最近研究了一下elf文件格式，发现好多资料写的都比较繁琐，可能会严重打击学习者的热情，我把自己研究的结果和大家分享，希望我的描述能够简洁一些。

一、基础知识

elf是一种文件格式，用于存储Linux程序. 它内部都有一些什么信息呢？大概包括编制好的计算机指令，数据，计算机在需要的时候把这个文件读取到内存中，cpu就可以从内存中一条一条的读取指令来执行了。

所以说想明白elf格式，我们应该了解一下计算机执行程序需要那些信息。所以这一节，我们补充一些计算机系统的基础知识。

进程和虚拟内存：

Linux系统给每个进程分配了4GB的空间，其中 0xC0000000到0xFFFFFFFF 这个地址段是留给系统使用的，主要用于系统（linux 内核)和进程通信和交换数据， 用户可以使用3GB的空间从(0x00000000-0xBFFFFFFF).

其实计算机的内存是没有那么大的，比如我们实际使用的计算机只有2G,以前更小，只有几百M,而且一台计算机上不只运行一个进程，一个占用4G，如果有10个进程，那就得着用40G了，哪有那么打的内存呢？其实这个不要紧，因为操作系统分配给用户的是虚拟内存，程序要可以使用3个G的内存。至于操作系统怎样把虚拟内存转化成物理内存，对于开发应用程序的工程师来说，是不需要了解的。我们直接使用虚拟内存就可以了，而不用担心其它进程会侵犯到你的内存空间。

进程的创建和运行进程的创建和运行：

大致经历了以下步骤

1.用户请求运行程序时，操作系统会读取存储在磁盘上的可执行文件，在linux系统上这个文件就是我们的elf格式文件，为用户分配4G的虚拟内存空间，

2. 根据文件的信息指示，把不同的文件内容放到为你分配的这3G虚拟内存

3. 然后根据文件的指示，系统设置设置代码段和数据段寄存器

4.然后根据文件的指示， 跳转到用户的代码的入口地址（一般就是我们的main函数)

5.从main开始，计算机就一条一条的执行我们给的指令，处理我们的数据了，直到我们程序结束。虽然在这个过程中，系统会多次切换到其他进程，但对用户程序来说没有影响，我们可以认为计算机只为我们服务。

通过以上我们多次看到计算机是根据文件指示这样的语言，所以学习elf 首先要理解elf指示了那些信息。

二、可执行的elf文件。

elf文件分三种类型： 1、目标文件（通常是.o); 2、可执行文件(我们的运行文件) 3、动态库(.so)

我们先讲一下可执行文件。

可执行文件一般分成4个部分，能扩展，我们理解这4部分就够了。

1、elf文件头，这个文件是对elf文件整体信息的描述，在32位系统下是56的字节，在64位系统下是64个字节。

对于可执行文件来说，文件头包含的一下信息与进程启动相关

e_entry　　程序入口地址

e_phoff　　segment偏移

e_phnum　segment数量

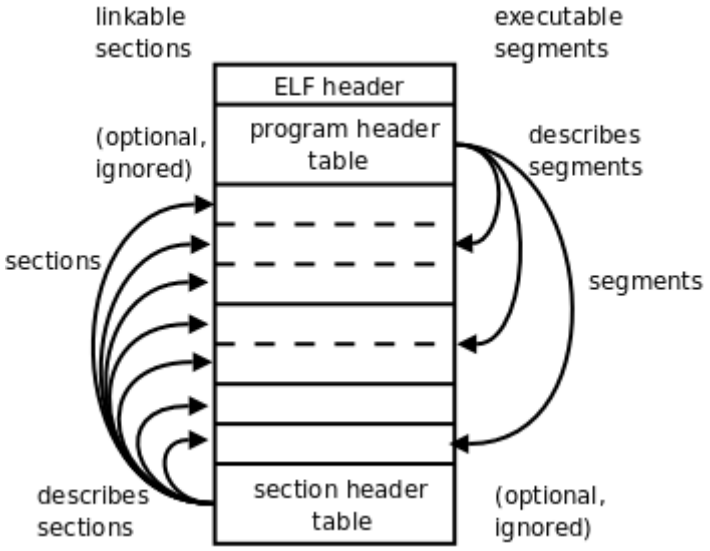
2. segment表， 这个表是加载指示器，操作系统（确切的说是加载器，有些elf文件，比如操作系统内核，是由其他程序加载的）,该表的结构非常重要。

```
typedef struct
{
  Elf64_Word    p_type;           /* Segment type */
  Elf64_Word    p_flags;       /* Segment flags */  /*segment权限，6表示可读写，5表示可读可执行
  Elf64_Off    p_offset;       /* Segment file offset */    /*段在文件中的偏移*/
  Elf64_Addr    p_vaddr;       /* Segment virtual address */   /*虚拟内存地址，这个表示内存中的
  Elf64_Addr    p_paddr;       /* Segment physical address  /*物理内存地址，对应用程序来说，这个字段无用*/
  Elf64_Xword    p_filesz;       /* Segment size in file */       /*段在文件中的长度*/
  Elf64_Xword    p_memsz;       /* Segment size in memory */       /在内存中的长度，一般和p_filesz的值一样*/
  Elf64_Xword    p_align;       /* Segment alignment */                /* 段对齐*/
} Elf64_Phdr;
```

3. elf的主题，对于可执行文件来说，最主要的就是数据段和代码段

4. section表，对可执行文件来说，没有用，在链接的时候有用，是对代码段数据段在链接是的一种描述。

整个elf文件的组成可以使用下图来描述



该图片使用的是Linux C编程作者 宋劲斌的图片

上图program header table 实际上就是我们说的segment table. segments 是从运行的角度来描述elf文件， sections是从链接的角度来描述elf文件的。

本节我们只将elf文件的执行，所以我们只讲segment相关的内容。

我们将通过一个例子来讲解系统加载elf的过程(64位平台)。

我们编写一个简单的汇编程序

```
 .section .data
.global data_item
data_item:
.long 3,67,28
.section .text
.global _start
_start:
    mov $1,%eax
    mov $4,%ebx
    int $0x80
```

编译链接后生成hello文件，我们分析hello文件.

执行：readelf -h ../asm/hello （readelf -h 是读取elf文件头的命令)

```
  ELF Header:
    Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
    Class:                             ELF64
    Data:                              2's complement, little endian
    Version:                           1 (current)
    OS/ABI:                            UNIX - System V
    ABI Version:                       0
```

Type: EXEC (Executable file)
Machine: Advanced Micro Devices X86-64
Version: 0x1
Entry point address: 0x4000b0 //程序的入口地址是0x4000b0
Start of program headers: 64 (bytes into file) //segment表在文件64字节偏移处
Start of section headers: 240 (bytes into file)
Flags: 0x0
Size of this header: 64 (bytes)
Size of program headers: 56 (bytes) //segment头项的长度是56字节（32系统是32字节)
Number of program headers: 2
Size of section headers: 64 (bytes)
Number of section headers: 6
Section header string table index: 3

对于程序的装载，我们关心这三项：

Entry point address: 0x4000b0 //程序的入口地址是0x4000b0
Start of program headers: 64 (bytes into file) //segment表在文件64字节偏移处
Size of program headers: 56 (bytes) //segment头项的长度是56字节（32系统是32字节)

以上内容告诉我们segment表在文件的64字节处，我们看看64字节处有什么内容。

执行 readelf -l ../asm/hello 输出segments信息。(readelf -l 读取segments)

Program Headers:

Type	Offset	VirtAddr	PhysAddr		
	FileSiz	MemSiz	Flags	Align	
LOAD	0x0000000000000000	0x0000000000400000	0x0000000000400000		
	0x00000000000000bc	0x00000000000000bc	R E	200000	
LOAD	0x00000000000000bc	0x00000000006000bc	0x00000000006000bc		
	0x000000000000000c	0x000000000000000c	RW	200000	

Section to Segment mapping:

Segment Sections...

00 .text

01 .data

我们看到程序有两个segment ， 分别叫做.text 和.data

.text的Offset是0，FileSiz是0x0,MemSiz是0xbc, VirtAddr是0x400000,Flags是R E,表示加载起将把elf文件中从0字节开始直到oxbc处的内容加载到虚拟内存中的0x400000处，占用0xbc长度的内存。设置该内存的权限是RE(可读，可执行），这一段的内容正好是elf头，segments table,和代码段。

在看看elfheader 的e_entry 的地址 0x4000b0，这个地址正好是代码段的起始地址。

.data的Offset是0，FileSiz是0xbc,MemSiz是0x0c, VirtAddr是0x6000bc,Flags是R W,表示加载起将把elf文件中从bc字节开始直到oxbc + 0xc处的内容加载到虚拟内存中的0x6000bc处，占用0x0c长度的内存。设置该内存的权限是RE(可读，可执行)

为什么数据段的其实地址是0x6000bc,而不是0x6000000呢，这是由Align决定的，Align决定内存和磁盘以1M为单位进行映射，在文件中.data 和.text处于一个页面中，在映射的时候，直接把整个页面都映射到了0x6000000处，所以把数据段的偏移设置成了0x60000bc,0x600000到0x6000bc的内容不使用。

有了以上内容，系统就可以根据elf文件创建进程了。

下一节，我们将讲述静态链接编译的过程。