ELF可重定位目标文件

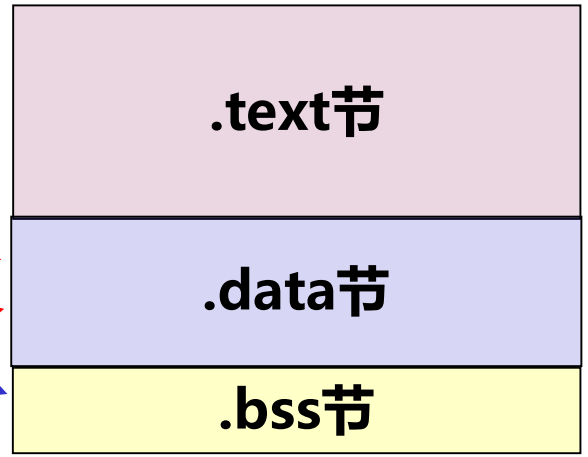
南京大学
计算机科学与技术系
袁春风
email: cfyuan@nju.edu.cn
2015.6

回顾：链接视图—可重定位目标文件

- 可被链接（合并）生成可执行文件或共享目标文件
- 静态链接库文件由若干个可重定位目标文件组成
- 包含代码、数据（已初始化全局变量和局部静态变量.data和未初始化的全局变量和局部静态变量.bss）
- 包含重定位信息（指出哪些符号引用处需要重定位）
- 文件扩展名为.o（相当于Windows中的 .obj文件）

```
int x=100;
int y;
void prn(int n)
{
   printf( "%d\n" ,n);
}

void main( )
{
    static int a=1;
    static int b;
    int i=200,j;
    prn(x+a+i);
}
```



为了进行链接，还需要其他许多信息，如符号表、重定位信息等许多其他的节（Section）

未初始化变量（.bss节）

- C语言规定：
  - 未初始化的全局变量和局部静态变量的默认初始值为0
- 将未初始化变量（.bss节）与已初始化变量（.data节）分开的好处
  - .data节中存放具体的初始值，需要占磁盘空间
  - .bss节中无需存放初始值，只要说明.bss中的每个变量将来在执行时占用几个字节即可，因此，.bss节实际上不占用磁盘空间，提高了磁盘空间利用率
- BSS（Block Started by Symbol）最初是UA-SAP汇编程序中所用的一个伪指令，用于为符号预留一块内存空间
- 所有未初始化的全局变量和局部静态变量都被汇总到.bss节中，通过专门的“节头表（Section header table）”来说明应该为.bss节预留多大的空间

可重定位目标文件格式

ELF 头

- ✓ 包括16字节标识信息、文件类型 (.o, exec, .so)、机器类型（如 IA-32）、节头表的偏移、节头表的表项大小以及表项个数

.text 节

- ✓ 编译后的代码部分

.rodata 节

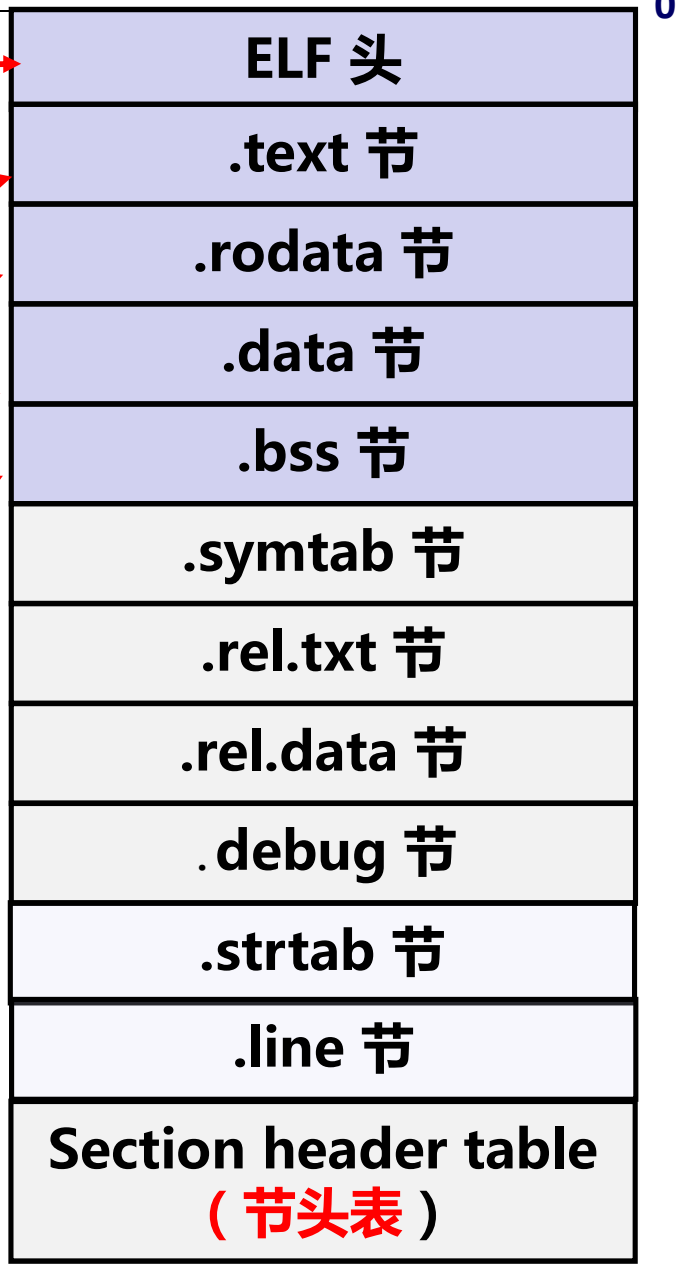
- ✓ 只读数据，如 printf 格式串、switch 跳转表等

.data 节

- ✓ 已初始化的全局变量

.bss 节

- ✓ 未初始化全局变量，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空间效率



switch-case语句举例

```
int sw_test(int a, int b, int c)
{
   int result;
   switch(a) {
   case 15:
      c=b&0x0f;
   case 10:
      result=c+50;
      break;
   case 12:
   case 17:
      result=b+50;
      break;
   case 14:
      result=b
      break;
   default:
      result=a;
   }
   return result;
}
```

```
  movl  8(%ebp), %eax
  subl  $10, %eax
  cmpl  $7, %eax
  ja    .L5
  jmp     *.L8( , %eax, 4)
.L1:
  movl  12(%ebp), %eax
  andl  $15, %eax
  movl  %eax, 16(%ebp)
.L2:
  movl  16(%ebp), %eax
  addl    $50, %eax
  jmp  .L7
.L3:
  movl  12(%ebp), %eax
  addl    $50, %eax
   jmp    .L7
.L4:
   movl  12(%ebp), %eax
   jmp  .L7
.L5:
   addl  $10, %eax
.L7:
```

R[eax]=a-10=i

if (a-10)>7 转 L5

转.L8+4*i 处的地址

跳转表在目标文件的只读数据节中，按4字节边界对齐

```
   .section   .rodata
   .align 4
.L8                    a=
   .long    .L2    10
   .long    .L5    11
   .long    .L3    12
   .long    .L5    13
   .long    .L4    14
   .long    .L1    15
   .long    .L5    16
   .long    .L3    17
```

可重定位目标文件格式

.symtab 节

- ✓ 存放函数和全局变量（符号表）信息，它不包括局部变量

.rel.text 节

- ✓ .text节的重定位信息，用于重新修改代码段的指令中的地址信息

.rel.data 节

- ✓ .data节的重定位信息，用于对被模块使用或定义的全局变量进行重定位的信息

.debug 节

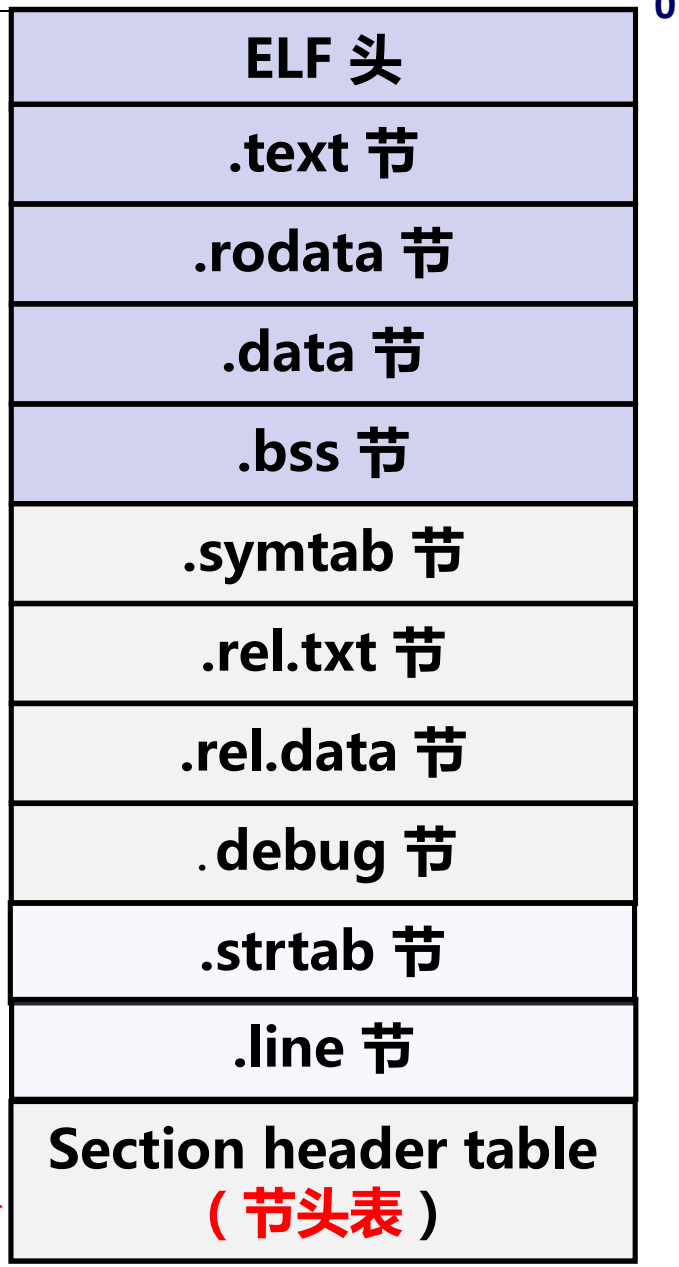
- ✓ 调试用符号表 (gcc -g)

strtab 节

- ✓ 包含symtab和debug节中符号及节名

Section header table（节头表）

- ✓ 每个节的节名、偏移和大小



ELF头（ELF Header）

- ELF头位于ELF文件开始，包含文件结构说明信息。分32位系统对应结构和64位系统对应结构（32位版本、64位版本）
- 以下是32位系统对应的数据结构

```
#define EI_NIDENT     16
typedef struct {
        unsigned char   e_ident[EI_NIDENT];
        Elf32_Half      e_type;
        Elf32_Half      e_machine;
        Elf32_Word      e_version;
        Elf32_Addr      e_entry;
        Elf32_Off       e_phoff;
        Elf32_Off       e_shoff;
        Elf32_Word      e_flags;
        Elf32_Half      e_ehsize;
        Elf32_Half      e_phentsize;
        Elf32_Half      e_phnum;
        Elf32_Half      e_shentsize;
        Elf32_Half      e_shnum;
        Elf32_Half      e_shstrndx;
} Elf32_Ehdr;
```

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、程序执行的入口地址、程序头表（段头表）的起始位置和长度、节头表的起始位置和长度等

魔数：文件开头几个字节通常用来确定文件的类型或格式

a.out的魔数：01H 07H

PE格式魔数：4DH 5AH

加载或读取文件时，可用魔数确认文件类型是否正确

ELF头信息举例

```
$ readelf -h main.o
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 0
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          516 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           40 (bytes)
  Number of section headers:         15
  Section header string table index: 12
```

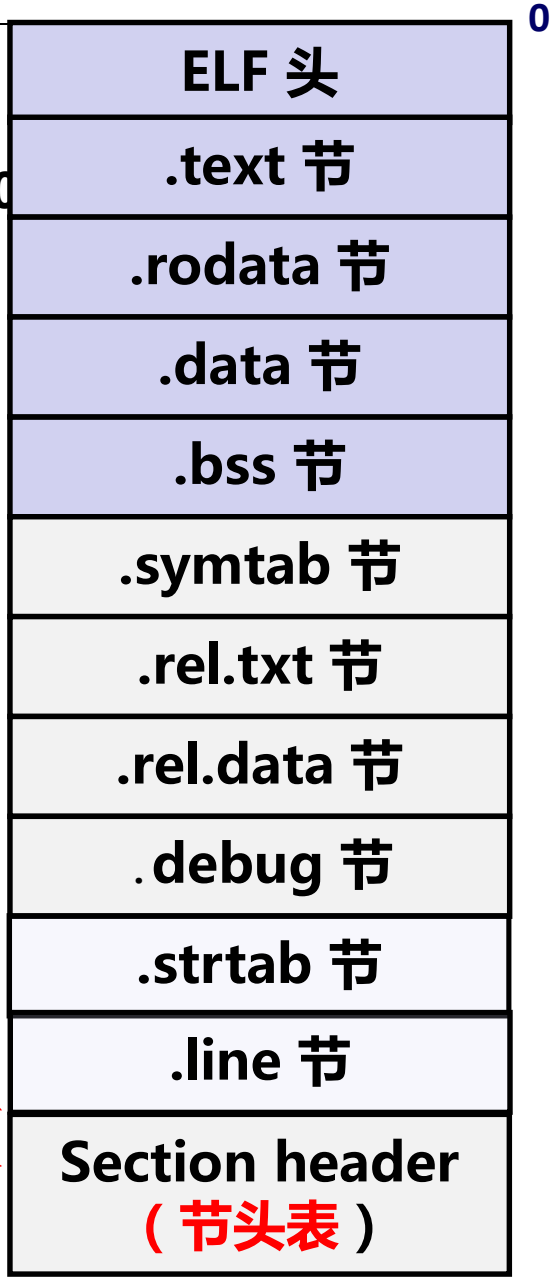
可重定位目标文件的ELF头

ELF文件的魔数

没有程序头表

15x40B

.strtab在节头表中的索引



节头表（Section Header Table）

- 除ELF头之外，节头表是ELF可重定位目标文件中最重要的部分内容
- 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32位系统对应的数据结构（每个表项占40B）

```
typedef struct {
    Elf32_Word      sh_name;    节名字符串在.strtab中的偏移
    Elf32_Word      sh_type;    节类型：无效/代码或数据/符号/字符串/...
    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节而言则无意义
    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; 节中每个表项的长度，0表示无固定长度表项
} Elf32_Shdr;
```

节头表信息举例

$ readelf -S test.o

There are 11 section headers, starting at offset 0x120:

Section Headers:

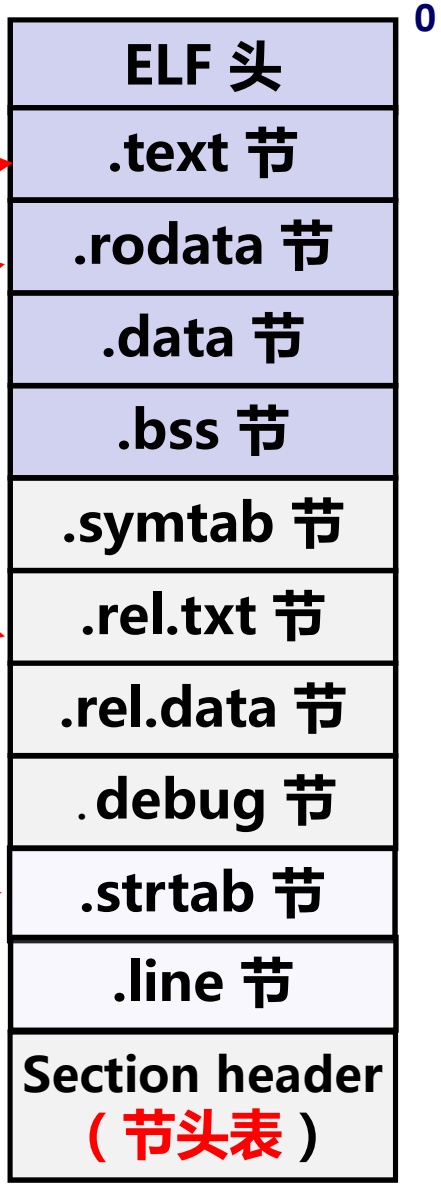
[Nr] Name	Type	Addr	Off	Size	ES	Flg	Lk	Inf	Al
[0]	NULL	00000000	000000	000000	00		0	0	0
[1] .text	PROGBITS	00000000	000034	00005b	00	AX	0	0	4
[2] .rel.text	REL	00000000	000498	000028	08		9	1	4
[3] .data	PROGBITS	00000000	000090	00000c	00	WA	0	0	4
[4] .bss	NOBITS	00000000	00009c	00000c	00	WA	0	0	4
[5] .rodata	PROGBITS	00000000	00009c	000004	00	A	0	0	1
[6] .comment	PROGBITS	00000000	0000a0	00002e	00		0	0	1
[7] .note.GNU-stack	PROGBITS	00000000	0000ce	000000	00		0	0	1
[8] .shstrtab	STRTAB	00000000	0000ce	000051	00		0	0	1
[9] .symtab	SYMTAB	00000000	0002d8	000120	10		10	13	4
[10] .strtab	STRTAB	00000000	0003f8	00009e	00		0	0	1

Key to Flags:

W (write), A (alloc), X (execute), M (merge), S (strings)

I (info), L (link order), G (group), x (unknown)

O (extra OS processing required) o (OS specific), p (processor specific)



可重定位目标文件中，每个可装入节的起始地址总是0

节头表信息举例

$ readelf -S test.o

There are 11 section headers, starting at offset 0x120:

Section Headers:

[Nr] Name	Off	Size	ES	Flg	Lk	Inf	Al
[0]	000000	000000	00		0	0	0
[1] .text	000034	00005b	00	AX	0	0	4
[2] .rel.text	000498	000028	08		9	1	4
[3] .data	000090	00000c	00	WA	0	0	4
[4] .bss	00009c	00000c	00	WA	0	0	4
[5] .rodata	00009c	000004	00	A	0	0	1
[6] .comment	0000a0	00002e	00		0	0	1
[7] .note.GNU-stack	0000ce	000000	00		0	0	1
[8] .shstrtab	0000ce	000051	00		0	0	1
[9] .symtab	0002d8	000120	10		10	13	4
[10] .strtab	0003f8	00009e	00		0	0	1

Key to Flags:

W (write), A (alloc), X (execute), M (merge), S (strings)

I (info), L (link order), G (group), x (unknown)

……

有4个节将会分配存储空间

.text：可执行

.data和.bss：可读可写

.rodata：可读

可重定位目标文件test.o的结构