ELF 中 symbol 的解析

前言

本文根据以前的笔记整理，分析了ELF格式可执行文件中symbol相关的数据结构及对其进行解释。本文是ELF Specification的学习笔记，最权威的信息来源当然是《**Executable and Linkable Format (ELF)**》。

例子源代码

本源代码没有任何特殊含义，当时分析时，随便找了一个我学习 Unix 编程时的代码。

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>

int main(int argc, char** argv)
{
    int z;
    int s[2];

    z = socketpair(AF_LOCAL, SOCK_STREAM, 0, s);
    if( z == -1)
    {
        fprintf(stderr, "%s: socketpair(AF_LOCAL, SOCK_STREAM, 0)\n", strerror(errno));
        return 1;
    }

    printf("s[0] = %d;\n", s[0]);
    printf("s[1] = %d;\n", s[1]);
    //system("netstat --unix -p");
```

```
    int d = dup(s[0]);
    shutdown(s[0], SHUT_RDWR);
    z = write(d, "Hello", 5);

    printf("We could reach here\n");

    if(z < 0)
    {
        printf("Error\n");
        return 1;
    }
    char buf[90];
    z = read(s[1], buf, sizeof(buf));
    if(z < 0)
    {
        printf("Error 2\n");
        return 1;
    }
    buf[z] = 0;
    printf("Receieve '%s' on s[1]", buf);


    return 0;
}
```

可执行文件中**symbol**

```
[wzhou@dcmp10 ~]$ readelf -S test
There are 35 section headers, starting at offset 0x1248:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        08048114 000114 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            08048128 000128 000020 00   A  0   0  4
  [ 3] .hash             HASH            08048148 000148 000050 04   A  4   0  4
  [ 4] .dynsym           DYNSYM          08048198 000198 0000f0 10   A  5   1  4
  [ 5] .dynstr           STRTAB          08048288 000288 0000a5 00   A  0   0  1
  [ 6] .gnu.version      VERSYM          0804832e 00032e 00001e 02   A  4   0  2
  [ 7] .gnu.version_r    VERNEED         0804834c 00034c 000020 00   A  5   1  4
  [ 8] .rel.dyn          REL             0804836c 00036c 000010 08   A  4   0  4
  [ 9] .rel.plt          REL             0804837c 00037c 000050 08   A  4  11  4
  [10] .init             PROGBITS        080483cc 0003cc 000017 00  AX  0   0  4
  [11] .plt              PROGBITS        080483e4 0003e4 0000b0 04  AX  0   0  4
  [12] .text             PROGBITS        08048494 000494 0002e0 00  AX  0   0  4
  [13] .fini             PROGBITS        08048774 000774 00001a 00  AX  0   0  4
  [14] .rodata           PROGBITS        08048790 000790 00008b 00   A  0   0  4
  [15] .eh_frame         PROGBITS        0804881c 00081c 000004 00   A  0   0  4
  [16] .ctors            PROGBITS        08049820 000820 000008 00  WA  0   0  4
```

```
  [17] .dtors            PROGBITS        08049828 000828 000008 00  WA  0   0  4
  [18] .jcr              PROGBITS        08049830 000830 000004 00  WA  0   0  4
  [19] .dynamic          DYNAMIC         08049834 000834 0000c8 08  WA  5   0  4
  [20] .got              PROGBITS        080498fc 0008fc 000004 04  WA  0   0  4
  [21] .got.plt          PROGBITS        08049900 000900 000034 04  WA  0   0  4
  [22] .data             PROGBITS        08049934 000934 00000c 00  WA  0   0  4
  [23] .bss              NOBITS          08049940 000940 000008 00  WA  0   0  4
  [24] .comment          PROGBITS        00000000 000940 000126 00      0   0  1
  [25] .debug_aranges    PROGBITS        00000000 000a66 000020 00      0   0  1
  [26] .debug_pubnames   PROGBITS        00000000 000a86 00001b 00      0   0  1
  [27] .debug_info       PROGBITS        00000000 000aa1 0004a8 00      0   0  1
  [28] .debug_abbrev     PROGBITS        00000000 000f49 0000e8 00      0   0  1
  [29] .debug_line       PROGBITS        00000000 001031 000091 00      0   0  1
  [30] .debug_frame      PROGBITS        00000000 0010c4 000038 00      0   0  4
  [31] .debug_str        PROGBITS        00000000 0010fc 000016 00      0   0  1
  [32] .shstrtab         STRTAB          00000000 001112 000134 00      0   0  1
  [33] .symtab           SYMTAB          00000000 0017c0 000560 10     34  51  4
  [34] .strtab           STRTAB          00000000 001d20 0002e5 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)
[wzhou@dcmp10 ~]$
```

从上面的输出中可以看到两个 symbol table。

1. dynsym
2. symtab

其中.dynsym section 中的 symbol 是被动态链接器用的，同时其用到的 symbol 的字符串也在.dynstr section 中；而.symtab section 中的 symbol 则是普通情况下应用的，即该程序执行时等（大部分 debugger 都要应用到）。

下面的结构摘自/usr/include/elf.h

```
typedef struct
{
  Elf32_Word    st_name;       /* Symbol name (string tbl index) */
  Elf32_Addr    st_value;      /* Symbol value */
  Elf32_Word    st_size;       /* Symbol size */
  unsigned char st_info;       /* Symbol type and binding */
  unsigned char st_other;      /* Symbol visibility */
  Elf32_Section st_shndx;      /* Section index */
} Elf32_Sym;
```

ELF Specification 中对各个成员的解释如下：

st_name This member holds an index into the object file’s symbol string table, which holds the character representations of the symbol names. If the value is non-zero, it represents a string table index that gives the symbol name. Otherwise, the symbol table entry has no name.

st_value This member gives the value of the associated symbol. Depending on the context, this may be an absolute value, an address, etc.; details appear below.

st_size Many symbols have associated sizes. For example, a data object’s size is the number of bytes contained in the object. This member holds 0 if the symbol has no size or an unknown size.

st_info This member specifies the symbol's type and binding attributes. A list of the values and meanings appears below. The following code shows how to manipulate the values.

```
/* How to extract and insert information held in the st_info field.  */

#define ELF32_ST_BIND(val)     (((unsigned char) (val)) >> 4)
#define ELF32_ST_TYPE(val)     ((val) & 0xf)
#define ELF32_ST_INFO(bind, type)   (((bind) << 4) + ((type) & 0xf))
```

st_other This member currently holds 0 and has no defined meaning.
st_shndx Every symbol table entry is ''defined'' in relation to some section; this member holds the relevant section header table index. As Figure 1-7 and the related text describe, some section indexes indicate special meanings.
A symbol's binding determines the linkage visibility and behavior.

```
/* Legal values for ST_BIND subfield of st_info (symbol binding).  */

 #define STB_LOCAL   0      /* Local symbol */
 #define STB_GLOBAL  1      /* Global symbol */
 #define STB_WEAK    2      /* Weak symbol */
 #define STB_NUM     3      /* Number of defined types.  */
 #define STB_LOOS    10     /* Start of OS-specific */
 #define STB_HIOS    12     /* End of OS-specific */
 #define STB_LOPROC  13     /* Start of processor-specific */
 #define STB_HIPROC  15     /* End of processor-specific */
```

```
/* Legal values for ST_TYPE subfield of st_info (symbol type).  */

#define STT_NOTYPE  0      /* Symbol type is unspecified */
#define STT_OBJECT  1      /* Symbol is a data object */
#define STT_FUNC    2      /* Symbol is a code object */
#define STT_SECTION 3      /* Symbol associated with a section */
#define STT_FILE    4      /* Symbol's name is file name */
#define STT_COMMON  5      /* Symbol is a common data object */
#define STT_TLS     6      /* Symbol is thread-local data object*/
#define STT_NUM     7      /* Number of defined types.  */
#define STT_LOOS    10     /* Start of OS-specific */
#define STT_HIOS    12     /* End of OS-specific */
#define STT_LOPROC  13     /* Start of processor-specific */
#define STT_HIPROC  15     /* End of processor-specific */
```

.dynsym section中的symbol

dump .dynsym section 的内容

```
[wzhou@dcmp10 ~]$ hexdump -C -s 0x198 -n 240 test
00000198  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001a8  37 00 00 00 00 00 00 00  7c 00 00 00 12 00 00 00  |7.......|.......|
000001b8  3d 00 00 00 00 00 00 00  21 00 00 00 12 00 00 00  |=.......!.......|
000001c8  2e 00 00 00 00 00 00 00  ad 00 00 00 12 00 00 00  |................|
000001d8  69 00 00 00 00 00 00 00  36 00 00 00 12 00 00 00  |i.......6.......|
000001e8  4a 00 00 00 00 00 00 00  39 00 00 00 12 00 00 00  |J.......9.......|
000001f8  57 00 00 00 40 99 04 08  04 00 00 00 11 00 17 00  |W...@...........|
00000208  89 00 00 00 00 00 00 00  ef 00 00 00 12 00 00 00  |................|
00000218  3e 00 00 00 00 00 00 00  36 00 00 00 12 00 00 00  |>.......6.......|
00000228  53 00 00 00 00 00 00 00  36 00 00 00 12 00 00 00  |S.......6.......|
00000238  7a 00 00 00 94 87 04 08  04 00 00 00 11 00 0e 00  |z...............|
00000248  01 00 00 00 00 00 00 00  00 00 00 00 20 00 00 00  |............ ...|
00000258  5e 00 00 00 00 00 00 00  39 00 00 00 12 00 00 00  |^.......9.......|
00000268  45 00 00 00 00 00 00 00  7c 00 00 00 12 00 00 00  |E.......|.......|
00000278  15 00 00 00 00 00 00 00  00 00 00 00 20 00 00 00  |............ ...|
```

上面每一行即是一个 Elf32_Sym 结构。比如

例一

```
000001b8  3d 00 00 00 00 00 00 00  21 00 00 00 12 00 00 00  |=.......!.......|
```

st_name = 0x3d，指向下面.dynstr section 中的“fprintf”字符串。
st_value = 0
st_size = 0x21
st_info = 0x12 = 0001 0010，即 bind = 1 (STB_GLOBAL), type = 2 (STT_FUNC)
st_other = 0 (目前总是 0)
st_shndx = 0 （SHN_UNDEF），Undefined section

“fprintf”是 glibc 库的函数，代码在/usr/lib/libc-2.3.4.so 中

```
[wzhou@dcmp10 ~]$ nm /lib/libc-2.3.4.so | grep " fprintf$"
00043840 T fprintf
```

```
[wzhou@dcmp10 ~]$ objdump -d  /lib/libc-2.3.4.so | less
00043840 <fprintf>:
   43840:      55                    push   %ebp
   43841:      89 e5                 mov    %esp,%ebp
   43843:      83 ec 0c              sub    $0xc,%esp
   43846:      8b 55 0c              mov    0xc(%ebp),%edx
   43849:      8d 4d 10              lea    0x10(%ebp),%ecx
   4384c:      8b 45 08              mov    0x8(%ebp),%eax
   4384f:      89 4c 24 08           mov    %ecx,0x8(%esp)
   43853:      89 54 24 04           mov    %edx,0x4(%esp)
   43857:      89 04 24              mov    %eax,(%esp)
```

```
  4385a:      e8 d1 71 ff ff          call   3aa30 <_IO_vfprintf>
  4385f:      c9                    leave
  43860:      c3                    ret
  43861:      90                    nop
  43862:      90                    nop
  43863:      90                    nop
  43864:      90                    nop
```

上面 st_size = 0x21 表示该 symbol 所代表的“东西”(这里是函数)的大小。该函数从 0x43840 开始，到 0x43861 结束，正好 0x21 个 bytes。

dump .dynstr section 内容

```
[wzhou@dcmp10 ~]$ hexdump -C -s 0x288 -n 165 test
00000288  00 5f 4a 76 5f 52 65 67  69 73 74 65 72 43 6c 61  |._Jv_RegisterCla|
00000298  73 73 65 73 00 5f 5f 67  6d 6f 6e 5f 73 74 61 72  |sses.__gmon_star|
000002a8  74 5f 5f 00 6c 69 62 63  2e 73 6f 2e 36 00 73 74  |t__.libc.so.6.st|
000002b8  72 65 72 72 6f 72 00 77  72 69 74 65 00 66 70 72  |rerror.write.fpr|
000002c8  69 6e 74 66 00 72 65 61  64 00 73 68 75 74 64 6f  |intf.read.shutdo|
000002d8  77 6e 00 64 75 70 00 73  74 64 65 72 72 00 73 6f  |wn.dup.stderr.so|
000002e8  63 6b 65 74 70 61 69 72  00 5f 5f 65 72 72 6e 6f  |cketpair.__errno|
000002f8  5f 6c 6f 63 61 74 69 6f  6e 00 5f 49 4f 5f 73 74  |_location._IO_st|
00000308  64 69 6e 5f 75 73 65 64  00 5f 5f 6c 69 62 63 5f  |din_used.__libc_|
00000318  73 74 61 72 74 5f 6d 61  69 6e 00 47 4c 49 42 43  |start_main.GLIBC|
00000328  5f 32 2e 30 00                                    |_2.0.|
```

再以下面的一个 symbol 为例

例二

```
00000238  7a 00 00 00 94 87 04 08  04 00 00 00 11 00 0e 00  |z...............|
```

st_name = 0x7a，指向上面.dynstr section 中的“_IO_stdin_used”字符串。
st_value = 0x08048794，即该 symbol 所代表的变量的地址。
st_size = 0x4，该 symbol 所代表的变量的大小为 4 个 byte。
st_info = 0x11 = 0001 0001，即 bind = 1 (STB_GLOBAL), type = 1 (STT_OBJECT)
st_other = 0 (目前总是 0)
st_shndx = 0x0e，即 section table 中的第 14 项。

```
[14] .rodata         PROGBITS        08048790 000790 00008b 00   A  0   0  4
```

该 section 从地址 0x08048790 开始，大小为 0x8b。而该 symbol 所代表的变量地址为 0x08048794，显然在.rodata section 内。
上面的 symbol 有点象在 C 语言的代码里定义了如下：

```
const int IO_stdin_used = 0;
```

变量 IO_stdin_used 被汇编后，一般会在前面加上“_”，所以 linker 看到的名称为“_IO_stdin_used”。该变量的类型是 int 型，在 32 位 CPU 上其占用 4 个 byte。由于是 const 型，并被初始化，所以该变量被放到 read only data 的数据段。当 programmer 在 debugger 中要显示变量“IO_stdin_used”中的值时，debugger 就是通过 symbol table 中的信息来找到对应的地址 0x08048794 的。

.symtab section中的symbol

dump .symtab section 中的内容

```
[wzhou@dcmp10 ~]$ readelf -S test | grep symtab
  [33] .symtab         SYMTAB          00000000 0017c0 000560 10    34  51  4
```

```
[wzhou@dcmp10 ~]$ hexdump -C -s 0x17c0 -n 1376 test
000017c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000017d0  00 00 00 00 14 81 04 08  00 00 00 00 03 00 01 00  |................|
000017e0  00 00 00 00 28 81 04 08  00 00 00 00 03 00 02 00  |....(...........|
000017f0  00 00 00 00 48 81 04 08  00 00 00 00 03 00 03 00  |....H...........|
00001800  00 00 00 00 98 81 04 08  00 00 00 00 03 00 04 00  |................|
00001810  00 00 00 00 88 82 04 08  00 00 00 00 03 00 05 00  |................|
00001820  00 00 00 00 2e 83 04 08  00 00 00 00 03 00 06 00  |................|
00001830  00 00 00 00 4c 83 04 08  00 00 00 00 03 00 07 00  |....L...........|
00001840  00 00 00 00 6c 83 04 08  00 00 00 00 03 00 08 00  |....l...........|
00001850  00 00 00 00 7c 83 04 08  00 00 00 00 03 00 09 00  |....|...........|
00001860  00 00 00 00 cc 83 04 08  00 00 00 00 03 00 0a 00  |................|
00001870  00 00 00 00 e4 83 04 08  00 00 00 00 03 00 0b 00  |................|
00001880  00 00 00 00 94 84 04 08  00 00 00 00 03 00 0c 00  |................|
00001890  00 00 00 00 74 87 04 08  00 00 00 00 03 00 0d 00  |....t...........|
000018a0  00 00 00 00 90 87 04 08  00 00 00 00 03 00 0e 00  |................|
000018b0  00 00 00 00 1c 88 04 08  00 00 00 00 03 00 0f 00  |................|
```

```
000018c0  00 00 00 00 20 98 04 08  00 00 00 00 03 00 10 00  |.... ...........|
000018d0  00 00 00 00 28 98 04 08  00 00 00 00 03 00 11 00  |....(...........|
000018e0  00 00 00 00 30 98 04 08  00 00 00 00 03 00 12 00  |....0...........|
000018f0  00 00 00 00 34 98 04 08  00 00 00 00 03 00 13 00  |....4...........|
00001900  00 00 00 00 fc 98 04 08  00 00 00 00 03 00 14 00  |................|
00001910  00 00 00 00 00 99 04 08  00 00 00 00 03 00 15 00  |................|
00001920  00 00 00 00 34 99 04 08  00 00 00 00 03 00 16 00  |....4...........|
00001930  00 00 00 00 40 99 04 08  00 00 00 00 03 00 17 00  |....@...........|
00001940  00 00 00 00 00 00 00 00  00 00 00 00 03 00 18 00  |................|
00001950  00 00 00 00 00 00 00 00  00 00 00 00 03 00 19 00  |................|
00001960  00 00 00 00 00 00 00 00  00 00 00 00 03 00 1a 00  |................|
00001970  00 00 00 00 00 00 00 00  00 00 00 00 03 00 1b 00  |................|
00001980  00 00 00 00 00 00 00 00  00 00 00 00 03 00 1c 00  |................|
00001990  00 00 00 00 00 00 00 00  00 00 00 00 03 00 1d 00  |................|
000019a0  00 00 00 00 00 00 00 00  00 00 00 00 03 00 1e 00  |................|
000019b0  00 00 00 00 00 00 00 00  00 00 00 00 03 00 1f 00  |................|
000019c0  00 00 00 00 00 00 00 00  00 00 00 00 03 00 20 00  |.............. .|
000019d0  00 00 00 00 00 00 00 00  00 00 00 00 03 00 21 00  |..............!.|
000019e0  00 00 00 00 00 00 00 00  00 00 00 00 03 00 22 00  |..............".|
000019f0  01 00 00 00 b8 84 04 08  00 00 00 00 02 00 0c 00  |................|
00001a00  11 00 00 00 00 00 00 00  00 00 00 00 04 00 f1 ff  |................|
00001a10  1c 00 00 00 20 98 04 08  00 00 00 00 01 00 10 00  |.... ...........|
00001a20  2a 00 00 00 28 98 04 08  00 00 00 00 01 00 11 00  |*...(...........|
00001a30  38 00 00 00 30 98 04 08  00 00 00 00 01 00 12 00  |8...0...........|
00001a40  45 00 00 00 3c 99 04 08  00 00 00 00 01 00 16 00  |E...<...........|
00001a50  49 00 00 00 44 99 04 08  01 00 00 00 01 00 17 00  |I...D...........|
```

```
00001a60  55 00 00 00 dc 84 04 08  00 00 00 00 02 00 0c 00  |U...............|
00001a70  6b 00 00 00 10 85 04 08  00 00 00 00 02 00 0c 00  |k...............|
00001a80  11 00 00 00 00 00 00 00  00 00 00 00 04 00 f1 ff  |................|
00001a90  77 00 00 00 24 98 04 08  00 00 00 00 01 00 10 00  |w...$...........|
00001aa0  84 00 00 00 2c 98 04 08  00 00 00 00 01 00 11 00  |....,...........|
00001ab0  91 00 00 00 1c 88 04 08  00 00 00 00 01 00 0f 00  |................|
00001ac0  9f 00 00 00 30 98 04 08  00 00 00 00 01 00 12 00  |....0...........|
00001ad0  ab 00 00 00 50 87 04 08  00 00 00 00 02 00 0c 00  |....P...........|
00001ae0  c1 00 00 00 00 00 00 00  00 00 00 00 04 00 f1 ff  |................|
00001af0  c8 00 00 00 34 98 04 08  00 00 00 00 11 00 13 00  |....4...........|
00001b00  d1 00 00 00 00 00 00 00  7c 00 00 00 12 00 00 00  |........|.......|
00001b10  e2 00 00 00 90 87 04 08  04 00 00 00 11 00 0e 00  |................|
00001b20  e9 00 00 00 00 00 00 00  21 00 00 00 12 00 00 00  |........!.......|
00001b30  fc 00 00 00 20 98 04 08  00 00 00 00 10 02 f1 ff  |.... ...........|
00001b40  0d 01 00 00 00 00 00 00  ad 00 00 00 12 00 00 00  |................|
00001b50  21 01 00 00 38 99 04 08  00 00 00 00 11 02 16 00  |!...8...........|
00001b60  2e 01 00 00 0c 87 04 08  42 00 00 00 12 00 0c 00  |........B.......|
00001b70  3e 01 00 00 00 00 00 00  36 00 00 00 12 00 00 00  |>.......6.......|
00001b80  5a 01 00 00 00 00 00 00  39 00 00 00 12 00 00 00  |Z.......9.......|
00001b90  6e 01 00 00 cc 83 04 08  00 00 00 00 12 00 0a 00  |n...............|
00001ba0  74 01 00 00 40 99 04 08  04 00 00 00 11 00 17 00  |t...@...........|
00001bb0  86 01 00 00 94 84 04 08  00 00 00 00 12 00 0c 00  |................|
00001bc0  8d 01 00 00 20 98 04 08  00 00 00 00 10 02 f1 ff  |.... ...........|
00001bd0  a0 01 00 00 b8 86 04 08  52 00 00 00 12 00 0c 00  |........R.......|
00001be0  b0 01 00 00 40 99 04 08  00 00 00 00 10 00 f1 ff  |....@...........|
00001bf0  bc 01 00 00 3c 85 04 08  7a 01 00 00 12 00 0c 00  |....<...z.......|
```

```
00001c00  c1 01 00 00 00 00 00 00  ef 00 00 00 12 00 00 00  |................|
00001c10  de 01 00 00 20 98 04 08  00 00 00 00 10 02 f1 ff  |.... ............|
00001c20  ef 01 00 00 34 99 04 08  00 00 00 00 20 00 16 00  |....4....... ...|
00001c30  fa 01 00 00 00 00 00 00  36 00 00 00 12 00 00 00  |........6.......|
00001c40  0c 02 00 00 74 87 04 08  00 00 00 00 12 00 0d 00  |....t...........|
00001c50  12 02 00 00 20 98 04 08  00 00 00 00 10 02 f1 ff  |.... ............|
00001c60  26 02 00 00 40 99 04 08  00 00 00 00 10 00 f1 ff  |&...@...........|
00001c70  2d 02 00 00 00 99 04 08  00 00 00 00 11 00 15 00  |-...............|
00001c80  43 02 00 00 48 99 04 08  00 00 00 00 10 00 f1 ff  |C...H...........|
00001c90  48 02 00 00 00 00 00 00  36 00 00 00 12 00 00 00  |H.......6.......|
00001ca0  57 02 00 00 20 98 04 08  00 00 00 00 10 02 f1 ff  |W... ............|
00001cb0  6a 02 00 00 94 87 04 08  04 00 00 00 11 00 0e 00  |j...............|
00001cc0  79 02 00 00 34 99 04 08  00 00 00 00 10 00 16 00  |y...4...........|
00001cd0  86 02 00 00 00 00 00 00  00 00 00 00 20 00 00 00  |............ ...|
00001ce0  9a 02 00 00 00 00 00 00  39 00 00 00 12 00 00 00  |........9.......|
00001cf0  b0 02 00 00 20 98 04 08  00 00 00 00 10 02 f1 ff  |.... ............|
00001d00  c6 02 00 00 00 00 00 00  7c 00 00 00 12 00 00 00  |........|.......|
00001d10  d6 02 00 00 00 00 00 00  00 00 00 00 20 00 00 00  |............ ...|
```

在.symtab section引用到的字符串在.strtab section中。

例一

在该symbol table中有很多无名的symbol，其没有字符串与某个地址关联；有的甚至无名也无地址对应，纯粹占用表中一项。比如

```
000017d0  00 00 00 00 14 81 04 08  00 00 00 00 03 00 01 00  |................|
```

该 symbol 没有名字，st_name = 0；但该 symbol 有地址对应，st_value = 0x08048114；该 symbol 没有大小，st_size = 0，可能只是个 label 吧；st_info = 0x03 = 0000 0011 = STB_LOCAL STT_SECTION，即该 symbol 只是指向一个 section。st_shndx = 1，表示指向.interp section。从下面的输出

```
Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        08048114 000114 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            08048128 000128 000020 00   A  0   0  4
  [ 3] .hash             HASH            08048148 000148 000050 04   A  4   0  4
  [ 4] .dynsym           DYNSYM          08048198 000198 0000f0 10   A  5   1  4
  [ 5] .dynstr           STRTAB          08048288 000288 0000a5 00   A  0   0  1
```

可看出该 symbol 实际上就指向.interp section 在被载入内存后的地址。但不知道派什么用处。

例二

```
000019f0  01 00 00 00 b8 84 04 08  00 00 00 00 02 00 0c 00  |................|
```

st_name = 0x01，指向下面.strtab section 中的“call_gmon_start”字符串。

st_value = 0x080484b8

st_size = 0

st_info = 0x02 = 0000 0010，即 bind = 0 (STB_LOCAL), type = 2 (STT_FUNC)

st_other = 0 (目前总是 0)

st_shndx = 0x0c，即.text section。

表明该 symbol 所代表的是 call_gmon_start 函数的首地址，其地址为 0x080484b8。

```
objdump -d test | less

080484b8 <call_gmon_start>:
 80484b8:      55                   push   %ebp
 80484b9:      89 e5                mov    %esp,%ebp
 80484bb:      53                   push   %ebx
 80484bc:      e8 00 00 00 00         call   80484c1 <call_gmon_start+0x9>
 80484c1:      5b                   pop    %ebx
 80484c2:      81 c3 3f 14 00 00      add    $0x143f,%ebx
 80484c8:      52                   push   %edx
 80484c9:      8b 83 fc ff ff ff      mov    0xfffffffc(%ebx),%eax
 80484cf:      85 c0                test   %eax,%eax
 80484d1:      74 02                je     80484d5 <call_gmon_start+0x1d>
 80484d3:      ff d0                call   *%eax
 80484d5:      58                   pop    %eax
 80484d6:      5b                   pop    %ebx
 80484d7:      c9                   leave
 80484d8:      c3                   ret
```

这里显然该 symbol 不关心它的大小，所以 st_size = 0。

用工具看 symbol

其实要看 ELF 可执行文件的 symbol 并不需要如我上面那样基于 hex dump 的来查看，GNU 的 binutil 工具链中的 readelf 提供了察看的方法。

```
[wzhou@dcmp10 ~]$ readelf -s test

Symbol table '.dynsym' contains 15 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000   124 FUNC    GLOBAL DEFAULT  UND write@GLIBC_2.0 (2)
     2: 00000000    33 FUNC    GLOBAL DEFAULT  UND fprintf@GLIBC_2.0 (2)[1]
     3: 00000000   173 FUNC    GLOBAL DEFAULT  UND strerror@GLIBC_2.0 (2)
     4: 00000000    54 FUNC    GLOBAL DEFAULT  UND __errno_location@GLIBC_2.0 (2)
     5: 00000000    57 FUNC    GLOBAL DEFAULT  UND shutdown@GLIBC_2.0 (2)
     6: 08049940     4 OBJECT  GLOBAL DEFAULT   23 stderr@GLIBC_2.0 (2)
     7: 00000000   239 FUNC    GLOBAL DEFAULT  UND __libc_start_main@GLIBC_2.0 (2)
     8: 00000000    54 FUNC    GLOBAL DEFAULT  UND printf@GLIBC_2.0 (2)
     9: 00000000    54 FUNC    GLOBAL DEFAULT  UND dup@GLIBC_2.0 (2)
    10: 08048794     4 OBJECT  GLOBAL DEFAULT   14 _IO_stdin_used[2]
    11: 00000000     0 NOTYPE  WEAK   DEFAULT  UND _Jv_RegisterClasses
    12: 00000000    57 FUNC    GLOBAL DEFAULT  UND socketpair@GLIBC_2.0 (2)
    13: 00000000   124 FUNC    GLOBAL DEFAULT  UND read@GLIBC_2.0 (2)
    14: 00000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__

Symbol table '.symtab' contains 86 entries:
```

[1]在.dynsym section 的例一中分析的就是这一行

[2]在.dynsym section 的例二中分析的就是这一行

```
  Num:    Value  Size Type    Bind   Vis      Ndx Name
    0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
    1: 08048114     0 SECTION LOCAL  DEFAULT    1[3]
    2: 08048128     0 SECTION LOCAL  DEFAULT    2
    3: 08048148     0 SECTION LOCAL  DEFAULT    3
    4: 08048198     0 SECTION LOCAL  DEFAULT    4
    5: 08048288     0 SECTION LOCAL  DEFAULT    5
    6: 0804832e     0 SECTION LOCAL  DEFAULT    6
    7: 0804834c     0 SECTION LOCAL  DEFAULT    7
    8: 0804836c     0 SECTION LOCAL  DEFAULT    8
    9: 0804837c     0 SECTION LOCAL  DEFAULT    9
   10: 080483cc     0 SECTION LOCAL  DEFAULT   10
   11: 080483e4     0 SECTION LOCAL  DEFAULT   11
   12: 08048494     0 SECTION LOCAL  DEFAULT   12
   13: 08048774     0 SECTION LOCAL  DEFAULT   13
   14: 08048790     0 SECTION LOCAL  DEFAULT   14
   15: 0804881c     0 SECTION LOCAL  DEFAULT   15
   16: 08049820     0 SECTION LOCAL  DEFAULT   16
   17: 08049828     0 SECTION LOCAL  DEFAULT   17
   18: 08049830     0 SECTION LOCAL  DEFAULT   18
   19: 08049834     0 SECTION LOCAL  DEFAULT   19
   20: 080498fc     0 SECTION LOCAL  DEFAULT   20
   21: 08049900     0 SECTION LOCAL  DEFAULT   21
   22: 08049934     0 SECTION LOCAL  DEFAULT   22
   23: 08049940     0 SECTION LOCAL  DEFAULT   23
```

[3] 在.symtab section 中的例一分析的就是这一行

```
   24: 00000000     0 SECTION LOCAL  DEFAULT   24
   25: 00000000     0 SECTION LOCAL  DEFAULT   25
   26: 00000000     0 SECTION LOCAL  DEFAULT   26
   27: 00000000     0 SECTION LOCAL  DEFAULT   27
   28: 00000000     0 SECTION LOCAL  DEFAULT   28
   29: 00000000     0 SECTION LOCAL  DEFAULT   29
   30: 00000000     0 SECTION LOCAL  DEFAULT   30
   31: 00000000     0 SECTION LOCAL  DEFAULT   31
   32: 00000000     0 SECTION LOCAL  DEFAULT   32
   33: 00000000     0 SECTION LOCAL  DEFAULT   33
   34: 00000000     0 SECTION LOCAL  DEFAULT   34
   35: 080484b8     0 FUNC    LOCAL  DEFAULT   12 call_gmon_start[4]
   36: 00000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
   37: 08049820     0 OBJECT  LOCAL  DEFAULT   16 __CTOR_LIST__
   38: 08049828     0 OBJECT  LOCAL  DEFAULT   17 __DTOR_LIST__
   39: 08049830     0 OBJECT  LOCAL  DEFAULT   18 __JCR_LIST__
   40: 0804993c     0 OBJECT  LOCAL  DEFAULT   22 p.0
   41: 08049944     1 OBJECT  LOCAL  DEFAULT   23 completed.1
   42: 080484dc     0 FUNC    LOCAL  DEFAULT   12 __do_global_dtors_aux
   43: 08048510     0 FUNC    LOCAL  DEFAULT   12 frame_dummy
   44: 00000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
   45: 08049824     0 OBJECT  LOCAL  DEFAULT   16 __CTOR_END__
   46: 0804982c     0 OBJECT  LOCAL  DEFAULT   17 __DTOR_END__
   47: 0804881c     0 OBJECT  LOCAL  DEFAULT   15 __FRAME_END__
   48: 08049830     0 OBJECT  LOCAL  DEFAULT   18 __JCR_END__
```

[4] 在.symtab section 中的例二分析的就是这一行

```
   49: 08048750    0 FUNC    LOCAL  DEFAULT   12 __do_global_ctors_aux
   50: 00000000    0 FILE    LOCAL  DEFAULT  ABS test.c
   51: 08049834    0 OBJECT  GLOBAL DEFAULT   19 _DYNAMIC
   52: 00000000  124 FUNC    GLOBAL DEFAULT  UND write@@GLIBC_2.0
   53: 08048790    4 OBJECT  GLOBAL DEFAULT   14 _fp_hw
   54: 00000000   33 FUNC    GLOBAL DEFAULT  UND fprintf@@GLIBC_2.0
   55: 08049820    0 NOTYPE  GLOBAL HIDDEN  ABS __fini_array_end
   56: 00000000  173 FUNC    GLOBAL DEFAULT  UND strerror@@GLIBC_2.0
   57: 08049938    0 OBJECT  GLOBAL HIDDEN   22 __dso_handle
   58: 0804870c   66 FUNC    GLOBAL DEFAULT   12 __libc_csu_fini
   59: 00000000   54 FUNC    GLOBAL DEFAULT  UND __errno_location@@GLIBC_2
   60: 00000000   57 FUNC    GLOBAL DEFAULT  UND shutdown@@GLIBC_2.0
   61: 080483cc    0 FUNC    GLOBAL DEFAULT   10 _init
   62: 08049940    4 OBJECT  GLOBAL DEFAULT   23 stderr@@GLIBC_2.0
   63: 08048494    0 FUNC    GLOBAL DEFAULT   12 _start
   64: 08049820    0 NOTYPE  GLOBAL HIDDEN  ABS __fini_array_start
   65: 080486b8   82 FUNC    GLOBAL DEFAULT   12 __libc_csu_init
   66: 08049940    0 NOTYPE  GLOBAL DEFAULT  ABS __bss_start
   67: 0804853c  378 FUNC    GLOBAL DEFAULT   12 main
   68: 00000000  239 FUNC    GLOBAL DEFAULT  UND __libc_start_main@@GLIBC_
   69: 08049820    0 NOTYPE  GLOBAL HIDDEN  ABS __init_array_end
   70: 08049934    0 NOTYPE  WEAK   DEFAULT   22 data_start
   71: 00000000   54 FUNC    GLOBAL DEFAULT  UND printf@@GLIBC_2.0
   72: 08048774    0 FUNC    GLOBAL DEFAULT   13 _fini
   73: 08049820    0 NOTYPE  GLOBAL HIDDEN  ABS __preinit_array_end
   74: 08049940    0 NOTYPE  GLOBAL DEFAULT  ABS _edata
```

```
   75: 08049900     0 OBJECT  GLOBAL DEFAULT   21 _GLOBAL_OFFSET_TABLE_
   76: 08049948     0 NOTYPE  GLOBAL DEFAULT  ABS _end
   77: 00000000    54 FUNC    GLOBAL DEFAULT  UND dup@@GLIBC_2.0
   78: 08049820     0 NOTYPE  GLOBAL HIDDEN  ABS __init_array_start
   79: 08048794     4 OBJECT  GLOBAL DEFAULT   14 _IO_stdin_used
   80: 08049934     0 NOTYPE  GLOBAL DEFAULT   22 __data_start
   81: 00000000     0 NOTYPE  WEAK   DEFAULT  UND _Jv_RegisterClasses
   82: 00000000    57 FUNC    GLOBAL DEFAULT  UND socketpair@@GLIBC_2.0
   83: 08049820     0 NOTYPE  GLOBAL HIDDEN  ABS __preinit_array_start
   84: 00000000   124 FUNC    GLOBAL DEFAULT  UND read@@GLIBC_2.0
   85: 00000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__
```

列出了.dynsym 与 .symtab 两个 section 中的 symbol。当然其列出的信息都是通过象我上面一样的分析来得到的。

在 GNU binutil 工具链中的 strip 能够剥除 ELF 文件中的部分 symbol(.symtab section),但.dynsym section 中的依然保留。因为.symtab section 中的 symbol 对程序的运行没有任何影响，它只是为调试而设的。但.dynsym section 中的 symbol 是动态链接器在载入该可执行文件时会用到的，没有它，载入会失败。

环境

系统

```
[wzhou@dcmp10 ~]$ uname -a
Linux dcmp10 2.6.9-5.ELsmp #1 SMP Wed Jan 5 19:30:39 EST 2005 i686 i686 i386 GNU/Linux
```

编译工具

```
[wzhou@dcmp10 ~]$ gcc -v
Reading specs from /usr/lib/gcc/i386-redhat-linux/3.4.3/specs
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --enable-shared
--enable-threads=posix --disable-checking --with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions
--enable-java-awt=gtk --host=i386-redhat-linux
Thread model: posix
gcc version 3.4.3 20041212 (Red Hat 3.4.3-9.EL4)
```

binutil

```
[wzhou@dcmp10 ~]$ readelf -v
GNU readelf 2.15.92.0.2 20040927
Copyright 2004 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License.  This program has absolutely no warranty.
```

联系

Walter Zhou

mailto:z-l-dragon@hotmail.com