hw期间，在公司事情比较少，就把之前没有系统研究过的二进制格式、linux c程序的从源码到二进制文件的转换以及二进制文件的装载和执行，再从头到尾捋一遍，把底层的原理弄清楚，让以前隐藏在迷雾中的摸棱两可的知识，一点一点呈现出来，也能让自己的基础更扎实些。

1. c源码设计

经过对各种设计类、架构类书籍的洗礼，应该可以“设计”出比较牛逼的软件架构了，然后就拿着各种编辑器啊、ide啊开始一顿写代码。
设计这块儿的内容，主要集中c的基础、c的高级技术、linux系统编程、linux网络编程、并行编程、IPC技术、内核编程等等，基础的编程技术的学习。还有操作系统、编译原理、网络原理、计算机体系结构、算法、架构设计、重构、系统分析、设计模式等等技术的学习。然后就是夜以继日无休止的撸代码，打副本升级。

2. c源码编写

然后在经历无数昼夜的百度、狗狗之后，终于把贼牛逼的架构实现了，虽然对写的什么东西一脸懵逼，但不耽误完成领导布置的任务，妹汁儿汁儿。

吭哧吭哧，终于把代码写完了，然后就是编译、执行。好像很自然的操作，但是这两部操作到底干了啥？我完全不知道，完全是傻子一样，等着计算机帮我处理好。所以，后面进入到linux c程序的编译阶段。

3. gcc预处理：cpp

4. gcc编译：cc1

5. gcc汇编：as

elf文件格式

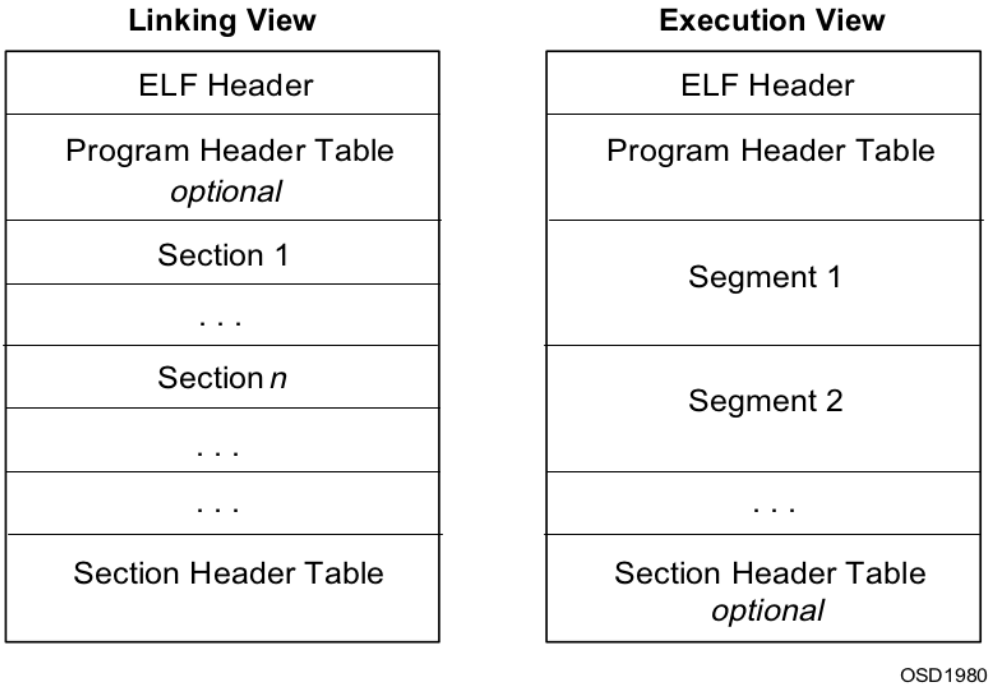
汇编之后会产生relocatable file，relocatable file是c程序生命周期中第一个以elf格式存在的文件，后面还有executable file和shared object file都是以elf格式存在，并且在elf定义中，都属于object file，因此在这里记录elf文件格式。
elf截至当前为止，分为两部分。一个是32位标准定义，一个是64位补充定义。

> elf 32位标准定义
> elf 64位补充定义

鉴于目前64位已比较普遍，所以在记录时，直接合并32位和64位定义中的相关数据结构定义。

---

object file会参与到程序的链接和执行过程中，因此elf文件划分出链接视图和执行视图，两种视图来体现链接和执行过程中的不同要素。



> **ELF header**：描述了整个elf的结构和组织。
> **Sections**：包含所有“链接视图”所需的信息。
> **Segments**：包含所有“执行视图”所需的信息。
> **program header table**：定义如何创建process image。
> **section header table**：包含所有section的全部信息。

Table 1. ELF-64 Data Types

Name	*Size*	*Alignment*	*Purpose*
Elf64_Addr	8	8	Unsigned program address
Elf64_Off	8	8	Unsigned file offset
Elf64_Half	2	2	Unsigned medium integer
Elf64_Word	4	4	Unsigned integer
Elf64_Sword	4	4	Signed integer
Elf64_Xword	8	8	Unsigned long integer
Elf64_Sxword	8	8	Signed long integer
unsigned char	1	1	Unsigned small integer

```
typedef struct
{
        unsigned char   e_ident[16];    /* ELF identification */
        Elf64_Half      e_type;         /* Object file type */
        Elf64_Half      e_machine;      /* Machine type */
        Elf64_Word      e_version;      /* Object file version */
        Elf64_Addr      e_entry;        /* Entry point address */
        Elf64_Off       e_phoff;        /* Program header offset */
        Elf64_Off       e_shoff;        /* Section header offset */
        Elf64_Word      e_flags;        /* Processor-specific flags */
        Elf64_Half      e_ehsize;       /* ELF header size */
        Elf64_Half      e_phentsize;    /* Size of program header entry */
        Elf64_Half      e_phnum;        /* Number of program header entries */
        Elf64_Half      e_shentsize;    /* Size of section header entry */
        Elf64_Half      e_shnum;        /* Number of section header entries */
        Elf64_Half      e_shstrndx;     /* Section name string table index */
} Elf64_Ehdr;
```

Figure 2. ELF-64 Header

Table 2. ELF Identification, e_ident

Name	*Value*	*Purpose*
EI_MAG0	0	File identification
EI_MAG1	1	
EI_MAG2	2	
EI_MAG3	3	
EI_CLASS	4	File class
EI_DATA	5	Data encoding
EI_VERSION	6	File version
EI_OSABI	7	OS/ABI identification
EI_ABIVERSION	8	ABI version
EI_PAD	9	Start of padding bytes
EI_NIDENT	16	Size of e_ident[]

Table 3. Object File Classes, e_ident[EI_CLASS]

Name	*Value*	*Meaning*
ELFCLASS32	1	32-bit objects
ELFCLASS64	2	64-bit objects

Table 4. Data Encodings, e_ident[EI_DATA]

Name	*Value*	*Meaning*
ELFDATA2LSB	1	Object file data structures are little-endian
ELFDATA2MSB	2	Object file data structures are big-endian

Table 5. Operating System and ABI Identifiers, e_ident[EI_OSABI]

Name	*Value*	*Meaning*
ELFOSABI_SYSV	0	System V ABI
ELFOSABI_HPUX	1	HP-UX operating system
ELFOSABI_STANDALONE	255	Standalone (embedded) application

Table 6. Object File Types, e_type

Name	*Value*	*Meaning*
ET_NONE	0	No file type
ET_REL	1	Relocatable object file
ET_EXEC	2	Executable file
ET_DYN	3	Shared object file
ET_CORE	4	Core file
ET_LOOS	0xFE00	Environment-specific use
ET_HIOS	0xFEFF	
ET_LOPROC	0xFF00	Processor-specific use
ET_HIPROC	0xFFFF	

实例:

```
[root@localhost tmp]# readelf -h /bin/ls
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:               0x404324
```

```
  Start of program headers:          64 (bytes into file)
  Start of section headers:          115688 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         9
  Size of section headers:           64 (bytes)
  Number of section headers:         30
  Section header string table index: 29
```

```
[root@localhost tmp]# xxd -s 0x0 -l 0x40 /bin/ls
0000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
0000010: 0200 3e00 0100 0000 2443 4000 0000 0000  ..>.....$C@.....
0000020: 4000 0000 0000 0000 e8c3 0100 0000 0000  @...............
0000030: 0000 0000 4000 3800 0900 4000 1e00 1d00  ....@.8...@.....
```

6. gcc链接：collect2:ld//lib64/ld-linux-x86-64.so.2

动态链接

实验：

`[root@localhost 3]# cat Program1.c`

```
#include "Lib.h"

int main(void) {
   foobar(1);
   return 0;
}
```

`[root@localhost 3]# cat Program2.c`

```
#include "Lib.h"

int main(void) {
   foobar(2);
   return 0;
}
```

`[root@localhost 3]# cat Lib.c`

```
#include <stdio.h>

void foobar(int i) {
    printf("Printing from Lib.so %d\n", i);
}
```

`[root@localhost 3]# cat Lib.h`

```
#ifndef LIB_H
#define LIB_H

void foobar(int i);

#endif
```

```
[root@localhost 3]# gcc -fPIC -shared -o Lib.so Lib.c
[root@localhost 3]# gcc -o Program1 Program1.c ./Lib.so
[root@localhost 3]# gcc -o Program2 Program2.c ./Lib.so
```

7. elf装载

program headers

```
int main(void) {
    int a = 0x12345678;

    return 0;
}
```

```
[root@localhost tmp]# readelf -l test

Elf file type is EXEC (Executable file)
Entry point 0x400400
There are 9 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000400040 0x0000000000400040
                 0x00000000000001f8 0x00000000000001f8  R E    8
  INTERP         0x0000000000000238 0x0000000000400238 0x0000000000400238
                 0x000000000000001c 0x000000000000001c  R      1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x000000000000069c 0x000000000000069c  R E    200000
  LOAD           0x0000000000000e18 0x0000000000600e18 0x0000000000600e18
                 0x0000000000000220 0x0000000000000228  RW     200000
  DYNAMIC        0x0000000000000e28 0x0000000000600e28 0x0000000000600e28
                 0x00000000000001d0 0x00000000000001d0  RW     8
  NOTE           0x0000000000000254 0x0000000000400254 0x0000000000400254
                 0x0000000000000044 0x0000000000000044  R      4
  GNU_EH_FRAME   0x0000000000000574 0x0000000000400574 0x0000000000400574
                 0x0000000000000034 0x0000000000000034  R      4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     10
  GNU_RELRO      0x0000000000000e18 0x0000000000600e18 0x0000000000600e18
                 0x00000000000001e8 0x00000000000001e8  R      1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
```

```
   02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr
.gnu.version .gnu.version_r .rela.dyn .rela.plt .init .plt .text .fini .rodata
.eh_frame_hdr .eh_frame
   03     .init_array .fini_array .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
   06     .eh_frame_hdr
   07
   08     .init_array .fini_array .dynamic .got
```

```
[root@localhost tmp]# xxd -s 0x40 -l 0x1f8 /tmp/test
0000040: 0600 0000 0500 0000 4000 0000 0000 0000  ........@.......
0000050: 4000 4000 0000 0000 4000 4000 0000 0000  @.@.....@.@.....
0000060: f801 0000 0000 0000 f801 0000 0000 0000  ................
0000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
0000080: 3802 0000 0000 0000 3802 4000 0000 0000  8.......8.@.....
0000090: 3802 4000 0000 0000 1c00 0000 0000 0000  8.@.............
00000a0: 1c00 0000 0000 0000 0100 0000 0000 0000  ................
00000b0: 0100 0000 0500 0000 0000 0000 0000 0000  ................
00000c0: 0000 4000 0000 0000 0000 4000 0000 0000  ..@.......@.....
00000d0: 9c06 0000 0000 0000 9c06 0000 0000 0000  ................
00000e0: 0000 2000 0000 0000 0100 0000 0600 0000  .. ............
00000f0: 180e 0000 0000 0000 180e 6000 0000 0000  ..........`.....
0000100: 180e 6000 0000 0000 2002 0000 0000 0000  ..`..... .......
0000110: 2802 0000 0000 0000 0000 2000 0000 0000  (......... .....
0000120: 0200 0000 0600 0000 280e 0000 0000 0000  ........(.......
0000130: 280e 6000 0000 0000 280e 6000 0000 0000  (.`.....(.`.....
0000140: d001 0000 0000 0000 d001 0000 0000 0000  ................
0000150: 0800 0000 0000 0000 0400 0000 0400 0000  ................
0000160: 5402 0000 0000 0000 5402 4000 0000 0000  T.......T.@.....
0000170: 5402 4000 0000 0000 4400 0000 0000 0000  T.@.....D.......
0000180: 4400 0000 0000 0000 0400 0000 0000 0000  D...............
0000190: 50e5 7464 0400 0000 7405 0000 0000 0000  P.td....t.......
00001a0: 7405 4000 0000 0000 7405 4000 0000 0000  t.@.....t.@.....
00001b0: 3400 0000 0000 0000 3400 0000 0000 0000  4.......4.......
00001c0: 0400 0000 0000 0000 51e5 7464 0600 0000  ........Q.td....
00001d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001f0: 0000 0000 0000 0000 1000 0000 0000 0000  ................
0000200: 52e5 7464 0400 0000 180e 0000 0000 0000  R.td............
0000210: 180e 6000 0000 0000 180e 6000 0000 0000  ..`.......`.....
0000220: e801 0000 0000 0000 e801 0000 0000 0000  ................
0000230: 0100 0000 0000 0000                      ........
```

```
[root@localhost tmp]# xxd -s 0x40 -l 56 /tmp/test
0000040: 0600 0000 0500 0000 4000 0000 0000 0000  ........@.......
0000050: 4000 4000 0000 0000 4000 4000 0000 0000  @.@.....@.@.....
0000060: f801 0000 0000 0000 f801 0000 0000 0000  ................
0000070: 0800 0000 0000 0000                      ........
```

> ELF二进制文件load到内存并执行，内核源码：
> https://github.com/GabrielJiang-J/study_information/blob/master/%E4%BA%8C%E8%BF%9B%E5%88%B6%E5%88%86%E6%9E%90/linux/elf-64-gen.pdf
> linux-2.6.34/fs/binfmt_elf.c:elf_format.load_binary
> linux-2.6.34/arch/ia64/kernel/process.c:sys_execve

8. elf执行