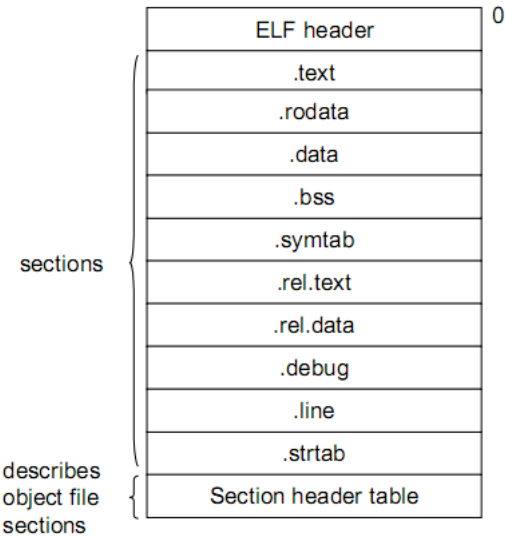


在于思考

如果一门语言不能影响你对编程的想法，那它就不值得去学。在具备基础之后，学习任何新东西，都要抓住主线，突出重点。对于关键理论的学习，要集中精力，速战速决。而旁枝末节和非本质性的知识内容，完全可以留给实践去零敲碎打。

ELF文件

1 ELF文件结构



图中显示了ELF可重定位文件的构成，ELF文件头的开始16个字节描述了文件中字的大小和字节序（大端模式还是小端模式）。文件头还包含了ELF头的大小，文件类型（可重定位，可执行和共享），机器类型，节头表的位置和大小。节头表中的每项对应于文件中的一个节，用于描述节的位置和大小。

ELF文件头：

```
root@ubuntu:~/test# readelf -h a.out
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00
  Class:                   ELF32
  Data:                     2's complement, little endian
  Version:                   1 (current)
  OS/ABI:                    UNIX - System V
  ABI Version:               0
  Type:                      EXEC (Executable file)
  Machine:                   Intel 80386
  Version:                   0x1
  Entry point address:       0x8048450
  Start of program headers:  52 (bytes into file)
  Start of section headers:  4416 (bytes into file)
  Flags:                      0x0
  Size of this header:       52 (bytes)
  Size of program headers:   32 (bytes)
  Number of program headers:  9
  Size of section headers:   40 (bytes)
  Number of section headers:  30
  Section header string table index: 27
```

ELF头对应的代码定义为：

随笔分类(153)
ARM(3)
c(17)
c++(5)
java(1)
linux(13)
linux编程(17)
linux内核(16)
Maven(1)
nginx源码分析(4)
opengl(2)
python(6)
redis源码分析(2)
vc++(12)
分布式(2)
汇编(8)
面试题目(3)
日常(6)
设计模式



```
#define EI_NIDENT    16

typedef struct elf32_hdr{
    unsigned char    e_ident[EI_NIDENT];  //开始的16个字节
    Elf32_Half       e_type;  //文件类型
    Elf32_Half       e_machine;  //运行的机器类型
    Elf32_Word       e_version;  //版本
    Elf32_Addr       e_entry;  //程序入口地址
    Elf32_Off        e_phoff;  //程序头表在文件中的偏移
    Elf32_Off        e_shoff;  //节头表在文件中的偏移
    Elf32_Word       e_flags;  //标记
    Elf32_Half       e_ehsize;  //elf文件头大小
    Elf32_Half       e_phentsize;  //程序头表项的大小
    Elf32_Half       e_phnum;  //程序头表中表项的个数
    Elf32_Half       e_shentsize;  //节头表项大小
    Elf32_Half       e_shnum;  //节头表中表项的个数
    Elf32_Half       e_shstrndx;  //节头表的字符串节所在节头表中下标
} Elf32_Ehdr;
```



节头表项对应的代码定义为：



```
typedef struct elf32_shdr {
    Elf32_Word       sh_name;  //节的名字，在符号表中的下标
    Elf32_Word       sh_type;  //节的类型，描述符号，代码，数据，重定位等
    Elf32_Word       sh_flags;  //读写执行标记
    Elf32_Addr       sh_addr;  //节在执行时的虚拟地址
    Elf32_Off        sh_offset;  //节在文件中的偏移量
    Elf32_Word       sh_size;  //节的大小
    Elf32_Word       sh_link;  //其它节的索引
    Elf32_Word       sh_info;  //节的其它信息
    Elf32_Word       sh_addralign;  //节对齐
    Elf32_Word       sh_entsize;  //节拥有固定大小项的大小
} Elf32_Shdr;
```



在ELF文件的ELF头和节头表直接是节头表描述的节本身。一个典型的重定位目标文件包含了下面的节：

.text: 机器代码

.rodata: 只读数据，例如printf的字符串常量参数，jump跳转表。

.data: 已初始化的全局C变量。本地C变量在运行程序的栈中。

.bss: 未初始化的全局变量。这个节在文件中并不占用实际的空间。

.symtab: 符号表，包含了本模块定义和引用的全局符号信息，不包含局部变量的信息。

.rel.text: .text节中需要重定位的信息。一般，任何调用外部函数或者引用全局变量的指令都需要被修改，但是，如果指令调用本地函数，则不需要被修改。重定位信息在可执行文件中可以不需要。

.rel.data: 重定位任何被这个模块定义和引用的全局变量信息。通常在一个全局变量的值是另一个全局变量地址或外部函数地址的情况下需要重新修改这个值。

.debug: 包含调试信息

.line: 在.text节中的机器指令与原始C代码所在的行之间的映射。

.strtab: 在.symtab和.debug节中所用的字符串表。

算法(30)

重温经典算法(2)

自己写操作系统(1)

综合(2)

积分与排名

积分 - 96357

排名 - 2430

最新评论

1. Re:腾讯面试小结

第一次在园子里看别人的博客，感觉很不错，希望以后我能体会当时的感觉。

2. Re:腾讯面试小结

楼主，你好，看过你的博客，感觉很不错，希望以后我能体会当时的感觉。

3. Re:连连看游戏（dfs实现）

楼主，你这分配空间都是怎么算的呀，嘿嘿

--S

4. Re:系统引导加载器的实现

@lugesot从事Linux c 方面的工作?;...

5. Re:用vmware运行操作系统

mark

阅读排行榜

1. MFC定时器(7257)

2. 地址空间分布(4454)

```
root@ubuntu:~/test# readelf -S test1.o
There are 13 section headers, starting at offset 0x12c:

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         00002e         00  AX  0  0  4
[ 2] .rel.text                           REL              00000000         000424         000020         08  11  1  4
[ 3] .data                               PROGBITS         00000000         000064         000000         00  WA  0  0  4
[ 4] .bss                               NOBITS           00000000         000064         000000         00  WA  0  0  4
[ 5] .rodata                            PROGBITS         00000000         000064         000008         00  A  0  0  1
[ 6] .comment                           PROGBITS         00000000         00006c         00002b         01  MS  0  0  1
[ 7] .note.GNU-stack                     PROGBITS         00000000         000097         000000         00  0  0  1
[ 8] .eh_frame                           PROGBITS         00000000         000098         000034         00  A  0  0  4
[ 9] .rel.eh_frame                       REL              00000000         000444         000008         08  11  8  4
[10] .shstrtab                           STRTAB           00000000         0000cc         00005f         00  0  0  1
[11] .symtab                             SYMTAB           00000000         000334         0000d0         10  12  9  4
[12] .strtab                             STRTAB           00000000         000404         00001d         00  0  0  1

Key to Flags:
W (write), A (alloc), X (execute), M (merge), S (strings)
I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
O (extra OS processing required) o (OS specific), p (processor specific)
```

程序头表项对应定义：

```
typedef struct elf32_phdr{
    Elf32_Word    p_type;    //段的类型，LOAD，DYNAMIC等
    Elf32_Off     p_offset;  //段在文件中的偏移量
    Elf32_Addr    p_vaddr;   //段的物理地址
    Elf32_Addr    p_paddr;   //段的虚拟地址
    Elf32_Word    p_filesz;  //段在文件中的大小
    Elf32_Word    p_memsz;   //段在内存中的大小
    Elf32_Word    p_flags;   //读写执行标记
    Elf32_Word    p_align;   //段的对齐
} Elf32_Phdr;
```

只有可执行文件和共享对象文件有程序头表，程序头表描述了程序中的段，以及程序文件中的节是映射到哪个段中的信息。当程序被装载进内存中时，是以段为单位进行虚拟内存地址映射的。

```
There are 9 program headers, starting at offset 52

Program Headers:
Type      Offset    VirtAddr    PhysAddr    FileSiz MemSiz  Flg Align
PHDR      0x000034  0x08048034  0x08048034  0x00120 0x00120 R E  0x4
INTERP    0x000154  0x08048154  0x08048154  0x00013 0x00013 R   0x1
  [Requesting program interpreter: /lib/ld-linux.so.2]
LOAD      0x000000  0x08048000  0x08048000  0x0070c 0x0070c R E  0x1000
LOAD      0x000f0c 0x08049f0c  0x08049f0c  0x0010c 0x00118 RW  0x1000
DYNAMIC   0x000f20  0x08049f20  0x08049f20  0x000d0 0x000d0 RW  0x4
NOTE      0x000168  0x08048168  0x08048168  0x00044 0x00044 R   0x4
GNU_EH_FRAME 0x000618  0x08048618  0x08048618  0x00034 0x00034 R   0x4
GNU_STACK 0x000000  0x00000000  0x00000000  0x00000 0x00000 RW  0x4
GNU_RELRO 0x000f0c  0x08049f0c  0x08049f0c  0x000f4 0x000f4 R   0x1

Section to Segment mapping:
Segment Sections...
00
01 .interp
02 .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r
.rel.dyn .rel.plt .init .plt .text .fini .rodata .eh_frame_hdr .eh_frame
03 .ctors .dtors .jcr .dynamic .got .got.plt .data .bss
04 .dynamic
05 .note.ABI-tag .note.gnu.build-id
06 .eh_frame_hdr
07
08 .ctors .dtors .jcr .dynamic .got
```

2 符号和符号表

每个重定位目标模块m都包含一个符号表来描述被m定义和引用的符号。在链接器上下文中，有以下三种符号：

- 1. 全局链接符号。模块m定义的，可以被其它模块引用。全局链接符号包括非静态的函数和非静态的全局变量。
- 2. 外部链接符号。被模块m引用但定义在其它模块的符号，可以是其它函数定义的全局链接符号。
- 3. 本地链接符号。只能被模块m内部使用，本地链接符号包括静态函数和静态变量（包括静态全局变量和静态局部变量）。

本地链接符号和本地程序符号是不同的，本地链接符号在.symtab表中出现，当本地程序符号不会出现在.symtab表中，而是出现在程序运行时栈中。

需要注意的是静态成员变量和全局变量并不会出现在程序运行时栈中，编译器会在.data或则.bss段分配空间给它们。

符号表是汇编器根据编译器编译到汇编语言中的符号建立的。符号表包含在节.symtab中，它是一个数组，数组的元素定义如下：

3. 腾讯面试小结(4096)

4. vmware安装win7提
drive to use:GCDROM
20)

5. 静态编译mysql库到
(2739)

评论排行榜

1. 腾讯面试小结(26)

2. MapReduce的理解(

3. win7下debug的常用

4. 平衡负载（2013年晋
竞赛题目一）(6)

5. 读《linux内核完全注

推荐排行榜


1. 腾讯面试小结(10)

2. 系统引导加载器的简


3. 读《linux内核完全注

4. 使用cmake自动构建

5. nginx源码分析之网



```
typedef struct elf32_sym{
    Elf32_Word    st_name;    //符号名字，在字符串表中的下标
    Elf32_Addr    st_value;    //符号在节的偏移量，或者虚拟地址
    Elf32_Word    st_size;    //对象的字节大小
    unsigned char    st_info;    //类型和绑定属性，变量，函数，节，源文件名等类型，全局或则局部属性，低四位表示类型，高四位表示属性
    unsigned char    st_other;    //没有使用
    Elf32_Half    st_shndx;    //符号所在的节在节头表中的下标
} Elf32_Sym;
```



st_shndx表示符号所在节在节头表中的下标，但存在三个特殊的虚拟节，这些节在节头表中并没有表项，如COMMON是表示未初始化的数据对象，UNDEF表示符号是未定义的，ABS表示符号不应该被重定位，就是用绝对地址表示。

```
root@ubuntu:~/test# readelf -s test1.o

Symbol table '.symtab' contains 14 entries:
   Num:      Value           Size Type      Bind   Vis      Ndx Name
   ---:      -
   0: 00000000             0 NOTYPE   LOCAL   DEFAULT UND
   1: 00000000             0 FILE    LOCAL   DEFAULT ABS test1.c
   2: 00000000             0 SECTION LOCAL   DEFAULT 1
   3: 00000000             0 SECTION LOCAL   DEFAULT 3
   4: 00000000             0 SECTION LOCAL   DEFAULT 4
   5: 00000000             4 OBJECT  LOCAL   DEFAULT 4 b
   6: 00000000             0 SECTION LOCAL   DEFAULT 5
   7: 00000004             4 OBJECT  LOCAL   DEFAULT 4 a.1687
   8: 00000000             0 SECTION LOCAL   DEFAULT 7
   9: 00000000             0 SECTION LOCAL   DEFAULT 8
  10: 00000000             0 SECTION LOCAL   DEFAULT 6
  11: 00000000            46 FUNC     GLOBAL  DEFAULT 1 main
  12: 00000000             0 NOTYPE   GLOBAL  DEFAULT UND printf
  13: 00000000             0 NOTYPE   GLOBAL  DEFAULT UND sleep
```

3 解决全局符号多重定义

在编译的时候，编译器会告诉汇编器每个全局符号是强符号还是弱符号，汇编器则将这些信息放入重定位文件对象的符号表中。函数和初始化的全局变量是强符号，未初始化的全局变量是弱符号。

在给出了符号的强弱表示后，链接器就根据下面的规则来处理符号的多重定义：

- 1. 多个强符号是不允许的
- 2. 有一个强符号和多个弱符号，选择强符号
- 3. 有多个弱符号，选择其中的任意一个

分类： linux编程

好文要顶

关注我

收藏该文



 在于思考
关注 - 0
粉丝 - 124

+加关注

20 0

« 上一篇: UCOSII内核代码分析
» 下一篇: ELF静态链接

posted @ 2013-12-14 19:50 在于思考 阅读(1390) 评论(1) 编辑 收藏

评论列表

#1楼 2013-12-14 22:36 pandaren

mark

支持(0) 反对(0)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问](#)网站首页。

- 【推荐】50万行VC++源码：大型组态工控、电力仿真CAD与GIS源码库
- 【推荐】开源大咖专业分享，一起感受开源之力！
- 【推荐】融云即时通讯云－豆果美食、Faceu等亿级APP都在用
- 【推荐】Google+GitHub联手打造前端工程师课程
- 【活动】一元专享1500元微软智能云Azure



最新IT新闻：

- Google宣布攻破SHA-1加密：证明哈希值可与PDF文件内容冲突
 - 百度交出史上最难成绩单：营收增长陷入瓶颈 成本不减压缩净利
 - 顺丰完成借壳上市：民营快递从市场竞争走到资本对决
 - 谷歌用AI技术识别恶意评论 助新闻机构大战嘴炮党
 - 联想终于出手：Motorola彻底死亡！
- » 更多新闻...



最新知识库文章：

- 程序员的沟通之痛
 - 技术文章如何写作才能有较好的阅读体验
 - 「代码家」的学习过程和学习经验分享
 - 写给未来的程序媛
 - 高质量的工程代码为什么难写
- » 更多知识库文章...