一个被替换的nop指令的偏移量）
2. 使用hexedit将phase2.o文件中上述位置0x103起始的字节内容（原为nop指令代码0x90）替换为调用指令代码序列

```
000000cc <do_phase>:
  cc: 55
  cd: 89 e5
  cf: 83 ec 28
  d2: c7 45 f0 49 6c 6f 56
  d9: c7 45 f4 45 63 53 00
  e0: c7 45 e6 31 32 33 34
  e7: c7 45 ea 35 36 37 38
  ee: 66 c7 45 ee 39 00
  f4: 83 ec 08
  f7: 8d 45 e6
  fa: 50
  fb: 8d 45 f0
  fe: 50
  ff: e8 98 ff ff ff
 104: 83 c4 10
 107: 90
 108: c9
 109: c3
 ……
```

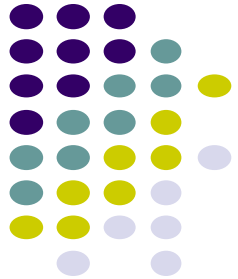
修改前

```
00000100   55 89 E5 90   90 90 90 90   90 90 90 90   90 90 90 90   U...............
00000110   90 90 90 90   90 90 90 90   90 90 90 90   90 90 90 90   ................
00000120   90 90 90 90   90 90 90 90   90 90 90 90   90 90 90 90   ................
00000130   90 90 90 90   90 90 90 90   90 90 90 90   90 90 90 90   ................
00000140   90 90 90 90   5D C3 00 00   00 00 00 00   00 00 00 00   ....]...........
00000150   32 00 49 6C   6F 56 45 63   53 00 00 47   43 43 3A 20   2.IloVEcS..GCC:
00000160   28 44 65 62   69 61 6E 20   36 2E 33 2E   30 2D 31 38   (Debian 6.3.0-18
```

修改后

```
00000100   55 89 E5 83   EC 28 C7 45   F0 49 6C 6F   56 C7 45 F4   U....(.E.IloV.E.
00000110   45 63 53 00   C7 45 E6 31   32 33 34 C7   45 EA 35 36   EcS..E.1234.E.56
00000120   37 38 66 C7   45 EE 39 00   83 EC 08 8D   45 E6 50 8D   78f.E.9.....E.P.
00000130   45 F0 50 E8   98 FF FF FF   83 C4 10 90   C9 C3 90 90   E.P.............
00000140   90 90 90 90   5D C3 00 00   00 00 00 00   00 00 00 00   ....]...........
00000150   32 00 49 6C   6F 56 45 63   53 00 00 47   43 43 3A 20   2.IloVEcS..GCC:
00000160   28 44 65 62   69 61 6E 20   36 2E 33 2E   30 2D 31 38   (Debian 6.3.0-18
```

实验阶段2



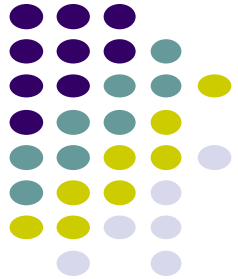
- 实验步骤4：使用调用输出函数的指令代码，替换**do_phase()**函数体中的**nop**指令，以获得期望输出

3. 重新链接、生成和运行程序，验证修改有效

```
ics@debian:~/linklab$ gcc -no-pie -o linkbomb main.o phase2.o
ics@debian:~/linklab$ ./linkbomb
123456789
```

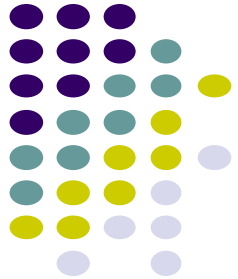
链接与ELF实验：课后实验

实验其余阶段



- 其余实验阶段作为课后实验自行完成（具体阶段以本学期课程的实验文档为准）
  - 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：符号解析**
  - **阶段4：switch语句与重定位**
  - **阶段5：可重定位目标文件**
  - **阶段6：位置无关代码（Position-Independent Code，PIC）**

实验结果提交



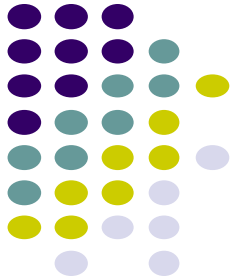
- 将实验数据包中的各目标模块文件（包括已完成修改的和未改动的）一起用**tar**工具打包并命名为“学号**.tar**”，例如（实际命令参数应与本学期实验各阶段相对应，详见实验说明文档）：

  linux> tar cf <学号>.tar main.o phase1.o phase2.o phase3.o phase3_patch.o phase4.o phase5.o phase6.o

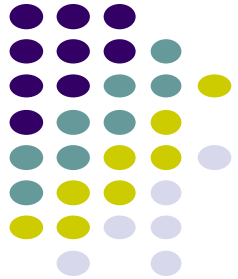
  - 注意：**TAR**文件中不要包含任何目录结构

- 提交结果**tar**文件

小结



- 本节课介绍了链接与ELF实验第2阶段“指令与ELF代码节”的基本过程和方法，并且演示了其中的主要操作步骤。从中，我们以实例分析与操作的方式，巩固了对课程中ELF文件结构与组成、二进制程序中指令代码的存储、过程调用及其参数传递等方面知识的掌握。此外，本节课将实验剩余的阶段布置为练习自行完成，并说明了实验结果的提交形式。



结 束