elf文件类型一 文件格式

来源: ChinaUnix博客 日期： 2008.01.28 15:12 (共有条评论) 我要评论

文件格式
Object文件参与程序的联接(创建一个程序)和程序的执行(运行一个程序)。
object 文件格式提供了一个方便有效的方法并行的视角看待文件的内容，
在他们的活动中，反映出不同的需要。例 1-1图显示了一个object文件的
组织图。
+ 图1-1: Object文件格式

Linking 视角	Execution 视角
ELF header	ELF header
Program header table (optional)	Program header table
Section 1	Segment 1
...	Segment 2
Section n	...
Section header table	Section header table (optional)

一个ELF头在文件的开始，保存了路线图(road map)，描述了该文件的组织情况。
sections保存着object 文件的信息，从连接角度看：包括指令，数据，
符号表，重定位信息等等。特别sections的描述会出项在以后的第一部分。
第二部分讨论了段和从程序的执行角度看文件。
假如一个程序头表（program header table）存在，那么它告诉系统如何来创建一
个进程的内存映象。被用来建立进程映象(执行一个程序)的文件必须要有一个程
序头表（program header table）；可重定位文件不需要这个头表。一个
section头表（section header table）包含了描述文件sections的信息。每个
section在这个表中有一个入口；每个入口给出了该section的名字，大小，
等等信息。在联接过程中的文件必须有一个section头表；其他object文件可要
可不要这个section头表。
注意：虽然图显示出程序头表立刻出现在一个ELF头后，section头表跟着其他
section部分出现，事实是的文件是可以不同的。此外，sections和段(segments)
没有特别的顺序。只有ELF头（elf header）是在文件的固定位置。
数据表示
object文件格式支持8位、32位不同的处理器。不过，它试图努力的在更大
或更小的体系上运行。因此，object文件描绘一些控制数据需要用与机器
无关的格式，使它尽可能的用一般的方法甄别object文件和描述他们的内容。
在object文件中剩余的数据使用目标处理器的编码方式，不管文件是在哪台
机子上创建的。
+ 图 1-2: 32-Bit Data Types

Name	Size	Alignment	Purpose
Elf32_Addr	4	4	Unsigned program address
Elf32_Half	2	2	Unsigned medium integer
Elf32_Off	4	4	Unsigned file offset
Elf32_Sword	4	4	Signed large integer
Elf32_Word	4	4	Unsigned large integer
unsigned char	1	1	Unsigned small integer

所有的object文件格式定义的数据结构是自然大小(natural size),为相关
的类型调整指针。如果需要，数据结构中明确的包含了确保4字节对齐的填
充字段。来使结构大小是4的倍数。数据从文件的开始也有适当的对齐。
例如，一个包含了Elf32_Addr成员的结构将会在文件内对齐到4字节的边界上。
因为移植性的原因，ELF不使用位字段。
========== ELF Header ==========
一些object文件的控制结构能够增长的，所以ELF头包含了他们目前的大小。
假如object文件格式改变，程序可能会碰到或大或小他们不希望的控制结构。

搜索

最新资讯 更多>>
· 谷歌劝说诺基亚采用Android操作..
· Apache 基金会确认退出 JCP 执..
· Chrome 10 新功能探秘：新增GP..
· 金山宣布开源其安全软件
· 女黑客在开源会议上抱受骚扰
· 21款值得关注的Linux游戏
· 马化腾：腾讯半年后彻底转型，..
· [多图] Chrome OS 预发布版本多..
· Lubuntu 11.04 默认应用抢先一览
· Red Hat宣布收购云计算软件提供..

论坛热点 更多>>
· do_execve时候用户栈中参数的..
· swapinfo -atm 问题
· Linux 的优点简述
· VM虚拟机上得Red Hat Linux上..
· 我看成了上海男人喜欢女人毛..
· 校车展览，看了你就知道
· 在遇到他之前，唯一需要做的..
· GRUB的疑问
· 从来没有人真正付足书价——..
· 云存储 vs 网盘

文档更新 更多>>
· orcale queue
· 谁可以推荐几本经典的操作系统的..
· 【北京】某物联网公司招云计算应..
· 【北京】某物联网公司招云计算应..
· 谁能推荐几本关于操作系统的书
· 如何添加网络接口eth1
· 葡萄牙语入门教材的选取与经验分享
· 葡萄牙语就业前景分析
· 葡萄牙语学习经验交流
· Ш

程序也有可能忽略额外（extra）的信息。
对待来历不明(missing)的信息依靠上下文来解释，假如扩展被定义，它们
将会被指定。

+ 图 1-3: ELF Header

```
#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;
```

* e_ident

这个最初的字段标示了该文件为一个object文件，提供了一个机器无关
的数据，解释文件的内容。在下面的ELF的鉴别（ELF Identification）
部分有更详细的信息。

* e_type

该成员确定该object的类型。

Name	Value	Meaning
ET_NONE	0	No file type
ET_REL	1	Relocatable file
ET_EXEC	2	Executable file
ET_DYN	3	Shared object file
ET_CORE	4	Core file
ET_LOPROC	0xff00	Processor-specific
ET_HIPROC	0xffff	Processor-specific

虽然CORE的文件内容未被指明，类型ET_CORE是保留的。
值从 ET_LOPROC 到 ET_HIPROC(包括ET_HIPROC)是为特殊的处理器保留的。
如有需要，其他保留的变量将用在新的object文件类型上。

* e_machine

该成员变量指出了运行该程序需要的体系结构。

Name	Value	Meaning
EM_NONE	0	No machine
EM_M32	1	AT&T WE 32100
EM_SPARC	2	SPARC
EM_386	3	Intel 80386
EM_68K	4	Motorola 68000
EM_88K	5	Motorola 88000
EM_860	7	Intel 80860
EM_MIPS	8	MIPS RS3000

如有需要，其他保留的值将用到新的机器类型上。特殊处理器名使用机器名来
区别他们。例如，下面将要被提到的成员flags使用前缀EF_;在一台EM_XYZ机器
上，flag称为WIDGET，那么就称为EF_XYZ_WIDGET。

* e_version

这个成员确定object文件的版本。

Name	Value	Meaning
EV_NONE	0	Invalid version
EV_CURRENT	1	Current version

值1表示原来的文件格式；创建新版本就用>1的数。EV_CURRENT值(上面给
出为1)如果需要将指向当前的版本号。

* e_entry

该成员是系统第一个传输控制的虚拟地址，在那启动进程。假如文件没有
如何关联的入口点，该成员就保持为0。

* e_phoff

该成员保持着程序头表（program header table）在文件中的偏移量(以字节计数)。
假如该文件没有程序头表的的话，该成员就保持为0。

* e_shoff

该成员保持着section头表（section header table）在文件中的偏移量(以字节计数)。假如该文件没有section头表的的话，该成员就保持为0。

* e_flags
该成员保存着相关文件的特定处理器标志。
flag的名字来自于EF__。看下机器信息"Machine Information"部分的flag的定义。

* e_ehsize
该成员保存着ELF头大小(以字节计数)。

* e_phentsize
该成员保存着在文件的程序头表（program header table）中一个入口的大小(以字节计数)。所有的入口都是同样的大小。

* e_phnum
该成员保存着在程序头表中入口的个数。因此，e_phentsize和e_phnum的乘机就是表的大小(以字节计数).假如没有程序头表（program header table），e_phnum变量为0。

* e_shentsize
该成员保存着section头的大小(以字节计数)。一个section头是在section头表(section header table)的一个入口；所有的入口都是同样的大小。

* e_shnum
该成员保存着在section header table中的入口数目。因此，e_shentsize和e_shnum的乘积就是section头表的大小(以字节计数)。
假如文件没有section头表，e_shnum值为0。

* e_shstrndx
该成员保存着跟section名字字符表相关入口的section头表(section header table)索引。假如文件中没有section名字字符表，该变量值为SHN_UNDEF。
更详细的信息 看下面"Sections"和字符串表("String Table") 。

ELF 鉴别(Identification)

在上面提到的，ELF提供了一个object文件的框架结构来支持多种处理机，多样的数据编码方式，多种机器类型。为了支持这个object文件家族，最初的几个字节指定用来说明如何解释该文件，独立于处理器，与文件剩下的内容无关。ELF头(也就是object文件)最初的几个字节与成员e_ident相一致。

+ 图 1-4: e_ident[] Identification Indexes

Name	Value	Purpose
EI_MAG0	0	File identification
EI_MAG1	1	File identification
EI_MAG2	2	File identification
EI_MAG3	3	File identification
EI_CLASS	4	File class
EI_DATA	5	Data encoding
EI_VERSION	6	File version
EI_PAD	7	Start of padding bytes
EI_NIDENT	16	Size of e_ident[]

通过索引访问字节，以下的变量被定义。

* EI_MAG0 to EI_MAG3
文件的前4个字符保存着一个魔术数(magic number)，用来确定该文件是否为ELF的目标文件。

Name	Value	Position
ELFMAG0	0x7f	e_ident[EI_MAG0]
ELFMAG1	'E'	e_ident[EI_MAG1]
ELFMAG2	'L'	e_ident[EI_MAG2]
ELFMAG3	'F'	e_ident[EI_MAG3]

* EI_CLASS
接下来的字节是e_ident[EI_CLASS]，用来确定文件的类型或者说是能力。

Name	Value	Meaning
ELFCLASSNONE	0	Invalid class
ELFCLASS32	1	32-bit objects
ELFCLASS64	2	64-bit objects

文件格式被设计成在不同大小机器中可移植的，在小型机上的不用大型机上的尺寸。类型ELFCLASS32支持虚拟地址空间最大可达4GB的机器；使用上面定义过的基本类型。
类型ELFCLASS64为64位体系的机器保留。它的出现表明了object文件可能改变，但是64位的格式还没有被定义。如果需要，其他类型将被定义，会有不同的类型和不同大小的数据尺寸。

* EI_DATA
字节e_ident[EI_DATA]指定了在object文件中特定处理器数据的编码方式。当前定义了以下编码方式。

Name	Value	Meaning
ELFDATANONE	0	Invalid data encoding
ELFDATA2LSB	1	See below
ELFDATA2MSB	2	See below

更多的关于编码的信息出现在下面。其他值保留，将被分配一个新的编码方式，当然如果必要的话。

* EI_VERSION

字节e_ident[EI_VERSION]表明了ELF头的版本号。
现在这个变量的是一定要设为EV_CURRENT，作为上面e_version的解释。

* EI_PAD

该变量标识了在e_ident中开始的未使用的字节。那些字节保留并被设置为0；程序把它们从object 文件中读出但应该忽略。假如当前未被使用的字节有了新的定义，EI_PAD变量将来会被改变。

一个文件的数据编码指出了如何来解释一个基本的object文件。在上述的描述中，类型ELFCLAS32文件使用占用1，2和4字节的目标文件。下面定义的编码方式，用下面的图来描绘。数据出现在左上角。

ELFDATA2LSB编码指定了2的补数值。
最小有意义的字节占有最低的地址。

+ 图1-5: Data Encoding ELFDATA2LSB

```
          0------+
    0x0102  | 01 |
          +------+
          0------1------+
   0x010204  | 02 | 01 |
          +------+------+
          0------1------2------3------+
 0x01020304  | 04 | 03 | 02 | 01 |
          +------+------+------+------+
```

ELFDATA2LSB编码指定了2的补数值。
最大有意义的字节占有最低的地址。

+ 图1-6: Data Encoding ELFDATA2MSB

```
          0------+
    0x0102  | 01 |
          +------+
          0------1------+
   0x010204  | 01 | 02 |
          +------+------+
          0------1------2------3------+
 0x01020304  | 01 | 02 | 03 | 04 |
          +------+------+------+------+
```

机器信息

为了确定文件，32位Intel体系结构的需要以下的变量。

+ 图1-7: 32-bit Intel Architecture Identification, e_ident

Position	Value
e_ident[EI_CLASS]	ELFCLASS32
e_ident[EI_DATA]	ELFDATA2LSB

处理器确认ELF头里的e_machine成员，该成员必须为EM_386。
ELF报头里的e_flags成员保存了和文件相关的位标记。32位Intel体系上未定义该标记；所以这个成员应该为0；

本文来自**ChinaUnix**博客，如果查看原文请点：http://blog.chinaunix.net/u/22754/showart_472557.html