ELF格式可执行程序的代码嵌入技术

本文通过对Linux的ELF格式可执行程序的介绍，详细描述了将代码嵌入目标文件的方法。

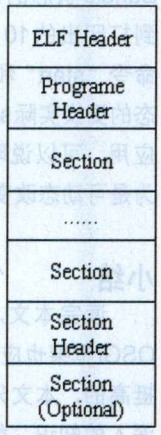
■文/杨广翔

简介

在我们的嵌入式Linux开发过程中，为了对多个目标程序进行测试，我们需要在目标程序中插入与测试有关的初始化代码。但是，我们没有目标程序的源代码，因此，必须直接在目标文件中修改。本文详细介绍了这一过程。

ELF格式介绍

可执行目标文件由ELF头部，包含多个单项的程序头部（Programe Header）、多个节区（Section）以及包含多个单项的节区头部（Section Header）所组成。下面简要介绍各个部分的位置和组成关系。



从目标文件的静态视图来看，ELF目标文件的结构如左图所示。

ELF头部为固定的52字节。它标明了程序头部相对文件起始点的偏移量和程序头部中单项的个数，同时也标明了节区头部的偏移量和个数。该头部的定义为：

```
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
```

各个字段的含义见下表：

字段	含义
e_ident	目标文件标识
e_type	目标文件类型
e_machine	目标文件所属体系结构，如80386
e_version	目标文件版本
e_entry	程序入口地址
e_phoff	程序头部的偏移量
e_shoff	节区头部的偏移量
e_flags	特定处理器标志
e_ehsize	ELF头部的大小
e_phentsize	单个程序头部的大小
e_phnum	目标文件中程序头部的个数
e_shentsize	单个节区头部的大小
e_shnum	目标文件中节区头部的个数
e_shstrndx	节区名称字符串所在节区的索引号

在ELF头部之后是程序头部。程序头部由多个单项组成。每个单项的大小由上表所示的e_phentsize确定。通常为固定的32字节。每个单项的结构是相同的。它的结构定义如下：

```
typedef struct
{
      Elf32_Word             p_type;
      Elf32_Off              p_offset;
      Elf32_Addr             p_vaddr;
      Elf32_Addr             p_paddr;
      Elf32_Word             p_filesz;
```

```
    Elf32_Word          p_memsz;
    Elf32_Word          p_flags;
    Elf32_Word          p_align;
} Elf32_Phdr;
```

结构中各个字段的含义见下表：

字段	含义
p_type	段类型
p_offset	属于该段的节区的偏移量
p_vaddr	段在内存中的虚拟地址
p_paddr	段在内存中的物理地址
p_filesz	属于该段的节区的总长度（可能包含多个节区）
p_memsz	段在内存中占用的大小
p_flags	段的标志
p_align	段的对齐方式

程序头部包含的每个单项指示了可执行的目标文件在运行时的一个段。p_offset和p_filesz指明了哪些节区属于某个段。一个段可能会包含多个节区。程序头部描述了目标文件动态状态下内部结构。

节区占用文件中的一个连续字节区域。它包含了目标文件的所有信息。例如，数据，代码，调试信息，等等。可执行目标文件会有多个节区。它们在程序头部之后，节区头部之前。也有部分节区会在节区头部之后。

节区头部位于节区之后，与程序头部一样，由多个单项组成。每个单项定义了一个节区。它们的结构也均是相同的。结构定义如下：

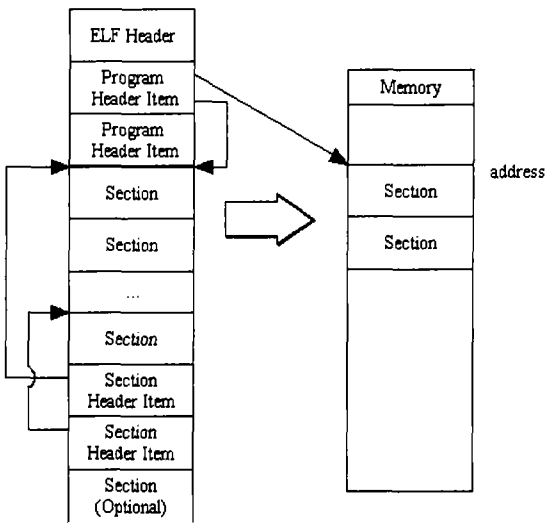
```
typedef struct
{
  Elf32_Word            sh_name;
  Elf32_Word            sh_type;
  Elf32_Word            sh_flags;
  Elf32_Addr            sh_addr;
  Elf32_Off             sh_offset;
  Elf32_Word            sh_size;
  Elf32_Word            sh_link;
  Elf32_Word            sh_info;
  Elf32_Word            sh_addralign;
  Elf32_Word            sh_entsize;
} Elf32_Shdr;
```

结构中各个字段的含义见下表：

字段	含义
sh_name	节区名称
sh_type	节区类型
sh_flags	节区的标志
sh_addr	节区出现在内存中的虚拟地址
sh_offset	节区在目标文件中的偏移量
sh_size	节区大小
sh_link	节区头部索引链接
sh_info	附加信息
sh_addralign	节区的对齐方式
sh_entsize	节区中包含表项的大小

前面介绍了ELF文件的四个主要组成部分：ELF头部、程序头部、节区和节区头部。这四个部分是相互关联的。ELF头部确定了程序头部和节区头部的位置和大小。程序头部定义了在运行状态时，每个段的属性和虚拟地址，以及包含什么节区。节区头部定义了每个节区在目标文件中的位置，特别定义了节区在内存的虚拟地址，以及是否具有执行属性。

这四个部分的复杂关系可以用下面的图表示：



图中的Program Header Item指明了段的起始虚拟地址，也指明了属于这个段的节区。运行状态时，属于这个段的节区将被装载到内存中。目标文件的Section Header Item指明了节区的偏移量，名称和其他的相关属性。

嵌入代码的方法

在了解了ELF文件格式之后，我们就可以开始考虑如何将代码嵌入到一个可执行目标文件中去。根据上文对ELF文件格式的说明，首先，需要嵌入的代码应该被添加到目标文件中的某个节区中，并且在可执行目标文件运行时，被加载到可运行的段中。然后，修改ELF头部，将程序入口地址指向嵌入代码在段中位置。这样，目标程序在被执行时，嵌入的代码将首先被运行。

操作的流程为：

1. 确定可执行目标文件的入口地址。
2. 根据入口地址找到可执行的段。
3. 根据段在文件中的偏移量和大小，找到属于这个段的最后一个节区。设为A。
4. 将嵌入代码添加到节区A中。
5. 增加段的大小。增加值为嵌入代码的长度。

6. 修改节区A的节区头部，增加节区A的大小。增加值为嵌入代码的长度。

7. 修改位于节区A之后所有节区的节区头部的偏移量，增加值为嵌入代码的长度。

8. 修改ELF头部的入口地址，指向添加的代码。

ELF头部，程序头部和节区头部的结构都在elf.h中被定义。这个头文件位于/usr/include目录下。在代码中简单的包含这个文件，就可对各个头部的数据进行处理。

首先，打开目标文件，读取ELF头部，得到最初的入口地址。

```
char elf_ehdr[sizeof(Elf32_Ehdr)];
Elf32_Ehdr *p_ehdr;
p_ehdr = (Elf32_Ehdr *)elf_ehdr;
origfile = open("elf", O_RDONLY); /* 目标文件名为elf */

ret = read(origfile, elf_ehdr, sizeof(elf_ehdr));
orgi_entry = p_ehdr->e_entry; /* 确定入口地址 */
```

读取了ELF头部后，立刻读取程序头部。程序头部由多个单项组成，因此，这里用循环处理。

```
char elf_phdr[sizeof(Elf32_Phdr)];
Elf32_Phdr *p_phdr;
p_phdr = (Elf32_Phdr *)elf_phdr;
for (i=0; i<(int)p_ehdr->e_phnum; i++){
/* e_phnum是程序头部单项的数量 */
       read(origfile, elf_phdr, sizeof(elf_phdr));
       if (p_phdr->p_paddr < orgi_entry \
       /* 如果入口地址在段的地址空间中 */
       && (p_phdr->p_paddr + p_phdr->p_filesz)
              > orgi_entry)
       {
       program_head_vaddr = p_phdr->p_vaddr;
       /* 找到所需的程序段 */
       program_head_size  = p_phdr->p_filesz;
       /* 保存段的虚拟地址和大小 */
       }
}
```

程序头部之后是节区，我们需要跳过这些节区，去读取节区的头部。根据前面得到段的虚拟地址和大小，找到我们需要的节区。新的入口地址将是这个节区的尾部。

```
ret = lseek(origfile, \
(int)p_ehdr->e_shoff - sizeof(elf_ehdr)-
(int)p_ehdr->e_phnum*sizeof(elf_phdr), \
SEEK_CUR);  /* 越过程序头部与节区头部之间的节区 */
for (i=0; i<(int)p_ehdr->e_shnum; i++){
       read(origfile, elf_shdr, sizeof(elf_shdr));
       if (p_shdr->sh_addr+p_shdr->sh_size ==
program_head_vaddr+program_head_size){
entry_section_offset = p_shdr->sh_offset;
/* 找到所要的节区 */
entry_section_size   = p_shdr->sh_size;
/ * 保存这个节区的偏移量和大小 */
new_entry = p_shdr->sh_addr + p_shdr->sh_size;
/* 新的入口地址 */
       }
}
```

至此，我们已经找到必要的位置，可以将我们的代码嵌入到目标文件中。嵌入的代码也必须是目标码，也就是编译过后的机器码。嵌入的代码在执行后，应该返回到最初的目标文件入口点。这通常是利用一个jmp指令来完成的。此外，ELF格式的可执行目标文件存在一个特殊的限制条件，即：段的虚拟地址和段在文件的偏移量对页大小的余数必须相等。举例来说，设某个段的偏移量为0x0014a0，虚拟地址为0x080494a0。Linux的页大小为4KB（4096字节）。那么：

```
0x0014a0 mod 4096 == 0x080494a0 mod 4096
```

假设我们嵌入的代码大小小于4096，但是为了满足这个限制，我们必须嵌入4096字节的代码。不足的可以使用nop指令填充。

在下面的例子中，我们嵌入两个指令，用于跳转到最初的入口地址。由于指令只有7字节，因此，我们用nop指令补足至4096字节。代码中的newfile是新创建的文件，用于保存嵌入代码之后的目标文件。在将parasize写入newfile时，目标文件elf中的其他部分已经被写到newfile中，相关的参数也已经被修改。下面的代码显示了如何写入补齐的指令，并跳转到目标文件的最初入口。

```
char nop[]={0x90};  /* nop */
char parasize[]={0xbd, 0x00, 0x00, 0x00, 0x00,
0xff, 0xe5}; /* mov $0x00000000, %ebp*/

 /* jmp %ebp */
struct _jump {
       char opcode_mov;
       int addr;
       short opcode_jmp;
}__attribute__((packed));
jump = (struct _jump *)parasize;

jump->addr = orgi_entry;  /* 最初的入口地址 */
write(newfile, parasize, sizeof(parasize));
for (i=0; i<page_size-sizeof(parasize); i++)
/* page_size为页大小，值为4096 */
write(newfile, nop, 1);
```

至此，我们新创建的newfile中已经包含了嵌入的代码。在这个文件的运行之初，嵌入的代码将先被执行。本例中，嵌入的代码只有两条：

```
mov $0xaddress, %ebp
jmp %ebp
```

在嵌入代码运行结束后，通过jmp指令跳转到最初的入口，执行原目标文件中代码。■

作者简介

杨广翔，中兴通讯的软件工程师。他目前在中国南京中兴通讯工作，从事嵌入式Linux产品的开发。

■ 责任编辑：赵健平 (zhaojp@csdn.net)