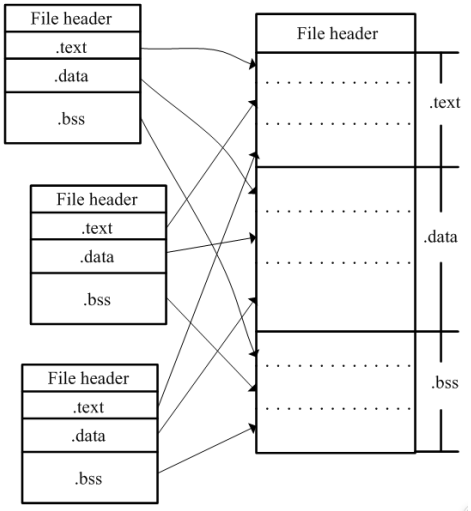
ELF静态链接

一直对ELF目标文件是怎样链接成可执行文件感到比较的疑惑，ELF文件里面的重定位段是怎样解决符号引用问题的？前几天偶然看了《深入理解计算机系统》里面讲了这个问题，看了之后对里面的实现机制终于有了一定的理解。

当有链接器链接多个可重定位的共享对象时，共享对象时怎样合并的呢？很简单，链接器将相同类型的节合并在一起，比如将所有输入文件的.text合并到输出文件的.text段，接着是.data段，.bss段等。



链接器扫描所有的输入目标文件，并且获得它们各个节的长度，属性和位置，并将输入目标文件中的符号表中所有的符号定义和符号引用收集起来，统一放到一个全局符号表中。链接器能够获得所有输入目标文件的节长度，并将它们合并，并计算出输出文件中各个节合并后的段长度和位置，并建立段和节之间的映射关系。也就是说，在将输入文件的各个节映射到段中后，输入文件中的各个节在链接后的虚拟地址就已经确定了，那么全局符号表中的符号地址就可以知道了。

知道了定义符号的虚拟地址，合并了各个节，是不是就大功告成了？显然没有这么简单，由于输入给链接器的文件都是可重定位的目标文件，这些目标文件中引用符号的地方存放的地址肯定不是最终的虚拟地址，因为这个时候符号地址还不确定。那么链接器在知道了符号的各个虚拟地址后怎样来修改引用符号的地址为实际的符号虚拟地址呢？这个工作是通过重定位目标文件中的重定位表来实现的。

对于每个要重定位的ELF节都有一个对应的重定位表，而一个重定位表往往就是ELF文件的一个节。比如代码节.text有要重定位的地方，那么会有一个相对应的.rel.text的节保存.text的重定位表；如果数据节.data有要被重定位的地方，也会有一个相对应的叫.rel.data的节与之对应。

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

Relocation section '.rel.text' at offset 0x44c contains 4 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
0000000b  00000401 R_386_32          00000000   .bss
00000010  00000601 R_386_32          00000000   .rodata
0000001c  00000d02 R_386_PC32        00000000   printf
00000028  00000e02 R_386_PC32        00000000   sleep
```

随笔分类(152)

ARM(3)

c(17)

c++(4)

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

每个要被重定位的地方叫做一个重定位入口（Relocation Entry），重定位入口的偏移（Offset）表示该入口在要被重定位的节中的位置。重定位表的结构很简单，它是一个Elf32_Rel结构的数组，每个数组元素对应一个重定位入口。Elf32_Rel定义如下：

```
typedef struct elf32_rel {
  Elf32_Addr    r_offset;
  Elf32_Word    r_info;
} Elf32_Rel;
```

r_offset表示重定位入口的偏移。对于可重定位文件来说，这个值是该重定位入口所要修正的位置的第一个字节相对于节起始的偏移；对于可执行文件或共享对象文件来说，这个值是该重定位入口所要修正的位置的第一个字节的虚拟地址。

r_info表示重定位入口的类型和符号。这个成员的高8位表示重定位入口的类型，低24位表示重定位入口的符号在符号表中的下标。

- R_386_PC32：重定位一个使用32位pc相对地址的引用。
- R_386_32：重定位使用32位绝对地址的引用。

链接器中根据重定位表来修改符号引用地址的算法是：

```
1 foreach section s {
2     foreach relocation entry r {
3         refptr = s + r.offset;  /* ptr to reference to be relocated */
4
5         /* relocate a PC-relative reference */
6         if (r.type == R_386_PC32) {
7             refaddr = ADDR(s) + r.offset; /* ref's runtime address */
8             *refptr = (unsigned) (ADDR(r.symbol) + *refptr - refaddr);
9         }
10
11        /* relocate an absolute reference */
12        if (r.type == R_386_32)
13            *refptr = (unsigned) (ADDR(r.symbol) + *refptr);
14    }
15 }
```

第一行迭代每个节，第二行迭代这个节中的每一个重定位表项，为了简单，我们假设节是一个字节数组，每个重定位表项都是Elf32_Rel类型，并假设当这个算法运行时，链接器也已经知道了每个节的运行时地址（定义为ADDR（s））和每个符号的运行时地址（定义为ADDR（r.symbol））。

根据重定位入口的类型，需要分两个情况来修改引用地址。

重定位**PC**相对引用

假设函数printf在main函数被调用，现在来看下它在main函数中是怎么被调用的：

```
1b:   e8 fc ff ff ff          call   7 <main+0x7>
```

我们可以看到call指令在节的偏移量为0x7，这条指令由1个字节的操作码0xe8和32比特的引用0xfffffffc（-4）组成。我们也可以看到对这个引用的重定位项：

```
r.offset = 0x7
r.symbol = printf
r.type = R_386_PC32
```

重定位的项告诉链接器修改32位PC相对引用在节的偏移量为0x7，现在假设链接器已经决定了：

```
ADDR(s) = ADDR(.text) = 0x80483b4
ADDR(r.symbol) = ADDR(printf) = 0x80483c8
```

使用前面的算法来计算引用的运行时地址：

```
refaddr = ADDR(s) + r.offset
        = 0x80483b4 + 0x7
        = 0x80483bb
```

修改引用符号地址的值，使得引用指向printf运行时的地址：

```
*refptr = (unsigned) (ADDR(r.symbol) + *refptr - refaddr)
        = (unsigned)(0x80483c8 + (-4) - 0x80483bb)
        = (unsigned)(0x9)
```

这样，printf的重定位值就为0x9。当程序运行时，由于call指令时相对pc寻址的，所以计算出的printf的实际地址为：

```
pc + 0x9 = 0x80483bf + 0x9 = 0x80483c8
```

重定位绝对地址引用

假设定义了下面一个全局变量：

```
int *bufp0 = &buf[0]
```

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 - 96230
排名 - 2431

最新评论
1. Re:腾讯面试小结
第一次在园子里看别人的
时候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
了，请问你在腾讯哪个部
3. Re:连连看游戏（dfs
目】
楼主，你这分配空间都是
呀，嘿嘿
--s
4. Re:系统引导加载器的
@lugesot从事Linux c
方面的工作?;...
5. Re:用vmware运行简
mark

阅读排行榜
1. MFC定时器(7255)
2. 地址空间分布(4451)

由于bufp0是一个初始化了的数据对象，它会被存放在.data节中。由于它被初始化为全局变量buf[0]的地址，因此bufp0需要被重定位。下面是bufp0在.data节中的反汇编代码：

```
00000000<bufp0>
   0:    00 00 00 00
```

我们可以看到.data节中包含了一个简单的32位的引用，指针bufp0的值位0x0。重定位表项告诉链接器这是一个32位的绝对地址引用，相对于.data节的偏移量为0。它必须被重定位以指向buf[0]的地址。假设链接器已经决定了：

```
ADDR(r.symbol) = ADDR(buf) = 0x8049454
```

链接器将根据上面的算法更新这个引用：

```
*refptr = (unsigned) (ADDR(r.symbol) + *refptr)
        = (unsigned) 0x8049454 + 0)
        = (unsigned)(0x8049454)
```

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

1 0

« 上一篇：ELF文件
» 下一篇：链接器怎样使用静态库来解决符号引用

posted @ 2013-12-18 11:18 在于思考 阅读(729) 评论(0) 编辑 收藏

刷新评论 刷新页面 返回顶部

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
【直播】支付宝、微博、阿里云专家联合解读红包浪潮下的核心技术架构
【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：
· 李开复：透过多次失败，我描绘出了最受VC追捧的创业者画像
· 万众期待，Google云计算平台终于支持云端GPU加速服务！
· 张勇湖畔大学授课：阿里历史上几个重要决策及背后思考
· AI利用现有代码学会自己写程序
· 微博把你对它的每一个槽点，都做成了赢利点
» 更多新闻...

3. 腾讯面试小结(4075)
4. vmware安装win7提
drive to use:GCDRON
03)
5. 静态编译mysql库到
(2738)

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年百
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源码分析之网络

最新知识库文章:
· 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
· 循序渐进地代码重构
» 更多知识库文章...

Copyright ©2017 在于思考